

# **The 802.11 subsystems – for kernel developers**

**Explaining wireless 802.11  
networking in the Linux kernel**

**Johannes Berg**

**`johannes@sipsolutions.net`**

# **The 802.11 subsystems – for kernel developers: Explaining wireless 802.11 net-working in the Linux kernel**

by Johannes Berg

Copyright © 2007-2009 Johannes Berg

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

This documentation is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this documentation; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more details see the file COPYING in the source distribution of Linux.

# **The cfg80211 subsystem**

## **The cfg80211 subsystem**

cfg80211 is the configuration API for 802.11 devices in Linux. It bridges userspace and drivers, and offers some utility functionality associated with 802.11. cfg80211 must, directly or indirectly via mac80211, be used by all modern wireless drivers in Linux, so that they offer a consistent API through nl80211. For backward compatibility, cfg80211 also offers wireless extensions to userspace, but hides them from drivers completely.

Additionally, cfg80211 contains code to help enforce regulatory spectrum use restrictions.

# Table of Contents

<b>1. Device registration .....</b>	<b>1</b>
enum ieee80211_band .....	1
enum ieee80211_channel_flags .....	2
struct ieee80211_channel .....	3
enum ieee80211_rate_flags .....	5
struct ieee80211_rate .....	6
struct ieee80211_sta_ht_cap .....	7
struct ieee80211_supported_band .....	9
enum cfg80211_signal_type .....	10
enum wiphy_params_flags .....	11
enum wiphy_flags .....	12
struct wiphy .....	13
struct wireless_dev .....	17
wiphy_new .....	19
wiphy_register .....	20
wiphy_unregister .....	21
wiphy_free .....	22
wiphy_name .....	23
wiphy_dev .....	23
wiphy_priv .....	24
priv_to_wiphy .....	25
set_wiphy_dev .....	25
wdev_priv .....	26
<b>2. Actions and configuration .....</b>	<b>29</b>
struct cfg80211_ops .....	29
struct vif_params .....	36
struct key_params .....	37
enum survey_info_flags .....	38
struct survey_info .....	39
struct beacon_parameters .....	41
enum plink_actions .....	42
struct station_parameters .....	43
enum station_info_flags .....	44
enum rate_info_flags .....	46
struct rate_info .....	47
struct station_info .....	48
enum monitor_flags .....	51
enum mpath_info_flags .....	52
struct mpath_info .....	53
struct bss_parameters .....	55

struct ieee80211_txq_params.....	56
struct cfg80211_crypto_settings .....	57
struct cfg80211_auth_request .....	58
struct cfg80211_assoc_request .....	60
struct cfg80211_deauth_request .....	61
struct cfg80211_disassoc_request.....	62
struct cfg80211_ibss_params.....	64
struct cfg80211_connect_params.....	65
struct cfg80211_pmksa .....	67
cfg80211_send_rx_auth.....	68
cfg80211_send_auth_timeout.....	69
__cfg80211_auth_canceled.....	70
cfg80211_send_rx_assoc .....	71
cfg80211_send_assoc_timeout .....	72
cfg80211_send_deauth .....	73
__cfg80211_send_deauth .....	74
cfg80211_send_disassoc.....	75
__cfg80211_send_disassoc.....	76
cfg80211_ibss_joined .....	77
cfg80211_connect_result .....	78
cfg80211_roamed .....	79
cfg80211_disconnected.....	81
cfg80211_ready_on_channel .....	82
cfg80211_remain_on_channel_expired .....	83
cfg80211_new_sta .....	84
cfg80211_rx_mgmt.....	85
cfg80211_mgmt_tx_status .....	86
cfg80211_cqm_rssi_notify .....	87
cfg80211_michael_mic_failure .....	88
<b>3. Scanning and BSS list handling.....</b>	<b>91</b>
struct cfg80211_ssid .....	91
struct cfg80211_scan_request.....	92
cfg80211_scan_done.....	93
struct cfg80211_bss .....	94
cfg80211_inform_bss_frame .....	96
cfg80211_inform_bss .....	97
cfg80211_unlink_bss .....	99
cfg80211_find_ie .....	100
ieee80211_bss_get_ie .....	101

<b>4. Utility functions.....</b>	<b>103</b>
ieee80211_channel_to_frequency.....	103
ieee80211_frequency_to_channel.....	103
ieee80211_get_channel.....	104
ieee80211_get_response_rate .....	105
ieee80211_hdrlen .....	106
ieee80211_get_hdrlen_from_skb.....	107
struct ieee80211_radiotap_iterator.....	107
<b>5. Data path helpers .....</b>	<b>111</b>
ieee80211_data_to_8023 .....	111
ieee80211_data_from_8023.....	111
ieee80211_amsdu_to_8023s .....	112
cfg80211_classify8021d .....	114
<b>6. Regulatory enforcement infrastructure .....</b>	<b>115</b>
regulatory_hint.....	115
wiphy_apply_custom_regulatory.....	116
freq_reg_info.....	117
<b>7. RFkill integration.....</b>	<b>119</b>
wiphy_rfkill_set_hw_state .....	119
wiphy_rfkill_start_polling .....	119
wiphy_rfkill_stop_polling.....	120
<b>8. Test mode .....</b>	<b>123</b>
cfg80211_testmode_alloc_reply_skb .....	123
cfg80211_testmode_reply.....	124
cfg80211_testmode_alloc_event_skb .....	125
cfg80211_testmode_event .....	126



# Chapter 1. Device registration

In order for a driver to use `cfg80211`, it must register the hardware device with `cfg80211`. This happens through a number of hardware capability structs described below.

The fundamental structure for each device is the 'wiphy', of which each instance describes a physical wireless device connected to the system. Each such wiphy can have zero, one, or many virtual interfaces associated with it, which need to be identified as such by pointing the network interface's `ieee80211_ptr` pointer to a struct `wireless_dev` which further describes the wireless part of the interface, normally this struct is embedded in the network interface's private data area. Drivers can optionally allow creating or destroying virtual interfaces on the fly, but without at least one or the ability to create some the wireless device isn't useful.

Each wiphy structure contains device capability information, and also has a pointer to the various operations the driver offers. The definitions and structures here describe these capabilities in detail.

## enum ieee80211\_band

**LINUX**

Kernel Hackers Manual July 2011

### Name

`enum ieee80211_band` — supported frequency bands

### Synopsis

```
enum ieee80211_band {
    IEEE80211_BAND_2GHZ,
    IEEE80211_BAND_5GHZ,
    IEEE80211_NUM_BANDS
};
```

## Constants

IEEE80211\_BAND\_2GHZ

2.4GHz ISM band

IEEE80211\_BAND\_5GHZ

around 5GHz band (4.9-5.7)

IEEE80211\_NUM\_BANDS

number of defined bands

## Device registration

The bands are assigned this way because the supported bitrates differ in these bands.

## enum ieee80211\_channel\_flags

### LINUX

Kernel Hackers Manual July 2011

### Name

enum ieee80211\_channel\_flags — channel flags

### Synopsis

```
enum ieee80211_channel_flags {  
    IEEE80211_CHAN_DISABLED,  
    IEEE80211_CHAN_PASSIVE_SCAN,  
    IEEE80211_CHAN_NO_IBSS,  
    IEEE80211_CHAN_RADAR,  
    IEEE80211_CHAN_NO_HT40PLUS,  
    IEEE80211_CHAN_NO_HT40MINUS  
};
```

## Constants

IEEE80211\_CHAN\_DISABLED

This channel is disabled.

IEEE80211\_CHAN\_PASSIVE\_SCAN

Only passive scanning is permitted on this channel.

IEEE80211\_CHAN\_NO\_IBSS

IBSS is not allowed on this channel.

IEEE80211\_CHAN\_RADAR

Radar detection is required on this channel.

IEEE80211\_CHAN\_NO\_HT40PLUS

extension channel above this channel is not permitted.

IEEE80211\_CHAN\_NO\_HT40MINUS

extension channel below this channel is not permitted.

## Description

Channel flags set by the regulatory control code.

## struct ieee80211\_channel

**LINUX**

Kernel Hackers Manual July 2011

### Name

struct ieee80211\_channel — channel definition

## Synopsis

```
struct ieee80211_channel {
    enum ieee80211_band band;
    u16 center_freq;
    u16 hw_value;
    u32 flags;
    int max_antenna_gain;
    int max_power;
    bool beacon_found;
    u32 orig_flags;
    int orig_mag;
    int orig_mpwr;
};
```

## Members

**band**

band this channel belongs to.

**center\_freq**

center frequency in MHz

**hw\_value**

hardware-specific value for the channel

**flags**

channel flags from enum `ieee80211_channel_flags`.

**max\_antenna\_gain**

maximum antenna gain in dBi

**max\_power**

maximum transmission power (in dBm)

**beacon\_found**

helper to regulatory code to indicate when a beacon has been found on this channel. Use `regulatory_hint_found_beacon` to enable this, this is useful only on 5 GHz band.

`orig_flags`

channel flags at registration time, used by regulatory code to support devices with additional restrictions

`orig_mag`

internal use

`orig_mpwr`

internal use

## Description

This structure describes a single channel for use with `cfg80211`.

## enum ieee80211\_rate\_flags

### LINUX

Kernel Hackers Manual July 2011

### Name

`enum ieee80211_rate_flags` — rate flags

### Synopsis

```
enum ieee80211_rate_flags {
    IEEE80211_RATE_SHORT_PREAMBLE,
    IEEE80211_RATE_MANDATORY_A,
    IEEE80211_RATE_MANDATORY_B,
    IEEE80211_RATE_MANDATORY_G,
    IEEE80211_RATE_ERP_G
};
```

## Constants

### IEEE80211\_RATE\_SHORT\_PREAMBLE

Hardware can send with short preamble on this bitrate; only relevant in 2.4GHz band and with CCK rates.

### IEEE80211\_RATE\_MANDATORY\_A

This bitrate is a mandatory rate when used with 802.11a (on the 5 GHz band); filled by the core code when registering the wiphy.

### IEEE80211\_RATE\_MANDATORY\_B

This bitrate is a mandatory rate when used with 802.11b (on the 2.4 GHz band); filled by the core code when registering the wiphy.

### IEEE80211\_RATE\_MANDATORY\_G

This bitrate is a mandatory rate when used with 802.11g (on the 2.4 GHz band); filled by the core code when registering the wiphy.

### IEEE80211\_RATE\_ERP\_G

This is an ERP rate in 802.11g mode.

## Description

Hardware/specification flags for rates. These are structured in a way that allows using the same bitrate structure for different bands/PHY modes.

## struct ieee80211\_rate

### LINUX

Kernel Hackers Manual July 2011

### Name

struct ieee80211\_rate — bitrate definition

## Synopsis

```
struct ieee80211_rate {
    u32 flags;
    u16 bitrate;
    u16 hw_value;
    u16 hw_value_short;
};
```

## Members

flags

rate-specific flags

bitrate

bitrate in units of 100 Kbps

hw\_value

driver/hardware value for this rate

hw\_value\_short

driver/hardware value for this rate when short preamble is used

## Description

This structure describes a bitrate that an 802.11 PHY can operate with. The two values *hw\_value* and *hw\_value\_short* are only for driver use when pointers to this structure are passed around.

## struct ieee80211\_sta\_ht\_cap

**LINUX**

## Name

`struct ieee80211_sta_ht_cap` — STA's HT capabilities

## Synopsis

```
struct ieee80211_sta_ht_cap {
    u16 cap;
    bool ht_supported;
    u8 ampdu_factor;
    u8 ampdu_density;
    struct ieee80211_mcs_info mcs;
};
```

## Members

`cap`

HT capabilities map as described in 802.11n spec

`ht_supported`

is HT supported by the STA

`ampdu_factor`

Maximum A-MPDU length factor

`ampdu_density`

Minimum A-MPDU spacing

`mcs`

Supported MCS rates

## Description

This structure describes most essential parameters needed to describe 802.11n HT capabilities for an STA.

# struct ieee80211\_supported\_band

## LINUX

Kernel Hackers Manual July 2011

### Name

struct ieee80211\_supported\_band — frequency band definition

### Synopsis

```
struct ieee80211_supported_band {
    struct ieee80211_channel * channels;
    struct ieee80211_rate * bitrates;
    enum ieee80211_band band;
    int n_channels;
    int n_bitrates;
    struct ieee80211_sta_ht_cap ht_cap;
};
```

### Members

channels

Array of channels the hardware can operate in in this band.

bitrates

Array of bitrates the hardware can operate with in this band. Must be sorted to give a valid “supported rates” IE, i.e. CCK rates first, then OFDM.

band

the band this structure represents

n\_channels

Number of channels in *channels*

n\_bitrates

Number of bitrates in *bitrates*

ht\_cap

HT capabilities in this band

## Description

This structure describes a frequency band a wiphy is able to operate in.

## enum cfg80211\_signal\_type

**LINUX**

Kernel Hackers Manual July 2011

## Name

enum cfg80211\_signal\_type — signal type

## Synopsis

```
enum cfg80211_signal_type {  
    CFG80211_SIGNAL_TYPE_NONE,  
    CFG80211_SIGNAL_TYPE_MBM,  
    CFG80211_SIGNAL_TYPE_UNSPEC  
};
```

## Constants

CFG80211\_SIGNAL\_TYPE\_NONE

no signal strength information available

CFG80211\_SIGNAL\_TYPE\_MBM

signal strength in mBm (100\*dBm)

CFG80211\_SIGNAL\_TYPE\_UNSPEC

signal strength, increasing from 0 through 100

## enum wiphy\_params\_flags

### LINUX

Kernel Hackers Manual July 2011

### Name

enum wiphy\_params\_flags — set\_wiphy\_params bitfield values

### Synopsis

```
enum wiphy_params_flags {
    WIPHY_PARAM_RETRY_SHORT,
    WIPHY_PARAM_RETRY_LONG,
    WIPHY_PARAM_FRAG_THRESHOLD,
    WIPHY_PARAM_RTS_THRESHOLD,
    WIPHY_PARAM_COVERAGE_CLASS
};
```

### Constants

WIPHY\_PARAM\_RETRY\_SHORT

wiphy->retry\_short has changed

WIPHY\_PARAM\_RETRY\_LONG

wiphy->retry\_long has changed

WIPHY\_PARAM\_FRAG\_THRESHOLD

wiphy->frag\_threshold has changed

WIPHY\_PARAM\_RTS\_THRESHOLD

wiphy->rts\_threshold has changed

WIPHY\_PARAM\_COVERAGE\_CLASS

coverage class changed

## enum wiphy\_flags

### LINUX

Kernel Hackers Manual July 2011

### Name

enum wiphy\_flags — wiphy capability flags

### Synopsis

```
enum wiphy_flags {
    WIPHY_FLAG_CUSTOM_REGULATORY,
    WIPHY_FLAG_STRICT_REGULATORY,
    WIPHY_FLAG_DISABLE_BEACON_HINTS,
    WIPHY_FLAG_NETNS_OK,
    WIPHY_FLAG_PS_ON_BY_DEFAULT,
    WIPHY_FLAG_4ADDR_AP,
    WIPHY_FLAG_4ADDR_STATION,
    WIPHY_FLAG_CONTROL_PORT_PROTOCOL,
    WIPHY_FLAG_IBSS_RSN
};
```

### Constants

WIPHY\_FLAG\_CUSTOM\_REGULATORY

tells us the driver for this device has its own custom regulatory domain and cannot identify the ISO / IEC 3166 alpha2 it belongs to. When this is enabled

we will disregard the first regulatory hint (when the initiator is `REGDOM_SET_BY_CORE`).

#### `WIPHY_FLAG_STRICT_REGULATORY`

tells us the driver for this device will ignore regulatory domain settings until it gets its own regulatory domain via its `regulatory_hint` unless the regulatory hint is from a country IE. After its gets its own regulatory domain it will only allow further regulatory domain settings to further enhance compliance. For example if channel 13 and 14 are disabled by this regulatory domain no user regulatory domain can enable these channels at a later time. This can be used for devices which do not have calibration information guaranteed for frequencies or settings outside of its regulatory domain.

#### `WIPHY_FLAG_DISABLE_BEACON_HINTS`

enable this if your driver needs to ensure that passive scan flags and beaconing flags may not be lifted by `cfg80211` due to regulatory beacon hints. For more information on beacon hints read the documentation for `regulatory_hint_found_beacon`

#### `WIPHY_FLAG_NETNS_OK`

if not set, do not allow changing the netns of this wiphy at all

#### `WIPHY_FLAG_PS_ON_BY_DEFAULT`

if set to true, powersave will be enabled by default -- this flag will be set depending on the kernel's default on `wiphy_new`, but can be changed by the driver if it has a good reason to override the default

#### `WIPHY_FLAG_4ADDR_AP`

supports 4addr mode even on AP (with a single station on a VLAN interface)

#### `WIPHY_FLAG_4ADDR_STATION`

supports 4addr mode even as a station

#### `WIPHY_FLAG_CONTROL_PORT_PROTOCOL`

This device supports setting the control port protocol ethertype. The device also honours the `control_port_no_encrypt` flag.

#### `WIPHY_FLAG_IBSS_RSN`

The device supports IBSS RSN.

# struct wiphy

## LINUX

Kernel Hackers Manual July 2011

## Name

struct wiphy — wireless hardware description

## Synopsis

```
struct wiphy {
    u8 perm_addr[ETH_ALEN];
    u8 addr_mask[ETH_ALEN];
    struct mac_address * addresses;
    const struct ieee80211_txrx_types * mgmt_types;
    u16 n_addresses;
    u16 interface_modes;
    u32 flags;
    enum cfg80211_signal_type signal_type;
    int bss_priv_size;
    u8 max_scan_ssids;
    u16 max_scan_ie_len;
    int n_cipher_suites;
    const u32 * cipher_suites;
    u8 retry_short;
    u8 retry_long;
    u32 frag_threshold;
    u32 rts_threshold;
    u8 coverage_class;
    char fw_version[ETHTOOL_BUSINFO_LEN];
    u32 hw_version;
    u8 max_num_pmkids;
    const void * privid;
    struct ieee80211_supported_band * bands[IEEE80211_NUM_BANDS];
    int (* reg_notifier) (struct wiphy *wiphy, struct regulatory_request *req);
    const struct ieee80211_regdomain * regd;
    struct device dev;
    struct dentry * debugfsdir;
#ifdef CONFIG_NET_NS
    struct net * _net;
#endif
#ifdef CONFIG_CFG80211_WEXT
    const struct iw_handler_def * wext;
#endif
}
```

```
#endif
char priv[0] __attribute__((__aligned__(NETDEV_ALIGN)));
};
```

## Members

`perm_addr[ETH_ALEN]`

permanent MAC address of this device

`addr_mask[ETH_ALEN]`

If the device supports multiple MAC addresses by masking, set this to a mask with variable bits set to 1, e.g. if the last

`addresses`

If the device has more than one address, set this pointer to a list of addresses (6 bytes each). The first one will be used by default for `perm_addr`. In this case, the mask should be set to all-zeroes. In this case it is assumed that the device can handle the same number of arbitrary MAC addresses.

`mgmt_stypes`

bitmasks of frame subtypes that can be subscribed to or transmitted through nl80211, points to an array indexed by interface type

`n_addresses`

number of addresses in *addresses*.

`interface_modes`

bitmask of interfaces types valid for this wiphy, must be set by driver

`flags`

wiphy flags, see enum `wiphy_flags`

`signal_type`

signal type reported in struct `cfg80211_bss`.

`bss_priv_size`

each BSS struct has private data allocated with it, this variable determines its size

## *Chapter 1. Device registration*

`max_scan_ssids`

maximum number of SSIDs the device can scan for in any given scan

`max_scan_ie_len`

maximum length of user-controlled IEs device can add to probe request frames transmitted during a scan, must not include fixed IEs like supported rates

`n_cipher_suites`

number of supported cipher suites

`cipher_suites`

supported cipher suites

`retry_short`

Retry limit for short frames (`dot11ShortRetryLimit`)

`retry_long`

Retry limit for long frames (`dot11LongRetryLimit`)

`frag_threshold`

Fragmentation threshold (`dot11FragmentationThreshold`); -1 = fragmentation disabled, only odd values  $\geq 256$  used

`rts_threshold`

RTS threshold (`dot11RTSThreshold`); -1 = RTS/CTS disabled

`coverage_class`

current coverage class

`fw_version[ETHTOOL_BUSINFO_LEN]`

firmware version for ethtool reporting

`hw_version`

hardware version for ethtool reporting

`max_num_pmkids`

maximum number of PMKIDs supported by device

`privid`

a pointer that drivers can use to identify if an arbitrary wiphy is theirs, e.g. in global notifiers

`bands[IEEE80211_NUM_BANDS]`

information about bands/channels supported by this device

`reg_notifier`

the driver's regulatory notification callback

`regd`

the driver's regulatory domain, if one was requested via the `regulatory_hint` API. This can be used by the driver on the `reg_notifier` if it chooses to ignore future regulatory domain changes caused by other drivers.

`dev`

(virtual) struct device for this wiphy

`debugfsdir`

debugfs directory used for this wiphy, will be renamed automatically on wiphy renames

`_net`

the network namespace this wiphy currently lives in

`wext`

wireless extension handlers

`priv[0] __attribute__((__aligned__(NETDEV_ALIGN)))`

driver private data (sized according to `wiphy_new` parameter)

## four bits are variable then set it to 00

...:00:0f. The actual variable bits shall be determined by the interfaces added, with interfaces not matching the mask being rejected to be brought up.

## struct wireless\_dev

**LINUX**

## Name

struct wireless\_dev — wireless per-netdev state

## Synopsis

```
struct wireless_dev {
    struct wiphy * wiphy;
    enum nl80211_iftype iftype;
    struct list_head list;
    struct net_device * netdev;
    struct list_head mgmt_registrations;
    spinlock_t mgmt_registrations_lock;
    struct mutex mtx;
    struct work_struct cleanup_work;
    bool use_4addr;
    u8 ssid[IEEE80211_MAX_SSID_LEN];
    u8 ssid_len;
    enum wext;
#ifdef
};
```

## Members

wiphy

pointer to hardware description

iftype

interface type

list

(private) Used to collect the interfaces

netdev

(private) Used to reference back to the netdev

mgmt\_registrations

list of registrations for management frames

`mgmt_registrations_lock`

lock for the list

`mtx`

mutex used to lock data in this struct

`cleanup_work`

work struct used for cleanup that can't be done directly

`use_4addr`

indicates 4addr mode is used on this interface, must be set by driver (if supported) on `add_interface` BEFORE registering the netdev and may otherwise be used by driver read-only, will be update by `cfg80211` on `change_interface`

`ssid[IEEE80211_MAX_SSID_LEN]`

(private) Used by the internal configuration code

`ssid_len`

(private) Used by the internal configuration code

`wext`

(private) Used by the internal wireless extensions compat code

## Description

This structure must be allocated by the driver/stack that uses the `ieee80211_ptr` field in `struct net_device` (this is intentional so it can be allocated along with the netdev.)

# wiphy\_new

**LINUX**

## Name

wiphy\_new — create a new wiphy for use with cfg80211

## Synopsis

```
struct wiphy * wiphy_new (const struct cfg80211_ops * ops, int
sizeof_priv);
```

## Arguments

*ops*

The configuration operations for this device

*sizeof\_priv*

The size of the private area to allocate

## Description

Create a new wiphy and associate the given operations with it. *sizeof\_priv* bytes are allocated for private use.

The returned pointer must be assigned to each netdev's ieee80211\_ptr for proper operation.

## wiphy\_register

**LINUX**

## Name

`wiphy_register` — register a wiphy with cfg80211

## Synopsis

```
int wiphy_register (struct wiphy * wiphy);
```

## Arguments

*wiphy*

The wiphy to register.

## Description

Returns a non-negative wiphy index or a negative error code.

# wiphy\_unregister

## LINUX

## Name

`wiphy_unregister` — deregister a wiphy from cfg80211

## Synopsis

```
void wiphy_unregister (struct wiphy * wiphy);
```

## Arguments

*wiphy*

The wiphy to unregister.

## Description

After this call, no more requests can be made with this priv pointer, but the call may sleep to wait for an outstanding request that is being handled.

# wiphy\_free

## LINUX

Kernel Hackers Manual July 2011

## Name

wiphy\_free — free wiphy

## Synopsis

```
void wiphy_free (struct wiphy * wiphy);
```

## Arguments

*wiphy*

The wiphy to free

## wiphy\_name

### LINUX

Kernel Hackers Manual July 2011

## Name

wiphy\_name — get wiphy name

## Synopsis

```
const char * wiphy_name (const struct wiphy * wiphy);
```

## Arguments

*wiphy*

The wiphy whose name to return

## wiphy\_dev

### LINUX

## Name

wiphy\_dev — get wiphy dev pointer

## Synopsis

```
struct device * wiphy_dev (struct wiphy * wiphy);
```

## Arguments

*wiphy*

The wiphy whose device struct to look up

# wiphy\_priv

## LINUX

## Name

wiphy\_priv — return priv from wiphy

## Synopsis

```
void * wiphy_priv (struct wiphy * wiphy);
```

## Arguments

*wiphy*

the wiphy whose priv pointer to return

## priv\_to\_wiphy

### LINUX

Kernel Hackers Manual July 2011

## Name

`priv_to_wiphy` — return the wiphy containing the priv

## Synopsis

```
struct wiphy * priv_to_wiphy (void * priv);
```

## Arguments

*priv*

a pointer previously returned by `wiphy_priv`

## set\_wiphy\_dev

### LINUX

## Name

`set_wiphy_dev` — set device pointer for wiphy

## Synopsis

```
void set_wiphy_dev (struct wiphy * wiphy, struct device *  
dev);
```

## Arguments

*wiphy*

The wiphy whose device to bind

*dev*

The device to parent it to

## wdev\_priv

### LINUX

## Name

`wdev_priv` — return wiphy priv from wireless\_dev

## Synopsis

```
void * wdev_priv (struct wireless_dev * wdev);
```

## **Arguments**

*wdev*

The wireless device whose wiphy's priv pointer to return



# Chapter 2. Actions and configuration

Each wireless device and each virtual interface offer a set of configuration operations and other actions that are invoked by userspace. Each of these actions is described in the operations structure, and the parameters these operations use are described separately.

Additionally, some operations are asynchronous and expect to get status information via some functions that drivers need to call.

Scanning and BSS list handling with its associated functionality is described in a separate chapter.

## struct cfg80211\_ops

### LINUX

Kernel Hackers Manual July 2011

### Name

struct cfg80211\_ops — backend description for wireless configuration

### Synopsis

```
struct cfg80211_ops {
    int (* suspend) (struct wiphy *wiphy);
    int (* resume) (struct wiphy *wiphy);
    int (* add_virtual_intf) (struct wiphy *wiphy, char *name, enum nl80211_
    int (* del_virtual_intf) (struct wiphy *wiphy, struct net_device *dev);
    int (* change_virtual_intf) (struct wiphy *wiphy, struct net_device *dev,
    int (* add_key) (struct wiphy *wiphy, struct net_device *netdev, u8 key_
    int (* get_key) (struct wiphy *wiphy, struct net_device *netdev, u8 key_
    int (* del_key) (struct wiphy *wiphy, struct net_device *netdev, u8 key_
    int (* set_default_key) (struct wiphy *wiphy, struct net_device *netdev, u
    int (* set_default_mgmt_key) (struct wiphy *wiphy, struct net_device *net
    int (* add_beacon) (struct wiphy *wiphy, struct net_device *dev, struct b
    int (* set_beacon) (struct wiphy *wiphy, struct net_device *dev, struct b
    int (* del_beacon) (struct wiphy *wiphy, struct net_device *dev);
    int (* add_station) (struct wiphy *wiphy, struct net_device *dev, u8 *mac
    int (* del_station) (struct wiphy *wiphy, struct net_device *dev, u8 *mac
    int (* change_station) (struct wiphy *wiphy, struct net_device *dev, u8 *
    int (* get_station) (struct wiphy *wiphy, struct net_device *dev, u8 *mac
```

```

int (* dump_station) (struct wiphy *wiphy, struct net_device *dev,int id);
int (* add_mpath) (struct wiphy *wiphy, struct net_device *dev,u8 *dst,
int (* del_mpath) (struct wiphy *wiphy, struct net_device *dev,u8 *dst);
int (* change_mpath) (struct wiphy *wiphy, struct net_device *dev,u8 *dst);
int (* get_mpath) (struct wiphy *wiphy, struct net_device *dev,u8 *dst,
int (* dump_mpath) (struct wiphy *wiphy, struct net_device *dev,int idx);
int (* get_mesh_params) (struct wiphy *wiphy,struct net_device *dev,struct
int (* set_mesh_params) (struct wiphy *wiphy,struct net_device *dev,const
int (* change_bss) (struct wiphy *wiphy, struct net_device *dev,struct bss
int (* set_txq_params) (struct wiphy *wiphy,struct ieee80211_txq_params
int (* set_channel) (struct wiphy *wiphy, struct net_device *dev,struct
int (* scan) (struct wiphy *wiphy, struct net_device *dev,struct cfg80211
int (* auth) (struct wiphy *wiphy, struct net_device *dev,struct cfg80211
int (* assoc) (struct wiphy *wiphy, struct net_device *dev,struct cfg80211
int (* deauth) (struct wiphy *wiphy, struct net_device *dev,struct cfg80211
int (* disassoc) (struct wiphy *wiphy, struct net_device *dev,struct cfg80211
int (* connect) (struct wiphy *wiphy, struct net_device *dev,struct cfg80211
int (* disconnect) (struct wiphy *wiphy, struct net_device *dev,u16 reason);
int (* join_ibss) (struct wiphy *wiphy, struct net_device *dev,struct cfg80211
int (* leave_ibss) (struct wiphy *wiphy, struct net_device *dev);
int (* set_wiphy_params) (struct wiphy *wiphy, u32 changed);
int (* set_tx_power) (struct wiphy *wiphy,enum nl80211_tx_power_setting
int (* get_tx_power) (struct wiphy *wiphy, int *dbm);
int (* set_wds_peer) (struct wiphy *wiphy, struct net_device *dev,const
void (* rfkill_poll) (struct wiphy *wiphy);
#ifdef CONFIG_NL80211_TESTMODE
int (* testmode_cmd) (struct wiphy *wiphy, void *data, int len);
#endif
int (* set_bitrate_mask) (struct wiphy *wiphy,struct net_device *dev,const
int (* dump_survey) (struct wiphy *wiphy, struct net_device *netdev,int
int (* set_pmksa) (struct wiphy *wiphy, struct net_device *netdev,struct
int (* del_pmksa) (struct wiphy *wiphy, struct net_device *netdev,struct
int (* flush_pmksa) (struct wiphy *wiphy, struct net_device *netdev);
int (* remain_on_channel) (struct wiphy *wiphy,struct net_device *dev,struct
int (* cancel_remain_on_channel) (struct wiphy *wiphy,struct net_device
int (* mgmt_tx) (struct wiphy *wiphy, struct net_device *dev,struct ieee
int (* set_power_mgmt) (struct wiphy *wiphy, struct net_device *dev,bool
int (* set_cqm_rssi_config) (struct wiphy *wiphy,struct net_device *dev,
void (* mgmt_frame_register) (struct wiphy *wiphy,struct net_device *dev,
};

```

## Members

suspend

wiphy device needs to be suspended

resume

wiphy device needs to be resumed

add\_virtual\_intf

create a new virtual interface with the given name, must set the struct wireless\_dev's iftype. Beware: You must create the new netdev in the wiphy's network namespace!

del\_virtual\_intf

remove the virtual interface determined by ifindex.

change\_virtual\_intf

change type/configuration of virtual interface, keep the struct wireless\_dev's iftype updated.

add\_key

add a key with the given parameters. *mac\_addr* will be NULL when adding a group key.

get\_key

get information about the key with the given parameters. *mac\_addr* will be NULL when requesting information for a group key. All pointers given to the *callback* function need not be valid after it returns. This function should return an error if it is not possible to retrieve the key, -ENOENT if it doesn't exist.

del\_key

remove a key given the *mac\_addr* (NULL for a group key) and *key\_index*, return -ENOENT if the key doesn't exist.

set\_default\_key

set the default key on an interface

set\_default\_mgmt\_key

set the default management frame key on an interface

## Chapter 2. Actions and configuration

### add\_beacon

Add a beacon with given parameters, *head*, *interval* and *dtim\_period* will be valid, *tail* is optional.

### set\_beacon

Change the beacon parameters for an access point mode interface. This should reject the call when no beacon has been configured.

### del\_beacon

Remove beacon configuration and stop sending the beacon.

### add\_station

Add a new station.

### del\_station

Remove a station; *mac* may be NULL to remove all stations.

### change\_station

Modify a given station.

### get\_station

get station information for the station identified by *mac*

### dump\_station

dump station callback -- resume dump at index *idx*

### add\_mpath

add a fixed mesh path

### del\_mpath

delete a given mesh path

### change\_mpath

change a given mesh path

### get\_mpath

get a mesh path for the given parameters

### dump\_mpath

dump mesh path callback -- resume dump at index *idx*

`get_mesh_params`

Put the current mesh parameters into `*params`

`set_mesh_params`

Set mesh parameters. The mask is a bitfield which tells us which parameters to set, and which to leave alone.

`change_bss`

Modify parameters for a given BSS.

`set_txq_params`

Set TX queue parameters

`set_channel`

Set channel for a given wireless interface. Some devices may support multi-channel operation (by channel hopping) so `cfg80211` doesn't verify much. Note, however, that the passed `netdev` may be `NULL` as well if the user requested changing the channel for the device itself, or for a monitor interface.

`scan`

Request to do a scan. If returning zero, the scan request is given the driver, and will be valid until passed to `cfg80211_scan_done`. For scan results, call `cfg80211_inform_bss`; you can call this outside the scan/scan\_done bracket too.

`auth`

Request to authenticate with the specified peer

`assoc`

Request to (re)associate with the specified peer

`deauth`

Request to deauthenticate from the specified peer

`disassoc`

Request to disassociate from the specified peer

`connect`

Connect to the ESS with the specified parameters. When connected, call `cfg80211_connect_result` with status code `WLAN_STATUS_SUCCESS`. If

## Chapter 2. Actions and configuration

the connection fails for some reason, call `cfg80211_connect_result` with the status from the AP.

`disconnect`

Disconnect from the BSS/ESS.

`join_ibss`

Join the specified IBSS (or create if necessary). Once done, call `cfg80211_ibss_joined`, also call that function when changing BSSID due to a merge.

`leave_ibss`

Leave the IBSS.

`set_wiphy_params`

Notify that wiphy parameters have changed; *changed* bitfield (see enum `wiphy_params_flags`) describes which values have changed. The actual parameter values are available in struct `wiphy`. If returning an error, no value should be changed.

`set_tx_power`

set the transmit power according to the parameters

`get_tx_power`

store the current TX power into the `dbm` variable; return 0 if successful

`set_wds_peer`

set the WDS peer for a WDS interface

`rkill_poll`

polls the hw rkill line, use `cfg80211` reporting functions to adjust rkill hw state

`testmode_cmd`

run a test mode command

`set_bitrate_mask`

set the bitrate mask configuration

`dump_survey`

get site survey information.

`set_pmksa`

Cache a PMKID for a BSSID. This is mostly useful for fullmac devices running firmwares capable of generating the (re) association RSN IE. It allows for faster roaming between WPA2 BSSIDs.

`del_pmksa`

Delete a cached PMKID.

`flush_pmksa`

Flush all cached PMKIDs.

`remain_on_channel`

Request the driver to remain awake on the specified channel for the specified duration to complete an off-channel operation (e.g., public action frame exchange). When the driver is ready on the requested channel, it must indicate this with an event notification by calling `cfg80211_ready_on_channel`.

`cancel_remain_on_channel`

Cancel an on-going remain-on-channel operation. This allows the operation to be terminated prior to timeout based on the duration value.

`mgmt_tx`

Transmit a management frame

`set_power_mgmt`

Configure WLAN power management. A timeout value of -1 allows the driver to adjust the dynamic ps timeout value.

`set_cqm_rssi_config`

Configure connection quality monitor RSSI threshold.

`mgmt_frame_register`

Notify driver that a management frame type was registered. Note that this callback may not sleep, and cannot run concurrently with itself.

## Description

This struct is registered by fullmac card drivers and/or wireless stacks in order to handle configuration requests on their interfaces.

All callbacks except where otherwise noted should return 0 on success or a negative error code.

All operations are currently invoked under `rtnl` for consistency with the wireless extensions but this is subject to reevaluation as soon as this code is used more widely and we have a first user without `wext`.

## struct vif\_params

### LINUX

Kernel Hackers Manual July 2011

### Name

`struct vif_params` — describes virtual interface parameters

### Synopsis

```
struct vif_params {  
    u8 * mesh_id;  
    int mesh_id_len;  
    int use_4addr;  
};
```

### Members

`mesh_id`

mesh ID to use

`mesh_id_len`

length of the mesh ID

`use_4addr`

use 4-address frames

# struct key\_params

## LINUX

Kernel Hackers Manual July 2011

### Name

struct key\_params — key information

### Synopsis

```
struct key_params {  
    u8 * key;  
    u8 * seq;  
    int key_len;  
    int seq_len;  
    u32 cipher;  
};
```

### Members

key

key material

seq

sequence counter (IV/PN) for TKIP and CCMP keys, only used with the `get_key` callback, must be in little endian, length given by `seq_len`.

key\_len

length of key material

seq\_len

length of `seq`.

cipher

cipher suite selector

## Description

Information about a key

# enum survey\_info\_flags

## LINUX

Kernel Hackers Manual July 2011

## Name

enum survey\_info\_flags — survey information flags

## Synopsis

```
enum survey_info_flags {  
    SURVEY_INFO_NOISE_DBM,  
    SURVEY_INFO_IN_USE,  
    SURVEY_INFO_CHANNEL_TIME,  
    SURVEY_INFO_CHANNEL_TIME_BUSY,  
    SURVEY_INFO_CHANNEL_TIME_EXT_BUSY,  
    SURVEY_INFO_CHANNEL_TIME_RX,  
    SURVEY_INFO_CHANNEL_TIME_TX  
};
```

## Constants

SURVEY\_INFO\_NOISE\_DBM

noise (in dBm) was filled in

SURVEY\_INFO\_IN\_USE

channel is currently being used

SURVEY\_INFO\_CHANNEL\_TIME

channel active time (in ms) was filled in

SURVEY\_INFO\_CHANNEL\_TIME\_BUSY

channel busy time was filled in

SURVEY\_INFO\_CHANNEL\_TIME\_EXT\_BUSY

extension channel busy time was filled in

SURVEY\_INFO\_CHANNEL\_TIME\_RX

channel receive time was filled in

SURVEY\_INFO\_CHANNEL\_TIME\_TX

channel transmit time was filled in

## Description

Used by the driver to indicate which info in struct `survey_info` it has filled in during the `get_survey`.

# struct survey\_info

## LINUX

Kernel Hackers Manual July 2011

## Name

`struct survey_info` — channel survey response

## Synopsis

```
struct survey_info {
```

```
struct ieee80211_channel * channel;  
u64 channel_time;  
u64 channel_time_busy;  
u64 channel_time_ext_busy;  
u64 channel_time_rx;  
u64 channel_time_tx;  
u32 filled;  
s8 noise;  
};
```

## Members

`channel`

the channel this survey record reports, mandatory

`channel_time`

amount of time in ms the radio spent on the channel

`channel_time_busy`

amount of time the primary channel was sensed busy

`channel_time_ext_busy`

amount of time the extension channel was sensed busy

`channel_time_rx`

amount of time the radio spent receiving data

`channel_time_tx`

amount of time the radio spent transmitting data

`filled`

bitflag of flags from enum `survey_info_flags`

`noise`

channel noise in dBm. This and all following fields are optional

## Description

Used by `dump_survey` to report back per-channel survey information.

This structure can later be expanded with things like channel duty cycle etc.

## struct beacon\_parameters

### LINUX

Kernel Hackers Manual July 2011

### Name

struct beacon\_parameters — beacon parameters

### Synopsis

```
struct beacon_parameters {  
    u8 * head;  
    u8 * tail;  
    int interval;  
    int dtim_period;  
    int head_len;  
    int tail_len;  
};
```

### Members

head

head portion of beacon (before TIM IE) or NULL if not changed

tail

tail portion of beacon (after TIM IE) or NULL if not changed

interval

beacon interval or zero if not changed

dtim\_period

DTIM period or zero if not changed

`head_len`

length of *head*

`tail_len`

length of *tail*

## Description

Used to configure the beacon for an interface.

# enum plink\_actions

## LINUX

Kernel Hackers Manual July 2011

## Name

`enum plink_actions` — actions to perform in mesh peers

## Synopsis

```
enum plink_actions {  
    PLINK_ACTION_INVALID,  
    PLINK_ACTION_OPEN,  
    PLINK_ACTION_BLOCK  
};
```

## Constants

`PLINK_ACTION_INVALID`

action 0 is reserved

PLINK\_ACTION\_OPEN

start mesh peer link establishment

PLINK\_ACTION\_BLOCK

block traffic from this mesh peer

## struct station\_parameters

### LINUX

Kernel Hackers Manual July 2011

### Name

struct station\_parameters — station parameters

### Synopsis

```
struct station_parameters {
    u8 * supported_rates;
    struct net_device * vlan;
    u32 sta_flags_mask;
    u32 sta_flags_set;
    int listen_interval;
    u16 aid;
    u8 supported_rates_len;
    u8 plink_action;
    struct ieee80211_ht_cap * ht_capa;
};
```

### Members

supported\_rates

supported rates in IEEE 802.11 format (or NULL for no change)

vlan

vlan interface station should belong to

sta\_flags\_mask

station flags that changed (bitmask of BIT(NL80211\_STA\_FLAG\_...))

sta\_flags\_set

station flags values (bitmask of BIT(NL80211\_STA\_FLAG\_...))

listen\_interval

listen interval or -1 for no change

aid

AID or zero for no change

supported\_rates\_len

number of supported rates

plink\_action

plink action to take

ht\_capa

HT capabilities of station

## **Description**

Used to change and create a new station.

## **enum station\_info\_flags**

**LINUX**

## Name

enum station\_info\_flags — station information flags

## Synopsis

```
enum station_info_flags {
    STATION_INFO_INACTIVE_TIME,
    STATION_INFO_RX_BYTES,
    STATION_INFO_TX_BYTES,
    STATION_INFO_LLID,
    STATION_INFO_PLID,
    STATION_INFO_PLINK_STATE,
    STATION_INFO_SIGNAL,
    STATION_INFO_TX_BITRATE,
    STATION_INFO_RX_PACKETS,
    STATION_INFO_TX_PACKETS,
    STATION_INFO_TX_RETRIES,
    STATION_INFO_TX_FAILED,
    STATION_INFO_RX_DROP_MISC
};
```

## Constants

STATION\_INFO\_INACTIVE\_TIME

*inactive\_time* filled

STATION\_INFO\_RX\_BYTES

*rx\_bytes* filled

STATION\_INFO\_TX\_BYTES

*tx\_bytes* filled

STATION\_INFO\_LLID

*llid* filled

STATION\_INFO\_PLID

*plid* filled

STATION\_INFO\_PLINK\_STATE

*plink\_state* filled

STATION\_INFO\_SIGNAL

*signal* filled

STATION\_INFO\_TX\_BITRATE

*tx\_bitrate* fields are filled (tx\_bitrate, tx\_bitrate\_flags and tx\_bitrate\_mcs)

STATION\_INFO\_RX\_PACKETS

*rx\_packets* filled

STATION\_INFO\_TX\_PACKETS

*tx\_packets* filled

STATION\_INFO\_TX\_RETRIES

*tx\_retries* filled

STATION\_INFO\_TX\_FAILED

*tx\_failed* filled

STATION\_INFO\_RX\_DROP\_MISC

*rx\_dropped\_misc* filled

## Description

Used by the driver to indicate which info in struct `station_info` it has filled in during `get_station` or `dump_station`.

## enum rate\_info\_flags

**LINUX**

## Name

enum rate\_info\_flags — bitrate info flags

## Synopsis

```
enum rate_info_flags {
    RATE_INFO_FLAGS_MCS,
    RATE_INFO_FLAGS_40_MHZ_WIDTH,
    RATE_INFO_FLAGS_SHORT_GI
};
```

## Constants

RATE\_INFO\_FLAGS\_MCS

*tx\_bitrate\_mcs* filled

RATE\_INFO\_FLAGS\_40\_MHZ\_WIDTH

40 Mhz width transmission

RATE\_INFO\_FLAGS\_SHORT\_GI

400ns guard interval

## Description

Used by the driver to indicate the specific rate transmission type for 802.11n transmissions.

## struct rate\_info

**LINUX**

## Name

`struct rate_info` — bitrate information

## Synopsis

```
struct rate_info {  
    u8 flags;  
    u8 mcs;  
    u16 legacy;  
};
```

## Members

`flags`

bitflag of flags from enum `rate_info_flags`

`mcs`

mcs index if struct describes a 802.11n bitrate

`legacy`

bitrate in 100kbit/s for 802.11abg

## Description

Information about a receiving or transmitting bitrate

## `struct station_info`

**LINUX**

## Name

`struct station_info` — station information

## Synopsis

```

struct station_info {
    u32 filled;
    u32 inactive_time;
    u32 rx_bytes;
    u32 tx_bytes;
    u16 llid;
    u16 plid;
    u8 plink_state;
    s8 signal;
    struct rate_info txrate;
    u32 rx_packets;
    u32 tx_packets;
    u32 tx_retries;
    u32 tx_failed;
    u32 rx_dropped_misc;
    int generation;
};

```

## Members

`filled`

bitflag of flags from enum `station_info_flags`

`inactive_time`

time since last station activity (tx/rx) in milliseconds

`rx_bytes`

bytes received from this station

`tx_bytes`

bytes transmitted to this station

llid

mesh local link id

plid

mesh peer link id

plink\_state

mesh peer link state

signal

signal strength of last received packet in dBm

txrate

current unicast bitrate to this station

rx\_packets

packets received from this station

tx\_packets

packets transmitted to this station

tx\_retries

cumulative retry counts

tx\_failed

number of failed transmissions (retries exceeded, no ACK)

rx\_dropped\_misc

Dropped for un-specified reason.

generation

generation number for nl80211 dumps. This number should increase every time the list of stations changes, i.e. when a station is added or removed, so that userspace can tell whether it got a consistent snapshot.

## **Description**

Station information filled by driver for `get_station` and `dump_station`.

# enum monitor\_flags

## LINUX

Kernel Hackers Manual July 2011

### Name

enum monitor\_flags — monitor flags

### Synopsis

```
enum monitor_flags {  
    MONITOR_FLAG_FCSFAIL,  
    MONITOR_FLAG_PLCPFAIL,  
    MONITOR_FLAG_CONTROL,  
    MONITOR_FLAG_OTHER_BSS,  
    MONITOR_FLAG_COOK_FRAMES  
};
```

### Constants

MONITOR\_FLAG\_FCSFAIL

pass frames with bad FCS

MONITOR\_FLAG\_PLCPFAIL

pass frames with bad PLCP

MONITOR\_FLAG\_CONTROL

pass control frames

MONITOR\_FLAG\_OTHER\_BSS

disable BSSID filtering

MONITOR\_FLAG\_COOK\_FRAMES

report frames after processing

## Description

Monitor interface configuration flags. Note that these must be the bits according to the nl80211 flags.

# enum mpath\_info\_flags

## LINUX

Kernel Hackers Manual July 2011

## Name

enum mpath\_info\_flags — mesh path information flags

## Synopsis

```
enum mpath_info_flags {  
    MPATH_INFO_FRAME_QLEN,  
    MPATH_INFO_SN,  
    MPATH_INFO_METRIC,  
    MPATH_INFO_EXPTIME,  
    MPATH_INFO_DISCOVERY_TIMEOUT,  
    MPATH_INFO_DISCOVERY_RETRIES,  
    MPATH_INFO_FLAGS  
};
```

## Constants

MPATH\_INFO\_FRAME\_QLEN

*frame\_qlen* filled

MPATH\_INFO\_SN

*sn* filled

MPATH\_INFO\_METRIC

*metric* filled

MPATH\_INFO\_EXPTIME

*exptime* filled

MPATH\_INFO\_DISCOVERY\_TIMEOUT

*discovery\_timeout* filled

MPATH\_INFO\_DISCOVERY\_RETRIES

*discovery\_retries* filled

MPATH\_INFO\_FLAGS

*flags* filled

## Description

Used by the driver to indicate which info in struct `mpath_info` it has filled in during `get_station` or `dump_station`.

# struct mpath\_info

## LINUX

Kernel Hackers Manual July 2011

## Name

struct `mpath_info` — mesh path information

## Synopsis

```
struct mpath_info {  
    u32 filled;  
    u32 frame_qlen;  
    u32 sn;
```

```
    u32 metric;  
    u32 exptime;  
    u32 discovery_timeout;  
    u8 discovery_retries;  
    u8 flags;  
    int generation;  
};
```

## Members

filled

bitfield of flags from enum mpath\_info\_flags

frame\_qlen

number of queued frames for this destination

sn

target sequence number

metric

metric (cost) of this mesh path

exptime

expiration time for the mesh path from now, in msecs

discovery\_timeout

total mesh path discovery timeout, in msecs

discovery\_retries

mesh path discovery retries

flags

mesh path flags

generation

generation number for nl80211 dumps. This number should increase every time the list of mesh paths changes, i.e. when a station is added or removed, so that userspace can tell whether it got a consistent snapshot.

## Description

Mesh path information filled by driver for `get_mpath` and `dump_mpath`.

## struct bss\_parameters

### LINUX

Kernel Hackers Manual July 2011

### Name

`struct bss_parameters` — BSS parameters

### Synopsis

```
struct bss_parameters {
    int use_cts_prot;
    int use_short_preamble;
    int use_short_slot_time;
    u8 * basic_rates;
    u8 basic_rates_len;
    int ap_isolate;
};
```

### Members

`use_cts_prot`

Whether to use CTS protection (0 = no, 1 = yes, -1 = do not change)

`use_short_preamble`

Whether the use of short preambles is allowed (0 = no, 1 = yes, -1 = do not change)

`use_short_slot_time`

Whether the use of short slot time is allowed (0 = no, 1 = yes, -1 = do not change)

`basic_rates`

basic rates in IEEE 802.11 format (or NULL for no change)

`basic_rates_len`

number of basic rates

`ap_isolate`

do not forward packets between connected stations

## Description

Used to change BSS parameters (mainly for AP mode).

# struct ieee80211\_txq\_params

## LINUX

Kernel Hackers Manual July 2011

## Name

`struct ieee80211_txq_params` — TX queue parameters

## Synopsis

```
struct ieee80211_txq_params {
    enum nl80211_txq_q queue;
    u16 txop;
    u16 cwmin;
    u16 cwmax;
    u8 aifs;
};
```

## Members

queue

TX queue identifier (NL80211\_TXQ\_Q\_\*)

txop

Maximum burst time in units of 32 usecs, 0 meaning disabled

cwmin

Minimum contention window [a value of the form  $2^{n-1}$  in the range 1..32767]

cwmax

Maximum contention window [a value of the form  $2^{n-1}$  in the range 1..32767]

aifs

Arbitration interframe space [0..255]

## struct cfg80211\_crypto\_settings

**LINUX**

Kernel Hackers Manual July 2011

### Name

struct cfg80211\_crypto\_settings — Crypto settings

### Synopsis

```
struct cfg80211_crypto_settings {
    u32 wpa_versions;
    u32 cipher_group;
    int n_ciphers_pairwise;
    u32 ciphers_pairwise[NL80211_MAX_NR_CIPHER_SUITES];
    int n_akm_suites;
    u32 akm_suites[NL80211_MAX_NR_AKM_SUITES];
    bool control_port;
    __be16 control_port_ethertype;
```

```
bool control_port_no_encrypt;  
};
```

## Members

wpa\_versions

indicates which, if any, WPA versions are enabled (from enum nl80211\_wpa\_versions)

cipher\_group

group key cipher suite (or 0 if unset)

n\_ciphers\_pairwise

number of AP supported unicast ciphers

ciphers\_pairwise[NL80211\_MAX\_NR\_CIPHER\_SUITES]

unicast key cipher suites

n\_akm\_suites

number of AKM suites

akm\_suites[NL80211\_MAX\_NR\_AKM\_SUITES]

AKM suites

control\_port

Whether user space controls IEEE 802.1X port, i.e., sets/clears NL80211\_STA\_FLAG\_AUTHORIZED. If true, the driver is required to assume that the port is unauthorized until authorized by user space. Otherwise, port is marked authorized by default.

control\_port\_ether\_type

the control port protocol that should be allowed through even on unauthorized ports

control\_port\_no\_encrypt

TRUE to prevent encryption of control port protocol frames.

# struct cfg80211\_auth\_request

## LINUX

Kernel Hackers Manual July 2011

### Name

struct cfg80211\_auth\_request — Authentication request data

### Synopsis

```
struct cfg80211_auth_request {  
    struct cfg80211_bss * bss;  
    const u8 * ie;  
    size_t ie_len;  
    enum nl80211_auth_type auth_type;  
    const u8 * key;  
    u8 key_len;  
    u8 key_idx;  
    bool local_state_change;  
};
```

### Members

bss

The BSS to authenticate with.

ie

Extra IEs to add to Authentication frame or NULL

ie\_len

Length of ie buffer in octets

auth\_type

Authentication type (algorithm)

key

WEP key for shared key authentication

key\_len

length of WEP key for shared key authentication

key\_idx

index of WEP key for shared key authentication

local\_state\_change

This is a request for a local state only, i.e., no Authentication frame is to be transmitted and authentication state is to be changed without having to wait for a response from the peer STA (AP).

## Description

This structure provides information needed to complete IEEE 802.11 authentication.

# struct cfg80211\_assoc\_request

## LINUX

Kernel Hackers Manual July 2011

## Name

struct cfg80211\_assoc\_request — (Re)Association request data

## Synopsis

```
struct cfg80211_assoc_request {
    struct cfg80211_bss * bss;
    const u8 * ie;
    const u8 * prev_bssid;
    size_t ie_len;
    struct cfg80211_crypto_settings crypto;
    bool use_mfp;
};
```

## Members

bss

The BSS to associate with.

ie

Extra IEs to add to (Re)Association Request frame or `NULL`

prev\_bssid

previous BSSID, if not `NULL` use reassociate frame

ie\_len

Length of ie buffer in octets

crypto

crypto settings

use\_mfp

Use management frame protection (IEEE 802.11w) in this association

## Description

This structure provides information needed to complete IEEE 802.11 (re)association.

## struct cfg80211\_deauth\_request

**LINUX**

Kernel Hackers Manual July 2011

## Name

`struct cfg80211_deauth_request` — Deauthentication request data

## Synopsis

```
struct cfg80211_deauth_request {  
    struct cfg80211_bss * bss;  
    const u8 * ie;  
    size_t ie_len;  
    u16 reason_code;  
    bool local_state_change;  
};
```

## Members

bss

the BSS to deauthenticate from

ie

Extra IEs to add to Deauthentication frame or `NULL`

ie\_len

Length of ie buffer in octets

reason\_code

The reason code for the deauthentication

local\_state\_change

This is a request for a local state only, i.e., no Deauthentication frame is to be transmitted.

## Description

This structure provides information needed to complete IEEE 802.11 deauthentication.

# struct cfg80211\_disassoc\_request

## LINUX

Kernel Hackers Manual July 2011

### Name

struct cfg80211\_disassoc\_request — Disassociation request data

### Synopsis

```
struct cfg80211_disassoc_request {  
    struct cfg80211_bss * bss;  
    const u8 * ie;  
    size_t ie_len;  
    u16 reason_code;  
    bool local_state_change;  
};
```

### Members

bss

the BSS to disassociate from

ie

Extra IEs to add to Disassociation frame or NULL

ie\_len

Length of ie buffer in octets

reason\_code

The reason code for the disassociation

local\_state\_change

This is a request for a local state only, i.e., no Disassociation frame is to be transmitted.

## Description

This structure provides information needed to complete IEEE 802.11 disassociation.

## struct cfg80211\_ibss\_params

### LINUX

Kernel Hackers Manual July 2011

### Name

struct cfg80211\_ibss\_params — IBSS parameters

### Synopsis

```
struct cfg80211_ibss_params {
    u8 * ssid;
    u8 * bssid;
    struct ieee80211_channel * channel;
    u8 * ie;
    u8 ssid_len;
    u8 ie_len;
    u16 beacon_interval;
    u32 basic_rates;
    bool channel_fixed;
    bool privacy;
};
```

### Members

ssid

The SSID, will always be non-null.

bssid

Fixed BSSID requested, maybe be `NULL`, if set do not search for IBSSs with a different BSSID.

channel

The channel to use if no IBSS can be found to join.

ie

information element(s) to include in the beacon

ssid\_len

The length of the SSID, will always be non-zero.

ie\_len

length of that

beacon\_interval

beacon interval to use

basic\_rates

bitmap of basic rates to use when creating the IBSS

channel\_fixed

The channel should be fixed -- do not search for IBSSs to join on other channels.

privacy

this is a protected network, keys will be configured after joining

## Description

This structure defines the IBSS parameters for the `join_ibss` method.

# struct cfg80211\_connect\_params

**LINUX**

## Name

struct cfg80211\_connect\_params — Connection parameters

## Synopsis

```
struct cfg80211_connect_params {
    struct ieee80211_channel * channel;
    u8 * bssid;
    u8 * ssid;
    size_t ssid_len;
    enum nl80211_auth_type auth_type;
    u8 * ie;
    size_t ie_len;
    bool privacy;
    struct cfg80211_crypto_settings crypto;
    const u8 * key;
    u8 key_len;
    u8 key_idx;
};
```

## Members

channel

The channel to use or `NULL` if not specified (auto-select based on scan results)

bssid

The AP BSSID or `NULL` if not specified (auto-select based on scan results)

ssid

SSID

ssid\_len

Length of ssid in octets

auth\_type

Authentication type (algorithm)

`ie`  
IEs for association request

`ie_len`  
Length of `assoc_ie` in octets

`privacy`  
indicates whether privacy-enabled APs should be used

`crypto`  
crypto settings

`key`  
WEP key for shared key authentication

`key_len`  
length of WEP key for shared key authentication

`key_idx`  
index of WEP key for shared key authentication

## Description

This structure provides information needed to complete IEEE 802.11 authentication and association.

## struct cfg80211\_pmksa

**LINUX**

Kernel Hackers Manual July 2011

## Name

`struct cfg80211_pmksa` — PMK Security Association

## Synopsis

```
struct cfg80211_pmksa {  
    u8 * bssid;  
    u8 * pmkid;  
};
```

## Members

bssid

The AP's BSSID.

pmkid

The PMK material itself.

## Description

This structure is passed to the `set/del_pmksa` method for PMKSA caching.

# cfg80211\_send\_rx\_auth

## LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_send_rx_auth` — notification of processed authentication

## Synopsis

```
void cfg80211_send_rx_auth (struct net_device * dev, const u8  
* buf, size_t len);
```

## Arguments

*dev*

network device

*buf*

authentication frame (header + body)

*len*

length of the frame data

## Description

This function is called whenever an authentication has been processed in station mode. The driver is required to call either this function or `cfg80211_send_auth_timeout` to indicate the result of `cfg80211_ops::auth` call. This function may sleep.

# cfg80211\_send\_auth\_timeout

## LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_send_auth_timeout` — notification of timed out authentication

## Synopsis

```
void cfg80211_send_auth_timeout (struct net_device * dev,  
const u8 * addr);
```

## Arguments

*dev*

network device

*addr*

The MAC address of the device with which the authentication timed out

## Description

This function may sleep.

# \_\_cfg80211\_auth\_canceled

## LINUX

Kernel Hackers Manual July 2011

## Name

`__cfg80211_auth_canceled` — notify cfg80211 that authentication was canceled

## Synopsis

```
void __cfg80211_auth_canceled (struct net_device * dev, const
u8 * addr);
```

## Arguments

*dev*

network device

*addr*

The MAC address of the device with which the authentication timed out

## Description

When a pending authentication had no action yet, the driver may decide to not send a deauth frame, but in that case must call this function to tell `cfg80211` about this decision. It is only valid to call this function within the `deauth` callback.

# cfg80211\_send\_rx\_assoc

## LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_send_rx_assoc` — notification of processed association

## Synopsis

```
void cfg80211_send_rx_assoc (struct net_device * dev, const u8  
* buf, size_t len);
```

## Arguments

*dev*

network device

*buf*

(re)association response frame (header + body)

*len*

length of the frame data

## Description

This function is called whenever a (re)association response has been processed in station mode. The driver is required to call either this function or `cfg80211_send_assoc_timeout` to indicate the result of

## `cfg80211_ops`

`:assoc` call. This function may sleep.

## `cfg80211_send_assoc_timeout`

### LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_send_assoc_timeout` — notification of timed out association

## Synopsis

```
void cfg80211_send_assoc_timeout (struct net_device * dev,  
const u8 * addr);
```

## Arguments

*dev*

network device

*addr*

The MAC address of the device with which the association timed out

## Description

This function may sleep.

# cfg80211\_send\_deauth

## LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_send_deauth` — notification of processed deauthentication

## Synopsis

```
void cfg80211_send_deauth (struct net_device * dev, const u8 *  
buf, size_t len);
```

## Arguments

*dev*

network device

*buf*

deauthentication frame (header + body)

*len*

length of the frame data

## Description

This function is called whenever deauthentication has been processed in station mode. This includes both received deauthentication frames and locally generated ones. This function may sleep.

## \_\_cfg80211\_send\_deauth

### LINUX

Kernel Hackers Manual July 2011

## Name

`__cfg80211_send_deauth` — notification of processed deauthentication

## Synopsis

```
void __cfg80211_send_deauth (struct net_device * dev, const u8
* buf, size_t len);
```

## Arguments

*dev*

network device

*buf*

deauthentication frame (header + body)

*len*

length of the frame data

## Description

Like `cfg80211_send_deauth`, but doesn't take the `wdev` lock.

# cfg80211\_send\_disassoc

## LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_send_disassoc` — notification of processed disassociation

## Synopsis

```
void cfg80211_send_disassoc (struct net_device * dev, const u8  
* buf, size_t len);
```

## Arguments

*dev*

network device

*buf*

disassociation response frame (header + body)

*len*

length of the frame data

## Description

This function is called whenever disassociation has been processed in station mode. This includes both received disassociation frames and locally generated ones. This function may sleep.

# \_\_cfg80211\_send\_disassoc

## LINUX

Kernel Hackers Manual July 2011

## Name

`__cfg80211_send_disassoc` — notification of processed disassociation

## Synopsis

```
void __cfg80211_send_disassoc (struct net_device * dev, const  
u8 * buf, size_t len);
```

## Arguments

*dev*

network device

*buf*

disassociation response frame (header + body)

*len*

length of the frame data

## Description

Like `cfg80211_send_disassoc`, but doesn't take the `wdev` lock.

# cfg80211\_ibss\_joined

## LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_ibss_joined` — notify `cfg80211` that device joined an IBSS

## Synopsis

```
void cfg80211_ibss_joined (struct net_device * dev, const u8 *  
bssid, gfp_t gfp);
```

## Arguments

*dev*

network device

*bssid*

the BSSID of the IBSS joined

*gfp*

allocation flags

## Description

This function notifies `cfg80211` that the device joined an IBSS or switched to a different BSSID. Before this function can be called, either a beacon has to have been received from the IBSS, or one of the `cfg80211_inform_bss{,_frame}` functions must have been called with the locally generated beacon -- this guarantees that there is always a scan result for this IBSS. `cfg80211` will handle the rest.

# cfg80211\_connect\_result

## LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_connect_result` — notify `cfg80211` of connection result

## Synopsis

```
void cfg80211_connect_result (struct net_device * dev, const
u8 * bssid, const u8 * req_ie, size_t req_ie_len, const u8 *
resp_ie, size_t resp_ie_len, u16 status, gfp_t gfp);
```

## Arguments

*dev*

network device

*bssid*

the BSSID of the AP

*req\_ie*

association request IEs (maybe be `NULL`)

*req\_ie\_len*

association request IEs length

*resp\_ie*

association response IEs (may be `NULL`)

*resp\_ie\_len*

assoc response IEs length

*status*

status code, 0 for successful connection, use `WLAN_STATUS_UNSPECIFIED_FAILURE` if your device cannot give you the real status code for failures.

*gfp*

allocation flags

## Description

It should be called by the underlying driver whenever `connect` has succeeded.

# cfg80211\_roamed

## LINUX

Kernel Hackers Manual July 2011

### Name

cfg80211\_roamed — notify cfg80211 of roaming

### Synopsis

```
void cfg80211_roamed (struct net_device * dev, const u8 *  
bssid, const u8 * req_ie, size_t req_ie_len, const u8 *  
resp_ie, size_t resp_ie_len, gfp_t gfp);
```

### Arguments

*dev*

network device

*bssid*

the BSSID of the new AP

*req\_ie*

association request IEs (maybe be NULL)

*req\_ie\_len*

association request IEs length

*resp\_ie*

association response IEs (may be NULL)

*resp\_ie\_len*

assoc response IEs length

*gfp*

allocation flags

## Description

It should be called by the underlying driver whenever it roamed from one AP to another while connected.

# cfg80211\_disconnected

## LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_disconnected` — notify `cfg80211` that connection was dropped

## Synopsis

```
void cfg80211_disconnected (struct net_device * dev, u16  
reason, u8 * ie, size_t ie_len, gfp_t gfp);
```

## Arguments

*dev*

network device

*reason*

reason code for the disconnection, set it to 0 if unknown

*ie*

information elements of the deauth/disassoc frame (may be `NULL`)

*ie\_len*

length of IEs

*gfp*

allocation flags

## Description

After it calls this function, the driver should enter an idle state and not try to connect to any AP any more.

# cfg80211\_ready\_on\_channel

## LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_ready_on_channel` — notification of `remain_on_channel` start

## Synopsis

```
void cfg80211_ready_on_channel (struct net_device * dev, u64
cookie, struct ieee80211_channel * chan, enum
nl80211_channel_type channel_type, unsigned int duration,
gfp_t gfp);
```

## Arguments

*dev*

network device

*cookie*

the request cookie

*chan*

The current channel (from `remain_on_channel` request)

*channel\_type*

Channel type

*duration*

Duration in milliseconds that the driver intends to remain on the channel

*gfp*

allocation flags

## cfg80211\_remain\_on\_channel\_expired

### LINUX

Kernel Hackers Manual July 2011

### Name

`cfg80211_remain_on_channel_expired` — `remain_on_channel` duration expired

### Synopsis

```
void cfg80211_remain_on_channel_expired (struct net_device *  
dev, u64 cookie, struct ieee80211_channel * chan, enum  
nl80211_channel_type channel_type, gfp_t gfp);
```

## Arguments

*dev*

network device

*cookie*

the request cookie

*chan*

The current channel (from `remain_on_channel` request)

*channel\_type*

Channel type

*gfp*

allocation flags

## cfg80211\_new\_sta

### LINUX

Kernel Hackers Manual July 2011

### Name

`cfg80211_new_sta` — notify userspace about station

### Synopsis

```
void cfg80211_new_sta (struct net_device * dev, const u8 *  
mac_addr, struct station_info * sinfo, gfp_t gfp);
```

## Arguments

*dev*

the netdev

*mac\_addr*

the station's address

*sinfo*

the station information

*gfp*

allocation flags

## cfg80211\_rx\_mgmt

**LINUX**

Kernel Hackers Manual July 2011

### Name

`cfg80211_rx_mgmt` — notification of received, unprocessed management frame

### Synopsis

```
bool cfg80211_rx_mgmt (struct net_device * dev, int freq,  
const u8 * buf, size_t len, gfp_t gfp);
```

## Arguments

*dev*

network device

*freq*

Frequency on which the frame was received in MHz

*buf*

Management frame (header + body)

*len*

length of the frame data

*gfp*

context flags

## Description

Returns `true` if a user space application has registered for this frame. For action frames, that makes it responsible for rejecting unrecognized action frames; `false` otherwise, in which case for action frames the driver is responsible for rejecting the frame.

This function is called whenever an Action frame is received for a station mode interface, but is not processed in kernel.

# cfg80211\_mgmt\_tx\_status

## LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_mgmt_tx_status` — notification of TX status for management frame

## Synopsis

```
void cfg80211_mgmt_tx_status (struct net_device * dev, u64  
cookie, const u8 * buf, size_t len, bool ack, gfp_t gfp);
```

## Arguments

*dev*

network device

*cookie*

Cookie returned by `cfg80211_ops::mgmt_tx`

*buf*

Management frame (header + body)

*len*

length of the frame data

*ack*

Whether frame was acknowledged

*gfp*

context flags

## Description

This function is called whenever a management frame was requested to be

## transmitted with `cfg80211_ops`

`:mgmt_tx` to report the TX status of the transmission attempt.

# cfg80211\_cqm\_rssi\_notify

## LINUX

Kernel Hackers Manual July 2011

### Name

`cfg80211_cqm_rssi_notify` — connection quality monitoring rssi event

### Synopsis

```
void cfg80211_cqm_rssi_notify (struct net_device * dev, enum  
nl80211_cqm_rssi_threshold_event rssi_event, gfp_t gfp);
```

### Arguments

*dev*

network device

*rssi\_event*

the triggered RSSI event

*gfp*

context flags

### Description

This function is called when a configured connection quality monitoring rssi threshold reached event occurs.

# cfg80211\_michael\_mic\_failure

## LINUX

Kernel Hackers Manual July 2011

### Name

`cfg80211_michael_mic_failure` — notification of Michael MIC failure (TKIP)

### Synopsis

```
void cfg80211_michael_mic_failure (struct net_device * dev,  
    const u8 * addr, enum nl80211_key_type key_type, int key_id,  
    const u8 * tsc, gfp_t gfp);
```

### Arguments

*dev*

network device

*addr*

The source MAC address of the frame

*key\_type*

The key type that the received frame used

*key\_id*

Key identifier (0..3)

*tsc*

The TSC value of the frame that generated the MIC failure (6 octets)

*gfp*

allocation flags

## **Description**

This function is called whenever the local MAC detects a MIC failure in a received frame. This matches with `MLME-MICHAELMICFAILURE.indication` primitive.

# Chapter 3. Scanning and BSS list handling

The scanning process itself is fairly simple, but `cfg80211` offers quite a bit of helper functionality. To start a scan, the scan operation will be invoked with a scan definition. This scan definition contains the channels to scan, and the SSIDs to send probe requests for (including the wildcard, if desired). A passive scan is indicated by having no SSIDs to probe. Additionally, a scan request may contain extra information elements that should be added to the probe request. The IEs are guaranteed to be well-formed, and will not exceed the maximum length the driver advertised in the `wiphy` structure.

When scanning finds a BSS, `cfg80211` needs to be notified of that, because it is responsible for maintaining the BSS list; the driver should not maintain a list itself. For this notification, various functions exist.

Since drivers do not maintain a BSS list, there are also a number of functions to search for a BSS and obtain information about it from the BSS structure `cfg80211` maintains. The BSS list is also made available to userspace.

## struct `cfg80211_ssid`

### LINUX

Kernel Hackers Manual July 2011

### Name

`struct cfg80211_ssid` — SSID description

### Synopsis

```
struct cfg80211_ssid {
    u8 ssid[IEEE80211_MAX_SSID_LEN];
    u8 ssid_len;
};
```

## Members

ssid[IEEE80211\_MAX\_SSID\_LEN]

the SSID

ssid\_len

length of the ssid

## struct cfg80211\_scan\_request

### LINUX

Kernel Hackers Manual July 2011

## Name

struct cfg80211\_scan\_request — scan request description

## Synopsis

```
struct cfg80211_scan_request {
    struct cfg80211_ssid * ssids;
    int n_ssids;
    u32 n_channels;
    const u8 * ie;
    size_t ie_len;
    struct wiphy * wiphy;
    struct net_device * dev;
    bool aborted;
    struct ieee80211_channel * channels[0];
};
```

## Members

ssids

SSIDs to scan for (active scan only)

n\_ssids

number of SSIDs

n\_channels

total number of channels to scan

ie

optional information element(s) to add into Probe Request or `NULL`

ie\_len

length of ie in octets

wiphy

the wiphy this was for

dev

the interface

aborted

(internal) scan request was notified as aborted

channels[0]

channels to scan on.

## cfg80211\_scan\_done

### LINUX

Kernel Hackers Manual July 2011

### Name

cfg80211\_scan\_done — notify that scan finished

## Synopsis

```
void cfg80211_scan_done (struct cfg80211_scan_request *  
request, bool aborted);
```

## Arguments

*request*

the corresponding scan request

*aborted*

set to true if the scan was aborted for any reason, userspace will be notified of that

## struct cfg80211\_bss

### LINUX

Kernel Hackers Manual July 2011

## Name

struct cfg80211\_bss — BSS description

## Synopsis

```
struct cfg80211_bss {  
    struct ieee80211_channel * channel;  
    u8 bssid[ETH_ALEN];  
    u64 tsf;  
    u16 beacon_interval;  
    u16 capability;  
    u8 * information_elements;  
    size_t len_information_elements;  
    u8 * beacon_ies;
```

```
size_t len_beacon_ies;  
u8 * proberesp_ies;  
size_t len_proberesp_ies;  
s32 signal;  
void (* free_priv) (struct cfg80211_bss *bss);  
u8 priv[0] __attribute__((__aligned__(sizeof(void *))));  
};
```

## Members

channel

channel this BSS is on

bssid[ETH\_ALEN]

BSSID of the BSS

tsf

timestamp of last received update

beacon\_interval

the beacon interval as from the frame

capability

the capability field in host byte order

information\_elements

the information elements (Note that there is no guarantee that these are well-formed!); this is a pointer to either the beacon\_ies or proberesp\_ies depending on whether Probe Response frame has been received

len\_information\_elements

total length of the information elements

beacon\_ies

the information elements from the last Beacon frame

len\_beacon\_ies

total length of the beacon\_ies

`proberesp_ies`

the information elements from the last Probe Response frame

`len_proberesp_ies`

total length of the `proberesp_ies`

`signal`

signal strength value (type depends on the wiphy's `signal_type`)

`free_priv`

function pointer to free private data

`priv[0] __attribute__((__aligned__(sizeof(void *))))`

private area for driver use, has at least `wiphy->bss_priv_size` bytes

## Description

This structure describes a BSS (which may also be a mesh network) for use in scan results and similar.

# cfg80211\_inform\_bss\_frame

## LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_inform_bss_frame` — inform `cfg80211` of a received BSS frame

## Synopsis

```
struct cfg80211_bss* cfg80211_inform_bss_frame (struct wiphy *  
wiphy, struct ieee80211_channel * channel, struct  
ieee80211_mgmt * mgmt, size_t len, s32 signal, gfp_t gfp);
```

## Arguments

*wiphy*

the wiphy reporting the BSS

*channel*

The channel the frame was received on

*mgmt*

the management frame (probe response or beacon)

*len*

length of the management frame

*signal*

the signal strength, type depends on the wiphy's `signal_type`

*gfp*

context flags

## Description

This informs `cfg80211` that BSS information was found and the BSS should be updated/added.

# cfg80211\_inform\_bss

**LINUX**

Kernel Hackers Manual July 2011

## Name

`cfg80211_inform_bss` — inform `cfg80211` of a new BSS

## Synopsis

```
struct cfg80211_bss* cfg80211_inform_bss (struct wiphy *  
wiphy, struct ieee80211_channel * channel, const u8 * bssid,  
u64 timestamp, u16 capability, u16 beacon_interval, const u8 *  
ie, size_t ielen, s32 signal, gfp_t gfp);
```

## Arguments

*wiphy*

the wiphy reporting the BSS

*channel*

The channel the frame was received on

*bssid*

the BSSID of the BSS

*timestamp*

the TSF timestamp sent by the peer

*capability*

the capability field sent by the peer

*beacon\_interval*

the beacon interval announced by the peer

*ie*

additional IEs sent by the peer

*ielen*

length of the additional IEs

*signal*

the signal strength, type depends on the wiphy's signal\_type

*gfp*

context flags

## Description

This informs cfg80211 that BSS information was found and the BSS should be updated/added.

# cfg80211\_unlink\_bss

## LINUX

Kernel Hackers Manual July 2011

## Name

cfg80211\_unlink\_bss — unlink BSS from internal data structures

## Synopsis

```
void cfg80211_unlink_bss (struct wiphy * wiphy, struct  
cfg80211_bss * bss);
```

## Arguments

*wiphy*

the wiphy

*bss*

the bss to remove

## Description

This function removes the given BSS from the internal data structures thereby making it no longer show up in scan results etc. Use this function when you detect a

BSS is gone. Normally BSSes will also time out, so it is not necessary to use this function at all.

## cfg80211\_find\_ie

### LINUX

Kernel Hackers Manual July 2011

### Name

`cfg80211_find_ie` — find information element in data

### Synopsis

```
const u8 * cfg80211_find_ie (u8 eid, const u8 * ies, int len);
```

### Arguments

*eid*

element ID

*ies*

data consisting of IEs

*len*

length of data

### Description

This function will return `NULL` if the element ID could not be found or if the element is invalid (claims to be longer than the given data), or a pointer to the first

byte of the requested element, that is the byte containing the element ID. There are no checks on the element length other than having to fit into the given data.

## ieee80211\_bss\_get\_ie

### LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_bss_get_ie` — find IE with given ID

### Synopsis

```
const u8 * ieee80211_bss_get_ie (struct cfg80211_bss * bss, u8
ie);
```

### Arguments

*bss*

the bss to search

*ie*

the IE ID Returns `NULL` if not found.



# Chapter 4. Utility functions

cfg80211 offers a number of utility functions that can be useful.

## ieee80211\_channel\_to\_frequency

**LINUX**

Kernel Hackers Manual July 2011

### Name

`ieee80211_channel_to_frequency` — convert channel number to frequency

### Synopsis

```
int ieee80211_channel_to_frequency (int chan);
```

### Arguments

*chan*

channel number

## ieee80211\_frequency\_to\_channel

**LINUX**

## Name

`ieee80211_frequency_to_channel` — convert frequency to channel number

## Synopsis

```
int ieee80211_frequency_to_channel (int freq);
```

## Arguments

*freq*

center frequency

# ieee80211\_get\_channel

## LINUX

## Name

`ieee80211_get_channel` — get channel struct from wiphy for specified frequency

## Synopsis

```
struct ieee80211_channel * ieee80211_get_channel (struct wiphy  
* wiphy, int freq);
```

## Arguments

*wiphy*

the struct `wiphy` to get the channel for

*freq*

the center frequency of the channel

# ieee80211\_get\_response\_rate

## LINUX

Kernel Hackers Manual July 2011

## Name

`ieee80211_get_response_rate` — get basic rate for a given rate

## Synopsis

```
struct ieee80211_rate * ieee80211_get_response_rate (struct
ieee80211_supported_band * sband, u32 basic_rates, int
bitrate);
```

## Arguments

*sband*

the band to look for rates in

*basic\_rates*

bitmap of basic rates

*bitrate*

the bitrate for which to find the basic rate

## Description

This function returns the basic rate corresponding to a given bitrate, that is the next lower bitrate contained in the basic rate map, which is, for this function, given as a bitmap of indices of rates in the band's bitrate table.

# ieee80211\_hdrlen

## LINUX

Kernel Hackers Manual July 2011

## Name

`ieee80211_hdrlen` — get header length in bytes from frame control

## Synopsis

```
unsigned int __attribute__((const)) ieee80211_hdrlen (__le16 fc);
```

## Arguments

*fc*

frame control field in little-endian format

# ieee80211\_get\_hdrlen\_from\_skb

## LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_get_hdrlen_from_skb` — get header length from data

### Synopsis

```
unsigned int ieee80211_get_hdrlen_from_skb (const struct
sk_buff * skb);
```

### Arguments

*skb*

the frame

### Description

Given an `skb` with a raw 802.11 header at the data pointer this function returns the 802.11 header length in bytes (not including encryption headers). If the data in the `sk_buff` is too short to contain a valid 802.11 header the function returns 0.

# struct ieee80211\_radiotap\_iterator

## LINUX

## Name

`struct ieee80211_radiotap_iterator` — tracks walk thru present radiotap args

## Synopsis

```
struct ieee80211_radiotap_iterator {
    struct ieee80211_radiotap_header * _rthdr;
    const struct ieee80211_radiotap_vendor_namespaces * _vns;
    const struct ieee80211_radiotap_namespace * current_namespace;
    unsigned char * _arg;
    unsigned char * _next_ns_data;
    __le32 * _next_bitmap;
    unsigned char * this_arg;
    int this_arg_index;
    int this_arg_size;
    int is_radiotap_ns;
    int _max_length;
    int _arg_index;
    uint32_t _bitmap_shifter;
    int _reset_on_ext;
};
```

## Members

`_rthdr`

pointer to the radiotap header we are walking through

`_vns`

vendor namespace definitions

`current_namespace`

pointer to the current namespace definition (or internally `NULL` if the current namespace is unknown)

`_arg`

next argument pointer

`_next_ns_data`

beginning of the next namespace's data

`_next_bitmap`

internal pointer to next present u32

`this_arg`

pointer to current radiotap arg; it is valid after each call to `ieee80211_radiotap_iterator_next` but also after `ieee80211_radiotap_iterator_init` where it will point to the beginning of the actual data portion

`this_arg_index`

index of current arg, valid after each successful call to `ieee80211_radiotap_iterator_next`

`this_arg_size`

length of the current arg, for convenience

`is_radiotap_ns`

indicates whether the current namespace is the default radiotap namespace or not

`_max_length`

length of radiotap header in cpu byte ordering

`_arg_index`

next argument index

`_bitmap_shifter`

internal shifter for curr u32 bitmap, b0 set == arg present

`_reset_on_ext`

internal; reset the arg index to 0 when going to the next bitmap word

## Description

Describes the radiotap parser state. Fields prefixed with an underscore must not be used by users of the parser, only by the parser internally.



# Chapter 5. Data path helpers

In addition to generic utilities, `cfg80211` also offers functions that help implement the data path for devices that do not do the 802.11/802.3 conversion on the device.

## ieee80211\_data\_to\_8023

### LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_data_to_8023` — convert an 802.11 data frame to 802.3

### Synopsis

```
int ieee80211_data_to_8023 (struct sk_buff * skb, const u8 *  
addr, enum nl80211_iftype iftype);
```

### Arguments

*skb*

the 802.11 data frame

*addr*

the device MAC address

*iftype*

the virtual interface type

# ieee80211\_data\_from\_8023

## LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_data_from_8023` — convert an 802.3 frame to 802.11

### Synopsis

```
int ieee80211_data_from_8023 (struct sk_buff * skb, const u8 *  
addr, enum nl80211_iftype iftype, u8 * bssid, bool qos);
```

### Arguments

*skb*

the 802.3 frame

*addr*

the device MAC address

*iftype*

the virtual interface type

*bssid*

the network bssid (used only for iftype STATION and ADHOC)

*qos*

build 802.11 QoS data frame

# ieee80211\_amsdu\_to\_8023s

## LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_amsdu_to_8023s` — decode an IEEE 802.11n A-MSDU frame

### Synopsis

```
void ieee80211_amsdu_to_8023s (struct sk_buff * skb, struct
sk_buff_head * list, const u8 * addr, enum nl80211_iftype
iftype, const unsigned int extra_headroom);
```

### Arguments

*skb*

The input IEEE 802.11n A-MSDU frame.

*list*

The output list of 802.3 frames. It must be allocated and initialized by the caller.

*addr*

The device MAC address.

*iftype*

The device interface type.

*extra\_headroom*

The hardware extra headroom for SKBs in the *list*.

## Description

Decode an IEEE 802.11n A-MSDU frame and convert it to a list of 802.3 frames. The *list* will be empty if the decode fails. The *skb* is consumed after the function returns.

## cfg80211\_classify8021d

### LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_classify8021d` — determine the 802.1p/1d tag for a data frame

## Synopsis

```
unsigned int cfg80211_classify8021d (struct sk_buff * skb);
```

## Arguments

*skb*

the data frame

# Chapter 6. Regulatory enforcement infrastructure

TODO

## regulatory\_hint

**LINUX**

Kernel Hackers Manual July 2011

### Name

`regulatory_hint` — driver hint to the wireless core a regulatory domain

### Synopsis

```
int regulatory_hint (struct wiphy * wiphy, const char *  
alpha2);
```

### Arguments

*wiphy*

the wireless device giving the hint (used only for reporting conflicts)

*alpha2*

the ISO/IEC 3166 alpha2 the driver claims its regulatory domain should be in. If *rd* is set this should be NULL. Note that if you set this to NULL you should still set *rd->alpha2* to some accepted alpha2.

## Description

Wireless drivers can use this function to hint to the wireless core what it believes should be the current regulatory domain by giving it an ISO/IEC 3166 alpha2 country code it knows its regulatory domain should be in or by providing a completely build regulatory domain. If the driver provides an ISO/IEC 3166 alpha2 userspace will be queried for a regulatory domain structure for the respective country.

The wiphy must have been registered to `cfg80211` prior to this call. For `cfg80211` drivers this means you must first use `wiphy_register`, for `mac80211` drivers you must first use `ieee80211_register_hw`.

Drivers should check the return value, its possible you can get an `-ENOMEM`.

# wiphy\_apply\_custom\_regulatory

## LINUX

Kernel Hackers Manual July 2011

## Name

`wiphy_apply_custom_regulatory` — apply a custom driver regulatory domain

## Synopsis

```
void wiphy_apply_custom_regulatory (struct wiphy * wiphy,  
const struct ieee80211_regdomain * regd);
```

## Arguments

*wiphy*

the wireless device we want to process the regulatory domain on

*regd*

the custom regulatory domain to use for this wiphy

## Description

Drivers can sometimes have custom regulatory domains which do not apply to a specific country. Drivers can use this to apply such custom regulatory domains. This routine must be called prior to wiphy registration. The custom regulatory domain will be trusted completely and as such previous default channel settings will be disregarded. If no rule is found for a channel on the regulatory domain the channel will be disabled.

## freq\_reg\_info

### LINUX

Kernel Hackers Manual July 2011

### Name

`freq_reg_info` — get regulatory information for the given frequency

### Synopsis

```
int freq_reg_info (struct wiphy * wiphy, u32 center_freq, u32
desired_bw_khz, const struct ieee80211_reg_rule ** reg_rule);
```

### Arguments

*wiphy*

the wiphy for which we want to process this rule for

*center\_freq*

Frequency in KHz for which we want regulatory information for

*desired\_bw\_khz*

the desired max bandwidth you want to use per channel. Note that this is still 20 MHz if you want to use HT40 as HT40 makes use of two channels for its 40 MHz width bandwidth. If set to 0 we'll assume you want the standard 20 MHz.

*reg\_rule*

the regulatory rule which we have for this frequency

## Description

Use this function to get the regulatory rule for a specific frequency on a given wireless device. If the device has a specific regulatory domain it wants to follow we respect that unless a country IE has been received and processed already.

Returns 0 if it was able to find a valid regulatory rule which does apply to the given *center\_freq* otherwise it returns non-zero. It will also return -ERANGE if we determine the given *center\_freq* does not even have a regulatory rule for a frequency range in the *center\_freq*'s band. See *freq\_in\_rule\_band* for our current definition of a band -- this is purely subjective and right now its 802.11 specific.

# Chapter 7. RFkill integration

RFkill integration in `cfg80211` is almost invisible to drivers, as `cfg80211` automatically registers an rfkill instance for each wireless device it knows about. Soft kill is also translated into disconnecting and turning all interfaces off, drivers are expected to turn off the device when all interfaces are down.

However, devices may have a hard RFkill line, in which case they also need to interact with the rfkill subsystem, via `cfg80211`. They can do this with a few helper functions documented here.

## wiphy\_rfkill\_set\_hw\_state

### LINUX

Kernel Hackers Manual July 2011

### Name

`wiphy_rfkill_set_hw_state` — notify `cfg80211` about hw block state

### Synopsis

```
void wiphy_rfkill_set_hw_state (struct wiphy * wiphy, bool  
blocked);
```

### Arguments

*wiphy*

the wiphy

*blocked*

block status

# wiphy\_rfkill\_start\_polling

## LINUX

Kernel Hackers Manual July 2011

### Name

wiphy\_rfkill\_start\_polling — start polling rfkill

### Synopsis

```
void wiphy_rfkill_start_polling (struct wiphy * wiphy);
```

### Arguments

*wiphy*

the wiphy

# wiphy\_rfkill\_stop\_polling

## LINUX

Kernel Hackers Manual July 2011

### Name

wiphy\_rfkill\_stop\_polling — stop polling rfkill

## Synopsis

```
void wiphy_rfkill_stop_polling (struct wiphy * wiphy);
```

## Arguments

*wiphy*

the wiphy



# Chapter 8. Test mode

Test mode is a set of utility functions to allow drivers to interact with driver-specific tools to aid, for instance, factory programming.

This chapter describes how drivers interact with it, for more information see the nl80211 book's chapter on it.

## cfg80211\_testmode\_alloc\_reply\_skb

### LINUX

Kernel Hackers Manual July 2011

### Name

`cfg80211_testmode_alloc_reply_skb` — allocate testmode reply

### Synopsis

```
struct sk_buff * cfg80211_testmode_alloc_reply_skb (struct wiphy * wiphy, int approxlen);
```

### Arguments

*wiphy*

the wiphy

*approxlen*

an upper bound of the length of the data that will be put into the skb

## Description

This function allocates and pre-fills an `skb` for a reply to the `testmode` command. Since it is intended for a reply, calling it outside of the `testmode_cmd` operation is invalid.

The returned `skb` (or `NULL` if any errors happen) is pre-filled with the `wiphy` index and set up in a way that any data that is put into the `skb` (with `skb_put`, `nla_put` or similar) will end up being within the `NL80211_ATTR_TESTDATA` attribute, so all that needs to be done with the `skb` is adding data for the corresponding userspace tool which can then read that data out of the `testdata` attribute. You must not modify the `skb` in any other way.

When done, call `cfg80211_testmode_reply` with the `skb` and return its error code as the result of the `testmode_cmd` operation.

# cfg80211\_testmode\_reply

## LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_testmode_reply` — send the reply `skb`

## Synopsis

```
int cfg80211_testmode_reply (struct sk_buff * skb);
```

## Arguments

*skb*

The `skb`, must have been allocated with  
`cfg80211_testmode_alloc_reply_skb`

## Description

Returns an error code or 0 on success, since calling this function will usually be the last thing before returning from the *testmode\_cmd* you should return the error code. Note that this function consumes the *skb* regardless of the return value.

# cfg80211\_testmode\_alloc\_event\_skb

## LINUX

Kernel Hackers Manual July 2011

## Name

`cfg80211_testmode_alloc_event_skb` — allocate testmode event

## Synopsis

```
struct sk_buff * cfg80211_testmode_alloc_event_skb (struct
wiphy * wiphy, int approxlen, gfp_t gfp);
```

## Arguments

*wiphy*

the wiphy

*approxlen*

an upper bound of the length of the data that will be put into the *skb*

*gfp*

allocation flags

## Description

This function allocates and pre-fills an skb for an event on the testmode multicast group.

The returned skb (or NULL if any errors happen) is set up in the same way as with `cfg80211_testmode_alloc_reply_skb` but prepared for an event. As there, you should simply add data to it that will then end up in the `NL80211_ATTR_TESTDATA` attribute. Again, you must not modify the skb in any other way.

When done filling the skb, call `cfg80211_testmode_event` with the skb to send the event.

## cfg80211\_testmode\_event

### LINUX

Kernel Hackers Manual July 2011

### Name

`cfg80211_testmode_event` — send the event

### Synopsis

```
void cfg80211_testmode_event (struct sk_buff * skb, gfp_t  
gfp);
```

### Arguments

*skb*

The skb, must have been allocated with  
`cfg80211_testmode_alloc_event_skb`

*gfp*

allocation flags

## Description

This function sends the given *skb*, which must have been allocated by `cfg80211_testmode_alloc_event_skb`, as an event. It always consumes it.



# The mac80211 subsystem

## **The mac80211 subsystem**

mac80211 is the Linux stack for 802.11 hardware that implements only partial functionality in hard- or firmware. This document defines the interface between mac80211 and low-level hardware drivers.

If you're reading this document and not the header file itself, it will be incomplete because not all documentation has been converted yet.

# Table of Contents

<b>I. The basic mac80211 driver interface.....</b>	<b>cxxxiii</b>
1. Basic hardware handling.....	1
struct ieee80211_hw .....	1
enum ieee80211_hw_flags.....	3
SET_IEEE80211_DEV.....	7
SET_IEEE80211_PERM_ADDR.....	8
struct ieee80211_ops.....	9
ieee80211_alloc_hw .....	15
ieee80211_register_hw .....	16
ieee80211_get_tx_led_name.....	16
ieee80211_get_rx_led_name .....	17
ieee80211_get_assoc_led_name .....	18
ieee80211_get_radio_led_name .....	19
ieee80211_unregister_hw .....	20
ieee80211_free_hw .....	21
2. PHY configuration .....	23
struct ieee80211_conf .....	23
enum ieee80211_conf_flags.....	25
3. Virtual interfaces.....	27
struct ieee80211_vif.....	27
4. Receive and transmit processing.....	29
4.1. what should be here .....	29
4.2. Frame format.....	29
4.3. Packet alignment.....	29
4.4. Calling into mac80211 from interrupts.....	30
4.5. functions/definitions.....	30
struct ieee80211_rx_status.....	30
enum mac80211_rx_flags .....	32
struct ieee80211_tx_info.....	33
ieee80211_rx.....	35
ieee80211_rx_irqsafe .....	36
ieee80211_tx_status .....	37
ieee80211_tx_status_irqsafe .....	38
ieee80211_rts_get .....	39
ieee80211_rts_duration.....	40
ieee80211_ctstoself_get.....	41
ieee80211_ctstoself_duration .....	42
ieee80211_generic_frame_duration.....	43
ieee80211_wake_queue .....	45
ieee80211_stop_queue.....	45
ieee80211_wake_queues.....	46

ieee80211_stop_queues .....	47
5. Frame filtering.....	49
enum ieee80211_filter_flags .....	49
<b>II. Advanced driver interface.....</b>	<b>53</b>
6. Hardware crypto acceleration .....	55
enum set_key_cmd.....	55
struct ieee80211_key_conf .....	56
enum ieee80211_key_flags .....	58
7. Powersave support .....	61
8. Beacon filter support.....	63
ieee80211_beacon_loss.....	64
9. Multiple queues and QoS support.....	65
struct ieee80211_tx_queue_params .....	65
10. Access point mode support .....	67
ieee80211_get_buffered_bc .....	67
ieee80211_beacon_get.....	68
11. Supporting multiple virtual interfaces .....	71
12. Hardware scan offload .....	73
ieee80211_scan_completed .....	73
<b>III. Rate control interface.....</b>	<b>75</b>
13. dummy chapter.....	77
<b>IV. Internals.....</b>	<b>79</b>
14. Key handling .....	81
14.1. Key handling basics .....	81
14.2. MORE TBD .....	81
15. Receive processing.....	83
16. Transmit processing .....	85
17. Station info handling.....	87
17.1. Programming information.....	87
struct sta_info.....	87
enum ieee80211_sta_info_flags.....	92
17.2. STA information lifetime rules .....	94
18. Synchronisation.....	97

# I. The basic mac80211 driver interface

## Table of Contents

<b>1. Basic hardware handling.....</b>	<b>1</b>
<b>2. PHY configuration .....</b>	<b>23</b>
<b>3. Virtual interfaces.....</b>	<b>27</b>
<b>4. Receive and transmit processing .....</b>	<b>29</b>
<b>5. Frame filtering.....</b>	<b>49</b>

You should read and understand the information contained within this part of the book while implementing a driver. In some chapters, advanced usage is noted, that may be skipped at first.

This part of the book only covers station and monitor mode functionality, additional information required to implement the other modes is covered in the second part of the book.



# Chapter 1. Basic hardware handling

TBD

This chapter shall contain information on getting a hw struct allocated and registered with mac80211.

Since it is required to allocate rates/modes before registering a hw struct, this chapter shall also contain information on setting up the rate/mode structs.

Additionally, some discussion about the callbacks and the general programming model should be in here, including the definition of ieee80211\_ops which will be referred to a lot.

Finally, a discussion of hardware capabilities should be done with references to other parts of the book.

## struct ieee80211\_hw

**LINUX**

Kernel Hackers Manual July 2011

### Name

struct ieee80211\_hw — hardware information and state

### Synopsis

```
struct ieee80211_hw {
    struct ieee80211_conf conf;
    struct wiphy * wiphy;
    const char * rate_control_algorithm;
    void * priv;
    u32 flags;
    unsigned int extra_tx_headroom;
    int channel_change_time;
    int vif_data_size;
    int sta_data_size;
    int napi_weight;
    u16 queues;
    u16 max_listen_interval;
    s8 max_signal;
    u8 max_rates;
```

```
    u8 max_report_rates;  
    u8 max_rate_tries;  
};
```

## Members

`conf`

struct `ieee80211_conf`, device configuration, don't use.

`wiphy`

This points to the struct `wiphy` allocated for this 802.11 PHY. You must fill in the `perm_addr` and `dev` members of this structure using `SET_IEEE80211_DEV` and `SET_IEEE80211_PERM_ADDR`. Additionally, all supported bands (with channels, bitrates) are registered here.

`rate_control_algorithm`

rate control algorithm for this hardware. If unset (NULL), the default algorithm will be used. Must be set before calling `ieee80211_register_hw`.

`priv`

pointer to private area that was allocated for driver use along with this structure.

`flags`

hardware flags, see enum `ieee80211_hw_flags`.

`extra_tx_headroom`

headroom to reserve in each transmit skb for use by the driver (e.g. for transmit headers.)

`channel_change_time`

time (in microseconds) it takes to change channels.

`vif_data_size`

size (in bytes) of the `drv_priv` data area within struct `ieee80211_vif`.

`sta_data_size`

size (in bytes) of the `drv_priv` data area within struct `ieee80211_sta`.

napi\_weight

weight used for NAPI polling. You must specify an appropriate value here if a napi\_poll operation is provided by your driver.

queues

number of available hardware transmit queues for data packets. WMM/QoS requires at least four, these queues need to have configurable access parameters.

max\_listen\_interval

max listen interval in units of beacon interval that HW supports

max\_signal

Maximum value for signal (rssi) in RX information, used only when *IEEE80211\_HW\_SIGNAL\_UNSPEC* or *IEEE80211\_HW\_SIGNAL\_DB*

max\_rates

maximum number of alternate rate retry stages the hw can handle.

max\_report\_rates

maximum number of alternate rate retry stages the hw can report back.

max\_rate\_tries

maximum number of tries for each stage

## Description

This structure contains the configuration and hardware information for an 802.11 PHY.

## enum ieee80211\_hw\_flags

**LINUX**

## Name

enum ieee80211\_hw\_flags — hardware flags

## Synopsis

```
enum ieee80211_hw_flags {
    IEEE80211_HW_HAS_RATE_CONTROL,
    IEEE80211_HW_RX_INCLUDES_FCS,
    IEEE80211_HW_HOST_BROADCAST_PS_BUFFERING,
    IEEE80211_HW_2GHZ_SHORT_SLOT_INCAPABLE,
    IEEE80211_HW_2GHZ_SHORT_PREAMBLE_INCAPABLE,
    IEEE80211_HW_SIGNAL_UNSPEC,
    IEEE80211_HW_SIGNAL_DBM,
    IEEE80211_HW_NEED_DTIM_PERIOD,
    IEEE80211_HW_SPECTRUM_MGMT,
    IEEE80211_HW_AMPDU_AGGREGATION,
    IEEE80211_HW_SUPPORTS_PS,
    IEEE80211_HW_PS_NULLFUNC_STACK,
    IEEE80211_HW_SUPPORTS_DYNAMIC_PS,
    IEEE80211_HW_MFP_CAPABLE,
    IEEE80211_HW_BEACON_FILTER,
    IEEE80211_HW_SUPPORTS_STATIC_SMPS,
    IEEE80211_HW_SUPPORTS_DYNAMIC_SMPS,
    IEEE80211_HW_SUPPORTS_UAPSD,
    IEEE80211_HW_REPORTS_TX_ACK_STATUS,
    IEEE80211_HW_CONNECTION_MONITOR,
    IEEE80211_HW_SUPPORTS_CQM_RSSI,
    IEEE80211_HW_SUPPORTS_PER_STA_GTK
};
```

## Constants

### IEEE80211\_HW\_HAS\_RATE\_CONTROL

The hardware or firmware includes rate control, and cannot be controlled by the stack. As such, no rate control algorithm should be instantiated, and the TX rate reported to userspace will be taken from the TX status instead of the rate control algorithm. Note that this requires that the driver implement a number of callbacks so it has the correct information, it needs to have the *set\_rts\_threshold* callback and must look at the BSS config

*use\_cts\_prot* for G/N protection, *use\_short\_slot* for slot timing in 2.4 GHz and *use\_short\_preamble* for preambles for CCK frames.

#### IEEE80211\_HW\_RX\_INCLUDES\_FCS

Indicates that received frames passed to the stack include the FCS at the end.

#### IEEE80211\_HW\_HOST\_BROADCAST\_PS\_BUFFERING

Some wireless LAN chipsets buffer broadcast/multicast frames for power saving stations in the hardware/firmware and others rely on the host system for such buffering. This option is used to configure the IEEE 802.11 upper layer to buffer broadcast and multicast frames when there are power saving stations so that the driver can fetch them with *ieee80211\_get\_buffered\_bc*.

#### IEEE80211\_HW\_2GHZ\_SHORT\_SLOT\_INCAPABLE

Hardware is not capable of short slot operation on the 2.4 GHz band.

#### IEEE80211\_HW\_2GHZ\_SHORT\_PREAMBLE\_INCAPABLE

Hardware is not capable of receiving frames with short preamble on the 2.4 GHz band.

#### IEEE80211\_HW\_SIGNAL\_UNSPEC

Hardware can provide signal values but we don't know its units. We expect values between 0 and *max\_signal*. If possible please provide dB or dBm instead.

#### IEEE80211\_HW\_SIGNAL\_DBM

Hardware gives signal values in dBm, decibel difference from one milliwatt. This is the preferred method since it is standardized between different devices. *max\_signal* does not need to be set.

#### IEEE80211\_HW\_NEED\_DTIM\_PERIOD

This device needs to know the DTIM period for the BSS before associating.

#### IEEE80211\_HW\_SPECTRUM\_MGMT

Hardware supports spectrum management defined in 802.11h Measurement, Channel Switch, Quieting, TPC

#### IEEE80211\_HW\_AMPDU\_AGGREGATION

Hardware supports 11n A-MPDU aggregation.

#### IEEE80211\_HW\_SUPPORTS\_PS

Hardware has power save support (i.e. can go to sleep).

IEEE80211\_HW\_PS\_NULLFUNC\_STACK

Hardware requires nullfunc frame handling in stack, implies stack support for dynamic PS.

IEEE80211\_HW\_SUPPORTS\_DYNAMIC\_PS

Hardware has support for dynamic PS.

IEEE80211\_HW\_MFP\_CAPABLE

Hardware supports management frame protection (MFP, IEEE 802.11w).

IEEE80211\_HW\_BEACON\_FILTER

Hardware supports dropping of irrelevant beacon frames to avoid waking up cpu.

IEEE80211\_HW\_SUPPORTS\_STATIC\_SMPS

Hardware supports static spatial multiplexing powersave, ie. can turn off all but one chain even on HT connections that should be using more chains.

IEEE80211\_HW\_SUPPORTS\_DYNAMIC\_SMPS

Hardware supports dynamic spatial multiplexing powersave, ie. can turn off all but one chain and then wake the rest up as required after, for example, rts/cts handshake.

IEEE80211\_HW\_SUPPORTS\_UAPSD

Hardware supports Unscheduled Automatic Power Save Delivery (U-APSD) in managed mode. The mode is configured with `conf_tx` operation.

IEEE80211\_HW\_REPORTS\_TX\_ACK\_STATUS

Hardware can provide ack status reports of Tx frames to the stack.

IEEE80211\_HW\_CONNECTION\_MONITOR

The hardware performs its own connection monitoring, including periodic keep-alives to the AP and probing the AP on beacon loss. When this flag is set, signaling beacon-loss will cause an immediate change to disassociated state.

IEEE80211\_HW\_SUPPORTS\_CQM\_RSSI

Hardware can do connection quality monitoring - i.e. it can monitor connection quality related parameters, such as the RSSI level and provide notifications if configured trigger levels are reached.

## IEEE80211\_HW\_SUPPORTS\_PER\_STA\_GTK

The device's crypto engine supports per-station GTKs as used by IBSS RSN or during fast transition. If the device doesn't support per-station GTKs, but can be asked not to decrypt group addressed frames, then IBSS RSN support is still possible but software crypto will be used. Advertise the wiphy flag only in that case.

## Description

These flags are used to indicate hardware capabilities to the stack. Generally, flags here should have their meaning done in a way that the simplest hardware doesn't need setting any particular flags. There are some exceptions to this rule, however, so you are advised to review these flags carefully.

# SET\_IEEE80211\_DEV

## LINUX

Kernel Hackers Manual July 2011

## Name

SET\_IEEE80211\_DEV — set device for 802.11 hardware

## Synopsis

```
void SET_IEEE80211_DEV (struct ieee80211_hw * hw, struct  
device * dev);
```

## Arguments

*hw*

the struct `ieee80211_hw` to set the device for

*dev*

the struct device of this 802.11 device

# SET\_IEEE80211\_PERM\_ADDR

## LINUX

Kernel Hackers Manual July 2011

## Name

`SET_IEEE80211_PERM_ADDR` — set the permanent MAC address for 802.11 hardware

## Synopsis

```
void SET_IEEE80211_PERM_ADDR (struct ieee80211_hw * hw, u8 *  
addr);
```

## Arguments

*hw*

the struct `ieee80211_hw` to set the MAC address for

*addr*

the address to set

# struct ieee80211\_ops

## LINUX

Kernel Hackers Manual July 2011

## Name

struct ieee80211\_ops — callbacks from mac80211 to the driver

## Synopsis

```
struct ieee80211_ops {
    int (* tx) (struct ieee80211_hw *hw, struct sk_buff *skb);
    int (* start) (struct ieee80211_hw *hw);
    void (* stop) (struct ieee80211_hw *hw);
    int (* add_interface) (struct ieee80211_hw *hw, struct ieee80211_vif *vif);
    int (* change_interface) (struct ieee80211_hw *hw, struct ieee80211_vif *vif);
    void (* remove_interface) (struct ieee80211_hw *hw, struct ieee80211_vif *vif);
    int (* config) (struct ieee80211_hw *hw, u32 changed);
    void (* bss_info_changed) (struct ieee80211_hw *hw, struct ieee80211_vif *vif,
        u64 (* prepare_multicast) (struct ieee80211_hw *hw, struct netdev_hw_addr *na,
        void (* configure_filter) (struct ieee80211_hw *hw, unsigned int changed);
    int (* set_tim) (struct ieee80211_hw *hw, struct ieee80211_sta *sta, bool tim_changed);
    int (* set_key) (struct ieee80211_hw *hw, enum set_key_cmd cmd, struct ieee80211_sta *sta,
        void (* update_tkip_key) (struct ieee80211_hw *hw, struct ieee80211_vif *vif,
        int (* hw_scan) (struct ieee80211_hw *hw, struct ieee80211_vif *vif, struct ieee80211_scan_req *req);
    void (* sw_scan_start) (struct ieee80211_hw *hw);
    void (* sw_scan_complete) (struct ieee80211_hw *hw);
    int (* get_stats) (struct ieee80211_hw *hw, struct ieee80211_low_level_stats *stats);
    void (* get_tkip_seq) (struct ieee80211_hw *hw, u8 hw_key_idx, u32 *iv32);
    int (* set_rts_threshold) (struct ieee80211_hw *hw, u32 value);
    int (* sta_add) (struct ieee80211_hw *hw, struct ieee80211_vif *vif, struct ieee80211_sta *sta);
    int (* sta_remove) (struct ieee80211_hw *hw, struct ieee80211_vif *vif, struct ieee80211_sta *sta);
    void (* sta_notify) (struct ieee80211_hw *hw, struct ieee80211_vif *vif, struct ieee80211_sta *sta);
    int (* conf_tx) (struct ieee80211_hw *hw, u16 queue, const struct ieee80211_tx_info *info,
        u64 (* get_tsf) (struct ieee80211_hw *hw);
    void (* set_tsf) (struct ieee80211_hw *hw, u64 tsf);
    void (* reset_tsf) (struct ieee80211_hw *hw);
    int (* tx_last_beacon) (struct ieee80211_hw *hw);
    int (* ampdu_action) (struct ieee80211_hw *hw, struct ieee80211_vif *vif, struct ieee80211_sta *sta,
        u16 queue, const struct ieee80211_tx_info *info, struct sk_buff *skb);
};
```

```
int (* get_survey) (struct ieee80211_hw *hw, int idx, struct survey_info *info);
void (* rfkill_poll) (struct ieee80211_hw *hw);
void (* set_coverage_class) (struct ieee80211_hw *hw, u8 coverage_class);
#ifdef CONFIG_NL80211_TESTMODE
int (* testmode_cmd) (struct ieee80211_hw *hw, void *data, int len);
#endif
void (* flush) (struct ieee80211_hw *hw, bool drop);
void (* channel_switch) (struct ieee80211_hw *hw, struct ieee80211_channel *chan, int *tx_rate);
int (* napi_poll) (struct ieee80211_hw *hw, int budget);
};
```

## Members

**tx**

Handler that 802.11 module calls for each transmitted frame. `skb` contains the buffer starting from the IEEE 802.11 header. The low-level driver should send the frame out based on configuration in the TX control data. This handler should, preferably, never fail and stop queues appropriately, more importantly, however, it must never fail for A-MPDU-queues. This function should return `NETDEV_TX_OK` except in very limited cases. Must be implemented and atomic.

**start**

Called before the first netdevice attached to the hardware is enabled. This should turn on the hardware and must turn on frame reception (for possibly enabled monitor interfaces.) Returns negative error codes, these may be seen in userspace, or zero. When the device is started it should not have a MAC address to avoid acknowledging frames before a non-monitor device is added. Must be implemented and can sleep.

**stop**

Called after last netdevice attached to the hardware is disabled. This should turn off the hardware (at least it must turn off frame reception.) May be called right after `add_interface` if that rejects an interface. If you added any work onto the `mac80211` workqueue you should ensure to cancel it on this callback. Must be implemented and can sleep.

**add\_interface**

Called when a netdevice attached to the hardware is enabled. Because it is not called for monitor mode devices, `start` and `stop` must be implemented. The driver should perform any initialization it needs before the device can be

enabled. The initial configuration for the interface is given in the `conf` parameter. The callback may refuse to add an interface by returning a negative error code (which will be seen in userspace.) Must be implemented and can sleep.

#### `change_interface`

Called when a netdevice changes type. This callback is optional, but only if it is supported can interface types be switched while the interface is UP. The callback may sleep. Note that while an interface is being switched, it will not be found by the interface iteration callbacks.

#### `remove_interface`

Notifies a driver that an interface is going down. The `stop` callback is called after this if it is the last interface and no monitor interfaces are present. When all interfaces are removed, the MAC address in the hardware must be cleared so the device no longer acknowledges packets, the `mac_addr` member of the `conf` structure is, however, set to the MAC address of the device going away. Hence, this callback must be implemented. It can sleep.

#### `config`

Handler for configuration requests. IEEE 802.11 code calls this function to change hardware configuration, e.g., channel. This function should never fail but returns a negative error code if it does. The callback can sleep.

#### `bss_info_changed`

Handler for configuration requests related to BSS parameters that may vary during BSS's lifespan, and may affect low level driver (e.g. assoc/disassoc status, erp parameters). This function should not be used if no BSS has been set, unless for association indication. The `changed` parameter indicates which of the bss parameters has changed when a call is made. The callback can sleep.

#### `prepare_multicast`

Prepare for multicast filter configuration. This callback is optional, and its return value is passed to `configure_filter`. This callback must be atomic.

#### `configure_filter`

Configure the device's RX filter. See the section "Frame filtering" for more information. This callback must be implemented and can sleep.

#### `set_tim`

Set TIM bit. `mac80211` calls this function when a TIM bit must be set or cleared for a given STA. Must be atomic.

#### set\_key

See the section “Hardware crypto acceleration” This callback is only called between `add_interface` and `remove_interface` calls, i.e. while the given virtual interface is enabled. Returns a negative error code if the key can’t be added. The callback can sleep.

#### update\_tkip\_key

See the section “Hardware crypto acceleration” This callback will be called in the context of Rx. Called for drivers which set `IEEE80211_KEY_FLAG_TKIP_REQ_RX_P1_KEY`. The callback must be atomic.

#### hw\_scan

Ask the hardware to service the scan request, no need to start the scan state machine in stack. The scan must honour the channel configuration done by the regulatory agent in the wiphy’s registered bands. The hardware (or the driver) needs to make sure that power save is disabled. The `req_ie/ie_len` members are rewritten by `mac80211` to contain the entire IEs after the SSID, so that drivers need not look at these at all but just send them after the SSID -- `mac80211` includes the (extended) supported rates and HT information (where applicable). When the scan finishes, `ieee80211_scan_completed` must be called; note that it also must be called when the scan cannot finish due to any error unless this callback returned a negative error code. The callback can sleep.

#### sw\_scan\_start

Notifier function that is called just before a software scan is started. Can be NULL, if the driver doesn’t need this notification. The callback can sleep.

#### sw\_scan\_complete

Notifier function that is called just after a software scan finished. Can be NULL, if the driver doesn’t need this notification. The callback can sleep.

#### get\_stats

Return low-level statistics. Returns zero if statistics are available. The callback can sleep.

#### get\_tkip\_seq

If your device implements TKIP encryption in hardware this callback should be provided to read the TKIP transmit IVs (both IV32 and IV16) for the given key from hardware. The callback must be atomic.

`set_rts_threshold`

Configuration of RTS threshold (if device needs it) The callback can sleep.

`sta_add`

Notifies low level driver about addition of an associated station, AP, IBSS/WDS/mesh peer etc. This callback can sleep.

`sta_remove`

Notifies low level driver about removal of an associated station, AP, IBSS/WDS/mesh peer etc. This callback can sleep.

`sta_notify`

Notifies low level driver about power state transition of an associated station, AP, IBSS/WDS/mesh peer etc. Must be atomic.

`conf_tx`

Configure TX queue parameters (EDCF (aifs, cw\_min, cw\_max), bursting) for a hardware TX queue. Returns a negative error code on failure. The callback can sleep.

`get_tsf`

Get the current TSF timer value from firmware/hardware. Currently, this is only used for IBSS mode BSSID merging and debugging. Is not a required function. The callback can sleep.

`set_tsf`

Set the TSF timer to the specified value in the firmware/hardware. Currently, this is only used for IBSS mode debugging. Is not a required function. The callback can sleep.

`reset_tsf`

Reset the TSF timer and allow firmware/hardware to synchronize with other STAs in the IBSS. This is only used in IBSS mode. This function is optional if the firmware/hardware takes full care of TSF synchronization. The callback can sleep.

`tx_last_beacon`

Determine whether the last IBSS beacon was sent by us. This is needed only for IBSS mode and the result of this function is used to determine whether to reply to Probe Requests. Returns non-zero if this device sent the last beacon. The callback can sleep.

#### `ampdu_action`

Perform a certain A-MPDU action. The RA/TID combination determines the destination and TID we want the ampdu action to be performed for. The action is defined through `ieee80211_ampdu_mlme_action`. Starting sequence number (*ssn*) is the first frame we expect to perform the action on. Notice that TX/RX\_STOP can pass NULL for this parameter. Returns a negative error code on failure. The callback can sleep.

#### `get_survey`

Return per-channel survey information

#### `rkill_poll`

Poll rkill hardware state. If you need this, you also need to set `wiphy->rkill_poll` to `true` before registration, and need to call `wiphy_rkill_set_hw_state` in the callback. The callback can sleep.

#### `set_coverage_class`

Set slot time for given coverage class as specified in IEEE 802.11-2007 section 17.3.8.6 and modify ACK timeout accordingly. This callback is not required and may sleep.

#### `testmode_cmd`

Implement a `cfg80211` test mode command. The callback can sleep.

#### `flush`

Flush all pending frames from the hardware queue, making sure that the hardware queues are empty. If the parameter *drop* is set to `true`, pending frames may be dropped. The callback can sleep.

#### `channel_switch`

Drivers that need (or want) to offload the channel switch operation for CSAs received from the AP may implement this callback. They must then call `ieee80211_chswitch_done` to indicate completion of the channel switch.

#### `napi_poll`

Poll Rx queue for incoming data frames.

## Description

This structure contains various callbacks that the driver may handle or, in some cases, must handle, for example to configure the hardware to a new channel or to transmit a frame.

## ieee80211\_alloc\_hw

### LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_alloc_hw` — Allocate a new hardware device

### Synopsis

```
struct ieee80211_hw * ieee80211_alloc_hw (size_t  
priv_data_len, const struct ieee80211_ops * ops);
```

### Arguments

*priv\_data\_len*

length of private data

*ops*

callbacks for this device

### Description

This must be called once for each hardware device. The returned pointer must be used to refer to this device when calling other functions. `mac80211` allocates a

private data area for the driver pointed to by *priv* in struct `ieee80211_hw`, the size of this area is given as *priv\_data\_len*.

## ieee80211\_register\_hw

### LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_register_hw` — Register hardware device

### Synopsis

```
int ieee80211_register_hw (struct ieee80211_hw * hw);
```

### Arguments

*hw*

the device to register as returned by `ieee80211_alloc_hw`

### Description

You must call this function before any other functions in `mac80211`. Note that before a hardware can be registered, you need to fill the contained wiphy's information.

# ieee80211\_get\_tx\_led\_name

## LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_get_tx_led_name` — get name of TX LED

### Synopsis

```
char * ieee80211_get_tx_led_name (struct ieee80211_hw * hw);
```

### Arguments

*hw*

the hardware to get the LED trigger name for

### Description

mac80211 creates a transmit LED trigger for each wireless hardware that can be used to drive LEDs if your driver registers a LED device. This function returns the name (or NULL if not configured for LEDs) of the trigger so you can automatically link the LED device.

# ieee80211\_get\_rx\_led\_name

## LINUX

## Name

`ieee80211_get_rx_led_name` — get name of RX LED

## Synopsis

```
char * ieee80211_get_rx_led_name (struct ieee80211_hw * hw);
```

## Arguments

*hw*

the hardware to get the LED trigger name for

## Description

mac80211 creates a receive LED trigger for each wireless hardware that can be used to drive LEDs if your driver registers a LED device. This function returns the name (or `NULL` if not configured for LEDs) of the trigger so you can automatically link the LED device.

# ieee80211\_get\_assoc\_led\_name

## LINUX

## Name

`ieee80211_get_assoc_led_name` — get name of association LED

## Synopsis

```
char * ieee80211_get_assoc_led_name (struct ieee80211_hw *  
hw);
```

## Arguments

*hw*

the hardware to get the LED trigger name for

## Description

mac80211 creates a association LED trigger for each wireless hardware that can be used to drive LEDs if your driver registers a LED device. This function returns the name (or `NULL` if not configured for LEDs) of the trigger so you can automatically link the LED device.

# ieee80211\_get\_radio\_led\_name

## LINUX

Kernel Hackers Manual July 2011

## Name

`ieee80211_get_radio_led_name` — get name of radio LED

## Synopsis

```
char * ieee80211_get_radio_led_name (struct ieee80211_hw *  
hw);
```

## Arguments

*hw*

the hardware to get the LED trigger name for

## Description

mac80211 creates a radio change LED trigger for each wireless hardware that can be used to drive LEDs if your driver registers a LED device. This function returns the name (or `NULL` if not configured for LEDs) of the trigger so you can automatically link the LED device.

# ieee80211\_unregister\_hw

## LINUX

Kernel Hackers Manual July 2011

## Name

`ieee80211_unregister_hw` — Unregister a hardware device

## Synopsis

```
void ieee80211_unregister_hw (struct ieee80211_hw * hw);
```

## Arguments

*hw*

the hardware to unregister

## Description

This function instructs mac80211 to free allocated resources and unregister netdevices from the networking subsystem.

# ieee80211\_free\_hw

## LINUX

Kernel Hackers Manual July 2011

## Name

`ieee80211_free_hw` — free hardware descriptor

## Synopsis

```
void ieee80211_free_hw (struct ieee80211_hw * hw);
```

## Arguments

*hw*

the hardware to free

## **Description**

This function frees everything that was allocated, including the private data for the driver. You must call `ieee80211_unregister_hw` before calling this function.

# Chapter 2. PHY configuration

TBD

This chapter should describe PHY handling including start/stop callbacks and the various structures used.

## struct ieee80211\_conf

**LINUX**

Kernel Hackers Manual July 2011

### Name

struct ieee80211\_conf — configuration of the device

### Synopsis

```
struct ieee80211_conf {
    u32 flags;
    int power_level;
    int dynamic_ps_timeout;
    int max_sleep_period;
    u16 listen_interval;
    u8 ps_dtim_period;
    u8 long_frame_max_tx_count;
    u8 short_frame_max_tx_count;
    struct ieee80211_channel * channel;
    enum nl80211_channel_type channel_type;
    enum ieee80211_smmps_mode smmps_mode;
};
```

### Members

flags

configuration flags defined above

## Chapter 2. PHY configuration

power\_level

requested transmit power (in dBm)

dynamic\_ps\_timeout

The dynamic powersave timeout (in ms), see the powersave documentation below. This variable is valid only when the CONF\_PS flag is set.

max\_sleep\_period

the maximum number of beacon intervals to sleep for before checking the beacon for a TIM bit (managed mode only); this value will be only achievable between DTIM frames, the hardware needs to check for the multicast traffic bit in DTIM beacons. This variable is valid only when the CONF\_PS flag is set.

listen\_interval

listen interval in units of beacon interval

ps\_dtim\_period

The DTIM period of the AP we're connected to, for use in power saving. Power saving will not be enabled until a beacon has been received and the DTIM period is known.

long\_frame\_max\_tx\_count

Maximum number of transmissions for a "long" frame (a frame not RTS protected), called "dot11LongRetryLimit" in 802.11, but actually means the number of transmissions not the number of retries

short\_frame\_max\_tx\_count

Maximum number of transmissions for a "short" frame, called "dot11ShortRetryLimit" in 802.11, but actually means the number of transmissions not the number of retries

channel

the channel to tune to

channel\_type

the channel (HT) type

smmps\_mode

spatial multiplexing powersave mode; note that IEEE80211\_SMPS\_STATIC is used when the device is not configured for an HT channel

## Description

This struct indicates how the driver shall configure the hardware.

## enum ieee80211\_conf\_flags

### LINUX

Kernel Hackers Manual July 2011

## Name

enum ieee80211\_conf\_flags — configuration flags

## Synopsis

```
enum ieee80211_conf_flags {
    IEEE80211_CONF_MONITOR,
    IEEE80211_CONF_PS,
    IEEE80211_CONF_IDLE,
    IEEE80211_CONF_OFFCHANNEL
};
```

## Constants

### IEEE80211\_CONF\_MONITOR

there's a monitor interface present -- use this to determine for example whether to calculate timestamps for packets or not, do not use instead of filter flags!

### IEEE80211\_CONF\_PS

Enable 802.11 power save mode (managed mode only). This is the power save mode defined by IEEE 802.11-2007 section 11.2, meaning that the hardware still wakes up for beacons, is able to transmit frames and receive the possible acknowledgment frames. Not to be confused with hardware specific wakeup/sleep states, driver is responsible for that. See the section “Powersave support” for more.

#### IEEE80211\_CONF\_IDLE

The device is running, but idle; if the flag is set the driver should be prepared to handle configuration requests but may turn the device off as much as possible. Typically, this flag will be set when an interface is set UP but not associated or scanning, but it can also be unset in that case when monitor interfaces are active.

#### IEEE80211\_CONF\_OFFCHANNEL

The device is currently not on its main operating channel.

## **Description**

Flags to define PHY configuration options

# Chapter 3. Virtual interfaces

TBD

This chapter should describe virtual interface basics that are relevant to the driver (VLANs, MGMT etc are not.) It should explain the use of the `add_iface/remove_iface` callbacks as well as the interface configuration callbacks.

Things related to AP mode should be discussed there.

Things related to supporting multiple interfaces should be in the appropriate chapter, a BIG FAT note should be here about this though and the recommendation to allow only a single interface in STA mode at first!

## struct ieee80211\_vif

**LINUX**

Kernel Hackers Manual July 2011

### Name

`struct ieee80211_vif` — per-interface data

### Synopsis

```
struct ieee80211_vif {
    enum nl80211_iftype type;
    struct ieee80211_bss_conf bss_conf;
    u8 addr[ETH_ALEN];
    bool p2p;
    u8 drv_priv[0] __attribute__((__aligned__(sizeof(void *)))));
};
```

### Members

`type`

type of this virtual interface

bss\_conf

BSS configuration for this interface, either our own or the BSS we're associated to

addr[ETH\_ALEN]

address of this interface

p2p

indicates whether this AP or STA interface is a p2p interface, i.e. a GO or p2p-sta respectively

drv\_priv[0] \_\_attribute\_\_((\_\_aligned\_\_(sizeof(void \*))))

data area for driver use, will always be aligned to sizeof(void \*).

## **Description**

Data in this structure is continually present for driver use during the life of a virtual interface.

# Chapter 4. Receive and transmit processing

## 4.1. what should be here

TBD

This should describe the receive and transmit paths in mac80211/the drivers as well as transmit status handling.

## 4.2. Frame format

As a general rule, when frames are passed between mac80211 and the driver, they start with the IEEE 802.11 header and include the same octets that are sent over the air except for the FCS which should be calculated by the hardware.

There are, however, various exceptions to this rule for advanced features:

The first exception is for hardware encryption and decryption offload where the IV/ICV may or may not be generated in hardware.

Secondly, when the hardware handles fragmentation, the frame handed to the driver from mac80211 is the MSDU, not the MPDU.

Finally, for received frames, the driver is able to indicate that it has filled a radiotap header and put that in front of the frame; if it does not do so then mac80211 may add this under certain circumstances.

## 4.3. Packet alignment

Drivers always need to pass packets that are aligned to two-byte boundaries to the stack.

Additionally, should, if possible, align the payload data in a way that guarantees that the contained IP header is aligned to a four-byte boundary. In the case of regular frames, this simply means aligning the payload to a four-byte boundary (because either the IP header is directly contained, or IV/RFC1042 headers that have a length divisible by four are in front of it). If the payload data is not properly aligned and the architecture doesn't support efficient unaligned operations, mac80211 will align the data.

With A-MSDU frames, however, the payload data address must yield two modulo four because there are 14-byte 802.3 headers within the A-MSDU frames that push the IP header further back to a multiple of four again. Thankfully, the specs were sane enough this time around to require padding each A-MSDU subframe to a length that is a multiple of four.

Padding like Atheros hardware adds which is inbetween the 802.11 header and the payload is not supported, the driver is required to move the 802.11 header to be directly in front of the payload in that case.

## 4.4. Calling into mac80211 from interrupts

Only `ieee80211_tx_status_irqsafe` and `ieee80211_rx_irqsafe` can be called in hardware interrupt context. The low-level driver must not call any other functions in hardware interrupt context. If there is a need for such call, the low-level driver should first ACK the interrupt and perform the IEEE 802.11 code call after this, e.g. from a scheduled workqueue or even tasklet function.

NOTE: If the driver opts to use the `_irqsafe` functions, it may not also use the non-IRQ-safe functions!

## 4.5. functions/definitions

### struct ieee80211\_rx\_status

#### LINUX

Kernel Hackers Manual July 2011

#### Name

struct `ieee80211_rx_status` — receive status

#### Synopsis

```
struct ieee80211_rx_status {
    u64 mactime;
    enum ieee80211_band band;
```

```
int freq;  
int signal;  
int antenna;  
int rate_idx;  
int flag;  
unsigned int rx_flags;  
};
```

## Members

mactime

value in microseconds of the 64-bit Time Synchronization Function (TSF) timer when the first data symbol (MPDU) arrived at the hardware.

band

the active band when this frame was received

freq

frequency the radio was tuned to when receiving this frame, in MHz

signal

signal strength when receiving this frame, either in dBm, in dB or unspecified depending on the hardware capabilities flags `IEEE80211_HW_SIGNAL_*`

antenna

antenna used

rate\_idx

index of data rate into band's supported rates or MCS index if HT rates are use (RX\_FLAG\_HT)

flag

RX\_FLAG\_\*

rx\_flags

internal RX flags for mac80211

## Description

The low-level driver should provide this information (the subset supported by hardware) to the 802.11 code with each received frame, in the skb's control buffer (cb).

## enum mac80211\_rx\_flags

### LINUX

Kernel Hackers Manual July 2011

### Name

enum mac80211\_rx\_flags — receive flags

### Synopsis

```
enum mac80211_rx_flags {
    RX_FLAG_MMIC_ERROR,
    RX_FLAG_DECRYPTED,
    RX_FLAG_MMIC_STRIPPED,
    RX_FLAG_IV_STRIPPED,
    RX_FLAG_FAILED_FCS_CRC,
    RX_FLAG_FAILED_PLCP_CRC,
    RX_FLAG_TSFT,
    RX_FLAG_SHORTPRE,
    RX_FLAG_HT,
    RX_FLAG_40MHZ,
    RX_FLAG_SHORT_GI
};
```

### Constants

#### RX\_FLAG\_MMIC\_ERROR

Michael MIC error was reported on this frame. Use together with RX\_FLAG\_MMIC\_STRIPPED.

#### RX\_FLAG\_DECRYPTED

This frame was decrypted in hardware.

#### RX\_FLAG\_MMIC\_STRIPPED

the Michael MIC is stripped off this frame, verification has been done by the hardware.

#### RX\_FLAG\_IV\_STRIPPED

The IV/ICV are stripped from this frame. If this flag is set, the stack cannot do any replay detection hence the driver or hardware will have to do that.

#### RX\_FLAG\_FAILED\_FCS\_CRC

Set this flag if the FCS check failed on the frame.

#### RX\_FLAG\_FAILED\_PLCP\_CRC

Set this flag if the PLCP check failed on the frame.

#### RX\_FLAG\_TSFT

The timestamp passed in the RX status (*mactime* field) is valid. This is useful in monitor mode and necessary for beacon frames to enable IBSS merging.

#### RX\_FLAG\_SHORTPRE

Short preamble was used for this frame

#### RX\_FLAG\_HT

HT MCS was used and *rate\_idx* is MCS index

#### RX\_FLAG\_40MHZ

HT40 (40 MHz) was used

#### RX\_FLAG\_SHORT\_GI

Short guard interval was used

## Description

These flags are used with the *flag* member of struct `ieee80211_rx_status`.

# struct ieee80211\_tx\_info

## LINUX

Kernel Hackers Manual July 2011

### Name

struct ieee80211\_tx\_info — skb transmit information

### Synopsis

```
struct ieee80211_tx_info {  
    u32 flags;  
    u8 band;  
    u8 antenna_sel_tx;  
    u8 pad[2];  
    union {unnamed_union};  
};
```

### Members

flags

transmit info flags, defined above

band

the band to transmit on (use for checking for races)

antenna\_sel\_tx

antenna to use, 0 for automatic diversity

pad[2]

padding, ignore

{unnamed\_union}

anonymous

## Description

This structure is placed in `skb->cb` for three uses: (1) `mac80211` TX control - `mac80211` tells the driver what to do (2) driver internal use (if applicable) (3) TX status information - driver tells `mac80211` what happened

The TX control's sta pointer is only valid during the `->tx` call, it may be `NULL`.

## ieee80211\_rx

### LINUX

Kernel Hackers Manual July 2011

## Name

`ieee80211_rx` — receive frame

## Synopsis

```
void ieee80211_rx (struct ieee80211_hw * hw, struct sk_buff *  
skb);
```

## Arguments

*hw*

the hardware this frame came in on

*skb*

the buffer to receive, owned by `mac80211` after this call

## Description

Use this function to hand received frames to mac80211. The receive buffer in *skb* must start with an IEEE 802.11 header. In case of a paged *skb* is used, the driver is recommended to put the ieee80211 header of the frame on the linear part of the *skb* to avoid memory allocation and/or memcpy by the stack.

This function may not be called in IRQ context. Calls to this function for a single hardware must be synchronized against each other. Calls to this function, `ieee80211_rx_ni` and `ieee80211_rx_irqsafe` may not be mixed for a single hardware.

In process context use instead `ieee80211_rx_ni`.

## ieee80211\_rx\_irqsafe

### LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_rx_irqsafe` — receive frame

### Synopsis

```
void ieee80211_rx_irqsafe (struct ieee80211_hw * hw, struct
sk_buff * skb);
```

### Arguments

*hw*

the hardware this frame came in on

*skb*

the buffer to receive, owned by mac80211 after this call

## Description

Like `ieee80211_rx` but can be called in IRQ context (internally defers to a tasklet.)

Calls to this function, `ieee80211_rx` or `ieee80211_rx_ni` may not be mixed for a single hardware.

## ieee80211\_tx\_status

### LINUX

Kernel Hackers Manual July 2011

## Name

`ieee80211_tx_status` — transmit status callback

## Synopsis

```
void ieee80211_tx_status (struct ieee80211_hw * hw, struct  
sk_buff * skb);
```

## Arguments

*hw*

the hardware the frame was transmitted by

*skb*

the frame that was transmitted, owned by mac80211 after this call

## Description

Call this function for all transmitted frames after they have been transmitted. It is permissible to not call this function for multicast frames but this can affect statistics.

This function may not be called in IRQ context. Calls to this function for a single hardware must be synchronized against each other. Calls to this function, `ieee80211_tx_status_ni` and `ieee80211_tx_status_irqsafe` may not be mixed for a single hardware.

## ieee80211\_tx\_status\_irqsafe

### LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_tx_status_irqsafe` — IRQ-safe transmit status callback

### Synopsis

```
void ieee80211_tx_status_irqsafe (struct ieee80211_hw * hw,  
struct sk_buff * skb);
```

### Arguments

*hw*

the hardware the frame was transmitted by

*skb*

the frame that was transmitted, owned by mac80211 after this call

### Description

Like `ieee80211_tx_status` but can be called in IRQ context (internally defers to a tasklet.)

Calls to this function, `ieee80211_tx_status` and `ieee80211_tx_status_ni` may not be mixed for a single hardware.

## ieee80211\_rts\_get

### LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_rts_get` — RTS frame generation function

### Synopsis

```
void ieee80211_rts_get (struct ieee80211_hw * hw, struct
ieee80211_vif * vif, const void * frame, size_t frame_len,
const struct ieee80211_tx_info * frame_txctl, struct
ieee80211_rts * rts);
```

### Arguments

*hw*

pointer obtained from `ieee80211_alloc_hw`.

*vif*

struct `ieee80211_vif` pointer from the `add_interface` callback.

*frame*

pointer to the frame that is going to be protected by the RTS.

*frame\_len*

the frame length (in octets).

*frame\_txctl*

struct `ieee80211_tx_info` of the frame.

*rts*

The buffer where to store the RTS frame.

## Description

If the RTS frames are generated by the host system (i.e., not in hardware/firmware), the low-level driver uses this function to receive the next RTS frame from the 802.11 code. The low-level is responsible for calling this function before and RTS frame is needed.

# ieee80211\_rts\_duration

## LINUX

Kernel Hackers Manual July 2011

## Name

`ieee80211_rts_duration` — Get the duration field for an RTS frame

## Synopsis

```
__le16 ieee80211_rts_duration (struct ieee80211_hw * hw,  
struct ieee80211_vif * vif, size_t frame_len, const struct  
ieee80211_tx_info * frame_txctl);
```

## Arguments

*hw*

pointer obtained from `ieee80211_alloc_hw`.

*vif*

struct `ieee80211_vif` pointer from the `add_interface` callback.

*frame\_len*

the length of the frame that is going to be protected by the RTS.

*frame\_txctl*

struct `ieee80211_tx_info` of the frame.

## Description

If the RTS is generated in firmware, but the host system must provide the duration field, the low-level driver uses this function to receive the duration field value in little-endian byteorder.

# ieee80211\_ctstoself\_get

## LINUX

Kernel Hackers Manual July 2011

## Name

`ieee80211_ctstoself_get` — CTS-to-self frame generation function

## Synopsis

```
void ieee80211_ctstoself_get (struct ieee80211_hw * hw, struct  
ieee80211_vif * vif, const void * frame, size_t frame_len,
```

```
const struct ieee80211_tx_info * frame_txctl, struct
ieee80211_cts * cts);
```

## Arguments

*hw*

pointer obtained from `ieee80211_alloc_hw`.

*vif*

struct `ieee80211_vif` pointer from the `add_interface` callback.

*frame*

pointer to the frame that is going to be protected by the CTS-to-self.

*frame\_len*

the frame length (in octets).

*frame\_txctl*

struct `ieee80211_tx_info` of the frame.

*cts*

The buffer where to store the CTS-to-self frame.

## Description

If the CTS-to-self frames are generated by the host system (i.e., not in hardware/firmware), the low-level driver uses this function to receive the next CTS-to-self frame from the 802.11 code. The low-level is responsible for calling this function before and CTS-to-self frame is needed.

## ieee80211\_ctstoself\_duration

**LINUX**

## Name

`ieee80211_ctstoself_duration` — Get the duration field for a CTS-to-self frame

## Synopsis

```
__le16 ieee80211_ctstoself_duration (struct ieee80211_hw * hw,  
struct ieee80211_vif * vif, size_t frame_len, const struct  
ieee80211_tx_info * frame_txctl);
```

## Arguments

*hw*

pointer obtained from `ieee80211_alloc_hw`.

*vif*

struct `ieee80211_vif` pointer from the `add_interface` callback.

*frame\_len*

the length of the frame that is going to be protected by the CTS-to-self.

*frame\_txctl*

struct `ieee80211_tx_info` of the frame.

## Description

If the CTS-to-self is generated in firmware, but the host system must provide the duration field, the low-level driver uses this function to receive the duration field value in little-endian byteorder.

# ieee80211\_generic\_frame\_duration

## LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_generic_frame_duration` — Calculate the duration field for a frame

### Synopsis

```
__le16 ieee80211_generic_frame_duration (struct ieee80211_hw *  
hw, struct ieee80211_vif * vif, size_t frame_len, struct  
ieee80211_rate * rate);
```

### Arguments

*hw*

pointer obtained from `ieee80211_alloc_hw`.

*vif*

struct `ieee80211_vif` pointer from the `add_interface` callback.

*frame\_len*

the length of the frame.

*rate*

the rate at which the frame is going to be transmitted.

### Description

Calculate the duration field of some generic frame, given its length and transmission rate (in 100kbps).

# ieee80211\_wake\_queue

## LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_wake_queue` — wake specific queue

### Synopsis

```
void ieee80211_wake_queue (struct ieee80211_hw * hw, int  
queue);
```

### Arguments

*hw*

pointer as obtained from `ieee80211_alloc_hw`.

*queue*

queue number (counted from zero).

### Description

Drivers should use this function instead of `netif_wake_queue`.

## ieee80211\_stop\_queue

### LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_stop_queue` — stop specific queue

### Synopsis

```
void ieee80211_stop_queue (struct ieee80211_hw * hw, int  
queue);
```

### Arguments

*hw*

pointer as obtained from `ieee80211_alloc_hw`.

*queue*

queue number (counted from zero).

### Description

Drivers should use this function instead of `netif_stop_queue`.

## ieee80211\_wake\_queues

### LINUX

## Name

`ieee80211_wake_queues` — wake all queues

## Synopsis

```
void ieee80211_wake_queues (struct ieee80211_hw * hw);
```

## Arguments

*hw*

pointer as obtained from `ieee80211_alloc_hw`.

## Description

Drivers should use this function instead of `netif_wake_queue`.

# ieee80211\_stop\_queues

## LINUX

## Name

`ieee80211_stop_queues` — stop all queues

## Synopsis

```
void ieee80211_stop_queues (struct ieee80211_hw * hw);
```

## Arguments

*hw*

pointer as obtained from `ieee80211_alloc_hw`.

## Description

Drivers should use this function instead of `netif_stop_queue`.

# Chapter 5. Frame filtering

mac80211 requires to see many management frames for proper operation, and users may want to see many more frames when in monitor mode. However, for best CPU usage and power consumption, having as few frames as possible percolate through the stack is desirable. Hence, the hardware should filter as much as possible.

To achieve this, mac80211 uses filter flags (see below) to tell the driver's `configure_filter` function which frames should be passed to mac80211 and which should be filtered out.

Before `configure_filter` is invoked, the `prepare_multicast` callback is invoked with the parameters `mc_count` and `mc_list` for the combined multicast address list of all virtual interfaces. It's use is optional, and it returns a u64 that is passed to `configure_filter`. Additionally, `configure_filter` has the arguments `changed_flags` telling which flags were changed and `total_flags` with the new flag states.

If your device has no multicast address filters your driver will need to check both the `FIF_ALLMULTI` flag and the `mc_count` parameter to see whether multicast frames should be accepted or dropped.

All unsupported flags in `total_flags` must be cleared. Hardware does not support a flag if it is incapable of `_passing_` the frame to the stack. Otherwise the driver must ignore the flag, but not clear it. You must `_only_` clear the flag (announce no support for the flag to mac80211) if you are not able to pass the packet type to the stack (so the hardware always filters it). So for example, you should clear `FIF_CONTROL`, if your hardware always filters control frames. If your hardware always passes control frames to the kernel and is incapable of filtering them, you do `_not_` clear the `FIF_CONTROL` flag. This rule applies to all other FIF flags as well.

## enum ieee80211\_filter\_flags

### LINUX

Kernel Hackers Manual July 2011

### Name

enum ieee80211\_filter\_flags — hardware filter flags

## Synopsis

```
enum ieee80211_filter_flags {  
    FIF_PROMISC_IN_BSS,  
    FIF_ALLMULTI,  
    FIF_FCSFAIL,  
    FIF_PLCPFAIL,  
    FIF_BCN_PRBRESP_PROMISC,  
    FIF_CONTROL,  
    FIF_OTHER_BSS,  
    FIF_PSPOLL,  
    FIF_PROBE_REQ  
};
```

## Constants

### FIF\_PROMISC\_IN\_BSS

promiscuous mode within your BSS, think of the BSS as your network segment and then this corresponds to the regular ethernet device promiscuous mode.

### FIF\_ALLMULTI

pass all multicast frames, this is used if requested by the user or if the hardware is not capable of filtering by multicast address.

### FIF\_FCSFAIL

pass frames with failed FCS (but you need to set the RX\_FLAG\_FAILED\_FCS\_CRC for them)

### FIF\_PLCPFAIL

pass frames with failed PLCP CRC (but you need to set the RX\_FLAG\_FAILED\_PLCP\_CRC for them)

### FIF\_BCN\_PRBRESP\_PROMISC

This flag is set during scanning to indicate to the hardware that it should not filter beacons or probe responses by BSSID. Filtering them can greatly reduce the amount of processing mac80211 needs to do and the amount of CPU wakeups, so you should honour this flag if possible.

#### FIF\_CONTROL

pass control frames (except for PS Poll), if PROMISC\_IN\_BSS is not set then only those addressed to this station.

#### FIF\_OTHER\_BSS

pass frames destined to other BSSes

#### FIF\_PSPOLL

pass PS Poll frames, if PROMISC\_IN\_BSS is not set then only those addressed to this station.

#### FIF\_PROBE\_REQ

pass probe request frames

## **Frame filtering**

These flags determine what the filter in hardware should be programmed to let through and what should not be passed to the stack. It is always safe to pass more frames than requested, but this has negative impact on power consumption.



# II. Advanced driver interface

## Table of Contents

<b>6. Hardware crypto acceleration .....</b>	<b>55</b>
<b>7. Powersave support .....</b>	<b>61</b>
<b>8. Beacon filter support.....</b>	<b>63</b>
<b>9. Multiple queues and QoS support.....</b>	<b>65</b>
<b>10. Access point mode support.....</b>	<b>67</b>
<b>11. Supporting multiple virtual interfaces.....</b>	<b>71</b>
<b>12. Hardware scan offload.....</b>	<b>73</b>

Information contained within this part of the book is of interest only for advanced interaction of mac80211 with drivers to exploit more hardware capabilities and improve performance.



# Chapter 6. Hardware crypto acceleration

mac80211 is capable of taking advantage of many hardware acceleration designs for encryption and decryption operations.

The `set_key` callback in the struct `ieee80211_ops` for a given device is called to enable hardware acceleration of encryption and decryption. The callback takes a `sta` parameter that will be `NULL` for default keys or keys used for transmission only, or point to the station information for the peer for individual keys. Multiple transmission keys with the same key index may be used when VLANs are configured for an access point.

When transmitting, the TX control data will use the `hw_key_idx` selected by the driver by modifying the struct `ieee80211_key_conf` pointed to by the `key` parameter to the `set_key` function.

The `set_key` call for the `SET_KEY` command should return 0 if the key is now in use, `-EOPNOTSUPP` or `-ENOSPC` if it couldn't be added; if you return 0 then `hw_key_idx` must be assigned to the hardware key index, you are free to use the full u8 range.

When the cmd is `DISABLE_KEY` then it must succeed.

Note that it is permissible to not decrypt a frame even if a key for it has been uploaded to hardware, the stack will not make any decision based on whether a key has been uploaded or not but rather based on the receive flags.

The struct `ieee80211_key_conf` structure pointed to by the `key` parameter is guaranteed to be valid until another call to `set_key` removes it, but it can only be used as a cookie to differentiate keys.

In TKIP some HW need to be provided a phase 1 key, for RX decryption acceleration (i.e. iwlwifi). Those drivers should provide `update_tkip_key` handler. The `update_tkip_key` call updates the driver with the new phase 1 key. This happens everytime the iv16 wraps around (every 65536 packets). The `set_key` call will happen only once for each key (unless the AP did rekeying), it will not include a valid phase 1 key. The valid phase 1 key is provided by `update_tkip_key` only. The trigger that makes mac80211 call this handler is software decryption with wrap around of iv16.

## enum set\_key\_cmd

### LINUX

Kernel Hackers Manual July 2011

### Name

enum set\_key\_cmd — key command

### Synopsis

```
enum set_key_cmd {  
    SET_KEY,  
    DISABLE_KEY  
};
```

### Constants

SET\_KEY

a key is set

DISABLE\_KEY

a key must be disabled

### Description

Used with the `set_key` callback in struct `ieee80211_ops`, this indicates whether a key is being removed or added.

## struct ieee80211\_key\_conf

### LINUX

## Name

`struct ieee80211_key_conf` — key information

## Synopsis

```
struct ieee80211_key_conf {  
    u32 cipher;  
    u8 icv_len;  
    u8 iv_len;  
    u8 hw_key_idx;  
    u8 flags;  
    s8 keyidx;  
    u8 keylen;  
    u8 key[0];  
};
```

## Members

`cipher`

The key's cipher suite selector.

`icv_len`

The ICV length for this key type

`iv_len`

The IV length for this key type

`hw_key_idx`

To be set by the driver, this is the key index the driver wants to be given when a frame is transmitted and needs to be encrypted in hardware.

`flags`

key flags, see enum `ieee80211_key_flags`.

`keyidx`

the key index (0-3)

keylen

key material length

key[0]

key material. For ALG\_TKIP the key is encoded as a 256-bit (32 byte)

## Description

This key information is given by mac80211 to the driver by the `set_key` callback in struct `ieee80211_ops`.

## data block

- Temporal Encryption Key (128 bits) - Temporal Authenticator Tx MIC Key (64 bits) - Temporal Authenticator Rx MIC Key (64 bits)

## enum ieee80211\_key\_flags

### LINUX

Kernel Hackers Manual July 2011

## Name

enum `ieee80211_key_flags` — key flags

## Synopsis

```
enum ieee80211_key_flags {
    IEEE80211_KEY_FLAG_WMM_STA,
    IEEE80211_KEY_FLAG_GENERATE_IV,
    IEEE80211_KEY_FLAG_GENERATE_MMIC,
    IEEE80211_KEY_FLAG_PAIRWISE,
    IEEE80211_KEY_FLAG_SW_MGMT
};
```

## Constants

### IEEE80211\_KEY\_FLAG\_WMM\_STA

Set by `mac80211`, this flag indicates that the STA this key will be used with could be using QoS.

### IEEE80211\_KEY\_FLAG\_GENERATE\_IV

This flag should be set by the driver to indicate that it requires IV generation for this particular key.

### IEEE80211\_KEY\_FLAG\_GENERATE\_MMIC

This flag should be set by the driver for a TKIP key if it requires Michael MIC generation in software.

### IEEE80211\_KEY\_FLAG\_PAIRWISE

Set by `mac80211`, this flag indicates that the key is pairwise rather than a shared key.

### IEEE80211\_KEY\_FLAG\_SW\_MGMT

This flag should be set by the driver for a CCMP key if it requires CCMP encryption of management frames (MFP) to be done in software.

## Description

These flags are used for communication about keys between the driver and `mac80211`, with the `flags` parameter of `struct ieee80211_key_conf`.



# Chapter 7. Powersave support

mac80211 has support for various powersave implementations.

First, it can support hardware that handles all powersaving by itself, such hardware should simply set the `IEEE80211_HW_SUPPORTS_PS` hardware flag. In that case, it will be told about the desired powersave mode with the `IEEE80211_CONF_PS` flag depending on the association status. The hardware must take care of sending nullfunc frames when necessary, i.e. when entering and leaving powersave mode. The hardware is required to look at the AID in beacons and signal to the AP that it woke up when it finds traffic directed to it.

`IEEE80211_CONF_PS` flag enabled means that the powersave mode defined in IEEE 802.11-2007 section 11.2 is enabled. This is not to be confused with hardware wakeup and sleep states. Driver is responsible for waking up the hardware before issuing commands to the hardware and putting it back to sleep at appropriate times.

When PS is enabled, hardware needs to wakeup for beacons and receive the buffered multicast/broadcast frames after the beacon. Also it must be possible to send frames and receive the acknowledgment frame.

Other hardware designs cannot send nullfunc frames by themselves and also need software support for parsing the TIM bitmap. This is also supported by mac80211 by combining the `IEEE80211_HW_SUPPORTS_PS` and `IEEE80211_HW_PS_NULLFUNC_STACK` flags. The hardware is of course still required to pass up beacons. The hardware is still required to handle waking up for multicast traffic; if it cannot the driver must handle that as best as it can, mac80211 is too slow to do that.

Dynamic powersave is an extension to normal powersave in which the hardware stays awake for a user-specified period of time after sending a frame so that reply frames need not be buffered and therefore delayed to the next wakeup. It's compromise of getting good enough latency when there's data traffic and still saving significantly power in idle periods.

Dynamic powersave is simply supported by mac80211 enabling and disabling PS based on traffic. Driver needs to only set `IEEE80211_HW_SUPPORTS_PS` flag and mac80211 will handle everything automatically. Additionally, hardware having support for the dynamic PS feature may set the

`IEEE80211_HW_SUPPORTS_DYNAMIC_PS` flag to indicate that it can support dynamic PS mode itself. The driver needs to look at the `dynamic_ps_timeout` hardware configuration value and use it that value whenever `IEEE80211_CONF_PS` is set. In this case mac80211 will disable dynamic PS feature in stack and will just keep `IEEE80211_CONF_PS` enabled whenever user has enabled powersave.

Some hardware need to toggle a single shared antenna between WLAN and

Bluetooth to facilitate co-existence. These types of hardware set limitations on the use of host controlled dynamic powersave whenever there is simultaneous WLAN and Bluetooth traffic. For these types of hardware, the driver may request temporarily going into full power save, in order to enable toggling the antenna between BT and WLAN. If the driver requests disabling dynamic powersave, the `dynamic_ps_timeout` value will be temporarily set to zero until the driver re-enables dynamic powersave.

Driver informs U-APSD client support by enabling `IEEE80211_HW_SUPPORTS_UAPSD` flag. The mode is configured through the `uapsd` parameter in `conf_tx` operation. Hardware needs to send the QoS Nullfunc frames and stay awake until the service period has ended. To utilize U-APSD, dynamic powersave is disabled for voip AC and all frames from that AC are transmitted with powersave enabled.

Note: U-APSD client mode is not yet supported with

`IEEE80211_HW_PS_NULLFUNC_STACK`.

# Chapter 8. Beacon filter support

Some hardware have beacon filter support to reduce host cpu wakeups which will reduce system power consumption. It usually works so that the firmware creates a checksum of the beacon but omits all constantly changing elements (TSF, TIM etc). Whenever the checksum changes the beacon is forwarded to the host, otherwise it will be just dropped. That way the host will only receive beacons where some relevant information (for example ERP protection or WMM settings) have changed.

Beacon filter support is advertised with the `IEEE80211_HW_BEACON_FILTER` hardware capability. The driver needs to enable beacon filter support whenever power save is enabled, that is `IEEE80211_CONF_PS` is set. When power save is enabled, the stack will not check for beacon loss and the driver needs to notify about loss of beacons with `ieee80211_beacon_loss`.

The time (or number of beacons missed) until the firmware notifies the driver of a beacon loss event (which in turn causes the driver to call `ieee80211_beacon_loss`) should be configurable and will be controlled by `mac80211` and the roaming algorithm in the future.

Since there may be constantly changing information elements that nothing in the software stack cares about, we will, in the future, have `mac80211` tell the driver which information elements are interesting in the sense that we want to see changes in them. This will include - a list of information element IDs - a list of OUIs for the vendor information element

Ideally, the hardware would filter out any beacons without changes in the requested elements, but if it cannot support that it may, at the expense of some efficiency, filter out only a subset. For example, if the device doesn't support checking for OUIs it should pass up all changes in all vendor information elements.

Note that change, for the sake of simplification, also includes information elements appearing or disappearing from the beacon.

Some hardware supports an "ignore list" instead, just make sure nothing that was requested is on the ignore list, and include commonly changing information element IDs in the ignore list, for example 11 (BSS load) and the various vendor-assigned IEs with unknown contents (128, 129, 133-136, 149, 150, 155, 156, 173, 176, 178, 179, 219); for forward compatibility it could also include some currently unused IDs.

In addition to these capabilities, hardware should support notifying the host of changes in the beacon RSSI. This is relevant to implement roaming when no traffic is flowing (when traffic is flowing we see the RSSI of the received data packets). This can consist in notifying the host when the RSSI changes significantly or when it drops below or rises above configurable thresholds. In the future these thresholds

will also be configured by mac80211 (which gets them from userspace) to implement them as the roaming algorithm requires.

If the hardware cannot implement this, the driver should ask it to periodically pass beacon frames to the host so that software can do the signal strength threshold checking.

## ieee80211\_beacon\_loss

### LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_beacon_loss` — inform hardware does not receive beacons

### Synopsis

```
void ieee80211_beacon_loss (struct ieee80211_vif * vif);
```

### Arguments

*vif*

struct `ieee80211_vif` pointer from the `add_interface` callback.

### Description

When beacon filtering is enabled with `IEEE80211_HW_BEACON_FILTER` and `IEEE80211_CONF_PS` is set, the driver needs to inform whenever the hardware is not receiving beacons with this function.

# Chapter 9. Multiple queues and QoS support

TBD

## struct ieee80211\_tx\_queue\_params

**LINUX**

Kernel Hackers Manual July 2011

### Name

struct ieee80211\_tx\_queue\_params — transmit queue configuration

### Synopsis

```
struct ieee80211_tx_queue_params {
    u16 txop;
    u16 cw_min;
    u16 cw_max;
    u8 aifs;
    bool uapsd;
};
```

### Members

txop

maximum burst time in units of 32 usecs, 0 meaning disabled

cw\_min

minimum contention window [a value of the form  $2^n - 1$  in the range 1..32767]

cw\_max

maximum contention window [like *cw\_min*]

aifs

arbitration interframe space [0..255]

uapsd

is U-APSD mode enabled for the queue

## **Description**

The information provided in this structure is required for QoS transmit queue configuration. Cf. IEEE 802.11 7.3.2.29.

# Chapter 10. Access point mode support

TBD

Some parts of the `if_conf` should be discussed here instead

Insert notes about VLAN interfaces with hw crypto here or in the hw crypto chapter.

## ieee80211\_get\_buffered\_bc

### LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_get_buffered_bc` — accessing buffered broadcast and multicast frames

### Synopsis

```
struct sk_buff * ieee80211_get_buffered_bc (struct
ieee80211_hw * hw, struct ieee80211_vif * vif);
```

### Arguments

*hw*

pointer as obtained from `ieee80211_alloc_hw`.

*vif*

struct `ieee80211_vif` pointer from the `add_interface` callback.

## Description

Function for accessing buffered broadcast and multicast frames. If hardware/firmware does not implement buffering of broadcast/multicast frames when power saving is used, 802.11 code buffers them in the host memory. The low-level driver uses this function to fetch next buffered frame. In most cases, this is used when generating beacon frame. This function returns a pointer to the next buffered skb or NULL if no more buffered frames are available.

## Note

buffered frames are returned only after DTIM beacon frame was generated with `ieee80211_beacon_get` and the low-level driver must thus call `ieee80211_beacon_get` first. `ieee80211_get_buffered_bc` returns NULL if the previous generated beacon was not DTIM, so the low-level driver does not need to check for DTIM beacons separately and should be able to use common code for all beacons.

## ieee80211\_beacon\_get

### LINUX

Kernel Hackers Manual July 2011

## Name

`ieee80211_beacon_get` — beacon generation function

## Synopsis

```
struct sk_buff * ieee80211_beacon_get (struct ieee80211_hw *  
hw, struct ieee80211_vif * vif);
```

## Arguments

*hw*

pointer obtained from `ieee80211_alloc_hw`.

*vif*

struct `ieee80211_vif` pointer from the `add_interface` callback.

## Description

See `ieee80211_beacon_get_tim`.



# Chapter 11. Supporting multiple virtual interfaces

TBD

Note: WDS with identical MAC address should almost always be OK

Insert notes about having multiple virtual interfaces with different MAC addresses here, note which configurations are supported by mac80211, add notes about supporting hw crypto with it.



# Chapter 12. Hardware scan offload

TBD

## ieee80211\_scan\_completed

### LINUX

Kernel Hackers Manual July 2011

### Name

`ieee80211_scan_completed` — completed hardware scan

### Synopsis

```
void ieee80211_scan_completed (struct ieee80211_hw * hw, bool  
aborted);
```

### Arguments

*hw*

the hardware that finished the scan

*aborted*

set to true if scan was aborted

### Description

When hardware scan offload is used (i.e. the `hw_scan` callback is assigned) this function needs to be called by the driver to notify mac80211 that the scan finished. This function can be called from any context, including hardirq context.



# III. Rate control interface

## Table of Contents

13. dummy chapter .....	77
-------------------------	----

TBD

This part of the book describes the rate control algorithm interface and how it relates to mac80211 and drivers.



# Chapter 13. dummy chapter

TBD



# IV. Internals

## Table of Contents

<b>14. Key handling.....</b>	<b>81</b>
<b>15. Receive processing.....</b>	<b>83</b>
<b>16. Transmit processing .....</b>	<b>85</b>
<b>17. Station info handling.....</b>	<b>87</b>
<b>18. Synchronisation.....</b>	<b>97</b>

TBD

This part of the book describes mac80211 internals.



# Chapter 14. Key handling

## 14.1. Key handling basics

Key handling in mac80211 is done based on per-interface (`sub_if_data`) keys and per-station keys. Since each station belongs to an interface, each station key also belongs to that interface.

Hardware acceleration is done on a best-effort basis, for each key that is eligible the hardware is asked to enable that key but if it cannot do that the key is simply kept for software encryption. There is currently no way of knowing this except by looking into debugfs.

All key operations are protected internally.

Within mac80211, key references are, just as STA structure references, protected by RCU. Note, however, that some things are unprotected, namely the `key->sta` dereferences within the hardware acceleration functions. This means that `sta_info_destroy` must remove the key which waits for an RCU grace period.

## 14.2. MORE TBD

TBD



# Chapter 15. Receive processing

TBD



# Chapter 16. Transmit processing

TBD



# Chapter 17. Station info handling

## 17.1. Programming information

### struct sta\_info

#### LINUX

Kernel Hackers Manual July 2011

#### Name

struct sta\_info — STA information

#### Synopsis

```
struct sta_info {
    struct list_head list;
    struct sta_info * hnext;
    struct ieee80211_local * local;
    struct ieee80211_sub_if_data * sdata;
    struct ieee80211_key * gtk[NUM_DEFAULT_KEYS + NUM_DEFAULT_MGMT_KEYS];
    struct ieee80211_key * ptk;
    struct rate_control_ref * rate_ctrl;
    void * rate_ctrl_priv;
    spinlock_t lock;
    spinlock_t flaglock;
    struct work_struct drv_unblock_wk;
    u16 listen_interval;
    bool dead;
    bool uploaded;
    u32 flags;
    struct sk_buff_head ps_tx_buf;
    struct sk_buff_head tx_filtered;
    unsigned long rx_packets;
    unsigned long rx_bytes;
    unsigned long wep_weak_iv_count;
    unsigned long last_rx;
    unsigned long num_duplicates;
    unsigned long rx_fragments;
```

```
    unsigned long rx_dropped;
    int last_signal;
    __le16 last_seq_ctrl[NUM_RX_DATA_QUEUES];
    unsigned long tx_filtered_count;
    unsigned long tx_retry_failed;
    unsigned long tx_retry_count;
    unsigned int fail_avg;
    unsigned long tx_packets;
    unsigned long tx_bytes;
    unsigned long tx_fragments;
    struct ieee80211_tx_rate last_tx_rate;
    u16 tid_seq[IEEE80211_QOS_CTL_TID_MASK + 1];
    struct sta_ampdu_mlme ampdu_mlme;
    u8 timer_to_tid[STA_TID_NUM];
#ifdef CONFIG_MAC80211_MESH
    __le16 llid;
    __le16 plid;
    __le16 reason;
    u8 plink_retries;
    bool ignore_plink_timer;
    bool plink_timer_was_running;
    enum plink_state plink_state;
    u32 plink_timeout;
    struct timer_list plink_timer;
#endif
#ifdef CONFIG_MAC80211_DEBUGFS
    struct sta_info_debugfsdentries debugfs;
#endif
    struct ieee80211_sta sta;
};
```

## Members

list

global linked list entry

hnext

hash table linked list pointer

local

pointer to the global information

sdata

virtual interface this station belongs to

gtk[NUM\_DEFAULT\_KEYS + NUM\_DEFAULT\_MGMT\_KEYS]

group keys negotiated with this station, if any

ptk

peer key negotiated with this station, if any

rate\_ctrl

rate control algorithm reference

rate\_ctrl\_priv

rate control private per-STA pointer

lock

used for locking all fields that require locking, see comments in the header file.

flaglock

spinlock for flags accesses

drv\_unblock\_wk

used for driver PS unblocking

listen\_interval

listen interval of this station, when we're acting as AP

dead

set to true when sta is unlinked

uploaded

set to true when sta is uploaded to the driver

flags

STA flags, see enum `ieee80211_sta_info_flags`

ps\_tx\_buf

buffer of frames to transmit to this station when it leaves power saving state

## *Chapter 17. Station info handling*

`tx_filtered`

buffer of frames we already tried to transmit but were filtered by hardware due to STA having entered power saving state

`rx_packets`

Number of MSDUs received from this STA

`rx_bytes`

Number of bytes received from this STA

`wep_weak_iv_count`

number of weak WEP IVs received from this station

`last_rx`

time (in jiffies) when last frame was received from this STA

`num_duplicates`

number of duplicate frames received from this STA

`rx_fragments`

number of received MPDUs

`rx_dropped`

number of dropped MPDUs from this STA

`last_signal`

signal of last received frame from this STA

`last_seq_ctrl[NUM_RX_DATA_QUEUES]`

last received seq/frag number from this STA (per RX queue)

`tx_filtered_count`

number of frames the hardware filtered for this STA

`tx_retry_failed`

number of frames that failed retry

`tx_retry_count`

total number of retries for frames to this STA

`fail_avg`  
moving percentage of failed MSDUs

`tx_packets`  
number of RX/TX MSDUs

`tx_bytes`  
number of bytes transmitted to this STA

`tx_fragments`  
number of transmitted MPDUs

`last_tx_rate`  
rate used for last transmit, to report to userspace as “the” transmit rate

`tid_seq[IEEE80211_QOS_CTL_TID_MASK + 1]`  
per-TID sequence numbers for sending to this STA

`ampdu_mlme`  
A-MPDU state machine state

`timer_to_tid[STA_TID_NUM]`  
identity mapping to ID timers

`llid`  
Local link ID

`plid`  
Peer link ID

`reason`  
Cancel reason on PLINK\_HOLDING state

`plink_retries`  
Retries in establishment

`ignore_plink_timer`  
ignore the peer-link timer (used internally)

`plink_timer_was_running`  
used by suspend/resume to restore timers

plink\_state

peer link state

plink\_timeout

timeout of peer link

plink\_timer

peer link watch timer

debugfs

debug filesystem info

sta

station information we share with the driver

## Description

This structure collects information about a station that mac80211 is communicating with.

## enum ieee80211\_sta\_info\_flags

### LINUX

Kernel Hackers Manual July 2011

## Name

enum ieee80211\_sta\_info\_flags — Stations flags

## Synopsis

```
enum ieee80211_sta_info_flags {  
    WLAN_STA_AUTH,  
    WLAN_STA_ASSOC,  
    WLAN_STA_PS_STA,
```

```
WLAN_STA_AUTHORIZED,  
WLAN_STA_SHORT_PREAMBLE,  
WLAN_STA_ASSOC_AP,  
WLAN_STA_WME,  
WLAN_STA_WDS,  
WLAN_STA_CLEAR_PS_FILT,  
WLAN_STA_MFP,  
WLAN_STA_BLOCK_BA,  
WLAN_STA_PS_DRIVER,  
WLAN_STA_PSPOLL  
};
```

## Constants

WLAN\_STA\_AUTH

Station is authenticated.

WLAN\_STA\_ASSOC

Station is associated.

WLAN\_STA\_PS\_STA

Station is in power-save mode

WLAN\_STA\_AUTHORIZED

Station is authorized to send/receive traffic. This bit is always checked so needs to be enabled for all stations when virtual port control is not in use.

WLAN\_STA\_SHORT\_PREAMBLE

Station is capable of receiving short-preamble frames.

WLAN\_STA\_ASSOC\_AP

We're associated to that station, it is an AP.

WLAN\_STA\_WME

Station is a QoS-STA.

WLAN\_STA\_WDS

Station is one of our WDS peers.

#### WLAN\_STA\_CLEAR\_PS\_FILT

Clear PS filter in hardware (using the IEEE80211\_TX\_CTL\_CLEAR\_PS\_FILT control flag) when the next frame to this station is transmitted.

#### WLAN\_STA\_MFP

Management frame protection is used with this STA.

#### WLAN\_STA\_BLOCK\_BA

Used to deny ADDBA requests (both TX and RX) during suspend/resume and station removal.

#### WLAN\_STA\_PS\_DRIVER

driver requires keeping this station in power-save mode logically to flush frames that might still be in the queues

#### WLAN\_STA\_PSPOLL

Station sent PS-poll while driver was keeping station in power-save mode, reply when the driver unblocks.

## Description

These flags are used with struct `sta_info`'s *flags* member.

## 17.2. STA information lifetime rules

STA info structures (struct `sta_info`) are managed in a hash table for faster lookup and a list for iteration. They are managed using RCU, i.e. access to the list and hash table is protected by RCU.

Upon allocating a STA info structure with `sta_info_alloc`, the caller owns that structure. It must then insert it into the hash table using either `sta_info_insert` or `sta_info_insert_rcu`; only in the latter case (which acquires an rcu read section but must not be called from within one) will the pointer still be valid after the call. Note that the caller may not do much with the STA info before inserting it, in particular, it may not start any mesh peer link management or add encryption keys.

When the insertion fails (`sta_info_insert`) returns non-zero), the structure will have been freed by `sta_info_insert`!

Station entries are added by `mac80211` when you establish a link with a peer. This means different things for the different type of interfaces we support. For a regular station this mean we add the AP sta when we receive an association response from the AP. For IBSS this occurs when get to know about a peer on the same IBSS. For WDS we add the sta for the peer immediately upon device open. When using AP mode we add stations for each respective station upon request from userspace through `nl80211`.

In order to remove a STA info structure, various `sta_info_destroy_*`() calls are available.

There is no concept of ownership on a STA entry, each structure is owned by the global hash table/list until it is removed. All users of the structure need to be RCU protected so that the structure won't be freed before they are done using it.



# Chapter 18. Synchronisation

TBD

Locking, lots of RCU

