RALINK TECHNOLOGY, CORP.

RALINK RT2800 USB Wireless Card

SOFTWARE DRIVER RELEASE NOTE

Contents

1 HISTORY

[V1.2.1.0]

1. Fixed rate issue on Adhoc mode.
2. Fixed other Adhoc STA re-join failed with security.
3. Add "iwconfig rate" setting for legacy rate.
4. ATE: Add command "iwpriv ra0 set ATELDE2P=1" to overwrite all EEPROM contents from a .bin file.
5. Fixed issue of showing SNR1 information.
6. Add make install/uninstall to Makfile.

[V1.2.0.0]

1. Add Legacy Power Saving Mode.
2. Add BaSmartHardTransmit mechanism.
3. Modify rate adaptation for fast ramp-up tuning.
4. Support custom wireless event.
5. Support Ethernet Convert Mechanism.
6. Support Linux Kernel 2.6 suspend and resume.
7. Fixed W52 with Activity scan issue in ABG_MIXED and ABGN_MIXED mode.
8. Fixed fixed rate issue in N mode and fixed rate ioctl.

2 README

- **ModelName:**

  RT2870 Wireless Lan Linux Driver

- **Driver lName:**

  Kernel 2.4.x:

  rt2870sta.o

  Kernel 2.6.x:

  rt2870sta.ko

- **Supporting Kernel:**

  linux kernel 2.4 and 2.6 series.

  Tested in Redhat 7.3 or later.

- **Description:**

  This is a linux device driver for Ralink RT2870 ABGN WLAN Card.

- **Contents:**

  Makefile: Makefile

  *.c: c files

  *.h: header files

- **Features:**

  This driver implements basic IEEE802.11.

  Infrastructure and Ad-Hoc mode with open or shared or WPA-PSK or WPA2-PSK authentication method.

  NONE, WEP, TKIP and AES encryption.

- **Build Instructions:**

  1> $tar -xvzf yyyy_mmdd_RT2870_Linux_STA_x.x.x.x.tgz
  go to "./yyyy_mmdd_RT2870_Linux_STA_x.x.x.x" directory.

  2> In Makefile
  - set the "MODE = STA" in Makefile
  - choose the TARGET to Linux by set "TARGET = LINUX"
  - define the linux kernel source include file path LINUX_SRC modify to meet your need.

  3> In os/linux/config.mk
  define the GCC and LD of the target machine.

define the compiler flags CFLAGS.
modify to meet your need.
** Build for being controlled by NetworkManager
Please set 'HAS_WPA_SUPPLICANT=y' and 'HAS_NATIVE_WPA_SUPPLICANT_SUPPORT=y'.
** Build for being controlled by WpaSupplicant with Ralink Driver
Please set 'HAS_WPA_SUPPLICANT=y' and 'HAS_NATIVE_WPA_SUPPLICANT_SUPPORT=n'.

4> compile driver source code
$make

5> $cp RT2870STA.dat /etc/Wireless/RT2870STA/RT2870STA.dat
!!!check **if it is a binary file before loading !!!**

6> load driver
#[kernel 2.4]
$/sbin/insmod rt2870sta.o
$/sbin/ifconfig ra0 inet YOUR_IP up

#[kernel 2.6]
$/sbin/insmod rt2870sta.ko
$/sbin/ifconfig ra0 inet YOUR_IP up

7> unload driver
$/sbin/ifconfig ra0 down
$/sbin/rmmod rt2870sta

3 CONFIGURATION:

RT2870 driver can be configured via following interfaces, i.e.

1. configuration file
2. "iwconfig" command
3. "iwpriv" command

Note:

1) modify configuration file "RT2870STA.dat" in /etc/Wireless/RT2870STA/RT2870STA.dat.
2) iwconfig/iwpriv comes with kernel.
3) iwpriv usage, please refer to below sections for details.

3.1 Configuration File : RT2870STA.dat

```
# Copy this file to /etc/Wireless/RT2870STA/RT2870STA.dat
# This file is a binary file and will be read on loading rt.o module.
#
# Use "vi -b RT2870STA.dat" to modify settings according to your need.
#
# 1.) set NetworkType to
#          "Adhoc" for using Adhoc-mode,
#           otherwise using Infrastructure
# 2.) set Channel to
#          "0" for auto-select on Infrastructure mode
# 3.) set SSID for connecting to your Accss-point.
# 4.) AuthMode can be
#          "WEPAUTO",
#           "OPEN",
#          "SHARED",
#          "WPAPSK",
#          "WPA2PSK",
#          "WPANONE"
# 5.) EncrypType can be
#          "NONE",
#          "WEP",
#          "TKIP",
#          "AES"
# for more information refer to the Readme file.
#
#The word of "Default" must not be removed
Default
CountryRegion=5
CountryRegionABand=7
CountryCode=
SSID=Dennis2870AP
NetworkType=Infra
WirelessMode=9
Channel=0
BasicRate=15
BeaconPeriod=100
TxPower=100
BGProtection=0
TxPreamble=0
RTSThreshold=2347
FragThreshold=2346
TxBurst=1
PktAggregate=0
WmmCapable=0
AckPolicy=0;0;0;0
AuthMode=OPEN
EncrypType=NONE
WPAPSK=
DefaultKeyID=1
Key1Type=0
```

```
Key1Str=
Key2Type=0
Key2Str=
Key3Type=0
Key3Str=
Key4Type=0
Key4Str=
PSMode=CAM
FastRoaming=0
RoamThreshold=70
HT_RDG=1
HT_EXTCHA=0
HT_OpMode=1
HT_MpduDensity=4
HT_BW=1
HT_AutoBA=1
HT_AMSDU=0
HT_BAWinSize=64
HT_GI=1
HT_MCS=33
HT_MIMOPSMode=3
IEEE80211H=0
TGnWifiTest=0
WirelessEvent=0
```

NOTE:

WMM parameters

WmmCapable	Set it as 1 to turn on WMM Qos support
AckPolicy1~4	Ack policy which support normal Ack or no Ack (AC_BK, AC_BE, AC_VI, AC_VO)

All WMM parameters do not support iwpriv command but 'WmmCapable'', please store all parameter to RT2870STA.dat, and restart driver.

3.2 Usage

Syntax is 'Param'='Value' and describes below.

SectionNumber **Param**
Value

. . .
. . .
. . .

3.2.1 CountryRegion

value

0: use 1 ~ 11 Channel
1: use 1 ~ 13 Channel
2: use 10 ~ 11 Channel
3: use 10 ~ 13 Channel
4: use 14 Channel
5: use 1 ~ 14 Channel
6: use 3 ~ 9 Channel
7: use 5 ~ 13 Channel

3.2.2 CountryRegionForABand

value

0: use 36, 40, 44, 48, 52, 56, 60, 64, 149, 153, 157, 161, 165 Channel
1: use 36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116,
20, 124, 128, 132, 136, 140 Channel
2: use 36, 40, 44, 48, 52, 56, 60, 64 Channel
3: use 52, 56, 60, 64, 149, 153, 157, 161 Channel
4: use 149, 153, 157, 161, 165 Channel
5: use 149, 153, 157, 161 Channel
6: use 36, 40, 44, 48 Channel
7: use 36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116,
120, 124, 128, 132, 136, 140, 149, 153, 157, 161, 165 Channel
8: use 52, 56, 60, 64 Channel
9: use 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140 Channel

3.2.3 SSID

value

0~z, 1~32 ascii characters.

3.2.4 WirelessMode

value

0: legacy 11b/g mixed
1: legacy 11B only
2: legacy 11A only // Not support in RfIcType=1(id=RFIC_5225)
// and RfIcType=2(id=RFIC_5325)
3: legacy 11a/b/g mixed // Not support in RfIcType=1(id=RFIC_5225)
// and RfIcType=2(id=RFIC_5325)
4: legacy 11G only
5: 11ABGN mixed
6: 11N only
7: 11GN mixed
8: 11AN mixed

9: 11BGN mixed
10: 11AGN mixed

3.2.5 Channel

value

depends on CountryRegion or CountryRegionForABand

3.2.6 BGProtection

value

0: Auto
1: Always on
2: Always off

3.2.7 TxPreamble

value

0:Preamble Long
1:Preamble Short
2:Auto

3.2.8 RTSThreshold

value

1~2347

3.2.9 FragThreshold

value

256~2346

3.2.10 TxBurst

value

0: Disable
1: Enable

3.2.11 PktAggregate

value

0: Disable
1: Enable

3.2.12 NetworkType

value

Infra: infrastructure mode
Adhoc: adhoc mode

3.2.13 AuthMode

value

OPEN	For open system
SHARED	For shared key system
WEPAUTO	Auto switch between OPEN and SHARED
WPAPSK	For WPA pre-shared key (Infra)
WPA2PSK	For WPA2 pre-shared key (Infra)
WPANONE	For WPA pre-shared key (Adhoc)
WPA	
WPA2	

3.2.14 EncrypType

value

NONE	For AuthMode=OPEN
WEP	For AuthMode=OPEN or AuthMode=SHARED
TKIP	For AuthMode=WPAPSK or WPA2PSK
AES	For AuthMode=WPAPSK or WPA2PSK

3.2.15 DefaultKeyID

value

1~4

3.2.16 WEP KeyType

Key1Type=vaule
Key2Type=value
Key3Type=vaule
Key4Type=vaule

value

0 hexadecimal type
1 assic type
(usage : reading profile only)

3.2.17 WEP Hex Key

Key1=value
Key2=value
Key3=value
Key4=value

value

10 or 26 hexadecimal characters eg: 012345678
5 or 13 ascii characters eg: passd
(usage : "iwpriv" only)

3.2.18 WEP Key String

Key1Str=value
Key2Str=value
Key3Str=vaule
Key4Str=vaule

value

10 or 26 characters (key type=0)
5 or 13 characters (key type=1)
(usage : reading profile only)

3.2.19 WPAPSK

value

8~63 ASCII or
64 HEX characters

3.2.20 WmmCapable

value

0: Disable WMM
1: Enable WMM

3.2.21 IEEE80211H

Enabel IEEE802.11h support

Value:

0:Disable

1:Enable

3.2.22 PSMode

value

CAM	Constantly Awake Mode
Max_PSP	Max Power Savings
Fast_PSP	Power Save Mode

3.2.23 FastRoaming

value

0: Disabled

1: Enabled

3.2.24 RoamThreshold

value

0 ~ 255

3.2.25 TGnWifiTest

value

0: Disabled

1: Enabled

3.2.26 WirelessEvent

value

0: Disabled

1: Enabled (send custom wireless event)

3.2.27 HT_RDG

value

0: Disabled

1: Enabled

3.2.28 HT_EXTCHA

value

0: Below

1: Above

3.2.29 HT_OpMode

value

0: HT mixed format

1: HT greenfield format

3.2.30 HT_MpduDensity

value

0 ~ 7

3.2.31 HT_BW

value

0: 20MHz

1: 40MHz

3.2.32 HT_AutoBA

value

0: Disabled
1: Enabled

3.2.33 HT_AMSDU

value

0: Disabled
1: Enabled

3.2.34 HT_BAWinSize

value

1 ~ 64

3.2.35 HT_GI

value

0: long GI
1: short GI

3.2.36 HT_MCS

value

0 ~ 15
33: auto

3.2.37 HT_MIMOPSMode

value

0: Static SM Power Save Mode
2: Reserved
1: Dynamic SM Power Save Mode
3: SM enabled
(not fully support yet)

3.3 MORE INFORMATION

If you want for rt2870 driver to auto-load at boot time:

A) choose ra0 for first RT2870 WLAN card, ra1 for second RT2870 WLAN card, etc.

B) create(edit) 'ifcfg-ra0' file in /etc/sysconfig/network-scripts/,
edit(or add the line) in /etc/modules.conf:
alias ra0 rt2870sta

C) edit(create) the file /etc/sysconfig/network-scripts/ifcfg-ra0
DEVICE='ra0'
ONBOOT='yes'
NOTE:
if you use dhcp, add this line too.
BOOTPROTO='dhcp'

D) To ease the Default Gateway setting,
add the line
GATEWAY=x.x.x.x
in /etc/sysconfig/network

4 Wireless Tools

4.1 Iwpriv Usage

This is detailed explanation of each parameters for iwpriv.
Before reading this document, make sure you already read README.

iwpriv ra0 set [parameters]=[Value]

NOTE:

Execute one iwpriv/set command simultaneously.

4.1.1 DriverVersion

Check driver version by issue iwpriv set command.

Range:

Any value

Value:

0

4.1.2 CountryRegion

Set country region.

Range:

{0~7}

Value:

0: 1 ~ 11 ch
1: 1 ~ 13 ch
2: 10, 11 ch
3: 10 ~ 13 ch
4: 14 ch
5: 1 ~ 14 ch
6: 3 ~ 9 ch
7: 5 ~ 13 Ch

4.1.3 CountryRegionABand

Set country region for A band.

Range:

{0~9}

Value:

0: 36, 40, 44, 48, 52, 56, 60, 64, 149, 153, 157, 161, 165 ch
1: 36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140 ch
2: 36, 40, 44, 48, 52, 56, 60, 64 ch
3: 52, 56, 60, 64, 149, 153, 157, 161 ch
4: 149, 153, 157, 161, 165 ch
5: 149, 153, 157, 161 ch
6: 36, 40, 44, 48 ch
7: 36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140, 149, 153, 157, 161, 165 ch
8: 52, 56, 60, 64 ch
9: 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140 ch

4.1.4 SSID

Set AP SSID

Range:
{0~z, 1~32 ascii characters}
Value:

4.1.5 WirelessMode

Set Wireless Mode
Range:
{0~10}
Value:
0: legacy 11b/g mixed
1: legacy 11B only
2: legacy 11A only
3: legacy 11a/b/g mixed
4: legacy 11G only
5: 11ABGN mixed
6: 11N only
7: 11GN mixed
8: 11AN mixed
9: 11BGN mixed
10: 11AGN mixed

4.1.6 TxBurst:

Set TxBurst Enable or Disable
Range:
{0,1}
Value:
0:Disable,
1:Enable

4.1.7 PktAggregate:

Set Tx Aggregate Enable or Disable
Range:
{0,1}
Value:
0:Disable,
1:Enable

4.1.8 TxPreamble:

Set TxPreamble
Range:
{0~2}
Value:
0:Preamble Long,
1:Preamble Short,
2:Auto

4.1.9 TxPower:

Set Tx power in percentage
Range:
{0~100}
Value:

4.1.10 Channel

Set Channel, depends on CountryRegion or CountryRegionABand

4.1.11 BGProtection:

Set 11B/11G Protection

Range:

{0~2}

Value:

0:Auto,
1:Always on,
2:Always off

4.1.12 RTSThreshold:

Set RTS Threshold

Range:

{1~2347}

Value:

4.1.13 FragThreshold:

Set Fragment Threshold

Range:

{256~2346}

Value:

4.1.14 NetworkType:

Set Network type

Range:

{Infra,Adhoc}

Value:

4.1.15 AuthMode:

Set Authentication Mode

Range:

{OPEN,SHARED,WEPAUTO,WPAPSK,WPA2PSK,WPANONE}

Value:

4.1.16 EncrypType:

Set Encryption Type

Range:

{NONE,WEP,TKIP,AES}

Value:

4.1.17 DefaultKeyID:

Set Default Key ID

Range:

{1~4}

Value:

4.1.18 Key1

Set Key1 String

Range:

{5 ascii characters or 10 hex number or
13 ascii characters or 26 hex numbers}

Value:

4.1.19 Key2

Set Key2 String

Range:

{5 ascii characters or 10 hex number or
13 ascii characters or 26 hex numbers}

Value:

4.1.20 Key3

Set Key3 String

Range:

{5 ascii characters or 10 hex number or
13 ascii characters or 26 hex numbers}

Value:

4.1.21 Key4

Set Key4 String

Range:

{5 ascii characters or 10 hex number or
13 ascii characters or 26 hex numbers}

Value:

4.1.22 WPAPSK

WPA Pre-Shared Key

Range:

{8~63 ascii or 64 hex characters}

Value:

4.1.23 WmmCapable

Set WMM Capable

Range:

{0,1}

Value:

0:Disable WMM,
1:Enable WMM

4.1.24 IEEE80211H

Enabel IEEE802.11h support

Range:

{0,1}

Value:

0:Disable
1:Enable

4.1.25 PSMode

Set Power Saving Mode

Range:

{CAM, MAX_PSP, FAST_PSP}

Value:

4.1.26 ResetCounter

Reset statistics counter

Range:
Any vlaue
Value:
0

4.1.27 Debug

Set on debug level
Range:
{0 ~ 5}
Value:

0: OFF	no debug message display
1: ERROR	display error message
2: WARN	display warning message
3: TRACE	display trace message, usually used.
4: INFO	display informatic message
5: LOUD	display all message

4.1.28 HtRdg

Enable HT Reverse Direction Grant.
value
0: Disabled
1: Enabled

4.1.29 HtExtcha

To locate the 40MHz channel in combination with the control.
value
0: Below
1: Above

4.1.30 HtOpMode

Change HT operation mode.
value
0: HT mixed format
1: HT greenfield format

4.1.31 HtMpduDensity

Minimum separation of MPDUs in an A-MPDU.
value
0 ~ 7
0: no restriction
1: 1/4 μs
2: 1/2 μs
3: 1 μs
4: 2 μs
5: 4 μs
6: 8 μs
7: 16 μs

4.1.32 HtBw

Support channel width.
value
0: 20MHz

1: 40MHz

4.1.33 HtAutoBa

Enable auto block acknowledgment (Block Ack).

value

0: Disabled
1: Enabled

4.1.34 HtAmsdu

Enable aggregation of multiple MSDUs in one MPDU.

value

0: Disabled
1: Enabled

4.1.35 HtBaWinSize

Set BA WinSize.

value

1 ~ 64

4.1.36 HtGi

Support Short/Long GI.

value

0: long GI
1: short GI

4.1.37 HtMcs

MCS rate selection.

value

0 ~ 15
33: auto

4.1.38 HtProtect

Enable HT protection for legacy device.

value

0: Disable
1: Enable

4.1.39 HtMimoPs

MIMO power save.

value

0: Disable
1: Enable

4.1.40 FixedTxMode

Set Fixed Tx Mode for fixed rate setting

value

Mode = CCK
MCS= 0 => 1Mbps
MCS= 1 => 2Mbps
MCS= 2 => 5.5 Mbps
MCS= 3 => 11 Mbps
Mode = OFDM
MCS= 0 => 6Mbps

MCS= 1 => 9Mbps
MCS= 2 => 12Mbps
MCS= 3 => 18Mbps
MCS= 4 => 24Mbps
MCS= 5 => 36Mbps
MCS= 6 => 48Mbps
MCS= 7 => 54Mbps

4.2 Iwpriv Examples

4.2.1 Infrastructure

4.2.1.1 OPEN/NONE

Config STA to link with AP which is OPEN/NONE(Authentication/Encryption)

1. iwpriv ra0 set NetworkType=Infra
2. iwpriv ra0 set AuthMode=OPEN
3. iwpriv ra0 set EncrypType=NONE
4. iwpriv ra0 set SSID="AP's SSID"

4.2.1.2 SHARED/WEP

Config STA to link with AP which is SHARED/WEP(Authentication/Encryption)

1. iwpriv ra0 set NetworkType=Infra
2. iwpriv ra0 set AuthMode=SHARED
3. iwpriv ra0 set EncrypType=WEP
4. iwpriv ra0 set DefaultKeyID=1
5. iwpriv ra0 set Key1="AP's wep key"
6. iwpriv ra0 set SSID="AP's SSID"

4.2.1.3 WPAPSK/TKIP

Config STA to link with AP which is WPAPSK/TKIP(Authentication/Encryption)

1. iwpriv ra0 set NetworkType=Infra
2. iwpriv ra0 set AuthMode=WPAPSK
3. iwpriv ra0 set EncrypType=TKIP
4. iwpriv ra0 set SSID="AP's SSID"
5. iwpriv ra0 set WPAPSK="AP's wpa-preshared key"
6. iwpriv ra0 set SSID="AP's SSID"

4.2.1.4 WPAPSK/AES

Config STA to link with AP which is WPAPSK/AES(Authentication/Encryption)

1. iwpriv ra0 set NetworkType=Infra
2. iwpriv ra0 set AuthMode=WPAPSK
3. iwpriv ra0 set EncrypType=AES
4. iwpriv ra0 set SSID="AP's SSID"
5. iwpriv ra0 set WPAPSK="AP's wpa-preshared key"
6. iwpriv ra0 set SSID="AP's SSID"

4.2.1.5 WPA2PSK/TKIP

Config STA to link with AP which is WPA2PSK/TKIP(Authentication/Encryption)

1. iwpriv ra0 set NetworkType=Infra
2. iwpriv ra0 set AuthMode=WPA2PSK
3. iwpriv ra0 set EncrypType=TKIP
4. iwpriv ra0 set SSID="AP's SSID"
5. iwpriv ra0 set WPAPSK=12345678
6. iwpriv ra0 set SSID="AP's SSID"

4.2.2 Ad-Hoc

4.2.2.1 OPEN/NONE

Config STA to create/link as adhoc mode, which is OPEN/NONE(Authentication/Encryption)

1. iwpriv ra0 set NetworkType=Adhoc

2. iwpriv ra0 set AuthMode=OPEN
3. iwpriv ra0 set EncrypType=NONE
4. iwpriv ra0 set SSID="Adhoc's SSID"

4.2.2.2 WPANONE/TKIP

Config STA to create/link as adhoc mode, which is WPANONE/TKIP(Authentication/Encryption)

1. iwpriv ra0 set NetworkType=Adhoc
2. iwpriv ra0 set AuthMode=WPANONE
3. iwpriv ra0 set EncrypType=TKIP
4. iwpriv ra0 set SSID="AP's SSID"
5. iwpriv ra0 set WPAPSK=12345678
6. iwpriv ra0 set SSID="AP's SSID"

4.2.3 Get site survey

usage:

iwpriv ra0 get_site_survey

4.2.4 Get Statistics

usage:

```
iwpriv ra0 stat                         ; read statistic counter
iwpriv ra0 set ResetCounter=0           ; reset statistic counter
```

4.2.5 ANY SSID

Link with an AP which is the largest strength, set ANY SSID (ssidLen=0)

usage:

```
iwconfig ra0 essid ""
or
iwpriv ra0 set SSID=""
```

4.3 iwlist

This is detailed explanation of each parameters for iwlist.

iwlist ra0 scanning ; list the results after scanning(manual rescan)

4.4 iwconfig

The following are our support in standard configuration – iwconfig

1) iwconfig ra0 essid {NN|on|off} ; set essid
2) iwconfig ra0 mode {managed|ad-hoc|...} ; set wireless mode
3) iwconfig ra0 freq N.NNNN[k|M|G]] ; set frequency
4) iwconfig ra0 channel N ; set channel
5) iwconfig ra0 ap {N|off|auto} ; set AP address
6) iwconfig ra0 nick N ; set nickname
7) iwconfig ra0 rate {N|auto|fixed} ; set rate
8) iwconfig ra0 rts {N|auto|fixed|off} ; set RTS threshold
9) iwconfig ra0 frag {N|auto|fixed|off} ; set Fragment threshold
10) iwconfig ra0 enc {NNNN-NNNN|off} ; set encryption type
11) iwconfig ra0 power {period N|timeout N} ; set power management modes

NOTE:

Wireless extension usage, please refer to man page of 'iwconfig', 'iwlist' and 'iwpriv'.

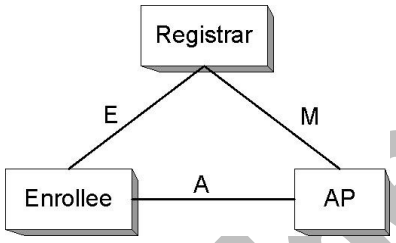
5 WPS – Wi-Fi Protected Setup

Simple Config Architectural Overview

This section presents a high-level description of the Simple Config architecture. Much of the material is taken directly from the Simple Config specification.

Figure 1 depicts the major components and their interfaces as defined by Wi-Fi Simple Config Spec. There are three logical components involved: the Registrar, the access point (AP), and the Enrollee.

- The **Enrollee** is a device seeking to join a WLAN domain. Once an Enrollee obtains a valid credential, it becomes a member.
- A **Registrar** is an entity with the authority to issue and revoke domain credentials. A registrar can be integrated into an AP.
- The **AP** can be either a WLAN AP or a wireless router.



Components and Interfaces

Registration initiation is ordinarily accomplished by a user action such as powering up the Enrollee and, optionally, running a setup wizard on the Registrar (PC).

5.1 Iwpriv Usage

This is detailed explanation of each parameters for iwpriv.
Before reading this document, make sure you already read README.

iwpriv ra0 [commands]=[Value]

NOTE:

Wireless extension private handlers.

5.1.1 wsc_conf_mode

Set WPS conf mode.

Range:

{0, 1, 2}

Value:

0: WPS Disabled
1: Enrollee
2: Registrar

5.1.2 wsc_mode

Set WPS mode, PIN or PBC.

Range:

{1, 2}

Value:

1: PIN
2: PBC

5.1.3 wsc_pin

Set Enrollee's PIN Code.

Range:

{00000000 ~ 99999999}

Value:

5.1.4 wsc_ ssid

Set WPS AP SSID.

Range:

{0~z, 1~32 ascii characters}

Value:

5.1.5 wsc_start

Trigger RT2870 STA driver to do WPS process.

Range:

NULL

Value:

5.1.6 wsc_ stop

Stop WPS process and don't wait upon two-minute timeout.

Range:

NULL

Value:

5.1.7 wsc_gen_pincode

Generate new PIN code.

Range:

NULL

Value:

5.1.8 wsc_cred_count

Set count of WPS credential, only support one credential for M8 in Registrar mode.

Range:

{1 ~ 8}

Value:

5.1.9 wsc_cred_ssid

Set SSID into credtentail[idx].

Range:

{"idx ssid_str"}

Value:

idx: 0 ~ 7

ssid_str: 0~z, 1~32 ascii characters

Example:

iwpriv ra0 wsc_cred_ssid "0 wps_ap1"

5.1.10 wsc_cred_auth

Set AuthMode into credtentail[idx].

Range:

{"idx auth_str"}

Value:

idx: 0 ~ 7

auth_str: OPEN, WPAPSK, WPA2PSK, SHARED, WPA, WPA2

Example:

iwpriv ra0 wsc_cred_auth "0 WPAPSK"

5.1.11 wsc_cred_encr

Set EncrypType into credtentail[idx].

Range:

{"idx encr_str"}

Value:

idx: 0 ~ 7

encr_str: NONE, WEP, TKIP, AES

Example:

iwpriv ra0 wsc_cred_encr "0 TKIP"

5.1.12 wsc_cred_keyIdx

Set Key Index into credtentail[idx].

Range:

{"idx key_index"}

Value:

idx: 0 ~ 7

key_index: 1 ~ 4

Example:

iwpriv ra0 wsc_cred_keyIdx "0 1"

5.1.13 wsc_cred_key

Set Key into credtentail[idx].

Range:

{"idx key"}

Value:

idx: 0 ~ 7

key: ASCII string (wep_key_len(=5,13), passphrase_len(=8~63))

OR

Hex string (wep_key_len(=10,26), passphrase_len(=64))

Example:

iwpriv ra0 wsc_cred_key "0 12345678" ;; Passphrase

iwpriv ra0 wsc_cred_key "0 abcd" ;; WEP Key

5.1.14 wsc_cred_mac

Set AP's MAC into credtentail[idx].

Range:

{"idx mac_str"}

Value:

idx: 0 ~ 7

mac_str: xx:xx:xx:xx:xx:xx

Example:

iwpriv ra0 wsc_cred_mac "0 00:11:22:33:44:55"

5.1.15 wsc_conn_by_idx

Connect AP by credential index.

Range:

{0 ~ 7}

Value:

idx: 0 ~ 7

5.1.16 wsc_auto_conn

If the registration is successful, driver will re-connect to AP or not.

Range:

{0, 1}

Value:

0: Disabled, driver won't re-connect to AP with new configurations.

1: Enabled, driver will re-connect to AP with new configurations.

5.2 WPS STA as an Enrollee or Registrar

Build WPS function. Please set 'HAS_WSC=y'.

5.2.1 Enrollee Mode

5.2.1.1 PIN mode:

Running Scenarios (case 'a' and 'b')

a. Adding an Enrollee to AP+Registrar (EAP)
[AP+Registrar]<----EAP--->[Enrollee Client]
b. Adding an Enrollee with external Registrar (UPnP/EAP)
[External Registrar]<----UPnP--->[AP_Proxy]<---EAP--->[Enrollee Client]

Note:
'EAP' indicates to use wireless medium and 'UPnP' indicates to use wired or wireless medium.

(i) [Registrar] or [AP+Registrar]
Enter the Enrollee PinCode on the Registrar and start WPS on the Registrar.
Note:
How to get the Enrollee PinCode? Use 'iwpriv ra0 stat' on the Enrollee.

(ii) [RT2870 Linux WPS STA]
iwpriv ra0 wsc_conf_mode 1 ;; Enrollee
iwpriv ra0 wsc_mode 1 ;; PIN
iwpriv ra0 wsc_ssid "AP's SSID"
iwpriv ra0 wsc_start

(iii) If the registration is successful, the Enrollee will be re-configured with the new parameters, and will connect to the AP with these new parameters.

5.2.1.2 PBC mode:

Running Scenarios (case 'a' only)

a. Adding an Enrollee to AP+Registrar (EAP)
[AP+Registrar]<----EAP--->[Client]

(i) [AP+Registrar]
Start PBC on the Registrar.

(ii) [RT2870 Linux WPS STA]
iwpriv ra0 wsc_conf_mode 1 ;; Enrollee
iwpriv ra0 wsc_mode 2 ;; PBC
iwpriv ra0 wsc_start

(iii) If the registration is successful, the Enrollee will be re-configured with the new parameters, and will connect to the AP with these new parameters.

5.2.2 Registrar Mode

5.2.2.1 PIN mode:

Running Scenarios (case 'a' and 'b')

a. Configure the un-configured AP
[Unconfigured AP]<----EAP--->[Registrar]
b. Configure the configured AP
Configured AP]<----EAP--->[Registrar]

(i) [AP]

Start PIN on the Enrollee WPS AP.

(ii) [RT2870 Linux WPS STA]

```
iwpriv ra0 wsc_conf_mode 2          ;; Registrar
iwpriv ra0 wsc_mode 1               ;; PIN
iwpriv ra0 wsc_pin xxxxxxxx         ;; AP's PIN Code
iwpriv ra0 wsc_ssid "AP's SSID"
iwpriv ra0 wsc_start
```

(iii) If the registration is successful;

in case 'a':

The Registrar will be re-configured with the new parameters, and will connect to the AP with these new parameters;

in case 'b':

The Registrar will be re-configured with AP's configurations, and will connect to the AP with these new parameters.

5.2.2.2 PBC mode:

Running Scenarios (case 'a' and 'b')

a. Configure the un-configured AP
   [Unconfigured AP]<----EAP--->[Registrar]
b. Configure the configured AP
   Configured AP]<----EAP--->[Registrar]

(i) [AP]

Start PBC on the Enrollee WPS AP.

(ii) [RT2870 Linux WPS STA]

```
iwpriv ra0 wsc_conf_mode 2          ;; Registrar
iwpriv ra0 wsc_mode 2               ;; PBC
iwpriv ra0 wsc_start
```

(iii) If the registration is successful;

in case 'a':

The Registrar will be re-configured with the new parameters, and will connect to the AP with these new parameters;

in case 'b':

The Registrar will be re-configured with AP's configurations, and will connect to the AP with these new parameters.

5.3 WPS IOCTL Usage

Detail parameters and arguments, please refer to above section for detail.

5.3.1 iwpriv commands without argument

1. iwpriv ra0 wsc_start
2. iwpriv ra0 wsc_stop
3. iwpriv ra0 wsc_gen_pincode

Example:

```
memset(&lwreq, 0, sizeof(lwreq));
sprintf(lwreq.ifr_name, "ra0", 3);
lwreq.u.mode = WSC_STOP;

/* Perform the private ioctl */
if(ioctl(skfd, RTPRIV_IOCTL_SET_WSC_PROFILE_U32_ITEM, &lwreq) < 0)
{
    fprintf(stderr, "Interface doesn't accept private ioctl...\n");
    return -1;
}
```

5.3.2 iwpriv commands with one INT argument

1. iwpriv ra0 wsc_cred_count 1
2. iwpriv ra0 wsc_conn_by_idx 1
3. iwpriv ra0 wsc_auto_conn 1
4. iwpriv ra0 wsc_conf_mode 1
5. iwpriv ra0 wsc_mode 1
6. iwpriv ra0 wsc_pin 12345678

Example:

```
memset(&lwreq, 0, sizeof(lwreq));
lwreq.u.data.length = 1;
cred_count = 1;
((int *) buffer)[i] = (int) cred_count;
offset = sizeof(int);

sprintf(lwreq.ifr_name, "ra0", 3);
lwreq.u.mode = WSC_CREDENTIAL_COUNT;
memcpy(lwreq.u.name + offset, buffer, IFNAMSIZ - offset);

/* Perform the private ioctl */
if(ioctl(skfd, RTPRIV_IOCTL_SET_WSC_PROFILE_U32_ITEM, &lwreq) < 0)
{
    fprintf(stderr, "Interface doesn't accept private ioctl...\n");
    return -1;
}
```

5.3.3 iwpriv commands with string argument

1. iwpriv ra0 wsc_ssid "0 xxxxx"
2. iwpriv ra0 wsc_cred_ssid "0 xxxxx"
3. iwpriv ra0 wsc_cred_auth "0 WPAPSK"
4. iwpriv ra0 wsc_cred_encr "0 TKIP"
5. iwpriv ra0 wsc_cred_keyIdx "0 1"
6. iwpriv ra0 wsc_cred_key "0 12345"
7. iwpriv ra0 wsc_cred_mac "0 00:11:22:33:44:55"

Example:

```
memset(&lwreq, 0, sizeof(lwreq));
memset(buffer, 0, 2048);
sprintf(lwreq.ifr_name, "ra0", 3);
sprintf(buffer, "0 wps_ssid_1");
lwreq.u.data.length = strlen(buffer) + 1;
lwreq.u.data.pointer = (caddr_t) buffer;
lwreq.u.data.flags = WSC_CREDENTIAL_SSID;

/* Perform the private ioctl */
if(ioctl(skfd, RTPRIV_IOCTL_SET_WSC_PROFILE_STRING_ITEM, &lwreq) < 0)
{
    fprintf(stderr, "Interface doesn't accept private ioctl...\n");
    return -1;
}
```

5.4 WPS IOCTL Sample Program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <netinet/in.h>   /* for sockaddr_in      */
#include <fcntl.h>
#include <time.h>
#include <sys/times.h>
#include <unistd.h>
#include <sys/socket.h>   /* for connect and socket*/
#include <sys/stat.h>
#include <err.h>
#include <errno.h>
#include <asm/types.h>
#include </usr/include/linux/wireless.h>
#include <sys/ioctl.h>

#define IFNAMSIZ 16

#define RTPRIV_IOCTL_SET_WSC_PROFILE_U32_ITEM      (SIOCIWFIRSTPRIV + 0x14)
#define RTPRIV_IOCTL_SET_WSC_PROFILE_STRING_ITEM   (SIOCIWFIRSTPRIV + 0x16)

enum {
            WSC_CREDENTIAL_COUNT = 1,
            WSC_CREDENTIAL_SSID = 2,
            WSC_CREDENTIAL_AUTH_MODE = 3,
            WSC_CREDENTIAL_ENCR_TYPE = 4,
            WSC_CREDENTIAL_KEY_INDEX = 5,
            WSC_CREDENTIAL_KEY = 6,
            WSC_CREDENTIAL_MAC = 7,
            WSC_SET_DRIVER_CONNECT_BY_CREDENTIAL_IDX = 8,
            WSC_SET_DRIVER_AUTO_CONNECT = 9,
            WSC_SET_CONF_MODE = 10, // Enrollee or Registrar
            WSC_SET_MODE = 11,     // PIN or PBC
            WSC_SET_PIN = 12,
            WSC_SET_SSID = 13,
            WSC_START = 14,
            WSC_STOP = 15,
            WSC_GEN_PIN_CODE = 16,
};

int main()
{
 struct iwreq lwreq;
 char        buffer[2048] = {0};
 int         cred_count;
 int         offset = 0;             /* Space for sub-ioctl index */
 int         skfd, i = 0;            /* generic raw socket desc. */

 skfd = socket(AF_INET, SOCK_DGRAM, 0);
 if (skfd < 0)
            return -1;

 ////////////// WSC_STOP //////////////
 memset(&lwreq, 0, sizeof(lwreq));
 sprintf(lwreq.ifr_name, "ra0", 3);
 lwreq.u.mode = WSC_STOP;

 /* Perform the private ioctl */
if(ioctl(skfd, RTPRIV_IOCTL_SET_WSC_PROFILE_U32_ITEM, &lwreq) < 0)
 {
            fprintf(stderr, "Interface doesn't accept private ioctl...\n");
            return -1;
 }
 ////////////////////////////////////

 /////// WSC_CREDENTIAL_COUNT ///////
 memset(&lwreq, 0, sizeof(lwreq));
 lwreq.u.data.length = 1;
```

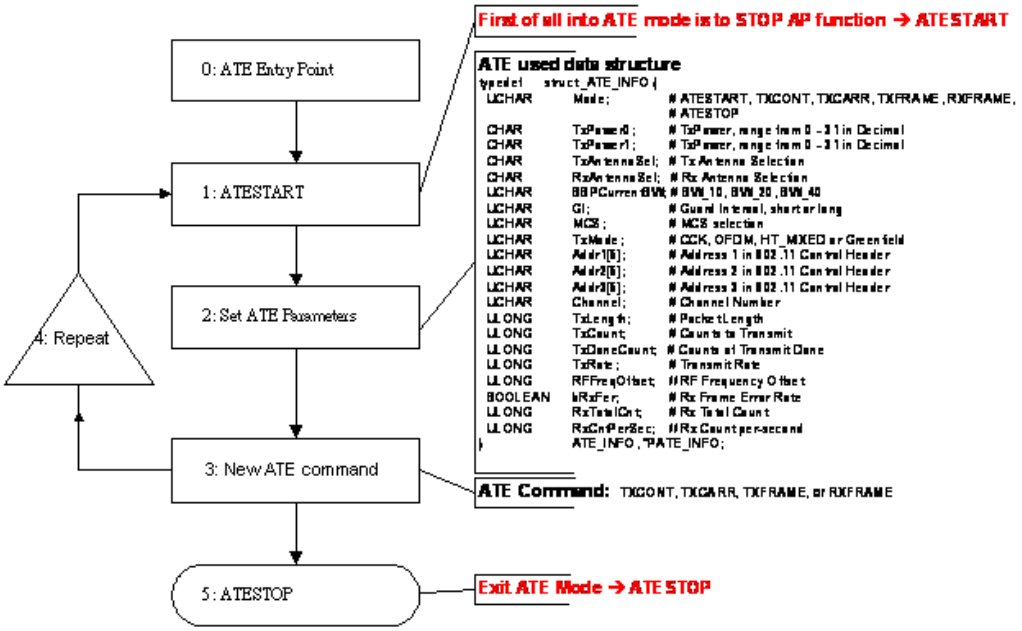
```
cred_count = 1;
((int *) buffer)[i] = (int) cred_count;
offset = sizeof(int);

sprintf(lwreq.ifr_name, "ra0", 3);
lwreq.u.mode = WSC_CREDENTIAL_COUNT;
memcpy(lwreq.u.name + offset, buffer, IFNAMSIZ - offset);

/* Perform the private ioctl */
if(ioctl(skfd, RTPRIV_IOCTL_SET_WSC_PROFILE_U32_ITEM, &lwreq) < 0)
{
        fprintf(stderr, "Interface doesn't accept private ioctl...\n");
        return -1;
}
////////////////////////////////////

/////// WSC_CREDENTIAL_SSID ///////
memset(&lwreq, 0, sizeof(lwreq));
memset(buffer, 0, 2048);
sprintf(lwreq.ifr_name, "ra0", 3);
sprintf(buffer, "0 wps_ssid_1");
lwreq.u.data.length = strlen(buffer) + 1;
lwreq.u.data.pointer = (caddr_t) buffer;
lwreq.u.data.flags = WSC_CREDENTIAL_SSID;

/* Perform the private ioctl */
if(ioctl(skfd, RTPRIV_IOCTL_SET_WSC_PROFILE_STRING_ITEM, &lwreq) < 0)
{
        fprintf(stderr, "Interface doesn't accept private ioctl...\n");
        return -1;
}
////////////////////////////////////

close(skfd);
return 0;
}
```

6 ATE Test Command Format

****** IMPORTANT ******

If you are not familiar with hardware, it is recommanded not to modify hardware default value.

Ralink ATE Operation Flow



Note:

1. Channel setting would take effect on next ATE command.
2. TxPower would take effect after frame transmit start.
   TxPower can be changed dynamically on any ATE command operating.
3. Any ATE parameters have to be included into ATE_INFO structure.
4. Enter ATE mode by set ATE command "ATESTART".
   a. Abort all TX rings
   b. AsicDisableSync → Stop Beacon.
   c. Stop REKEYTimer
   d. Stop CounterMeasureTimer
   e. MacTableReset
5. Use TXCONT to check transmit power mask.
6. Use TXCARR to check frequency lock (under 25ppm).

6.2. iwpriv ra0 set [parameters]=[val]

Syntax:			Example	
Section#	parameters		11.1.5	ATECHANNEL
		Explanation		Set ATE channel.
	Value:			Value:
	0:	...		1:
	1:	...		2:
	.:	...		.:

6.2.1. ATE

Set ATE actions.

Value:

ATESTART	- Stop AP & ATE function.
ATESTOP	- Start AP function.
TXCONT	- Start AP continuous TX, for power mask.
TXCARR	- Start AP carrier test, for frequency calibration.
TXFRAME	- Transmit frame, for EVM.
RXFRAME	- Continuous RX, for PER/FER.

6.2.2. ATEDA

Set ATE frame header addr1.

Value:

xx:xx:xx:xx:xx:xx ; hex

6.2.3. ATESA

Set ATE frame header addr2.

Value:

xx:xx:xx:xx:xx:xx ; hex

6.2.4. ATEBSSID

Set ATE frame header addr3.

Value:

xx:xx:xx:xx:xx:xx ; hex

6.2.5. ATECHANNEL

Set ATE Channel, deimal.

Value:

802.11b/g: 1 ~ 14 depends on CountryRegion setting

6.2.6. ATETXPOW0

Set ATE Tx power for Antenna 1.

Value:

0 ~ 31 ; 5-bits only, deimal

6.2.7. ATETXPOW1

Set ATE Tx power for Antenna 2.

Value:

0 ~ 31 ; 5-bits only, decimal

6.2.8. ATETXFREQOFFSET

Set ATE RF frequency offset.

Value:

0 ~ 63 ; unit: 2KHz, deimal

6.2.9. ATETXLEN

Set ATE frame length.

Value:

24 ~ 1500 ; decimal

6.2.10. ATETXCNT

Set ATE frame Tx count.

Value:

1 ~ ; 32-bit, decimal

6.2.11. ATETXMODE (Refer to TxMode)

Set ATE Tx Mode.

Value:

0:	CCK	802.11b
1:	OFDM	802.11g
2:	HT_MIX	802.11b/g/n
3:	Green Field	802.11n

6.2.12. ATETXBW (Refer to TxMode)

Set ATE Tx Bandwidth.

Value:

0:	20MHz
1:	40MHz

6.2.13. ATETXGI (Refer to TxMode)

Set ATE Tx Guard Interval.

Value:

0 :	Long
1 :	Short

6.2.14. ATETXMCS (Refer to TxMode)

Set ATE Tx MCS type.

Value:

0 ~ 15

6.2.15. ATETXANT

Set ATE TX antenna.

Value:

0 :	All
1 :	Antenna one
2 :	Antenna two

6.2.16. ATERXANT

Set ATE RX antenna.

Value:

0 :	All
1 :	Antenna one
2 :	Antenna two
3 :	Antenna three

6.2.17. ATERXFER

Set ATE to periodic show up RxCount(per-second) and RxTotalCount.

Value:

0 : Disable counter show up

1 : Enable counter show up

6.2.18. ATESHOW

Show all parameters of ATE.

Value:

1

6.2.19. ATEHELP

List all commands of ATE.

Value:

1

6.2.20. ResetCounter

Reset statistic counter.

Value:

0

6.2.21. ATERRF

Read all of the RF registers.

Value:

1

6.2.22. ATEWRF1

Write the RF register 1.

Value:

xxxxxxxx ;32-bit, hex

6.2.23. ATEWRF2

Write the RF register 2.

Value:

xxxxxxxx ;32-bit, hex

6.2.24. ATEWRF3

Write the RF register 3.

Value:

xxxxxxxx ;32-bit, hex

6.2.25. ATEWRF4

Write the RF register 4.

Value:

xxxxxxxx ;32-bit, hex

6.3. Tx Mode, MCS, BW and GI Selection Table

6.3.1. MODE = 0, Legacy CCK	
MCS = 0	Long Preamble CCK 1Mbps
MCS = 1	Long Preamble CCK 2Mbps
MCS = 2	Long Preamble CCK 5.5Mbps
MCS = 3	Long Preamble CCK 11Mbps
MCS = 8	Short Preamble CCK 1Mbps, * illegal rate
MCS = 9	Short Preamble CCK 2Mbps
MCS = 10	Short Preamble 5.5Mbps
MCS = 11	Short Preamble 11Mbps
Notes: 1. Other MCS codes are reserved in legacy CCK mode. 2. BW, SGI and STBC are reserved in legacy CCK mode.	
6.3.2. MODE = 1, Legacy OFDM	
MCS = 0	6Mbps
MCS = 1	9Mbps
MCS = 2	12Mbps
MCS = 3	18Mbps
MCS = 4	24Mbps
MCS = 5	36Mbps
MCS = 6	48Mbps
MCS = 7	54Mbps
Notes: 1. Other MCS code in legacy CCK mode are reserved. 2. When BW = 1, duplicate legacy OFDM is sent. 3. SGI, STBC are reserved in legacy OFDM mode.	
6.3.3. MODE = 2, HT Mixed Mode **6.3.4. MODE = 3, HT Greenfield**	
MCS = 0 (1S)	(BW=0, SGI=0) 6.5Mbps
MCS = 1	(BW=0, SGI=0) 13Mbps
MCS = 2	(BW=0, SGI=0) 19.5Mbps
MCS = 3	(BW=0, SGI=0) 26Mbps
MCS = 4	(BW=0, SGI=0) 39Mbps
MCS = 5	(BW=0, SGI=0) 52Mbps
MCS = 6	(BW=0, SGI=0) 58.5Mbps
MCS = 7	(BW=0, SGI=0) 65Mbps
MCS = 8 (2S)	(BW=0, SGI=0) 13Mbps
MCS = 9	(BW=0, SGI=0) 26Mbps
MCS = 10	(BW=0, SGI=0) 39Mbps
MCS = 11	(BW=0, SGI=0) 52Mbps
MCS = 12	(BW=0, SGI=0) 78Mbps
MCS = 13	(BW=0, SGI=0) 104Mbps
MCS = 14	(BW=0, SGI=0) 117Mbps
MCS = 15	(BW=0, SGI=0) 130Mbps
MCS = 32	(BW=1, SGI=0) HT duplicate 6Mbps

Notes:

1. When BW=1, PHY_RATE = PHY_RATE * 2
2. When SGI=1, PHY_RATE = PHY_RATE * 10/9
3. The effects of BW and SGI are accumulative.
4. When MCS=0~7(1S, One Tx Stream), STBC option is supported. SGI option is supported. BW option is supported.
5. When MCS=8~15(2S, Two Tx Stream), STBC option is NOT supported. SGI option is supported. BW option is supported.
6. When MCS=32, only SGI option is supported. BW and STBC option are not supported. (BW =1, STBC=0)
7. Other MCS code in HT mode are reserved.
8. When STBC is supported. Only STBC = 1 is allowed. STBC will extend the transmission range but will not increase transmission rate.

6.4. Examples

6.4.1. Check EVM & Power

```
iwpriv ra0 set ATE=ATESTART
iwpriv ra0 set ATEDA=00:11:22:33:44:55
iwpriv ra0 set ATESA=00:aa:bb:cc:dd:ee
iwpriv ra0 set ATEBSSID=00:11:22:33:44:55
iwpriv ra0 set ATECHANNEL=1              ; set Channel
iwpriv ra0 set ATETXMODE=1               ; set TX-Mode.
iwpriv ra0 set ATETXMCS=7                ; set MCS type.
iwpriv ra0 set ATETXBW=0                 ; set Bandwidth
iwpriv ra0 set ATETXGI=0                 ; set Long GI.
iwpriv ra0 set ATETXLEN=1024             ; set packet length.
iwpriv ra0 set ATETXPOW0=18
iwpriv ra0 set ATETXPOW1=18
iwpriv ra0 set ATETXCNT=100000
iwpriv ra0 set ATETXFRAME
…
iwpriv ra0 set ATETXPOW0=19
…
iwpriv ra0 set ATETXPOW0=20
…
iwpriv ra0 set ATE=ATESTART
```

6.4.2. Check Carrier

```
iwpriv ra0 set ATE=ATESTART
iwpriv ra0 set ATECHANNEL=1              ; set Channel
iwpriv ra0 set ATETXMODE=1               ; set TX-Mode.
iwpriv ra0 set ATETXMCS=7                ; set MCS type.
iwpriv ra0 set ATETXBW=0                 ; set Bandwidth
iwpriv ra0 set ATETXCNT=200              ; Tx frame count(decmial)
iwpriv ra0 set ATE=TXFRAME               ; Start Tx Frame(inform BBP to change, modulation mode)
iwpriv ra0 set ATE=TXCARR                ; Start Tx carrier, Measure carrier with instrument
iwpriv ra0 set ATETXPOW0=05
iwpriv ra0 set ATETXPOW1=05
iwpriv ra0 set ATETXFREQOFFSET=19
iwpriv ra0 set ATE=ATESTART
```

6.4.3. Check specturm mask

```
iwpriv ra0 set ATE=ATESTART
iwpriv ra0 set ATECHANNEL=1              ; set Channel
iwpriv ra0 set ATETXMODE=1               ; set TX-Mode.
iwpriv ra0 set ATETXMCS=7                ; set MCS type.
iwpriv ra0 set ATETXBW=0                 ; set Bandwidth
iwpriv ra0 set ATETXCNT=200              ; Tx frame count(decmial)
iwpriv ra0 set ATE=TXFRAME               ; Start Tx Frame(inform BBP to change, modulation mode)
iwpriv ra0 set ATE=TXCONT                ; Start continuous TX, Measure specturm mask with instrument
iwpriv ra0 set ATETXPOW0=5
iwpriv ra0 set ATETXPOW1=5
iwpriv ra0 set ATE=ATESTART
```

6.4.4. Frequency offset tuning

```
iwpriv ra0 set ATE=ATESTART
iwpriv ra0 set ATECHANNEL=1              ; set Channel
iwpriv ra0 set ATETXMODE=1               ; set TX-Mode.
iwpriv ra0 set ATETXMCS=7                ; set MCS type.
iwpriv ra0 set ATETXCNT=200              ; Tx frame count(decmial)
iwpriv ra0 set ATETXFREQOFFSET=0         ; Set frequency offset 0(decimal)
iwpriv ra0 set ATE=TXFRAME               ; Start Tx Frame
iwpriv ra0 set ATE=TXCARR                ; Start Tx carrier, Measure carrier frequency with instrument
iwpriv ra0 set ATETXFREQOFFSET=10        ; Dynamic turning frequency offset, 10(decimal)
iwpriv ra0 set ATETXFREQOFFSET=20        ; Dynamic turning frequency offset, 20(decimal)
iwpriv ra0 set ATE=ATESTART              ; Stop, Store the tuning result to EEPROM
```

6.4.5. Rx

```
iwpriv ra0 set ATE=ATESTART
iwpriv ra0 set ATECHANNEL=1              ; set Channel
```

```
iwpriv ra0 set ResetCounter=0          ; Reset statistic counter
iwpriv ra0 set ATETXMODE=1             ; set TX-Mode.
iwpriv ra0 set ATETXMCS=7              ; set MCS type.
iwpriv ra0 set ATETXBW=0               ; set Bandwidth
iwpriv ra0 set ATE=RXFRAME             ; Start Rx,
iwpriv ra0 set ATERXFER=1              ; show RxCnt and RSSI/per-antenna, Transmit test packets
iwpriv ra0 set ATE=ATESTART            ; Stop
iwpriv ra0 stat                        ; get statistics counter
iwpriv ra0 set ATERXFER=1
iwpriv ra0 set ATERXANT=1

iwpriv ra0 set ATE=ATESTART
iwpriv ra0 set ATERXANT=0
iwpriv ra0 set ATE=RXFRAME
```

6.4.6. Show all ate parameters

iwpriv ra0 set ATESHOW=1

```
Mode=4
TxPower0=0
TxPower1=0
TxAntennaSel=0
RxAntennaSel=0
BBPCurrentBW=0
GI=0
MCS=7
TxMode=1
Addr1=00:11:22:aa:bb:cc
Addr2=00:11:22:aa:bb:cc
Addr3=00:11:22:aa:bb:cc
Channel=1
TxLength=1024
TxCount=40000
TxRate=11
RFFreqOffset=0
```

6.4.7. Online help

iwpriv ra0 set ATEHELP=1

```
ATE=ATESTART, ATESTOP, TXCONT, TXCARR, TXFRAME, RXFRAME
ATEDA
ATESA
ATEBSSID
ATECHANNEL, range:0~14
ATETXPOW0, set power level of antenna 1.
ATETXPOW1, set power level of antenna 2.
ATETXANT, set TX antenna. 0:all, 1:antenna one, 2:antenna two.
ATERXANT, set RX antenna.0:all, 1:antenna one, 2:antenna tow, 3:antenna three.
ATETXFREQOFFSET, set frequency offset, range 0~63
ATETXBW, set BandWidth, 0:20MHz, 1:40MHz.
ATETXLEN, set Frame length, range 24~1500
ATETXCNT, set how many frame going to transmit.
ATETXRATE, set rate, reference to rate table.
ATETXMCS, set MCS, reference to rate table.
ATETXMODE, set Mode 0:CCK, 1:OFDM, 2:HT-Mix, 3:GreenField, reference to rate table.
ATETXGI, set GI interval, 0:Long, 1:Short
ATERXFER, 0:disable Rx Frame error rate. 1:enable Rx Frame error rate.
ATESHOW, display all parameters of ATE.
ATEHELP, online help.
```

6.4.8. Display Rx Packet Count and RSSI

iwpriv ra0 set ATE=RXFRAME ➔ ***Start Rx***
iwpriv ra0 set ATERXANT=0 ➔ ***Enable All Three Rx Antennas***
iwpriv ra0 set ATERXFER=1 ➔ ***Enable Rx Frame Error Rate: RxCnt/RxTotal***

```
MlmePeriodicExec: Rx packet cnt = 2/4
MlmePeriodicExec: Rx AvgRssi0=-88, AvgRssi1=-80, AvgRssi2=-91
```

MlmePeriodicExec: Rx packet cnt = 2/6
MlmePeriodicExec: Rx AvgRssi0=-86, AvgRssi1=-77, AvgRssi2=-89…
…

iwpriv ra0 set ATE=RXFRAME ➔ ***Start Rx***
iwpriv ra0 set ATERXANT=1 ➔ ***Enable Three Rx Antenna-1***
iwpriv ra0 set ATERXFER=1 ➔ ***Enable Rx Frame Error Rate: RxCnt/RxTotal***

MlmePeriodicExec: Rx packet cnt = 0/7
MlmePeriodicExec: Rx AvgRssi=-87

MlmePeriodicExec: Rx packet cnt = 7/14
MlmePeriodicExec: Rx AvgRssi=-90
…
…

6.5. iwpriv ra0 bbp [parameters]=[Value]

Read/Write BBP register by ID number.

6.5.1. BBPID

Read BBP register, BBPID only, no "=" symbol.

BBPID:

0 ~ xx ; decimal, 8-bit

6.5.2. BBPID=Value

Write BBP register.

BBPID:

0 ~ xx ; decimal, 8-bit

Value:

00 ~FF ; hexdecimal, 8-bit

6.6. iwpriv ra0 mac [parameters]=[val]

Read/Write MAC register by offset.

6.6.1. MAC_OFFSET

Read MAC register, MAC_OFFSET only, no "=" symbol.

MAC_OFFSET:

0000 ~ FFFF ; hexdecimal, 16-bit

6.6.2. MAC_OFFSET=Value

Write MAC register.

MAC_OFFSET:

0000 ~ FFFF ; hexdecimal, 16-bit

Value:

0000 ~FFFF ; hexdecimal, 32-bit

6.7. iwpriv ra0 e2p [parameters]=[val]

Read/Write EEPROM content by address.

6.7.1. EEP_ADDR

Read EEPROM content, EEP_ADDR only, no "=" symbol.

EEP_ADDR:

00 ~ FF ; hexdecimal, 16-bit alignment (0, 2, 4, 6, 8, A, C, …)

6.7.2. EEP_ADDR=Value

Write EEPROM content.

EEP_ADDR:

00 ~ FF ; hexdecimal, 16-bit alignment (0, 2, 4, 6, 8, A, C, …)

Value:

0000 ~FFFF ; hexdecimal, 16-bit

6.8. Example

6.8.1. Hardware access

```
iwpriv ra0 bbp 0                    # read BBP register 0
iwpriv ra0 bbp 0=12                 # write BBP register 0 as 0x12
iwpriv ra0 mac 0                    # read MAC register 0
iwpriv ra0 mac 0=1234abcd           # write MAC register 0 as 0x1234abcd
iwpriv ra0 e2p 0                    # read E2PROM 0
iwpriv ra0 e2p c=12ab               # write E2PROM 0xc as 0x12ab
```

6.8.2. Statistic counter operation

```
iwpriv ra0 stat                     # read statistic counter
iwpriv ra0 set ResetCounter=0       # reset statistic counter
```

6.8.3. Suggestion:

1. To turn on ATE functionality, you have to add compile flag "RALINK_ATE" to Makefile
2. Before doing ATE testing, please stop AP function
3. If you want to test another ATE action, prefer to stop AP & ATE function
4. All ATE function settings will lose efficacy after reboot.
5. Before hardware register access, please reference hardware spec.

Note.

In ATE mode, the channel must set via "ATECHANNEL"

6.9. ated

ated - user space ATE agent program for RT2870 linux driver, Ralink Tech. Corp.

RT2870 ATE daemon - ated, which comes with RT2870 linux driver.

Here will explains the relationship between the linux driver, Windows QA GUI and RT2870 ATE daemon.

In addtion, this will teach you how to use this ATE daemon.

6.9.1. Introduction

The ated is an optional user space component for RT2870 linux driver.
When ated starts, AP enters ATE mode (i.e.,ATESTART) immediately.
It behaves as a proxy between Windows QA GUI and RT2870 linux driver when ATE process proceeds.
And ated will be killed automatically when Windows QA GUI is closed.

You can kill it manually, too(for example, type '$killall ated').
RT2870 linux driver will leave ATE mode either ated is killed or QA GUI is closed.

6.9.2. Environment setup

1. Connect the platform you want to test directly with a Windows host by ether network line.
2. In the Windows host, run WinPcap_4_0.exe for the QA GUI.

6.9.3. How to use ated for ATE purpose

1. First you should set **both "HAS_ATE=y" and "HAS_2870_QA=y"** in the file ~/Module/os/linux/**config.mk** and compile the driver.
2. Modify the Makefile according to our target "PLATFORM".
3. Change the path of "CROSS_COMPILE" if needed.
4. Remove "-I$(INCLUDE)" about in line 39 if your target "PLATFORM" is not "PC".
5. Then type 'make' command to compile the source code of the daemon.
6. After the driver interface "ra0" has started up, attach both of "ra0" and the ethernet interface to the bridge interface "br0".
7. Manually start ated, type '$ated -bbrX -iraX'.(For further usage of options, type $ated -h)
8. In the Windows host, run RT2870QA_ATE.exe.
9. Select the wired network adapter.
10. Choose 2870_ATE, then press OK.

Note :

The names of WLAN interface(default is "ra0") and Bridge interface(default is "br0") must be specified manually (for example : '$ated –b br1 –ira2') if your WLAN interface or Bridge interface is not "ra0" or "br0" respectively !

7 IOCTL

7.1 Parameters for iwconfig

Access	Description	ID	Parameters
Get	BSSID, MAC Address	SIOCGIFHWADDR	wrq->u.name, (length = 6)
	WLAN Name	SIOCGIWNAME	wrq->u.name = "RT2870 Wireless", length = strlen(wrq->u.name)

	SSID	SIOCGIWESSID	erq = &wrq->u.essid;
			
			if(OPSTATUS_TEST_FLAG(pAd,fOP_STATUS_MEDIA_STATE_CONNECTED))
			{
			erq->flags=1;
			erq->length = pAd-> CommonCfg.SsidLen;
			Status = copy_to_user(erq->pointer,
			pAd-> CommonCfg.Ssid, erq->length);
			}
			else
			{
			erq->flags=0;
			erq->length=0;
			}

	Channel / Frequency (Hz)	SIOCGIWFREQ	wrq->u.freq.m = pAd-> CommonCfg.Channel;
			wrq->u.freq.e = 0;
			wrq->u.freq.i = 0;

	Node name/nickname	SIOCGIWNICKN	erq = &wrq->u.data;
			erq->length = strlen(pAd->nickn);
			Status = copy_to_user(erq->pointer, pAd->nickn, erq->length);

	Bit Rate (bps)	SIOCGIWRATE	wrq->u.bitrate.value = RateIdTo500Kbps[pAd-> CommonCfg.TxRate] * 500000; wrq->u.bitrate.disabled = 0;

	RTS/CTS threshold	SIOCGIWRTS	wrq->u.rts.value = (INT) pAd-> CommonCfg.RtsThreshold;
			wrq->u.rts.disabled = (wrq->u.rts.value == MAX_RTS_THRESHOLD);
			wrq->u.rts.fixed = 1;

	Fragmentation threshold (bytes)	SIOCGIWFRAG	wrq->u.frag.value = (INT) pAd-> CommonCfg.FragmentThreshold;
			wrq->u.frag.disabled = (wrq->u.frag.value >= MAX_FRAG_THRESHOLD);
			wrq->u.frag.fixed = 1;

	Encoding token & mode	SIOCGIWENCODE	index = (wrq->u.encoding.flags & IW_ENCODE_INDEX) - 1;
			if ((index < 0) || (index >= NR_WEP_KEYS))
			index = pAd-> CommonCfg.DefaultKeyId; // Default key for tx (shared key)
			if (pAd-> CommonCfg.AuthMode == Ndis802_11AuthModeOpen)
			wrq->u.encoding.flags = IW_ENCODE_OPEN;
			else if (pAd-> CommonCfg.AuthMode == Ndis802_11AuthModeShared)
			wrq->u.encoding.flags = IW_ENCODE_RESTRICTED;
			if (pAd-> CommonCfg.WepStatus == Ndis802_11WEPDisabled)
			wrq->u.encoding.flags |= IW_ENCODE_DISABLED;
			else
			{
			if(wrq->u.encoding.pointer)
			{
			wrq->u.encoding.length = pAd->SharedKey[index].KeyLen;
			Status = copy_to_user(wrq->u.encoding.pointer,
			pAd->SharedKey[index].Key,
			pAd->SharedKey[index].KeyLen);
			wrq->u.encoding.flags |= (index + 1);
			}
			}

	AP's MAC address	SIOCGIWAP	wrq->u.ap_addr.sa_family = ARPHRD_ETHER; memcpy(wrq->u.ap_addr.sa_data, &pAd-> CommonCfg.Bssid, ETH_ALEN);

	Operation Mode	SIOCGIWMODE	if (ADHOC_ON(pAd))
			{
			BssType = Ndis802_11IBSS;
			wrq->u.mode = IW_MODE_ADHOC;
			}
			else if (INFRA_ON(pAd))
			{
			BssType = Ndis802_11Infrastructure;

			wrq->u.mode = IW_MODE_INFRA;
			}
			else
			{
			BssType = Ndis802_11AutoUnknown;
			wrq->u.mode = IW_MODE_AUTO;
			}

Access	**Description**	**ID**	**Parameters**

Set	SSID	SIOCSIWESSID	erq = &wrq->u.essid;
			memset(&Ssid, 0x00, sizeof(NDIS_802_11_SSID));
			if (erq->flags)
			{
			if (erq->length > IW_ESSID_MAX_SIZE)
			{
			Status = -E2BIG;
			break;
			}
			Status = copy_from_user(Ssid.Ssid, erq->pointer, (erq->length - 1));
			Ssid.SsidLength = erq->length - 1; //minus null character.
			}
			else
			{
			Ssid.SsidLength = 0; // ANY ssid
			memcpy(pSsid->Ssid, "", 0);
			pAd->CommonCfg.BssType = BSS_INFRA;
			pAd->CommonCfg.AuthMode = Ndis802_11AuthModeOpen;
			pAd->CommonCfg.WepStatus = Ndis802_11EncryptionDisabled;
			
			}
			pSsid = &Ssid;
			if (pAd->Mlme.CntlMachine.CurrState != CNTL_IDLE)
			{
			MlmeRestartStateMachine(pAd);
			}
			pAd->MlmeAux.CurrReqIsFromNdis = FALSE;
			MlmeEnqueue(pAd,
			MLME_CNTL_STATE_MACHINE,
			OID_802_11_SSID,
			sizeof(NDIS_802_11_SSID),
			(VOID *)pSsid);
			Status = NDIS_STATUS_SUCCESS;
			StateMachineTouched = TRUE;

	Channel / Frequency (Hz)	SIOCSIWFREQ	frq = &wrq->u.freq;
			if((frq->e == 0) && (frq->m <= 1000))
			chan = frq->m; // Setting by channel number
			else
			MAP_KHZ_TO_CHANNEL_ID((frq->m /100) , chan);
			pAd->CommonCfg.Channel = chan;

	node name/nickname	SIOCSIWNICKN	erq = &wrq->u.data;
			if (erq->flags)
			{
			if (erq->length <= IW_ESSID_MAX_SIZE)
			Status = copy_from_user(pAd->nickn, erq->pointer, erq->length);
			else
			Status = -E2BIG;
			}

	Bit Rate (bps)	SIOCSIWRATE	RTMPSetDesiredRates(pAd, wrq->u.bitrate.value);

	RTS/CTS threshold	SIOCSIWRTS	RtsThresh = wrq->u.rts.value;
			if (wrq->u.rts.disabled)
			RtsThresh = MAX_RTS_THRESHOLD;
			if((RtsThresh > 0) && (RtsThresh <= MAX_RTS_THRESHOLD))
			pAd->CommonCfg.RtsThreshold = (USHORT)RtsThresh;
			else if (RtsThresh == 0)
			pAd->CommonCfg.RtsThreshold = MAX_RTS_THRESHOLD;

	Fragmentation threshold (bytes)	SIOCSIWFRAG	FragThresh = wrq->u.frag.value;
			if (wrq->u.rts.disabled)
			FragThresh = MAX_FRAG_THRESHOLD;
			if ((FragThresh >= MIN_FRAG_THRESHOLD) &&

			(FragThresh <= MAX_FRAG_THRESHOLD))
			pAd->CommonCfg.FragmentThreshold = (USHORT)FragThresh;
			else if (FragThresh == 0)
			pAd->CommonCfg.FragmentThreshold = MAX_FRAG_THRESHOLD;
			if (pAd->CommonCfg.FragmentThreshold == MAX_FRAG_THRESHOLD)
			pAd->CommonCfg.bFragmentZeroDisable = TRUE;
			else
			pAd->CommonCfg.bFragmentZeroDisable = FALSE;

	Encoding
	token & mode
		SIOCSIWENCODE	index = (wrq->u.encoding.flags & IW_ENCODE_INDEX) - 1;
			if((index < 0) || (index >= NR_WEP_KEYS))
			index = pAd->CommonCfg.DefaultKeyId; // Default key for tx (shared key)
			
			if(wrq->u.encoding.pointer)
			{
			len = wrq->u.encoding.length;
			if(len > WEP_LARGE_KEY_LEN)
			len = WEP_LARGE_KEY_LEN;
			
			memset(pAd->SharedKey[index].Key, 0x00, MAX_LEN_OF_KEY);
			Status = copy_from_user(pAd->SharedKey[index].Key,
			wrq->u.encoding.pointer, len);
			pAd->SharedKey[index].KeyLen = len <= WEP_SMALL_KEY_LEN ?
			WEP_SMALL_KEY_LEN :
			WEP_LARGE_KEY_LEN;
			}
			pAd->CommonCfg.DefaultKeyId = (UCHAR) index;
			if (wrq->u.encoding.flags & IW_ENCODE_DISABLED)
			pAd->CommonCfg.WepStatus = Ndis802_11WEPDisabled;
			else
			pAd->CommonCfg.WepStatus = Ndis802_11WEPEnabled;
			
			if (wrq->u.encoding.flags & IW_ENCODE_RESTRICTED)
			pAd->CommonCfg.AuthMode = Ndis802_11AuthModeShared;
			else
			pAd->CommonCfg.AuthMode = Ndis802_11AuthModeOpen;
			
			if(pAd->CommonCfg.WepStatus == Ndis802_11WEPDisabled)
			pAd->CommonCfg.AuthMode = Ndis802_11AuthModeOpen;

	AP's MAC address	SIOCSIWAP	Status = copy_from_user(&Bssid, &wrq->u.ap_addr.sa_data,
			sizeof(NDIS_802_11_MAC_ADDRESS));
			if (pAd->Mlme.CntlMachine.CurrState != CNTL_IDLE)
			{
			MlmeRestartStateMachine(pAd);
			}
			pAd->MlmeAux.CurrReqIsFromNdis = FALSE;
			MlmeEnqueue(pAd,
			MLME_CNTL_STATE_MACHINE,
			OID_802_11_BSSID,
			sizeof(NDIS_802_11_MAC_ADDRESS),
			(VOID *)&Bssid);
			Status = NDIS_STATUS_SUCCESS;
			StateMachineTouched = TRUE;

	Operation Mode	SIOCSIWMODE	if(wrq->u.mode == IW_MODE_ADHOC)
			{
			  if (pAd->CommonCfg.BssType != BSS_ADHOC)
			  {
			  pAd->bConfigChanged = TRUE;
			  }
			pAd->CommonCfg.BssType = BSS_ADHOC;
			}
			else if (wrq->u.mode == IW_MODE_INFRA)
			{
			  if (pAd->CommonCfg.BssType != BSS_INFRA)
			  {
			  pAd->bConfigChanged = TRUE;
			  }
			  pAd->CommonCfg.BssType = BSS_INFRA;
			}
			else
			{
			  Status = -EINVAL;
			}
			pAd->CommonCfg.WpaState = SS_NOTUSE;

7.2 Parameters for iwpriv

Please refer section 3 to have iwpriv parameters and values.

Parameters:

```
int     socket_id;
char    name[25];                   // interface name
char    data[255];                  // command string
struct  iwreq wrq;
```

Default setting:

```
wrq.ifr_name = name = "ra0";        // interface name
wrq.u.data.pointer = data;          // data buffer of command string
wrq.u.data.length = strlen(data);   // length of command string
wrq.u.data.flags = 0;
```

Data Structure:

Please refer to "./include/oid.h" for update and detail definition.

7.2.1 Set Data, Parameters is Same as iwpriv

Command and IOCTL Function		
Set Data		
Function Type	**Command**	**IOCTL**

RTPRIV_IOCTL_SET	iwpriv ra0 **set SSID=RT2870AP**	sprintf(name, "ra0");
		strcpy(data, "**SSID=RT2870AP**");
		strcpy(wrq.ifr_name, name);
		wrq.u.data.length = strlen(data);
		wrq.u.data.pointer = data;
		wrq.u.data.flags = 0;
		ioctl(socket_id, RTPRIV_IOCTL_SET, &wrq);

7.2.2 Get Data, Parameters is Same as iwpriv

Command and IOCTL Function		
Get Data		
Function Type	**Command**	**IOCTL**

RTPRIV_IOCTL_STATISTICS	Iwpriv ra0 **stat**	sprintf(name, "ra0");
		strcpy(data, "**stat**");
		strcpy(wrq.ifr_name, name);
		wrq.u.data.length = strlen(data);
		wrq.u.data.pointer = data;
		wrq.u.data.flags = 0;
		ioctl(socket_id, RTPRIV_IOCTL_STATISTICS, &wrq);

RTPRIV_IOCTL_GSITESURVEY	Iwpriv ra0 **get_site_survey**	sprintf(name, "ra0");
		strcpy(data, "**get_site_survey**");
		strcpy(wrq.ifr_name, name);
		wrq.u.data.length = strlen(data);
		wrq.u.data.pointer = data;
		wrq.u.data.flags = 0;
		ioctl(socket_id, RTPRIV_IOCTL_GSITESURVEY, &wrq);

7.2.3 Set Raw Data with Flags

IOCTL Function	
Set Raw Data by I/O Control Interface with Flags	
Function Type	**IOCTL**

RT_OID_802_11_COUNTRY_REGION	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(UCHAR));
	wrq.u.data.length = sizeof(UCHAR);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_COUNTRY_REGION;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_BSSID_LIST_SCAN	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	wrq.u.data.length = 0;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_BSSID_LIST_SCAN;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_SSID	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_SSID));
	wrq.u.data.length = sizeof(NDIS_802_11_SSID);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_SSID;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_BSSID	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_MAC_ADDRESS));
	wrq.u.data.length = sizeof(NDIS_802_11_MAC_ADDRESS);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_BSSID;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_RADIO	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(BOOLEAN));
	wrq.u.data.length = sizeof(BOOLEAN);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_RADIO;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_PHY_MODE	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(RT_802_11_PHY_MODE));
	wrq.u.data.length = sizeof(RT_802_11_PHY_MODE);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_PHY_MODE;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_STA_CONFIG	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(RT_802_11_STA_CONFIG));
	wrq.u.data.length = sizeof(RT_802_11_STA_CONFIG);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_STA_CONFIG;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_DESIRED_RATES	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_RATES));
	wrq.u.data.length = sizeof(NDIS_802_11_RATES);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_DESIRED_RATES;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_PREAMBLE	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(RT_802_11_PREAMBLE));
	wrq.u.data.length = sizeof(RT_802_11_PREAMBLE);

	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_OID_802_11_PREAMBLE**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_WEP_STATUS	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_WEP_STATUS));
	wrq.u.data.length = sizeof(NDIS_802_11_WEP_STATUS);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_WEP_STATUS**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_AUTHENTICATION_MODE	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_AUTHENTICATION_MODE));
	wrq.u.data.length = sizeof(NDIS_802_11_AUTHENTICATION_MODE);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_AUTHENTICATION_MODE**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_INFRASTRUCTURE_MODE	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_NETWORK_INFRASTRUCTURE));
	wrq.u.data.length =
	sizeof(NDIS_802_11_NETWORK_INFRASTRUCTURE);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_INFRASTRUCTURE_MODE**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_REMOVE_WEP	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_KEY_INDEX));
	wrq.u.data.length = sizeof(NDIS_802_11_KEY_INDEX);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_REMOVE_WEP**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_802_11_RESET_COUNTERS	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	wrq.u.data.length = 0;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_OID_802_11_RESET_COUNTERS**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_RTS_THRESHOLD	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_RTS_THRESHOLD));
	wrq.u.data.length = sizeof(NDIS_802_11_RTS_THRESHOLD);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_RTS_THRESHOLD**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_FRAGMENTATION_THRESHOLD	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0,
	sizeof(NDIS_802_11_FRAGMENTATION_THRESHOLD));
	wrq.u.data.length =
	sizeof(NDIS_802_11_FRAGMENTATION_THRESHOLD);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_FRAGMENTATION_THRESHOLD**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_POWER_MODE	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_POWER_MODE));
	wrq.u.data.length = sizeof(NDIS_802_11_POWER_MODE);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_POWER_MODE**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_TX_POWER_LEVEL	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_TX_POWER_LEVEL));
	wrq.u.data.length = sizeof(NDIS_802_11_TX_POWER_LEVEL);
	wrq.u.data.pointer = data;

	wrq.u.data.flags = OID_802_11_TX_POWER_LEVEL; ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_TX_POWER_LEVEL_1	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ULONG));
	wrq.u.data.length = sizeof(ULONG);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_TX_POWER_LEVEL_1;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_NETWORK_TYPE_IN_USE	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_NETWORK_TYPE));
	wrq.u.data.length = / sizeof(NDIS_802_11_NETWORK_TYPE);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_NETWORK_TYPE_IN_USE;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_RX_ANTENNA_SELECTED	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_ANTENNA));
	wrq.u.data.length = sizeof(NDIS_802_11_ANTENNA);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_RX_ANTENNA_SELECTED;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_TX_ANTENNA_SELECTED	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_ANTENNA));
	wrq.u.data.length = sizeof(NDIS_802_11_ANTENNA);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_TX_ANTENNA_SELECTED;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_ADD_WPA	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, 32);
	wrq.u.data.length = 32;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_ADD_WPA;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_REMOVE_KEY	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_REMOVE_KEY));
	wrq.u.data.length = sizeof(NDIS_802_11_REMOVE_KEY);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_REMOVE_KEY;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_ADD_KEY	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, keylength); //5,10,13,26
	wrq.u.data.length = keylength L;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_ADD_KEY;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_SET_IEEE8021X	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(BOOLEAN));
	wrq.u.data.length = sizeof(BOOLEAN);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_SET_IEEE8021X;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_SET_IEEE8021X_REQUIRE_KEY	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(BOOLEAN));
	wrq.u.data.length = sizeof(BOOLEAN);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_SET_IEEE8021X_REQUIRE_KEY;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_ADD_WEP	printf(name, "ra0");

	strcpy(wrq.ifr_name, name);
	memset(data, 0, keylength); //5,10,13,26
	wrq.u.data.length = keylength;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_OID_802_11_RADIO**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_CONFIGURATION	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_CONFIGURATION));
	wrq.u.data.length = sizeof(NDIS_802_11_CONFIGURATION);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_CONFIGURATION**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_SET_COUNTERMEASURES	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	wrq.u.data.length = 0;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_SET_COUNTERMEASURES**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_DISASSOCIATE	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	wrq.u.data.length = 0;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_DISASSOCIATE**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_PMKID	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	wrq.u.data.length = keylength; //follow your setting
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_PMKID**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_WPA_SUPPLICANT_SUPPORT	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(BOOLEAN));
	wrq.u.data.length = sizeof(BOOLEAN);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_OID_WPA_SUPPLICANT_SUPPORT**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_WPA_SUPPLICANT_SUPPORT	printf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ULONG));
	wrq.u.data.length = sizeof(ULONG);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_OID_WPA_SUPPLICANT_SUPPORT**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_SET_DEL_MAC_ENTRY	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0xdd, 6);
	strcpy(wrq.ifr_name, name);
	wrq.u.data.length = 6;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_SET_DEL_MAC_ENTRY**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_802_11_SET_HT_PHYMODE | OID_GET_SET_TOGGLE	typedef struct {
	RT_802_11_PHY_MODE PhyMode;
	UCHAR TransmitNo;
	UCHAR HtMode; //HTMODE_GF or HTMODE_MM
	UCHAR ExtOffset; //extension channel above or below
	UCHAR MCS;
	UCHAR BW;
	UCHAR STBC;
	UCHAR SHORTGI;
	UCHAR rsv;
	} OID_SET_HT_PHYMODE ;

	RT_802_11_PHY_MODE tmp_ht_mode;
	sprintf(wrq.ifr_name, "ra0");
	wrq.u.data.pointer = (caddr_t) & tmp_ht_mode;
	wrq.u.data.length = sizeof(RT_802_11_PHY_MODE);
	wrq.u.data.flags = **RT_OID_802_11_SET_HT_PHYMODE | OID_GET_SET_TOGGLE**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

7.2.4 Get Raw Data with Flags

IOCTL Function	
Get Raw Data by I/O Control Interface with Flags	
Function Type	**IOCTL**

RT_OID_DEVICE_NAME	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, 255);
	wrq.u.data.length = 255;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_DEVICE_NAME;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_VERSION_INFO	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(RT_VERSION_INFO));
	wrq.u.data.length = sizeof(RT_VERSION_INFO);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_VERSION_INFO;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_BSSID_LIST	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, BssLen);
	wrq.u.data.length = BssLen;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_BSSID_LIST;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_3_CURRENT_ADDRESS	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(CurrentAddress));
	wrq.u.data.length = sizeof(CurrentAddress);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_3_CURRENT_ADDRESS;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_GEN_MEDIA_CONNECT_STATUS	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_MEDIA_STATE));
	wrq.u.data.length = sizeof(NDIS_MEDIA_STATE);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_GEN_MEDIA_CONNECT_STATUS;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_BSSID	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_MAC_ADDRESS));
	wrq.u.data.length = sizeof(NDIS_802_11_MAC_ADDRESS);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_BSSID;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_SSID	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_SSID));
	wrq.u.data.length = sizeof(NDIS_802_11_SSID);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_SSID;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_QUERY_LINK_STATUS	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(RT_802_11_LINK_STATUS));
	wrq.u.data.length = sizeof(RT_802_11_LINK_STATUS);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_QUERY_LINK_STATUS;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_CONFIGURATION	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_CONFIGURATION));

	wrq.u.data.length = sizeof(NDIS_802_11_CONFIGURATION);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_CONFIGURATION**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_RSSI_TRIGGER	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ulInfo));
	wrq.u.data.length = sizeof(ulInfo);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_RSSI_TRIGGER**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_802_11_RSSI	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ulInfo));
	wrq.u.data.length = sizeof(ulInfo);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_OID_802_11_RSSI**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_802_11_RSSI_1	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ulInfo));
	wrq.u.data.length = sizeof(ulInfo);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_OID_802_11_RSSI_1**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_802_11_RSSI_2	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ulInfo));
	wrq.u.data.length = sizeof(ulInfo);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_OID_802_11_RSSI_2**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_STATISTICS	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(NDIS_802_11_STATISTICS));
	wrq.u.data.length = sizeof(NDIS_802_11_STATISTICS);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_STATISTICS**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_GEN_RCV_OK	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ulInfo));
	wrq.u.data.length = sizeof(ulInfo);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_GEN_RCV_OK**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_GEN_RCV_NO_BUFFER	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ulInfo));
	wrq.u.data.length = sizeof(ulInfo);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_GEN_RCV_NO_BUFFER**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_802_11_PHY_MODE	typedef enum _RT_802_11_PHY_MODE { PHY_11BG_MIXED = 0, PHY_11B, PHY_11A, PHY_11ABG_MIXED, PHY_11G, PHY_11ABGN_MIXED, // both band 5 PHY_11N, // 6 PHY_11GN_MIXED, // 2.4G band 7 PHY_11AN_MIXED, // 5G band 8 PHY_11BGN_MIXED, // if check 802.11b. 9 PHY_11AGN_MIXED, // if check 802.11b. 10 } RT_802_11_PHY_MODE

	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ulInfo));
	wrq.u.data.length = sizeof(ulInfo);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_OID_802_11_PHY_MODE**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_802_11_STA_CONFIG	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(RT_802_11_STA_CONFIG));
	wrq.u.data.length = sizeof(RT_802_11_STA_CONFIG);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_OID_802_11_STA_CONFIG**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_RTS_THRESHOLD	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(RtsThresh));
	wrq.u.data.length = sizeof(RtsThresh);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_RTS_THRESHOLD**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_FRAGMENTATION_THRESHOLD	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(FragThresh));
	wrq.u.data.length = sizeof(FragThresh);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_FRAGMENTATION_THRESHOLD**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_POWER_MODE	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(PowerMode));
	wrq.u.data.length = sizeof(PowerMode);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_POWER_MODE**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_802_11_RADIO	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(RadioState));
	wrq.u.data.length = sizeof(RadioState);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_OID_802_11_RADIO**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_INFRASTRUCTURE_MODE	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(BssType));
	wrq.u.data.length = sizeof(BssType);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_INFRASTRUCTURE_MODE**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_802_11_PREAMBLE	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(PreamType));
	wrq.u.data.length = sizeof(PreamType);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_OID_802_11_PREAMBLE**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_AUTHENTICATION_MODE	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(AuthMode));
	wrq.u.data.length = sizeof(AuthMode);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **OID_802_11_AUTHENTICATION_MODE**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

OID_802_11_WEP_STATUS	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(WepStatus));

	wrq.u.data.length = sizeof(WepStatus);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_WEP_STATUS;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_TX_POWER_LEVEL	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ULONG));
	wrq.u.data.length = sizeof(ULONG);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_TX_POWER_LEVEL;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_TX_POWER_LEVEL_1	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ULONG));
	wrq.u.data.length = sizeof(ULONG);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_TX_POWER_LEVEL_1;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_NETWORK_TYPES_SUPPORTED	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, 16);
	wrq.u.data.length = 16;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_NETWORK_TYPES_SUPPORTED;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_NETWORK_TYPE_IN_USE	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ULONG));
	wrq.u.data.length = sizeof(ULONG);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_NETWORK_TYPE_IN_USE;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_QUERY_EEPROM_VERSION	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ULONG));
	wrq.u.data.length = sizeof(ULONG);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_QUERY_EEPROM_VERSION;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_QUERY_FIRMWARE_VERSION	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ULONG));
	wrq.u.data.length = sizeof(ULONG);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_QUERY_FIRMWARE_VERSION;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_QUERY_NOISE_LEVEL	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(UCHAR));
	wrq.u.data.length = sizeof(UCHAR);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_QUERY_NOISE_LEVEL;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_EXTRA_INFO	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ULONG));
	wrq.u.data.length = sizeof(ULONG);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_EXTRA_INFO;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_QUERY_PIDVID	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(ULONG));
	wrq.u.data.length = sizeof(ULONG);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_QUERY_PIDVID;

	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_WE_VERSION_COMPILED	sprintf(name, "ra0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(UINT));
	wrq.u.data.length = sizeof(UINT);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = **RT_OID_WE_VERSION_COMPILED**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_802_11_QUERY_LAST_TX_RATE	HTTRANSMIT_SETTING tmpHT;
	sprintf(wrq.ifr_name, "ra0");
	wrq.u.data.pointer = (caddr_t) & tmpHT;
	wrq.u.data.flags = **RT_OID_802_11_QUERY_LAST_TX_RATE**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

RT_OID_802_11_QUERY_LAST_RX_RATE	HTTRANSMIT_SETTING tmpHT;
	sprintf(wrq.ifr_name, "ra0");
	wrq.u.data.pointer = (caddr_t) & tmpHT;
	wrq.u.data.flags = **RT_OID_802_11_QUERY_LAST_RX_RATE**;
	ioctl(socket_id, **RT_PRIV_IOCTL**, &wrq);

SHOW_CONN_STATUS	u_char buffer[IW_PRIV_SIZE_MASK];
	sprintf(wrq.ifr_name, "ra0");
	wrq.u.data.pointer = (caddr_t) buffer;
	wrq.u.data.flags = **SHOW_CONN_STATUS**;
	ioctl(socket_id, **RTPRIV_IOCTL_SHOW**, &wrq);

7.2.5 Set Raw Data with Flags

IOCTL Function	
Get Raw Data by I/O Control Interface with Flags	
Function Type	**IOCTL**

RT_OID_802_11_SET_HT_PHYMODE | OID_GET_SET_TOGGLE	typedef struct {
	RT_802_11_PHY_MODE PhyMode;
	UCHAR TransmitNo;
	UCHAR HtMode; //HTMODE_GF or HTMODE_MM
	UCHAR ExtOffset; //extension channel above or below
	UCHAR MCS;
	UCHAR BW;
	UCHAR STBC;
	UCHAR SHORTGI;
	UCHAR rsv;
	} OID_SET_HT_PHYMODE ;
	
	RT_802_11_PHY_MODE tmp_ht_mode;
	sprintf(wrq.ifr_name, "ra0");
	wrq.u.data.pointer = (caddr_t) & tmp_ht_mode;
	wrq.u.data.length = sizeof(RT_802_11_PHY_MODE);
	wrq.u.data.flags = RT_OID_802_11_SET_HT_PHYMODE | OID_GET_SET_TOGGLE;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

8 IOCTL How To

8.1 Get Data

8.1.1 GET station connection status:

Linux console command: iwpriv ra0 connStatus

sample code =>

```
u_char buffer[IW_PRIV_SIZE_MASK];
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) buffer;
wrq.u.data.flags = SHOW_CONN_STATUS;
ioctl(socket_id, RTPRIV_IOCTL_SHOW, &wrq);
```

8.1.2 GET station statistics information:

Linux console command: iwpriv ra0 stat

sample code =>

```
u_char buffer[IW_PRIV_SIZE_MASK];
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) buffer;
wrq.u.data.flags = 0;
ioctl(socket_id, RTPRIV_IOCTL_STATISTICS, &wrq);
```

8.1.3 GET AP list table:

Linux console command: iwpriv ra0 get_site_survey

sample code =>

```
u_char buffer[4096];
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) buffer;
wrq.u.data.flags = 0;
ioctl(socket_id, RTPRIV_IOCTL_GSITESURVEY, &wrq);
```

8.1.4 GET scan table:

sample code =>

```
u_char buffer[4096];
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) buffer;
wrq.u.data.length = 4096;
wrq.u.data.flags = OID_802_11_BSSID_LIST;
ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
PNDIS_802_11_BSSID_LIST_EX          pBssidList = (PNDIS_802_11_BSSID_LIST_EX) buffer ;
```

8.1.5 GET station's MAC:

sample code =>

```
u_char buffer[6];
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) buffer;
wrq.u.data.flags = OID_802_3_CURRENT_ADDRESS;
ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.1.6 GET station connection status:

Sample code =>

```
#define NdisMediaStateConnected      1
#define NdisMediaStateDisconnected   0
NDIS_MEDIA_STATE MediaState;
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) & MediaState;
wrq.u.data.flags = OID_GEN_MEDIA_CONNECT_STATUS;
```

```
ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.1.7 GET AP's BSSID

Sample code =>

```
char BSSID[6];
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) BSSID;
wrq.u.data.flags = OID_802_11_BSSID;
ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.1.8 GET SSID

Sample code =>

```
NDIS_802_11_SSID SSID;
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) &SSID;
wrq.u.data.flags = OID_802_11_SSID;
ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.1.9 GET station's last TX related information:

Sample code =>

```
HTTRANSMIT_SETTING tmpHT;
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) & tmpHT;
wrq.u.data.flags = RT_OID_802_11_QUERY_LAST_TX_RATE;
ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.1.10 GET station's last RX related information:

Sample code =>

```
HTTRANSMIT_SETTING tmpHT;
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) & tmpHT;
wrq.u.data.flags = RT_OID_802_11_QUERY_LAST_RX_RATE;
ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.1.11 GET station's wireless mode:

Sample code =>

```
typedef enum _RT_802_11_PHY_MODE {
    PHY_11BG_MIXED = 0,
    PHY_11B,
    PHY_11A,
    PHY_11ABG_MIXED,
    PHY_11G,
    PHY_11ABGN_MIXED,       // both band        5
    PHY_11N,                //                  6
    PHY_11GN_MIXED,         // 2.4G band        7
    PHY_11AN_MIXED,         // 5G  band         8
    PHY_11BGN_MIXED,        // if check 802.11b. 9
    PHY_11AGN_MIXED,        // if check 802.11b. 10
} RT_802_11_PHY_MODE

unsigned long tmp_mode;
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) & tmp_mode;
wrq.u.data.flags = RT_OID_802_11_PHY_MODE;
ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.1.12 GET Bss type:

Sample code =>

```
typedef enum _NDIS_802_11_NETWORK_INFRASTRUCTURE
{
```

```
        Ndis802_11IBSS,
        Ndis802_11Infrastructure,
        Ndis802_11AutoUnknown,
        Ndis802_11Monitor,
        Ndis802_11InfrastructureMax     // Not a real value, defined as upper bound
    } NDIS_802_11_NETWORK_INFRASTRUCTURE

    NDIS_802_11_NETWORK_INFRASTRUCTURE  BssType;
    sprintf(wrq.ifr_name, "ra0");
    wrq.u.data.pointer = (caddr_t) & BssType;
    wrq.u.data.flags = OID_802_11_INFRASTRUCTURE_MODE;
    ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.1.13 GET Authentication Mode:

Sample code =>

```
    typedef enum _NDIS_802_11_AUTHENTICATION_MODE
    {
        Ndis802_11AuthModeOpen,
        Ndis802_11AuthModeShared,
        Ndis802_11AuthModeAutoSwitch,
        Ndis802_11AuthModeWPA,
        Ndis802_11AuthModeWPAPSK,
        Ndis802_11AuthModeWPANone,
        Ndis802_11AuthModeWPA2,
        Ndis802_11AuthModeWPA2PSK,
        Ndis802_11AuthModeWPA1WPA2,
        Ndis802_11AuthModeWPA1PSKWPA2PSK,
        Ndis802_11AuthModeMax         // Not a real mode, defined as upper bound
    } NDIS_802_11_AUTHENTICATION_MODE

    NDIS_802_11_AUTHENTICATION_MODE     AuthMode;
    sprintf(wrq.ifr_name, "ra0");
    wrq.u.data.pointer = (caddr_t) & AuthMode;
    wrq.u.data.flags = OID_802_11_AUTHENTICATION_MODE;
    ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.1.14 GET Encryption Type:

Sample code =>

```
    typedef enum _NDIS_802_11_WEP_STATUS
    {
        Ndis802_11WEPEnabled,
        Ndis802_11Encryption1Enabled = Ndis802_11WEPEnabled,
        Ndis802_11WEPDisabled,
        Ndis802_11EncryptionDisabled = Ndis802_11WEPDisabled,
        Ndis802_11WEPKeyAbsent,
        Ndis802_11Encryption1KeyAbsent = Ndis802_11WEPKeyAbsent,
        Ndis802_11WEPNotSupported,
        Ndis802_11EncryptionNotSupported = Ndis802_11WEPNotSupported,
        Ndis802_11Encryption2Enabled,
        Ndis802_11Encryption2KeyAbsent,
        Ndis802_11Encryption3Enabled,
        Ndis802_11Encryption3KeyAbsent,
        Ndis802_11Encryption4Enabled,  // TKIP or AES mix
        Ndis802_11Encryption4KeyAbsent,
    } NDIS_802_11_WEP_STATUS, *PNDIS_802_11_WEP_STATUS,

    NDIS_802_11_WEP_STATUS    WepStatus;
    sprintf(wrq.ifr_name, "ra0");
    wrq.u.data.pointer = (caddr_t) & WepStatus;
    wrq.u.data.flags = OID_802_11_WEP_STATUS;
    ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.1.15 GET RSSI 0 (unit: db)

Sample code =>

```
long rssi_0
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) & rssi_0;
wrq.u.data.flags = RT_OID_802_11_RSSI;
ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.1.16 GET RSSI 1 (unit: db)

Sample code =>

```
long rssi_1
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) & rssi_1;
wrq.u.data.flags = RT_OID_802_11_RSSI_1;
ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.1.17 GET RSSI 2 (unit: db)

Sample code =>

```
long rssi_2
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) & rssi_2;
wrq.u.data.flags = RT_OID_802_11_RSSI_2;
ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.1.18 GET Driver wireless extension version

Sample code =>

```
Unsigned int wext_version;
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) & wext_version;
wrq.u.data.flags = RT_OID_WE_VERSION_COMPILED;
ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```

8.2 How to display rate, BW:

```
HTTRANSMIT_SETTING HTSetting;
Double Rate;
double b_mode[] ={1, 2, 5.5, 11};
float g_Rate[] = { 6,9,12,18,24,36,48,54};
switch(HTSetting.field.MODE)
{
        case 0:
                if (HTSetting.field.MCS >=0 && HTSetting.field.MCS<=3)
                        Rate = b_mode[HTSetting.field.MCS];
                else if (HTSetting.field.MCS >=8 && HTSetting.field.MCS<=11)
                        Rate = b_mode[HTSetting.field.MCS-8];
                else
                        Rate = 0;
                break;
        case 1:
                if ((HTSetting.field.MCS >= 0) && (HTSetting.field.MCS < 8))
                        Rate = g_Rate[HTSetting.field.MCS];
                else
                        Rate = 0;
                break;
        case 2:
        case 3:
                if (0 == bGetHTTxRateByBW_GI_MCS(HTSetting.field.BW, HTSetting.field.ShortGI,
                                        HTSetting.field.MCS,
                                        &Rate))
                Rate = 0;
                break;
        default:
                Rate = 0;
                break;
}

char bGetHTTxRateByBW_GI_MCS(int nBW, int nGI, int nMCS, double* dRate)
{
        double HTTxRate20_800[16]={6.5, 13.0, 19.5, 26.0, 39.0, 52.0, 58.5, 65.0, 13.0, 26.0, 39.0, 52.0, 78.0, 104.0, 117.0,
        130.0};
        double HTTxRate20_400[16]={7.2, 14.4, 21.7, 28.9, 43.3, 57.8, 65.0, 72.2, 14.444, 28.889, 43.333, 57.778, 86.667,
        115.556, 130.000, 144.444};
        double HTTxRate40_800[18]={13.5, 27.0, 40.5, 54.0, 81.0, 108.0, 121.5, 135.0, 27.0, 54.0, 81.0, 108.0, 162.0, 216.0,
        243.0, 270.0, 6.0, 39.0};
        double HTTxRate40_400[18]={15.0, 30.0, 45.0, 60.0, 90.0, 120.0, 135.0, 150.0, 30.0, 60.0, 90.0, 120.0, 180.0, 240.0,
        270.0, 300.0, 6.7, 43.3};

        // no TxRate for (BW = 20, GI = 400, MCS = 32) & (BW = 20, GI = 400, MCS = 32)
        if (((nBW == BW_20) && (nGI == GI_400) && (nMCS == 32)) ||
                        ((nBW == BW_20) && (nGI == GI_800) && (nMCS == 32)))
                return 0; //false

        if( nBW == BW_20 && nGI == GI_800)
                *dRate = HTTxRate20_800[nMCS];
        else if( nBW == BW_20 && nGI == GI_400)
                *dRate = HTTxRate20_400[nMCS];
        else if( nBW == BW_40 && nGI == GI_800)
                *dRate = HTTxRate40_800[nMCS];
        else if( nBW == BW_40 && nGI == GI_400)
                *dRate = HTTxRate40_400[nMCS];
        else
                return 0; //false

        return 1; //true
}
```

8.3 Set Data for N mode

7.3.1. SET HT mode:

Sample code =>

```
typedef struct {
 RT_802_11_PHY_MODE      PhyMode;
 UCHAR                   TransmitNo;
 UCHAR                   HtMode;          //HTMODE_GF or HTMODE_MM
 UCHAR                   ExtOffset;       //extension channel above or below
 UCHAR                   MCS;
 UCHAR                   BW;
 UCHAR                   STBC;
 UCHAR                   SHORTGI;
 UCHAR                   rsv;
} OID_SET_HT_PHYMODE ;

RT_802_11_PHY_MODE tmp_ht_mode;
sprintf(wrq.ifr_name, "ra0");
wrq.u.data.pointer = (caddr_t) & tmp_ht_mode;
wrq.u.data.length = sizeof(RT_802_11_PHY_MODE);
wrq.u.data.flags = RT_OID_802_11_SET_HT_PHYMODE | OID_GET_SET_TOGGLE;
ioctl(socket_id, RT_PRIV_IOCTL, &wrq);
```