IC WLAN USB Module

IEEE 802.11b, IEEE 802.11g ,IEEE 802.11n

1 Specification

- **Operating voltage：** 3.3 V +/- 10%
- **Power consumption：** < 500 mA Tx (All mode), < 500 mA (11b Rx mode)
- **Storage temperature：** -55~+150 ℃
- **Operation temperature：** -10~+85 ℃
- **Transmission rate：**

IEEE 802.11b: 1,2, 5.5 and 11 Mbps / IEEE 802.11g: 6,9,12,18,24,36,48,54 Mbps

IEEE 802.11n HT 20MHz: 6.50,13.00,19.50,26.00,39.00,52.00,58.50,65.00 Mbps

IEEE 802.11 HT 40MHz: 13.50,27.00,40.50,54.00,81.00,108.0,121.5,135.0 Mbps

- **Dimensions：**11.6 ±0.2mm(W) x 16.9±0.2 mm(L) x 1.4 mm(H)

FCC Warning statement

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

-Reorient or relocate the receiving antenna.

-Increase the separation between the equipment and receiver.

-Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.

-Consult the dealer or an experienced radio/TV technician for help.

You are cautioned that changes or modifications not expressly approved by the party responsible for compliance could void your authority to operate the equipment.

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions:

(1) this device may not cause harmful interference and

(2) this device must accept any interference received, including interference that may cause undesired operation

2 Installation Instructions

When you install the platform, make sure that the USB port of the module is connected to the platform USB in order to ensure the normal functioning.

3 CONFIGURATION

RT3070 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 use, please refer to below sections for details.

3.1 Configuration File RT2870STA.dat

```
# Copy this file to /etc/Wireless/RT2870STA/RT2870STA.dat
# This file will be read on loading driver module.
#
# Use "vi RT2870STA.dat" to modify settings according to your need.
#
#
#The word of "Default" must not be removed
Default
CountryRegion=5
CountryRegionABand=7
CountryCode=
ChannelGeography=1
SSID=11n-AP
NetworkType=Infra
WirelessMode=5
Channel=0
BeaconPeriod=100
TxPower=100
BGProtection=0
TxPreamble=0
RTSThreshold=2347
FragThreshold=2346
TxBurst=1
PktAggregate=0
WmmCapable=1
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
AutoRoaming=0
RoamThreshold=70
APSDCapable=0
APSDAC=0;0;0;0
HT_RDG=1
HT_EXTCHA=0
HT_OpMode=1
HT_MpduDensity=4
HT_BW=1
HT_BADecline=0
HT_AutoBA=1
HT_BADecline=0
HT_AMSDU=0
HT_BAWinSize=64
HT_GI=1
HT_MCS=33
HT_MIMOPSMode=3
HT_DisallowTKIP=1
IEEE80211H=0
TGnWifiTest=0
WirelessEvent=0
CarrierDetect=0
AntDiversity=0
BeaconLostTime=4
FtSupport=1

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 Configuration file use

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

SectionNumber
Param
Value
. . .
. . .
. . .

3.2.1 CountryRegion

Value:

Region	Channels
0	1-11

1	1-13
2	10-11
3	10-13
4	14
5	1-14
6	3-9
7	5-13
31	1-14
32	1-11 active scan, 12 and 13 passive scan
33	1-14 all active scan, 14 b mode only

3.2.2 CountryRegionForABand

Value:

Region	Channels
0	36, 40, 44, 48, 52, 56, 60, 64, 149, 153, 157, 161, 165
1	36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140
2	36, 40, 44, 48, 52, 56, 60, 64
3	52, 56, 60, 64, 149, 153, 157, 161
4	149, 153, 157, 161, 165
5	149, 153, 157, 161
6	36, 40, 44, 48
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
8	52, 56, 60, 64
9	36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 132, 136, 140, 149, 153, 157, 161, 165
10	36, 40, 44, 48, 149, 153, 157, 161, 165
11	36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 120, 149, 153, 157, 161

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
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
11: 11N only in 5G band only

3.2.5 Channel

Value:
Depends on CountryRegion or CountryRegionForABand

3.2.6 HwAntDiv

Value:
0: Disable
1: HW RX antenna diversity
2: Fixed RX at AUX ANT
3: Fixed RX at main ANT

3.2.7 BGProtection

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

3.2.8 TxPreamble

Value:
0:Preamble Long
1:Preamble Short
2:Auto

3.2.9 RTSThreshold

Value:
1~2347

3.2.10 FragThreshold

Value:
256~2346

3.2.11 TxBurst

Value:
0: Disable
1: Enable

3.2.12 PktAggregate

Value:

0: Disable

1: Enable

3.2.13 NetworkType

Value:

Infra: infrastructure mode

Adhoc: adhoc mode

3.2.14 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	

3.2.15 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

3.2.16 DefaultKeyID

Value:

1~4

3.2.17 WEP KeyType

Key1Type=value

Key2Type=value

Key3Type=value

Key4Type=value

Value:

0 hexadecimal type

1 assic type

(use: reading profile only)

3.2.18 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
(use: "iwpriv" only)

3.2.19 WEP Key String

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

Value:

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

3.2.20 WPAPSK

Value:

8~63 ASCII or
64 HEX characters

3.2.21 WmmCapable

Value:

0: Disable WMM
1: Enable WMM

3.2.22 IEEE80211H

Enabel IEEE802.11h support

Value:

0:Disable
1:Enable

3.2.23 PSMode

Value:

CAM	Constantly Awake Mode
Max_PSP	Max Power Saving
Fast_PSP	Fast Power Saving
Legacy_PSP	Legacy Power Saving

3.2.24 FastRoaming

Value:

0: Disabled
1: Enabled

3.2.25 RoamThreshold

Value:

0 ~ 255

3.2.26 TGnWifiTest

Value:

0: Disabled
1: Enabled

3.2.27 WirelessEvent

Value:

0: Disabled
1: Enabled (send custom wireless event)

3.2.28 CarrierDetect

Value:

0: Disabled
1: Enabled

3.2.29 HT_RDG

Value:

0: Disabled
1: Enabled

3.2.30 HT_EXTCHA

Value:

0: Below
1: Above

3.2.31 HT_OpMode

Value:

0: HT mixed format
1: HT greenfield format
(Note) If you want to do TGn WIFI green field item, please set HT_OpMode=1

3.2.32 HT_MpduDensity

Value:

0 ~ 7

3.2.33 HT_BW

Value:

0: 20MHz
1: 40MHz

3.2.34 HT_AutoBA

Value:

0: Disabled
1: Enabled

3.2.35 HT_AMSDU

Value:

0: Disabled
1: Enabled

3.2.36 HT_BAWinSize

Value:

1 ~ 64

3.2.37 HT_GI

Value:

0: long GI
1: short GI

3.2.38 HT_MCS

Value:

0 ~ 15
33: auto

3.2.39 HT_MIMOPSEnable

Enable/Disable the 802.11n SM power save function.

Value:

0:Disable
1:Enable (Default)

3.2.40 HT_MIMOPSMode

Value:

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

3.2.41 HT_DisallowTKIP

Enable/Disable N rate with 11N ap when cipher is WEP or TKIP.

Value:

0: FALSE
1: TRUE

Default setting is disable.

3.2.42 HT_RxStream

Set the number of spatial streams for reception

Value:

1: 1 Rx stream
2: 2 Rx stream

3.2.43 HT_TxStream

Set the number of spatial streams for transimtion

Value:

1: 1 Tx stream
2: 2 Tx stream

3.2.44 HT_LinkAdapt

Enable/Disable HT Link Adaptation Control

Value:

0:Disable (Default)
1:Enable

3.2.45 HT_HTC

Enable/disable HTC field of data frames send with 802.11n data rates

Value:

0:Disable (Default)
1:Enable

3.2.46 HT_DisableReordering

Disable AMPDU re-ordering handling mechanism

Value:

0:Disable (Default)
1:Enable

3.2.47 BeaconLostTime

Change Beacon Lost Time

Value:

1 ~ 60 seconds
Default value is 4 seconds

3.2.48 AutoRoaming

Enable/disable auto roaming mechanism

Value:

0: disable
1: enable
Default setting is disable.

3.2.49 MacAddress

MacAddress=value

Value:

XX:XX:XX:XX:XX:XX

3.2.50 TDLSCapable

Enable/disable TDLS Capable function

Value:

0: disable
1: enable

3.2.51 AutoConnect

Enable/Disable driver connect to ANY AP when SSID is null.

Value:

0: disable (default)
1: enable

3.2.52 HT_40MHZ_INTOLERANT

Set to disable the 40MHz channel bandwidth operation and also indicate other 20/40MHz BSS Coex awared

Value:

0:Disable (default)
1:Enable

3.2.53 AntGain

Define peak antenna gain (dBi) for Single SKU setting.

Value:

0: Disable Single SKU TxPower Adjustment.
1~255: Enable Single SKU TxPower Adjustment.

3.2.54 BandedgeDelta

Define delta conducted power value which can pass bandeage of FCC certification at Ch1 and Ch11 (dBm) within HT_40 Bandwidth for Single SKU setting.

Value:

1~255: Delta value between HT_20 and HT_40 power value.

3.2.55 P2P_GOIntent

Relative value between 0 and 15 used to indicate the desire of the P2P device to be the P2P Group Owner, with a larger value indicating a higher desire.

Value:

0~15: GO Intent.

4 WIRELESS TOOLS

4.1 Iwpriv ra0 set use

This section describes parameters set using iwpriv. Please refer to the Readme section for more general data.

iwpriv ra0 set [parameters]=[Value]

Note: Execute one iwpriv/set command at a time.

4.1.1 DriverVersion

Check driver version by issue iwpriv set command.

Range:

Any value

Value:

0

Example:

#iwpriv ra0 set DriverVersion=1

4.1.2 CountryRegion

Set country region.

Range:

{0~7}

Value:

Region	Channels
0	1-11
1	1-13
2	10-11
3	10-13
4	14
5	1-14
6	3-9
7	5-13
31	1-14
32	1-11 active scan, 12 and 13 passive scan
33	1-14 all active scan, 14 b mode only

4.1.3 CountryRegionABand

Set country region for A band.

Range:

{0~9}

Value:

Region	Channels
0	36, 40, 44, 48, 52, 56, 60, 64, 149, 153, 157, 161, 165

1	36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140
2	36, 40, 44, 48, 52, 56, 60, 64
3	52, 56, 60, 64, 149, 153, 157, 161
4	149, 153, 157, 161, 165
5	149, 153, 157, 161
6	36, 40, 44, 48
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
8	52, 56, 60, 64
9	36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 132, 136, 140, 149, 153, 157, 161, 165
10	36, 40, 44, 48, 149, 153, 157, 161, 165
11	36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 120, 149, 153, 157, 161

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
11: 11N only in 5G band only

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 HwAntDiv

Setting H/W Antenna Diversity Diable or Enable

Value:

0: Disable
1: HW RX antenna diversity
2: Fixed RX at AUX ANT
3: Fixed RX at main ANT

4.1.12 BGProtection:

Set 11B/11G Protection

Range:

{0~2}

Value:

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

4.1.13 RTSThreshold:

Set RTS Threshold

Range:

{1~2347}

Value:

4.1.14 FragThreshold:

Set Fragment Threshold

Range:

{256~2346}

Value:

4.1.15 NetworkType:

Set Network type

Range:

{Infra,Adhoc}

Value:

4.1.16 AuthMode:

Set Authentication Mode

Range:

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

Value:

4.1.17 EncrypType:

Set Encryption Type

Range:

{NONE,WEP,TKIP,AES}

Value:

4.1.18 DefaultKeyID:

Set Default Key ID

Range:

{1~4}

Value:

4.1.19 Key1

Set Key1 String

Range:

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

Value:

4.1.20 Key2

Set Key2 String

Range:

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

Value:

4.1.21 Key3

Set Key3 String
Range:
{5 ascii characters or 10 hex number or
13 ascii characters or 26 hex numbers}
Value:

4.1.22 Key4

Set Key4 String
Range:
{5 ascii characters or 10 hex number or
13 ascii characters or 26 hex numbers}
Value:

4.1.23 WPAPSK

WPA Pre-Shared Key
Range:
{8~63 ascii or 64 hex characters}
Value:

4.1.24 WmmCapable

Set WMM Capable
Range:
{0,1}
Value:
0:Disable WMM,
1:Enable WMM

4.1.25 IEEE80211H

Enabel IEEE802.11h support
Range:
{0,1}
Value:
0:Disable
1:Enable

4.1.26 PSMode

Set Power Saving Mode
Range:
{CAM, MAX_PSP, FAST_PSP}
Value:

4.1.27 ResetCounter

Reset statistics counter

Range:

Any vlaue

Value:

0

4.1.28 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

4.1.29 CarrierDetect

Value

0: Disabled
1: Enabled

4.1.30 HtRdg

Enable HT Reverse Direction Grant.

Value:

0: Disabled
1: Enabled

4.1.31 HtExtcha

To locate the 40MHz channel in combination with the control.

Value:

0: Below
1: Above

4.1.32 HtOpMode

Change HT operation mode.

Value:

0: HT mixed format
1: HT greenfield format

4.1.33 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

4.1.34 HtBw

Support channel width.

Value:

0: 20MHz
1: 40MHz

4.1.35 HtAutoBa

Enable auto block acknowledgment (Block Ack).

Value:

0: Disabled
1: Enabled

4.1.36 HtAmsdu

Enable aggregation of multiple MSDUs in one MPDU.

Value:

0: Disabled
1: Enabled

4.1.37 HtBaWinSize

Set BA WinSize.

Value:

1 ~ 64

4.1.38 HtGi

Support Short/Long GI.

Value:

0: long GI
1: short GI

4.1.39 HtMcs

MCS rate selection.

Value:

0 ~ 15
33: auto

4.1.40 HtProtect

Enable HT protection for legacy device.

Value:

0: Disable

1: Enable

4.1.41 HtMimoPs

MIMO power save.

Value:

0: Disable

1: Enable

4.1.42 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

4.1.43 LongRetry

USE:

iwpriv ra0 set LongRetry=value

Value:

0~255

4.1.44 ShortRetry

USE:

iwpriv ra0 set ShortRetry=value

Value:

0~255

4.1.45 HtTxStream=value

Value:

1: Support 1-Tx Stream for MCS0 ~ MCS7

2: Support 2-Tx Stream for MCS0 ~ MCS15

4.1.46 HtRxStream=value

Value:

1: Support 1-Rx Stream for MCS0 ~ MCS7
2: Support 2-Rx Stream for MCS0 ~ MCS15

4.1.47 HtDisallowTKIP=value

Enable/Disable N rate with 11N ap when cipher is WEP or TKIP.

Value:

0: FALSE
1: TRUE
Default setting is disable.

4.1.48 HtBaDecline

Reject all Recipient's BA requests.

Value:

0: Disable (Default)
1: Enable

4.1.49 BeaconLostTime=value

Change Beacon Lost Time

Value:

1 ~ 60 seconds
Default value is 4 seconds

4.1.50 AutoRoaming=value

Enable/disable auto roaming mechanism

Value:

0: disable
1: enable
Default setting is disable.

4.1.51 SiteSurvey=value

Scan with specific SSID after link up

Value:

0~z, 1~32 ascii characters

4.1.52 TdlsCapable=value

Enable/disable TDLS capable

Value:

0: disable
1: enable
Example: iwpriv ra0 set TdlsCapable=0

4.1.53 TdlsSetup=value

Manually add TDLS link
Value: MAC address
Example: iwpriv ra0 set TdlsSetup=00:11:22:33:44:55

4.1.54 AutoReconnect=value

Description: Enable/Disable driver auto reconnect functionality
Valid Range: 0-1
Default Value: 1
0: Disable, 1: Enable

4.1.55 AdhocN=value

Description: Enable/Disable Adhoc to support N or not
Valid Range: 0-1
Default Value: 1
0: Disable, 1: Enable

4.1.56 AntGain

Define peak antenna gain (dBi) for Single SKU setting.
Value:
0: Disable Single SKU TxPower Adjustment.
1~255: Enable Single SKU TxPower Adjustment.

4.2 Iwpriv ra0 show use

This section describes parameters set using iwpriv. Please refer to the Readme section for more general data.

A detailed explanation of each parameter for iwpriv is shown subsequently. Refer to the Readme before using this section.

iwpriv ra0 show [parameters]

4.2.1 connStatus

Show STA connection Status

4.2.2 driverVer

Show STA current driver version

4.2.3 bainfo

Show STA current BA information

4.2.4 rxbulk

Show STA current rxbluk information

4.2.5 txbulk

Show STA current txbluk information

4.2.6 AutoReconnect

Show bAutoReconnect flag

4.2.7 WPAPSK

Show WPA Passphrase

4.2.8 PMK

Show PMK key

4.3 Iwpriv ra0 use

This section describes parameters set using iwpriv. Please refer to the Readme section for more general data.

iwpriv ra0 show [parameters]

4.3.1 radio_off

Turn STA radio off

4.3.2 radio_on

Turn STA radio on

4.4 Iwpriv Examples

4.4.1 Infrastructure

4.4.1.1 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"

4.4.1.2 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"

4.4.1.3 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"

4.4.1.4 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"

4.4.1.5 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"

4.4.2 Ad-Hoc

4.4.2.1 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"

4.4.2.2 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"

4.4.3 Get site survey

use:

iwpriv ra0 get_site_survey

4.4.4 Get Statistics

use:

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

4.4.5 ANY SSID

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

use:

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

4.5 iwlist

This section describes parameters set using iwlist. Please refer to the Readme section for more general data.

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

4.6 iwconfig

The subsequent settings are used in the standard iwconfig configuration

```
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: Refer to the 'iwconfig', 'iwlist' and 'iwpriv' sections for wireless extension instructions.

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.

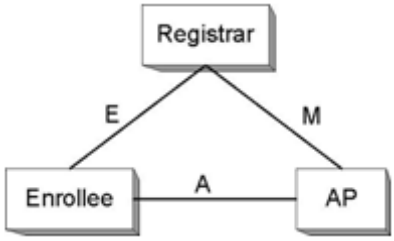


Figure 1. Components and Interfaces

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 use

This section describes parameters set using iwpriv. Please refer to the Readme section for more general data.

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_bssid

BSSID of WSC AP that STA wants to do WPS with

Value:

xx:xx:xx:xx:xx:xx

5.1.6 wsc_start

Trigger RT5370 STA driver to do WPS process.

Range:

NULL

Value:

5.1.7 wsc_stop

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

Range:

NULL

Value:

5.1.8 wsc_gen_pincode

Generate new PIN code.

Range:

NULL

Value:

5.1.9 wsc_cred_count

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

Range:

{1 ~ 8}

Value:

5.1.10 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"

5.1.11 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"

5.1.12 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"

5.1.13 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"

5.1.14 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

5.1.15 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"

5.1.16 wsc_conn_by_idx

Connect AP by credential index.

Range:

{0 ~ 7}

Value:

idx: 0 ~ 7

5.1.17 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.

5.1.18 wsc_ap_band

Setting prefer band to do WPS with dual band WPS AP.

Range:

{0, 1,2}

Value:

0: prefer 2.4G

1: prefer 5G

2: auto

Default value is auto (2)

5.1.19 Wsc4digitPinCode

Generate WPS 4-digits PIN

Value:

0: Disable

1: Enable

5.2 WPS STA as an Enrollee or Registrar

Build WPS function. Please set the “HAS_WSC” parameter value to “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) [RT5370 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) [RT5370 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) [RT5370 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) [RT5370 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 use

This section describes specific parameters and arguments. Please refer to the previous section for more general data.

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;
```

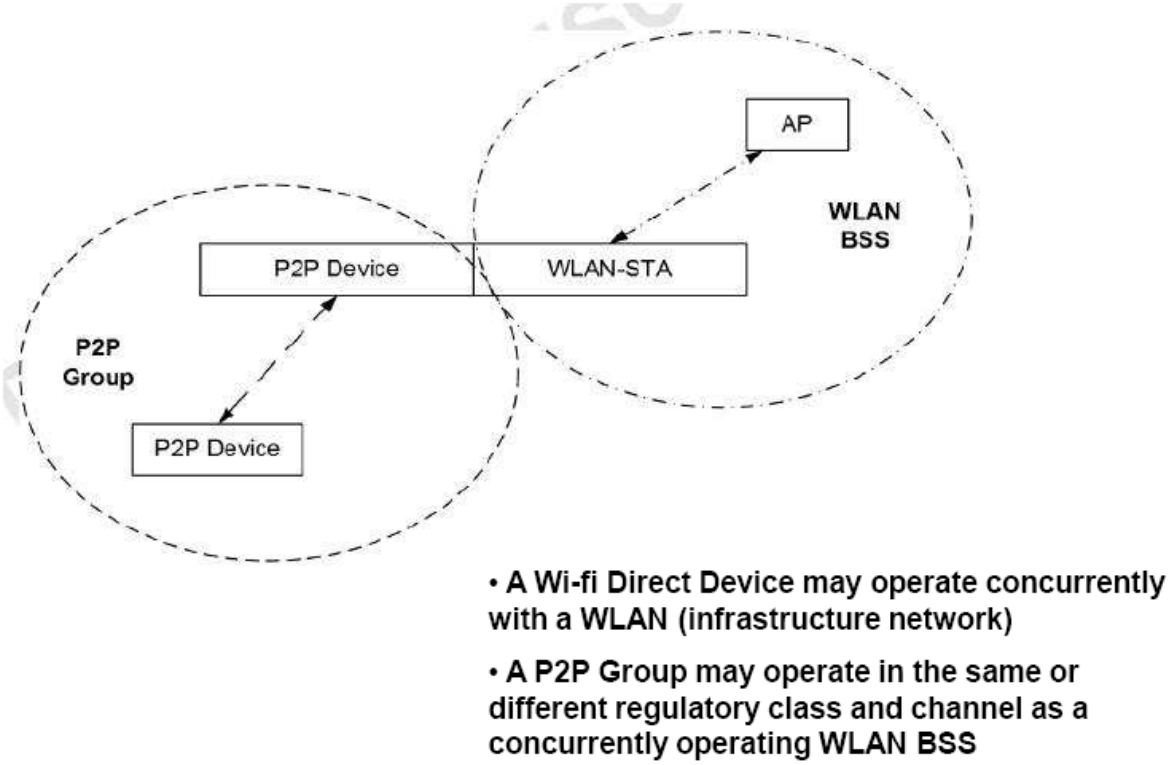
```
/* Perform the private ioctl */
if(ioctl(skfd, RTPRIV_IOCTL_SET_WSC_PROFILE_STRING_ITEM, &lwreq) < 0)
{
    fprintf(stderr, "Interface doesn't accept private ioctl...\n");
    return -1;
}
```

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 WIFI DIRECT - P2P COMMAND



Wifi direct feature Makes direct connections to one another quickly and conveniently to do things like print, sync, and share content even when an access point or router is unavailable.

6.1 Iwpriv use

6.1.1 P2pOpMode

Set p2p interface operate mode to GO.

Value:

1: Auto (Force) GO mode

Example:

#iwpriv p2p0 set P2pOpMode=1

6.1.2 p2pLisCh

Set p2p device Channel in Listen stage.

Value:

1, 6, 11 (Define in P2P spec Page 26 & 36)

Example:

#iwpriv p2p0 set P2pLisCh=x

6.1.3 p2pOpCh

Set p2p Operation Channel if negotiate as GO

Value:

Based on country region

Example:

iwpriv p2p0 set p2pOpCh=1

6.1.4 p2pGoInt

Set p2p device GO Intent value

- This value is set to nego the art for become GO or Client

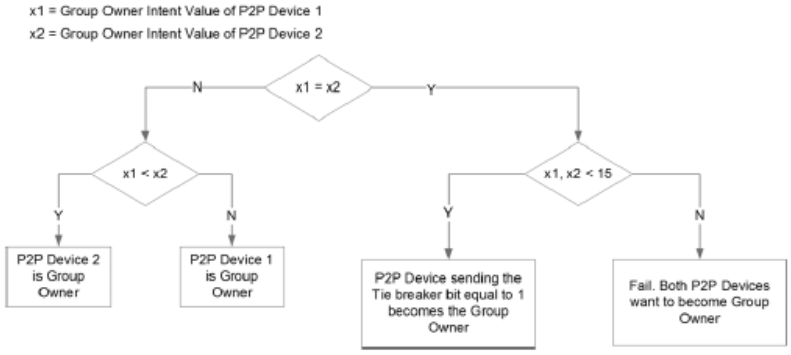


Figure 7—Group Owner determination flowchart

Value:

0~15

Example:

#iwpriv p2p0 set p2pGoInt=x

6.1.5 p2pDevName

Set p2p device display Device Name

Value:

0~Z, less than 32 characters.

Example:

#iwpriv p2p0 set p2pDevName=xxxxx

6.1.6 p2pWscMode

Set p2p device WSC Mode for P2P negotiate.

Value:

1: PIN

2: PBC

Example:

#iwpriv p2p0 set p2pWscMode=x

6.1.7 p2pWscConf

Set p2p device WSC Configure Method

Value:

1: Display

2: KeyPad

3: PBC

Example:

#iwpriv p2p0 set p2pWscConf =x

6.1.8 p2pScan

Set p2p device to start P2P Scanning.

Value:

0: Disable (Force Stop this action and cancel timer)

1: Enable (Do action)

Example:

#iwpriv p2p0 set p2pScan=x

6.1.9 p2pInv

Select p2p device ID to Invite (send p2p Invite to peer p2p device join our p2p Gorup)

Value:

0~29 (Software setting)

Example:

#iwpriv p2p0 set p2pInv =x

6.1.10 p2pDevDisc

Send p2p device discoverability to GO query specific CLIENT is in Group or not(debug use)

Value:

0~29 (Software setting)

Example:

#iwpriv p2p0 set p2pDevDisc=x

6.1.11 p2pLink

Select p2p device ID to do GO Negotiation

Value:

0~29 (Software setting)

Example:

#iwpriv p2p0 set p2pLink =x

6.1.12 p2pCfg

Dump/Show p2p configuration (In kernel background message)

Value:

Any

Example:

#iwpriv p2p0 set p2pCfg

6.1.13 p2pTab

Show Group Table status in kernel background message.

Value:

Any

Example:

#iwpriv p2p0 set p2pTab

6.1.14 p2pProv

Select p2p device ID to Provision

Value:

0~29 (Software setting)

Example:

#iwpriv p2p0 set p2pProv=x

6.1.15 p2pStat

Dump/Show p2p current rule, state machine status(In kernel background message)

Value:

Any

Example:

#iwpriv p2p0 set p2pStat

6.1.16 p2pReset

Reset p2p configuration the stat machine set to initial stage.

Value:

Any

Example:

#iwpriv p2p0 set p2pReset

6.1.17 p2pPerTab

Show p2p Persistent Table in kernel background message.

Value:

Any

Example:

#iwpriv p2p0 set p2pPerTab

6.1.18 p2pDefConfMthd

Set default WSC Config Method to Provision

Value:

1: Display

2: KeyPad

3: PBC

Example:

#iwpriv p2p0 set p2pDefConfMthd=x

6.1.19 p2pLinkDown

Tear down p2p session and change as p2p device mode.

Value:

Any

Example:

#iwpriv p2p0 set p2pLinkDown=x

6.1.20 p2pSigmaEnable

For p2p Sigma auto testing, we need enable some flag to pass WPS.

Value:

Any

Example:

#iwpriv p2p0 set p2pSigmaEnable=x

6.1.21 Other P2P command

The other command wrote in source code which is for debug only. Please ignore them.

6.1.22 P2P example:

P2P device enable as autonomous GO :

autonomous GO:

#iwpriv p2p0 set p2pOpCh=11

#iwpriv p2p0 set P2pOpMode=1

P2P device reset to default setting:

#iwpriv p2p0 set p2pReset=1

P2P device start to scan and Listen as Channel 11:

#iwpriv p2p0 set p2pLisCh=11

#iwpriv p2p0 set p2pScan=1

P2P Device Start Device Discovery.

#iwpriv p2p0 set p2pScan=1

P2P Device Stop Device Discovery.

#iwpriv p2p0 set p2pScan=0

Connect the P2P Device on Scan Table of index 0.

```
Table.Client[0]: DeviceName[R A L I N K - P C ]
                        DevCapability = 32.
                                ServiceDiscovery = 0.        P2P Client Discoverability = 0
                                ConcurrentOperation = 0.        InfraManaged = 0.
                                DeviceLimit = 0.        InvitationProcedure = 1.
                        GroupCapability = 8.
                                GroupOwner = 0.        PersistentP2PGroup = 0
                                GroupLimit = 0.        IntraBSS = 1.
                                CrossConnection = 0.        PersistentReconnect = 0.
                                GroupFormation = 0.
                        Addr[00:0c:43:21:64:81]
                        BSSID[00:0c:43:21:64:81]
                        InterfaceAddr[00:0c:43:21:64:81]
                        SSID[D I R E C T - l 2 ]
                        WscMode = PBC.        PIN = 00 00 00 00 00 00 00 00.
                        CfgMethod = PBC   PBC.        GoIntent = 6.
                        PrimDevType[00 01 00 50 f2 04 00 01] SecDevType[00 00 00 00 00 00 00 00]
                        ChNumber = 0.        OpChannel = 1.        ListenChannel = 1.
                        P2pClientState = P2PSTATE_GO_WPS.        MyGOIndex = 255.
                        P2pIP = 00000000.        P2pFlag = 2.
                        Dpid = 65535.        StateCount = 1.        Rule = I am P2P GO.
                        GeneralToken = 76.        NoAToken = 0.
                        RegClass = 0.        ConfigTimeOut = 0.
                        ExtListenPeriod = 0.        ExtListenInterval = 0.
```

PIN:

#iwpriv p2p0 set p2pWscMode=1

#iwpriv p2p0 set p2pLink=0

PBC:

#iwpriv p2p0 set p2pWscMode=2

#iwpriv p2p0 set p2pLink=0

P**2P scan and select p2p device do link:**

#iwpriv p2p0 set p2pScan=1

#sleep 10

#iwpriv p2p0 set p2pTab=1

#iwpriv p2p0 set p2pLink=2

P2P device Show P2P Scan Table:

#iwpriv p2p0 set p2pTab=1

```
Table.Client[0]: DeviceName[R A L I N K - P C ]
                        DevCapability = 32.
                                ServiceDiscovery = 0.        P2P Client Discoverability = 0
                                ConcurrentOperation = 0.        InfraManaged = 0.
                                DeviceLimit = 0.        InvitationProcedure = 1.
                        GroupCapability = 8.
                                GroupOwner = 0.        PersistentP2PGroup = 0
                                GroupLimit = 0.        IntraBSS = 1.
                                CrossConnection = 0.        PersistentReconnect = 0.
                                GroupFormation = 0.
                        Addr[00:0c:43:21:64:81]
                        BSSID[00:0c:43:21:64:81]
                        InterfaceAddr[00:0c:43:21:64:81]
                        SSID[D I R E C T - l 2 ]
                        WscMode = PBC.        PIN = 00 00 00 00 00 00 00 00.
                        CfgMethod = PBC   PBC.        GoIntent = 6.
                        PrimDevType[00 01 00 50 f2 04 00 01] SecDevType[00 00 00 00 00 00 00 00]
                        ChNumber = 0.        OpChannel = 1.        ListenChannel = 1.
                        P2pClientState = P2PSTATE_GO_WPS.        MyGOIndex = 255.
                        P2pIP = 00000000.        P2pFlag = 2.
                        Dpid = 65535.        StateCount = 1.        Rule = I am P2P GO.
                        GeneralToken = 76.        NoAToken = 0.
                        RegClass = 0.        ConfigTimeOut = 0.
                        ExtListenPeriod = 0.        ExtListenInterval = 0.
```

P2P device Show P2P configuration:

#iwpriv p2p0 set p2pCfg

```
P2P Device Config :
=====================================================================
Device Name[9] = P2P-linux.
Device Addr = 00:0c:43:32:40:45.
OpChannel = 1.        Listen Channel = 11.
My Go Intent = 0.
WscMode = PBC.        ConfigMethod = PBC   PBC.
SSID[9] = DIRECT-l2.
NoA_Count = 1.        NoA_Duration = 51200.        NoA_Interval = 102400.        StartTime = 0.
ExtListenPeriod = 0.        ExtListenInterval = 0.
Intra-Bss = 0.
ConenctMAC = 00:00:00:00:00:00.
```

P2P Show current rule and state machine status

#iwpriv p2p0 set p2pStat

```
P2P Current State
=====================================================================
My Rule = I am P2P Client
CTRL Machine State = P2P_CTRL_IDLE.
DISC Machine State = P2P_DISC_IDLE.
GO_FORM Machine State = P2P_GO_FORM_IDLE.
```

P2P device GO security setting change:

#iwpriv p2p0 set p2pReset=1
#iwpriv p2p0 set p2pOpCh=1
#iwpriv p2p0 set P2pOpMode=1
#iwpriv p2p0 set p2pWscMode=2
#iwpriv p2p0 set p2pWscConf=3
#iwpriv p2p0 set p2pDevName=Ralink-P2P-Device
#iwpriv p2p0 set SSID=DIRECT- Ralink
#iwpriv p2p0 set AuthMode=WPA2PSK
#iwpriv p2p0 set EncrypType=AES
#iwpriv p2p0 set WPAPSK=12345678
iwpriv p2p0 set SSID=DIRECT- Ralink
#iwpriv p2p0 set p2pScan=1

Ralink P2P module provides three WPS configuration methods such as PBC, PIN-Displya, PIN-Keypad.

Case 1: Enable autonomous GO on Channel 11 start WPS (PBC):

```
#iwpriv p2p0 set p2pOpCh=11
#iwpriv p2p0 set P2pOpMode=1
#iwpriv p2p0 set p2pWscMode=2
#iwpriv p2p0 set p2pWscConf=3
#iwpriv p2p0 set WscConfMode=7
#iwpriv p2p0 set WscMode=2
#iwpriv p2p0 set WscGetConf=1
#iwpriv p2p0 set p2pScan=1
```

Case 2: Enable autonomous GO on Channel 11 start WPS (PIN-Display):

```
#iwpriv p2p0 set p2pOpCh=11
#iwpriv p2p0 set P2pOpMode=1
#iwpriv p2p0 set p2pWscMode=1
#iwpriv p2p0 set p2pWscConf=1
#iwpriv p2p0 set WscConfMode=7
#iwpriv p2p0 set WscMode=1
#iwpriv p2p0 set WscGetConf=1
#iwpriv p2p0 set p2pScan=1
```

Case 3: Enable autonomous GO on Channel 11 start WPS (PIN-Keypad):

```
#iwpriv p2p0 set p2pOpCh=11
#iwpriv p2p0 set p2pWscMode=1
#iwpriv p2p0 set p2pWscConf=2
#iwpriv p2p0 set WscConfMode=7
#iwpriv p2p0 set WscMode=1
#iwpriv p2p0 set p2pLink=0                  (The index on P2P Scan Table)
#iwpriv p2p0 set WscPinCode=12345670 (read from enrollee's PIN Code)
#iwpriv p2p0 set WscGetConf=1
#iwpriv p2p0 set p2pScan=1
```

P2P device GO Negotiation as GO or CLIENT:

Case 1: To Do P2P GO Negotiation start WPS (PBC):

```
#iwpriv p2p0 set p2pOpCh=11
#iwpriv p2p0 set p2pLisCh=1
#iwpriv p2p0 set p2pGoInt=0              (Default is 0)
#iwpriv p2p0 set p2pWscMode=2
#iwpriv p2p0 set p2pWscConf=3
#iwpriv p2p0 set WscConfMode=7
#iwpriv p2p0 set WscMode=2
#iwpriv p2p0 set WscGetConf=1
#iwpriv p2p0 set p2pScan=1
```

Case 2: To Do P2P GO Negotiation start WPS (PIN-Display):

```
#iwpriv p2p0 set p2pOpCh=11
#iwpriv p2p0 set p2pLisCh=1
#iwpriv p2p0 set p2pGoInt=0              (Default is 0)
#iwpriv p2p0 set p2pWscMode=1
#iwpriv p2p0 set p2pWscConf=1
#iwpriv p2p0 set WscConfMode=7
#iwpriv p2p0 set WscMode=1
#iwpriv p2p0 set WscGetConf=1
#iwpriv p2p0 set p2pScan=1
```

Case 3: To Do P2P GO Negotiation start WPS (PIN-Keypad):

```
#iwpriv p2p0 set p2pOpCh=11
#iwpriv p2p0 set p2pLisCh=1
#iwpriv p2p0 set p2pGoInt=0              (Default is 0)
#iwpriv p2p0 set p2pWscMode=1
#iwpriv p2p0 set p2pWscConf=2
#iwpriv p2p0 set WscConfMode=7
#iwpriv p2p0 set WscMode=1
#iwpriv p2p0 set p2pLink=0               (The index on P2P Scan Table)
#iwpriv p2p0 set WscPinCode=12345670 (read from enrollee's PIN Code)
#iwpriv p2p0 set WscGetConf=1
#iwpriv p2p0 set p2pScan=1
```

8 IOCTL

8.1 Parameters for iwconfig

Access	Description	ID	Parameters
Get	BSSID, MAC Address	SIOCGIFHWADDR	wrq->u.name, (length = 6)
	WLAN Name	SIOCGIWNAME	wrq->u.name = "RT5370Wireless", 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	SIOCGIWFRAG	wrq->u.frag.value = (INT) pAd-> CommonCfg.FragmentThreshold;

	(bytes)		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;

8.2 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.

8.2.1 Set Data, Parameters is Same as iwpriv

Command and IOCTL Function

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

RTPRIV_IOCTL_SET	iwpriv ra0 **set SSID=RT3572AP**	sprintf(name, "ra0");
		strcpy(data, "**SSID=RT3572AP**");
		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);

8.2.2 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);

8.2.3 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);

8.2.4 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);

8.2.5 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);

9 IOCTL INSTRUCTIONS

9.1 Get Data

9.1.1 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);
```

9.1.2 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);
```

9.1.3 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);
```

9.1.4 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 ;
```

9.1.5 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);
```

9.1.6 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);
```

9.1.7 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);
```

9.1.8 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);
```

9.1.9 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);
```

9.1.10 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);
```

9.1.11 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);
```

9.1.12 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);
```

9.1.13 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);
```

9.1.14 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);
```

9.1.15 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);
```

9.1.16 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);
```

9.1.17 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);
```

9.1.18 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);
```

9.2 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
}
```