



Realtek Wi-Fi Direct/Miracast Programming Guide

Support Linux Kernel <= 2.6.34

Date: 2013/01/29

Version: 1.2

This document is subject to change without notice. The document contains Realtek confidential information and must not be disclosed to any third party without appropriate NDA.

Document Version	Note
V1.2	<ol style="list-style-type: none"><li data-bbox="641 237 992 268">1. Refine the P2P description<li data-bbox="641 279 943 310">2. Added P2P commands<li data-bbox="641 321 1000 352">3. Added Miracast commands

Wi-Fi Direct (P2P) is the new technology developed by Wi-Fi Alliance. It is a solution for Wi-Fi device-to-device connectivity. And it is also backward compatible with existing Wi-Fi Certified devices.

The following picture is the overview for Wi-Fi Direct architecture of Realtek Linux Wi-Fi Direct software. This software architecture is based on the linux standard interface “wireless extension”. If the system platform will perform the P2P/Miracast feature on cfg80211/Android, please refer to another document (Miracast on Android.pdf).

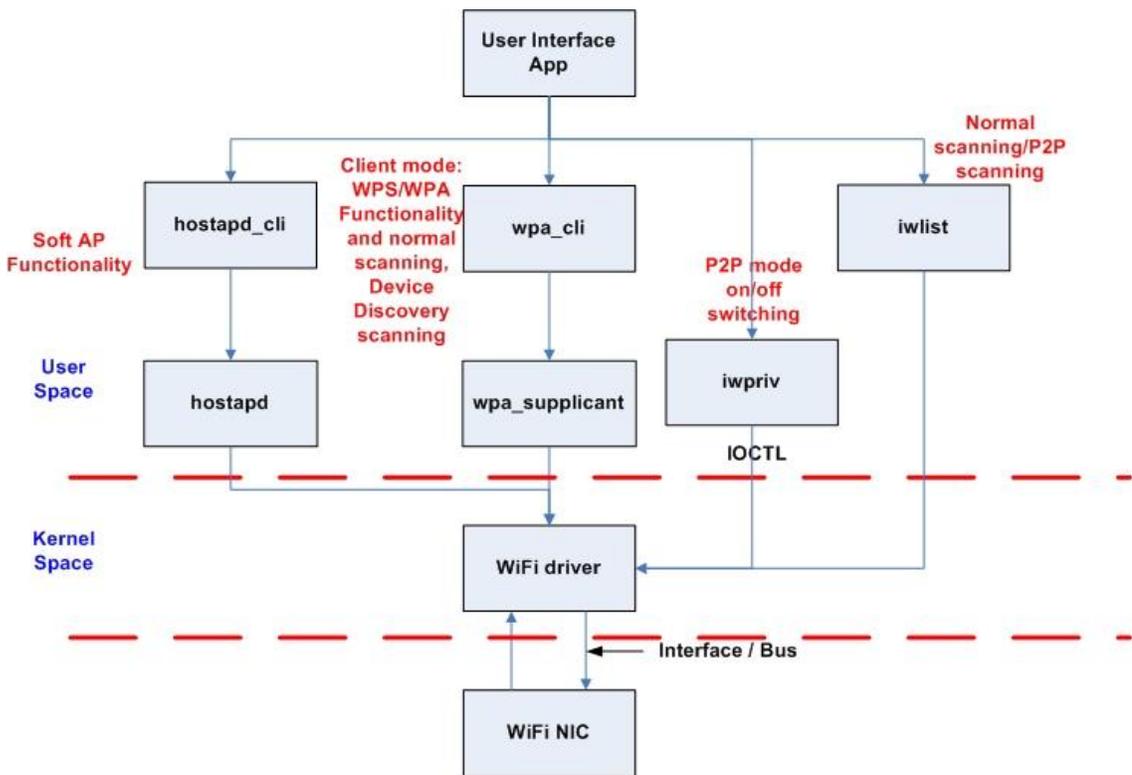


Figure1: Software Architecture for Realtek Linux Wi-Fi Direct

Basically, there are two roles in the Miracast connection. One is Miracast source device; another is Miracast display (sink) device. The TDLS & Wi-Fi Direct can be used to establish the Miracast connection. The Wi-Fi Direct is mandatory feature for Miracast, TDLS is optional. This document will focus on the Wi-Fi Direct technology when creating the Miracast connection.

“User Interface App (UI)” is an application which should be designed by customer for their product. UI should use the iwpriv application to get/set whole the P2P

information from/to the Realtek Wi-Fi driver. Iwpriv application is available in the wpa_supplicant_hostapd folder and we also provide the porting guide under document folder. (Please refer to the [Wireless tools porting guide.doc](#))

Of course, the UI can use the iwpriv application to issue the P2P information to driver based on the Wi-Fi Direct APIs described in this document.

wpa_cli and wpa_supplicant are the standard application to handle whole the 802.11 station mode connection. Hostapd and hostapd_cli are also the standard application to handle whole the 802.11 AP mode connections.

In usual, there are 4 stages in the Wi-Fi Direct scenario.

1. “Device Discovery”
2. “Provision Discovery”
3. “Group Formation”
4. “Provisioning”

The following picture will provide the overall concept for Wi-Fi Direct functionality and it will also contain these 4 stages described above.

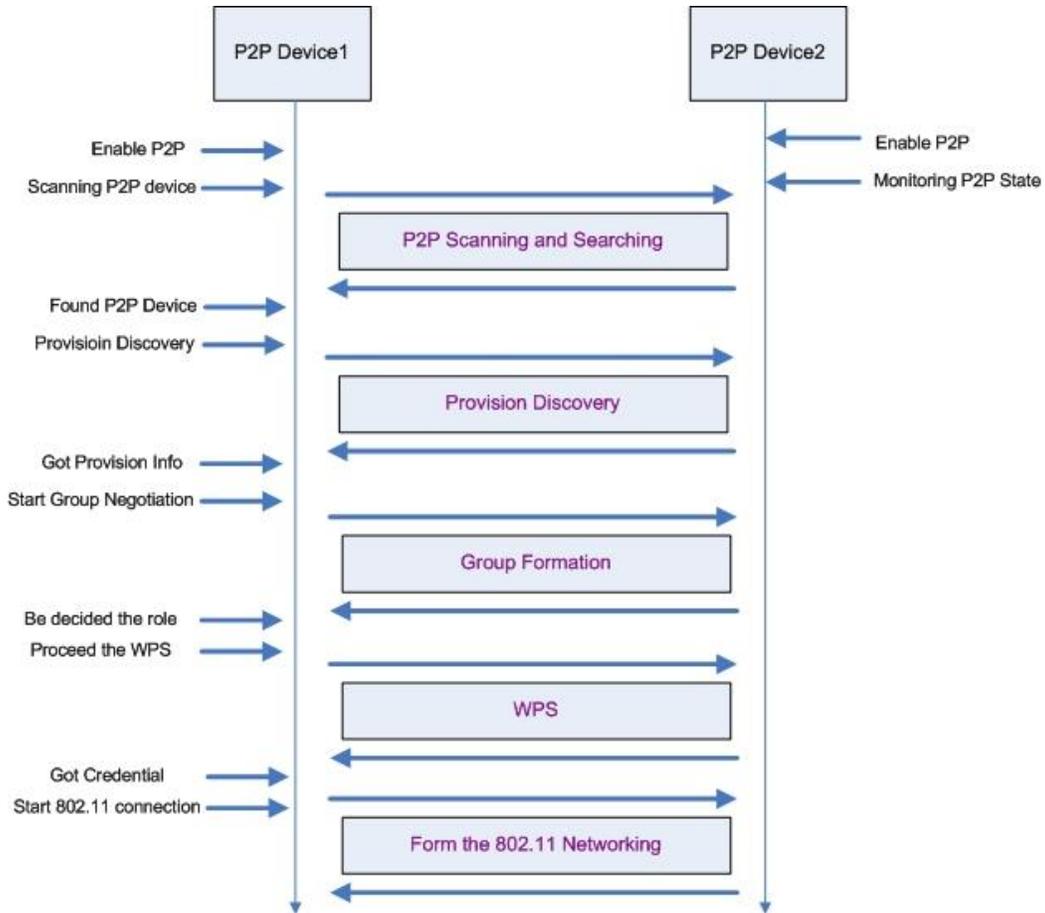


Figure2: Wi-Fi Direct Overview

The figure2 describes the basic Wi-Fi Direct scenario and this document will use this figure to explanation the Wi-Fi Direct functionality and APIs.

1. Enable P2P

In this case, there are two Wi-Fi devices which both support the Wi-Fi Direct functionality. We can use the iwpriv to enable the Wi-Fi Direct function of Realtek Wi-Fi driver (enable P2P).

```
#> iwpriv wlan0 p2p_set enable=n
```

“wlan0” is the network interface for Realtek Wi-Fi device on the system. “p2p_set” command is used to pass the settings information to the driver. “enable” is the actual command to tell the driver which information the application is trying to set. “n” is a number to set the P2P functionality.

“n=0” means to disable the P2P functionality

```
Ex: #> iwpriv wlan0 p2p_set enable=0
```

“n=1” means to turn the P2P functionality on and the Wi-Fi driver will be the P2P device mode.

```
Ex: #> iwpriv wlan0 p2p_set enable=1
```

“n=2” means to turn the P2P functionality on and the Wi-Fi driver will be the P2P client mode

```
Ex: #> iwpriv wlan0 p2p_set enable=2
```

“n=3” means to turn the P2P functionality on and the Wi-Fi driver will be the P2P group owner mode.

```
Ex: #> iwpriv wlan0 p2p_set enable=3
```

“n” had been defined in the P2P.h file of the document folder. This document also copies that definition here for the reference.

```
enum P2P_ROLE {  
    P2P_ROLE_DISABLE = 0,  
    P2P_ROLE_DEVICE = 1,  
    P2P_ROLE_CLIENT = 2,  
    P2P_ROLE_GO = 3  
};
```

The Realtek Wi-Fi driver supports the concurrent mode. It means there are two network interfaces on your system when the Wi-Fi driver is installed on your platform. However, the P2P functionality can just be enabled on one network interface. The P2P functionality can't be enabled on both two network interface at the same time. For example: If you had enabled the P2P functionality on wlan0, the Wi-Fi driver will ignore the P2P enable instruction automatically when you try to enable the P2P functionality on wlan1.

2. Scanning P2P Device

After enabling the P2P functionality of the Wi-Fi driver, the P2P device1 got to find out how many other P2P devices exist in the environment. The UI can do the scan via wpa_supplicant or iwlist command.

For wpa_supplicant:

```
Ex: #> wpa_cli scan // Ask wpa_supplicant to do the scanning
```

Ex: #> wpa_cli scan_results // Get the scanning result

For iwlist:

Ex: #> iwlist wlan0 scan// Issue the scanning request to driver and result the scanning result.

When P2P is enabled, the Wi-Fi driver can provide the different scanning result based on the “scan_type” command.

Ex: #> iwpriv wlan0 p2p_set scan_type=n

When n=0, the Wi-Fi driver will provide the device list which has the P2P capability. When n=1, the Wi-Fi driver will list all the Access Points and P2P devices (This case is related to the WiDi Gen2 NB). When n=2, the Wi-Fi driver will provide the Miracast source device list if the Wi-Fi driver is set to Miracast sink device. When n=2, the Wi-Fi driver will provide the Miracast sink device list if the Wi-Fi driver is set to the Miracast source device. The command “wfd_type” can be used to set the Wi-Fi driver to Miracast source/sink device. When n=2

Ex: #> iwpriv wlan0 p2p_set wfd_type=n

When n=0, the Wi-Fi driver will play the Miracast source device. When n=1, the Wi-Fi driver will play the Miracast sink device.

After having the scanning list, Realtek Wi-Fi driver provides the following commands to get more detail information for each found device. (the following command will just work fine when the MAC exists in the scanning result)

The wps_CM command will provide the WPS config method which the P2P device with the MAC address supports. The MAC format is XX:XX:XX:XX:XX:XX. The MAC address is the mac address shows up in the scanning result generated by iwlist or wpa_supplicant.

For example: 00:01:22:33:44:55

#> iwpriv wlan0 p2p_get2 wps_CM=MAC

The devN command will provide the readable device name of specific P2P device.

#> iwpriv wlan0 p2p_get2 devN=MAC

The dev_type command will provide the WPS primary device type of specific

P2P devic.

```
#> iwpriv wlan0 p2p_get2 dev_type=MAC
```

The following information is the meaning returned by dev_type command.

N=00 -> device doesn't exist in the scanning list

N=01 -> Computer

N=02 -> Input Device

N=03 -> Printers, Scanners, Faxes, Copiers

N=04 -> Camera

N=05 -> Storage

N=06 -> Network Infrastructure

N=07 -> Displays

N=08 -> Multimedia Devices

N=09 -> Gaming Devices

N=10 -> Telephone

N=11 -> Audio Device

The go_devadd command will provide the P2P device address for specific P2P GO. (The SSID for P2P device is "DIRECT-", the length of SSID is 7. The prefix of SSID for P2P GO is "DIRECT-", the length of P2P GO's SSID is bigger than 8.)

```
#> iwpriv wlan0 p2p_get2 go_devadd=MAC
```

The InvProc command will provide the information which the specific P2P device/P2P GO supports the P2P Invitation Procedure or not.

```
#> iwpriv wlan0 p2p_get2 InvProc=MAC
```

However, the InvProc will be 1 on some Android smart phones even they doesn't support the P2P persistent function.

3. Provision Discovery

The purpose for the provision discovery is to get the WPS Pin Code or WPS push button for the following WPS procedure.

"prov_disc" is the command to start the provision procedure. "00:11:22:33:44:55" is the P2P device address of peer P2P device (P2P Device2 in the figure2) which you want to get the WPS information. "-" is a connector. "display" means the peer P2P device should display its PIN CODE on its screen and the user should key-in this PIN CODE on the local P2P device (P2P Device1 in the figure1). "keypad" means the local P2P device should display its PIN CODE on its screen and

the user should key-in this PIN CODE on the peer P2P device. “pbc” means these two P2P device will use the WPS push button for the following WPS procedure. “label” means the user should read the PIN CODE from the label of peer P2P device and key-in this PIN CODE of this label on the local P2P device. And now, both local P2P device and peer P2P device had got the PIN CODE or PBC and should be ready to form an 802.11 network.

```
Ex: #> iwpriv wlan0 p2p_set prov_disc=00:11:22:33:44:55_display
```

```
Ex: #> iwpriv wlan0 p2p_set prov_disc=00:11:22:33:44:55_keypad
```

```
Ex: #> iwpriv wlan0 p2p_set prov_disc=00:11:22:33:44:55_pbc
```

```
Ex: #> iwpriv wlan0 p2p_set prov_disc=00:11:22:33:44:55_label
```

However, the peer P2P device is possible to be the P2P GO (The SSID can be used to check whether this P2P device is P2P device or P2P GO). The MAC address of wpa_supplicant & iwlist is the interface address of that P2P GO. In this case, the UI should use the “go_devadd” command to get the P2P GO’s device address then use the “prov_disc” command with this P2P GO’s device address to send the provision discovery request to P2P GO.

After getting the WPS PIN CODE or PBC, the UI should use the “got_wpsinfo” command to inform the Wi-Fi driver for this.

“got_wpsinfo=1” means the UI got the WPS PIN CODE from peer P2P device’s screen or label and uses key-in this PIN CODE on the local P2P device.

“got_wpsinfo=2” means the PIN CODE is displayed from the local P2P device and user had key-in this PIN CODE on the peer P2P device.

“got_wpsinfo=3” means the UI got the WPS PBC.

```
Ex: #> iwpriv wlan0 p2p_set got_wpsinfo=1
```

```
Ex: #> iwpriv wlan0 p2p_set got_wpsinfo=2
```

```
Ex: #> iwpriv wlan0 p2p_set got_wpsinfo=3
```

The P2P.h file also defined the value and meaning for the “got_wpsinfo” command.

```
enum P2P_WPSINFO {
    P2P_NO_WPSINFO                = 0,
    P2P_GOT_WPSINFO_PEER_DISPLAY_PIN = 1,
    P2P_GOT_WPSINFO_SELF_DISPLAY_PIN = 2,
    P2P_GOT_WPSINFO_PBC           = 3,
};
```

On the P2P device2 side of figure2, it can use the “status” command to check the current P2P state. If the status string is the “Status=08”, it means the driver received the provision discovery request from certain P2P device. At this moment, the UI should use the “req_cm” command to know which WPS method the certain P2P device is purpose to do.

```
Ex: #> iwpriv wlan0 p2p_get req_cm  
Return String: CM=dis or CM=lab or CM=pbcc or CM=pad
```

If the return string is “CM=dis”, it means the peer P2P device want to this P2P device to show up the PIN CODE on the local screen so that the user can key-in this PIN CODE on the peer P2P device. If the return string is “CM=lab”, it means the peer P2P device want to use the PIN CODE printed on the label of this P2P device and user can key-in this PIN CODE on the peer P2P device. If the return string is “CM=pbcc”, it means the peer P2P device wants to use the PBC for the following WPS procedure. If the return string is “CM=pad”, it means the peer P2P device will show its PIN CODE on the peer P2P device side and the user should key-in this PIN CODE on the local P2P device.

4. Start Group Negotiation

In the Wi-Fi Direct scenario, one of the P2P devices will become a group owner (almost the same as the SoftAP) and the other P2P device will become an 802.11 client to connect to that group owner. The stage4 “Start Group Negotiation” is the procedure to determine which P2P device should be the group owner/client.

“intent” is a value from 0 ~ 15. This value will provide the degree information to want to be the group owner. “intent=15” means this Wi-Fi driver must be the group owner. The default intent value is 1 and this default value will be assigned by enabling the P2P functionality.

```
Ex: #> iwpriv wlan0 p2p_set intent=n
```

Beside the intent value, the UI should determine the SSID which will be used when this P2P device becomes the group owner with SoftAP functionality in the future. After UI determine the SSID, the UI should pass that SSID to the driver by using the ssid command. This information must be passed to driver before calling the “nego” command.

```
Ex: #> iwpriv wlan0 p2p_set ssid=SsidString
```

“nego” command will inform the Wi-Fi driver to perform the group negotiation procedure.

```
Ex: #> iwpriv wlan0 p2p_set nego=00:11:22:33:44:55
```

In the figure2, the P2P Device2 is using the “status” and “role” commands to monitor the P2P state machine so that the UI of P2P Device2 just is able to know what kinds of information the P2P Device1 sent.

```
Ex: #> iwpriv wlan0 p2p_get status  
Return string : Status=02 or Status=10
```

```
Ex: #> iwpriv wlan0 p2p_get role  
Return string: Role=01
```

The following two enum are the definition for the “status” and “role” commands.

```
enum P2P_STATE {  
    P2P_STATE_NONE = 0,    //    P2P disable  
    P2P_STATE_IDLE = 1,    //    P2P had enabled and do nothing  
    P2P_STATE_LISTEN = 2,    //    In pure listen state  
    P2P_STATE_SCAN = 3,    //    In scan phase  
    P2P_STATE_FIND_PHASE_LISTEN = 4,    //    In the listen state of find phase  
    P2P_STATE_FIND_PHASE_SEARCH = 5,    //    In the search state of find phase  
    P2P_STATE_TX_PROVISIONING_DIS_REQ = 6, //    In P2P provisioning discovery  
    P2P_STATE_RX_PROVISIONING_DIS_RSP = 7,  
    P2P_STATE_RX_PROVISIONING_DIS_REQ = 8,  
    P2P_STATE_GONEGO_ING = 9, //    Doing the group owner negotiation handshake  
    P2P_STATE_GONEGO_OK = 10, //    finish the group negotiation handshake with success  
    P2P_STATE_GONEGO_FAIL = 11, //    finish the group negotiation handshake with failure  
    P2P_STATE_RECV_INVITE_REQ_MATCH = 12, //    receiving the P2P Invitation request  
    and match with the profile.  
    P2P_STATE_PROVISIONING_ING = 13,    //    Doing the P2P WPS  
    P2P_STATE_PROVISIONING_DONE = 14, //    Finish the P2P WPS  
    P2P_STATE_TX_INVITE_REQ = 15,    //    Transmit the P2P Invitation request  
    P2P_STATE_RX_INVITE_RESP_OK = 16, //Receiving the P2P Invitation response with success  
    P2P_STATE_RECV_INVITE_REQ_DISMATCH = 17, //    receiving the P2P Invitation  
    request and mismatch with the profile.  
    P2P_STATE_RECV_INVITE_REQ_GO = 18, //receiving the P2P Invitation request and this wifi
```

is GO.

```
P2P_STATE_RECV_INVITE_REQ_JOIN = 19, //receiving the P2P Invitation request to join an existing P2P Group.
```

```
P2P_STATE_RX_INVITE_RESP_FAIL = 20, // recveing the P2P Invitation response with failure
```

```
P2P_STATE_RX_INFOR_NOREADY = 21, // receiving p2p negoitation response with information is not available
```

```
P2P_STATE_TX_INFOR_NOREADY = 22, // sending p2p negoitation response with information is not available
```

```
};
```

```
enum P2P_ROLE {
```

```
    P2P_ROLE_DISABLE = 0,
```

```
    P2P_ROLE_DEVICE = 1,
```

```
    P2P_ROLE_CLIENT = 2,
```

```
    P2P_ROLE_GO = 3
```

```
};
```

For example, the UI of P2P Device2 will get the “Status=08” when the P2P Device1 proceeds the Provision Discovery procedure. “State=10” means the group negotiation procedure is finished with success. “State=11” means the group negotiation procedure is finished with failure.

After the P2P Device1 and P2P Device2 found the group negotiation finished, they can use the “role” command to know the role they should play in the following operation.

```
Ex: #> iwpriv wlan0 p2p_get role
```

```
Return string: Role=02
```

If the return string is “Role=02”, it means this P2P device should play the 802.11 client role in the following operation. The UI should launch wpa_supplicant to perform the WPS procedure with the peer P2P device.

If the return string is “Role=03”, it means this P2P device should be the 802.11 AP role in the following operation. The UI should launch the hostapd to enable the SoftAP functionality and enable the WPS procedure.

In some UIs design, the UI will inform the user there is a connection request

from peer P2P device and need user's confirmation to accept this request or not. In this case, the UI won't use the "got_wpsinfo" command to inform the Wi-Fi driver it already got the WPS config method until user accepts the connection request. It is possible to get the status=22 in this case. It means the Wi-Fi driver receives the P2P negotiation request sent from peer P2P device and the "got_wpsinfo" command doesn't be called to set the WPS config method. So, the Wi-Fi driver already sent the P2P negotiation response to peer P2P device with "information is unavailable" error status. When this case happens, the UI should use the "peer_deva" command. After having the device address for peer P2P device, the UI can start the P2P scanning function continuously until the scanning result contains this device address and use the "p2p_connect" command to connect back to peer P2P device.

5. Proceed the WPS

After confirming the role for both P2P Device1 and P2P Device2, the P2P device which got the "Role=02" should launch the wpa_supplicant in the background and use the wpa_cli with PIN CODE or PBC to perform the WPS procedure. (Please refer to [wpa_cli_with_wpa_supplicant_20100728.doc](#) for further information.) However, the P2P device address and P2P interface address won't be the same for some P2P/Miracast devices. After the UI got the status=10, the UI can use the "peer_ifa" to get the interface address for peer P2P/Miracast device. This interface address can be used for the following WPS procedure to improve the connection success rate.

```
#> iwpriv wlan0 p2p_get peer_ifa
```

In the same word, the P2P device which got the "Role=03" should launch the hostapd in the background and use the hostapd_cli with PIN CODE or PBC to perform the WPS procedure. (Please refer to [Quick_Start_Guide_for_SoftAP.doc](#) for further information.)

6. DHCP

The Wi-Fi Direct Specification required that the P2P device which becomes the group owner should also provide the DHCP server application in their system. The DHCP server should be launched and be ready to provide the IP address to the DHCP client. The specification also required that the P2P device which becomes the P2P client should launch the DHCP client application to acquire the IP address from the P2P group owner after the wpa_supplicant established the 802.11 connection with AP successfully.

7. Other P2P commands

1. “inv_peer_deva” command:

When the Wi-Fi driver receives the P2P Invitation request from a P2P GO and this invitation request is asking to join the P2P GO’s group, the P2P state will be 19. The Wi-Fi driver can use the “peer_ifa” command to get interface address of that P2P GO. After having the P2P GO’s interface address, the UI can do the scanning continuously until the scanning result contains this interface address. Then, the “inv_peer_deva” can be used to get the device address of that P2P GO. The UI can use the “prov_disc” to issue the provision discovery request with this device address then do the WPS procedure with this P2P GO.

2. “op_ch” get command:

When the P2P state is 10, it means the P2P negotiation had finished successfully. The “op_ch” command can be used to know the final operating channel whether the local Wi-Fi device is P2P GO or P2P Client.

Ex: #> iwpriv wlan0 p2p_get op_ch

3. “peer_port” command:

“peer_port” is the Miracast command and is used to get the control port for Miracast source device. If the local Wi-Fi driver is playing the P2P Client, the UI can use this command after the wpa state becomes “COMPLETED”. If the local Wi-Fi driver is playing the P2P GO, the UI can use this command after the P2P Client had connected to it (by using the hostapd_cli all_sta command).

4. “wfd_sa” command:

“wfd_sa” command is used to know the peer Miracast device is available or not. If the peer Miracast device is not available, the Wi-Fi driver can’t use the prov_disc and p2p_connect command to send the P2P packets to peer device.

5. “profilefound” command:

This command is used to support the P2P persistent function. When the P2P/Miracast connection is established and the Wi-Fi driver plays the P2P Client, the UI should store the current network setting to a profile. When the UI enters the re-initialization state, the UI should use this “profilefound” command to set the profile information to Wi-Fi driver. After that, it is possible to get the P2P state=12. The P2P state=12 means the Wi-Fi driver received a P2P Invitation Request frame to perform a P2P persistent group and the Wi-Fi driver can find a profile out to be a P2P Client to rebuild the P2P connection. The P2P state=17 means the Wi-Fi driver received a P2P Invitation Request to perform a P2P persistent group but this profile can’t be found in the profile information. P2P state=18 means the Wi-Fi driver received a P2P Invitation Request to perform a P2P persistent group and the Wi-Fi driver plays as the P2P GO.

Ex: #> iwpriv wlan0 p2p_set profilefound=0 // for clear all the profile

information of Wi-Fi driver

```
Ex: #> iwpriv wlan0 p2p_set profilefound=1MACYYSSID
```

MAC is the interface address of peer P2P GO in XX:XX:XX:XX:XX:XX format.

YY is the length of SSID.

SSID is the SSID string for the P2P persistent group.

6. “listen_ch” command:

This command can be used to set the P2P listen channel to Wi-Fi driver. The listen channel value should be 1 or 6 or 11. The Wi-Fi driver will determine the listen channel by itself in two cases.

Case 1: The input listen channel is not 1, 6, 11.

Case 2: The Wi-Fi driver supports the concurrent mode and another network interface had connected to an AP which stands on channel 1 or 6 or 11.

7. “op_ch” set command:

This command can be used to set the desired operating channel to Wi-Fi driver. However, the final operating channel should be confirmed by using the “op_ch” get command.

```
Ex: #> iwpriv wlan0 p2p_set op_ch=6
```

8. “invite” command:

This command is used to send the P2P Invitation Request to perform a P2P persistent group.

```
Ex: #> iwpriv wlan0 p2p_set invite="MAC1 GOMAC GOSSID"
```

The MAC1 is the P2P device address for peer P2P device. The GOMAC is the interface address for that persistent group. The GOSSID is the SSID for persistent GO.

9. “persistent” command:

This command will enable/disable the P2P persistent function so that other P2P devices will know this P2P device supports the persistent function or not.

```
Ex: #> iwpriv wlan0 p2p_set persistent=1
```

10. “sa” command:

This command is for Miracast functionality. It is used to let other Miracast devices know the Wi-Fi is ready for Miracast connection or not.

```
Ex: iwpriv wlan0 p2p_set sa=1
```

11. “wfd_type” command:

This command is for Miracast functionality and can be used to set the Miracast role.

```
// Set the Miracast role to source device
```

```
Ex: #> iwpriv wlan0 p2p_set wfd_type=0
```

// Set the Miracast role to display (sink) device

Ex: #> iwpriv wlan0 p2p_set wfd_type=1

12. “scan_type” command:

This command is used to control the scanning result list.

Ex: #> iwpriv wlan0 p2p_set scan_type=0; iwlist wlan0 scan

The scanning result will be the found P2P devices.

Ex: #> iwpriv wlan0 p2p_set scan_type=1; iwlist wlan0 scan

The scanning result will be found Access Points and P2P devices. This mode will be compatible with the Intel WiDi Gen2.

Ex: #> iwpriv wlan0 p2p_set scan_type=2; iwlist wlan0 scan

If the Wi-Fi driver is Miracast source device, the scanning result will be found Miracast display device. If the Wi-Fi driver is Miracast display device, the scanning result will be the found Miracast source device.