AN13680

Wi-Fi Tx Power Table and Channel Scan Management for RW61x

Rev. 1 — 10 August 2022

Application note
CONFIDENTIAL

Document information

Information	Content
Keywords	Transmit (Tx) power levels, Tx power table, Wi-Fi channel list, data structure, CLI command
Abstract	Describes how to configure the Wi-Fi Tx power table and Wi-Fi channel list in the product software.

1 Revision history

Revision history

Rev	Date	Description
v.1	20220810	Initial version

2 Introduction

The Wi-Fi TX power table defines the target transmit power levels for all operating conditions of the product. The transmit power levels are determined based on regulatory compliance, IEEE 802.11 requirements, and product design constraints. The general goal is to adjust the power table to achieve the highest transmit power level within these constraints.

The target power level is defined at the antenna connector and set per Wi-Fi channel, bandwidth, and modulation.

The Wi-Fi channel list defines the channels allowed for the product. The regulatory domain and country-specific requirements determine the channel list and its adjustment. For example, a product certified for use in Europe can operate on channels 1-13 in the 2.4 band, whereas operation in US is restricted to channels 1-11.

Channel selection during product operation requires the radio to scan the channel list. Two scan types can be selected based on the desired behavior.

2.1 Scope

This application note describes how to configure the Wi-Fi TX power table and Wi-Fi channel list in product software.

It includes information on the APIs, data structures, and CLI commands that can be used to update the power table and channel list. Before using this document, review the software architecture of your product.

2.2 Supported chipsets

- RW610
- RW612

3 Tx power tables

3.1 TX power table in OTP

The Wi-Fi TX power table may optionally be programmed into the OTP memory on the radio.

If you are using a RW61x-based wireless module, check with the module vendor if the power table has already been programmed in the OTP memory. Also ask if the Region Enforcement bit is set.

If the power table is already programmed in the OTP memory, and if the Region Enforcement bit is set, update the power table using the API defined in this document. Note:

- The `get` command (which reads the power table) returns the power table defined in OTP.
- The `set` command (which sets the power table) does not update the power table and it does not report an error.

3.2 TX power table configuration

This section shows how to configure the TX power tables. To customize the transmission power levels for each wireless channel, make sure to follow the system requirements and regulatory domain rules. The steps to configure the channel and power level are explained in this section.

The TX power levels are set using `wlan_set_txpwrlimit` API, and queried using `wlan_get_txpwrlimit` API. These functions use the following data structures:

- `tx_pwrlimit_2g_cfg` (power levels in 2.4 GHz band)
- `tx_pwrlimit_5g_cfg` (power levels in 5 GHz band)

Both data structures are defined in the file:

<sdk_path>/components/wifi_bt_module/AzureWave/tx_pwr_limits/wlan_txpwrlimit_cfg_WW_rw610.h

The information within these structures allows the user to specify transmit power levels for specific:

- band
- channels
- data rate
- bandwidth

See below an example of `tx_pwrlimit_2g_cfg` and `tx_pwrlimit_5g_cfg` data structures.

Refer to *<sdk_path>/middleware/wifi/wlcmgr/wlan_enhanced_tests.c* for an example of how the data structures are populated and used in the APIs for TX power configuration.

```
static wlan_txpwrlimit_t tx_pwrlimit_2g_cfg =
     {
          .subband   = (wifi_SubBand_t)0x00,
          .num_chans = 1,
          .txpwrlimit_config[0] =
          {
               .num_mod_grps = 20,
               .chan_desc =
               {
                  .start_freq = 2407,
                  .chan_width = 20,
                  .chan_num   = 1,
                },
               .txpwrlimit_entry = {{0, 19}, {1, 15}, {2, 15}, {3, 15}, {4, 13}, {5, 13},
               {6, 13}, {7, 0}, {8, 0}, {9, 0}, {10, 13}, {11, 0}, {12, 0}, {13, 0}, {14, 0},
               {15, 0}, {16, 13}, {17, 0}, {18, 0}, {19, 0}},
          },
     };

#ifdef CONFIG_5GHz_SUPPORT
static wlan_txpwrlimit_t tx_pwrlimit_5g_cfg =
     {
          .subband   = (wifi_SubBand_t)0x00,
          .num_chans = 3,
          .txpwrlimit_config[0] =
          {
               .num_mod_grps = 20,
               .chan_desc =
               {
                    .start_freq = 5000,
                    .chan_width = 20,
                    .chan_num   = 36,
               },
               .txpwrlimit_entry = {{0, 0},{1, 20}, {2, 20}, {3, 20}, {4, 20}, {5, 20},
               {6, 19}, {7, 0}, {8, 0}, {9, 0}, {10, 16}, {11, 0}, {12, 0}, {13, 0},
               {14, 0}, {15, 0}, {16, 16}, {17, 0}, {18, 0}, {19, 0}},
          },
          .txpwrlimit_config[1] =
          {
               .num_mod_grps = 9,
               .chan_desc =
               {
                    .start_freq = 5000,
                    .chan_width = 20,
                    .chan_num   = 100,
               },
               .txpwrlimit_entry = {{0, 0},{1, 20}, {2, 20}, {3, 20}, {4, 20}, {5, 20},
               {6, 19}, {7, 0}, {8, 0}, {9, 0}, {10, 16}, {11, 0}, {12, 0}, {13, 0},
               {14, 0}, {15, 0}, {16, 16}, {17, 0}, {18, 0}, {19, 0}},
          },
          .txpwrlimit_config[2] =
          {
               .num_mod_grps = 9,
               .chan_desc =
               {
                    .start_freq = 5000,
                    .chan_width = 20,
                    .chan_num   = 149,
               },
               .txpwrlimit_entry = {{0, 0},{1, 20}, {2, 20}, {3, 20}, {4, 20}, {5, 20},
               {6, 19}, {7, 0}, {8, 0}, {9, 0}, {10, 16}, {11, 0}, {12, 0}, {13, 0},
               {14, 0}, {15, 0}, {16, 16}, {17, 0}, {18, 0}, {19, 0}},
          },
         };
#endif
```

The main parameters in the `tx_pwrlimit_2g_cfg` and `tx_pwrlimit_5g_cfg` data structures are defined in Table 1.

Table 1. Main parameters in txpwrlimit data structures

Parameter	Description
`num_chans`	Number of wireless channels configured in this structure. Up to 13 channels for 2.4 GHz and up to 39 channel for 5 GHz are supported.
`num_mod_groups`	Number of rate groups to configure the power level for

Table 1. Main parameters in txpwrlimit data structures*...continued*

Parameter	Description
`chan_num`	Channel number
`start_freq`	Starting frequency for a channel
`chan_width`	Channel bandwidth in MHz (remains 20 MHz)
`txpwrlimit_entry`	Specifies the power levels corresponding to rate groups
`txpwrlimit_config`	Configuration entry

The above code sample sets TX Power table for channel 1 for the 2.4 GHz band. And it sets TX power table for channels 36, 100, and 149 for the 5 GHz band. Each channel within the structure is associated to a configuration entry (parameter `txpwrlimit_config`). The configuration entry includes the channel number, the frequency, and the parameter `txpwrlimit_entry`.

The TX power table consists of all target power levels, organized by RF channel and rate group index. To reduce the size of the power table, multiple data rates are grouped into a single rate group. For example, 24 Mbit/s and 36 Mbit/s legacy OFDM rates are combined in a single rate group. And they have the same target power level. See Table 2.

Table 2. Rate group information

Rate group	Description
0	CCK modulation (data rates: 1, 2, 5.5, 11 Mbit/s)
1	OFDM modulation (data rates: 6, 9, 12, 18 Mbit/s)
2	OFDM modulation (data rates: 24, 36 Mbit/s)
3	OFDM modulation (data rates: 48, 54 Mbit/s)
4	Channel bandwidth 20 MHz, 1 spatial stream, HT/VHT/HE MCS: 0, 1, 2
5	Channel bandwidth 20 MHz, 1 spatial stream, HT/VHT/HE MCS: 3, 4
6	Channel bandwidth 20 MHz, 1 spatial stream, HT/VHT/HE MCS: 5, 6, 7
7	Channel bandwidth 40 MHz, 1 spatial stream, HT/VHT/HE MCS: 0, 1, 2[1]
8	Channel bandwidth 40 MHz, 1 spatial stream, HT/VHT/HE MCS: 3, 4[1]
9	Channel bandwidth 40 MHz, 1 spatial stream, HT/VHT/HE MCS: 5, 6, 7[1]
10	Channel bandwidth 20 MHz, 1 spatial stream, VHT MCS: 8, 9
11	Channel bandwidth 40 MHz, 1 spatial stream, VHT MCS: 8, 9[1]
12	Channel bandwidth 80 MHz, 1 spatial stream, VHT MCS: 0, 1, 2[2]
13	Channel bandwidth 80 MHz, 1 spatial stream, VHT MCS: 3, 4[2]
14	Channel bandwidth 80 MHz, 1 spatial stream, VHT MCS: 5, 6, 7[2]
15	Channel bandwidth 80 MHz, 1 spatial stream, VHT MCS: 8, 9[2]
16	Channel bandwidth 20 MHz, 1 spatial stream, HE MCS: 8, 9
17	Channel bandwidth 80 MHz, 1 spatial stream, HE MCS: 10,11[3]

[1] RW61x does not support rates in 40 MHz bandwidth.
[2] RW610x does not support rates in 80 MHz bandwidth.
[3] RW61x supports 802.11ax HE rate up to MCS9 in 20 MHz bandwidth

For example, if for a given channel OFDM modulation at 54 Mbit/s rate is used, this refers to the rate group **3**.

The following code sample shows an example with the data structure `tx_pwrlimit_2g_cfg` where TX power level is changed to 14 dBm for the channel 2 and rate group 3.

```
 .txpwrlimit_config[0] =
        {
         .num_mod_grps = 20,
         .chan_desc =
          {
           .start_freq = 2407,
           .chan_width = 20,
           .chan_num   = 2,
          },
         .txpwrlimit_entry = {{0, 17}, {1, 18}, {2, 17}, {3, 14}, {4, 18}, {5, 16},
{6, 14}, {7, 0}, {8, 0}, {9, 0}, {10, 12}, {11, 0}, {12, 0}, {13, 0}, {14, 0}, {15, 0},
{16, 12}, {17, 0}, {18, 0}, {19, 0}},
         },
```

In the example above, the power table for the rate group 3 is set to 14 dBm. This value can be edited to fit the user needs. Notice that the number of elements in `txpwrlimit_entry` reflects the number specified by the parameter `num_mod_grps`.

Similarly for 5 GHz, the following code sample shows an example with the data structure `tx_pwrlimit_5g_cfg` for channel 36, channel bandwidth 20 MHz, and rate group 4 to set TX power to 14 dBm.

```
 static wlan_txpwrlimit_t tx_pwrlimit_5g_cfg =
        {
         .num_mod_grps = 20,
         .chan_desc =
          {
           .start_freq = 5000,
           .chan_width = 20,
           .chan_num   = 36,
          },
         .txpwrlimit_entry = {{0, 0},{1, 16}, {2, 16}, {3, 15}, {4, 14}, {5, 15}, {6,
14}, {7, 0}, {8, 0}, {9, 0}, {10, 16}, {11, 0}, {12, 0}, {13, 0}, {14, 0}, {15, 0},
{16, 16}, {17, 0}, {18, 0}, {19, 0}},
         },
```

After changing the power table, rebuild and load the SDK needs to the platform for the change to take effect.

Note: *In the example above, `chan_width` value remains 20 MHz but `num_mod_grps` value is updated for 80 MHz, and the pair for 40/80 MHz bandwidth is added to `txpwrlimit_entry` array with power value 0.*

3.3 Customization of TX power table configuration

Note: *When using a module provided by a third party, consult the module vendor for guidance on the transmit power limitations that were used for regulatory compliance. Setting the TX power to levels beyond what the module is certified to support voids the modular certification on the device.*

When the board boots up, the default regulatory region is set to worldwide (WW). And the default power level is set to 8 dBm for all channels.

You can change the TX power levels either using the API calls or using the Command Line Interface.

The next sections detail the two methods.

3.3.1 Customization using API calls

The API used to set the power limit of the Wi-Fi device uses the data structures defined in the file *<sdk_root>/middleware/wifi/wlcmgr/wlan_enhanced_tests.c*. To set the TX power to the desired level, follow the steps in Section 3.2 "TX power table configuration".

To set the TX power table, use the driver APIs with the required arguments:

```
int wlan_set_txpwrlimit (wlan_txpwrlimit_t * txpwrlimit)
```

Where:

Parameter	Description
`[in] txpwrlimit`	A pointer to `wlan_txpwrlimit_t` structure to supply TX Power Table configuration. Read more in Section 3.2 "TX power table configuration".

Returns:

WM_SUCCESS: the call is successful

WM_FAIL: the call failed

Example:

The following example sets TX power limit for 5 GHz bandwidth. The data structure `tx_pwrlimit_5g_cfg` is passed as parameter.

```
ret = wlan_set_txpwrlimit(&tx_pwrlimit_5g_cfg); if (WM_SUCCESS != ret)
   PRINTF("Unable to set 5G TX PWR Limit configuration\r\n");
```

The current implementation provides a return status, that can be used to check whether the function call is successful or not.

Use the following function to **get** the TX power table information:

```
int wlan_get_txpwrlimit (wifi_subband_t subband,
wifi_txpwrlimit_t * txpwrlimit)
```

Where:

Parameter	Description
`[in] subband`	Input parameter with the subband information detailed in Table 3.
`[Out] txpwrlimit`	A pointer to `wlan_txpwrlimit_t` structure used to supply TX power limit configuration. Read more in Section 3.2 "TX power table configuration".

Returns:

WM_SUCCESS: the call is successful

WM_FAIL: the call failed

Example:

The following example gets TX power limit for a subband.

```
int ret = wlan_get_txpwrlimit(subband, &txpwrlimit);

if (WM_SUCCESS != ret)
{
  PRINTF("Unable to get Tx PWR Limit configuration\r\n");
}
else
{
  PRINTF("Get txpwrlimit: sub_band=%x \r\n", txpwrlimit.subband);
  for (i = 0; i < txpwrlimit.num_chans; i++)
  {
    PRINTF("StartFreq: %d\r\n",
           txpwrlimit.txpwrlimit_config[i].chan_desc.start_freq);
    PRINTF("ChanWidth: %d\r\n",
           txpwrlimit.txpwrlimit_config[i].chan_desc.chan_width);
    PRINTF("ChanNum:%d\r\n",
           txpwrlimit.txpwrlimit_config[i].chan_desc.chan_num);
    PRINTF("Pwr:");
    for (j = 0; j < txpwrlimit.txpwrlimit_config[i].num_mod_grps; j++)
    {
      if (j == (txpwrlimit.txpwrlimit_config[i].num_mod_grps - 1))
        PRINTF("%d,%d",
          txpwrlimit.txpwrlimit_config[i].txpwrlimit_entry[j].mod_group,
          txpwrlimit.txpwrlimit_config[i].txpwrlimit_entry[j].tx_power);
      else
        PRINTF("%d,%d,",
          txpwrlimit.txpwrlimit_config[i].txpwrlimit_entry[j].mod_group,
          txpwrlimit.txpwrlimit_config[i].txpwrlimit_entry[j].tx_power);
    }
    PRINTF("\r\n");
  }
}
```

As input, this function takes the subband and the pointer to the `wlan_txpwrlimit_t` type structure (which corresponds to `tx_pwrlimit_2g_cfg` for 2.4 GHz and `tx_pwrlimit_5g_cfg` for 5 GHz band). The output includes:

- Subband
- Starting frequency
- Channel width
- Channel number
- Power levels

The following is an example of system output:

```
# wlan-get-txpwrlimit 00
------------------------------------------------------------------------------
Get txpwrlimit: sub_band=0
StartFreq: 2407
ChanWidth: 20
ChanNum:   1
Pwr:0,19,1,15,2,15,3,15,4,13,5,13,6,13,7,0,8,0,9,0,10,13,11,0,12,0,13,0,14,
0,15,0,16,13,17,0,18,0,19,0
```

Returns:

WM_SUCCESS: the call is successful

WM_FAIL: the call failed

Table 3. Subband definitions

Subband	Description
0x00	2G subband (2.4 GHz: channels 1-14)
0x10	5G subband0 (5 GHz: channels 36, 40, 44, 48, 52, 56, 60, 64)
0x11	5G subband1 ((5 GHz: channels 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140, 144)
0x12	5G subband2 (5 GHz: channels 149, 153, 157, 161, 165, 172)
0x13	5G subband3 (5G: channels 183, 184, 185, 187, 188, 189, 192, 196; 5G: channel 7 ,8, 11, 12, 16, 34)

3.3.2 Customization using the command-line interface

By default, the *wifi_cert* sample application is available for RW61x. The application supports set/get TX power limit command-line interface. This section shows how to add the command-line interface using *wifi_cli* sample application.

- Edit the file *<sdk_root>/middleware/wifi/wlcmgr/wlan_enhanced_tests.c* in the SDK, and add the following line:

```
{"wlan-set-txpwrlimit", NULL, test_wlan_set_txpwrlimit},
```

The example below shows the added line in **bold**.

```
static struct cli_command tests[] = {
    .
    .
    .
    {"wlan-set-txpwrlimit", NULL, test_wlan_set_txpwrlimit},

}
```

- Check that the two structures `tx_pwrlimit_2g_cfg` and `tx_pwrlimit_5g_cfg` are correctly set. Refer to Section 3.2 "TX power table configuration".
- Define the handler for the CLI command `wlan-set-txpwrlimit` to set TX power table:

```
static void test_wlan_set_txpwrlimit(int argc, char **argv)
{
   int rv = wlan_set_txpwrlimit(&tx_pwrlimit_2g_cfg);
   if (rv != WM_SUCCESS)
      PRINTF("Unable to set 2G TX PWR Limit configuration\r\n");
   else
      PRINTF("Successfully configured 2G TX PWR Limit\r\n");
#ifdef CONFIG_5GHz_SUPPORT
   rv = wlan_set_txpwrlimit(&tx_pwrlimit_5g_cfg);
   if (rv != WM_SUCCESS)
      PRINTF("Unable to set 5G TX PWR Limit configuration\r\n");
   else
      PRINTF("Successfully configured 5G TX PWR Limit\r\n");
#endif
}
```

Finally, run `wlan-set-txpwrlimit` command from the command-line interface and check that the output is as shown below:

```
# wlan-set-txpwrlimit
Successfully configured 2G TX PWR Limit
Successfully configured 5G TX PWR Limit
```

The current SDK implementation does not include the CLI commands to **get** the TX power levels. Follow the steps below to add the feature.

- Edit the file *<sdk_root>/middleware/wifi/wlcmgr/wlan_enhanced_tests.c*
- Find the tests structure and add the following line:

```
{"wlan-get-txpwrlimit", "<subband>", test_wlan_get_txpwrlimit},
```

The example below shows the added line in **bold**.

```
static struct cli_command tests[] = {
    .
    .
    .
   {"wlan-get-txpwrlimit", "<subband>", test_wlan_get_txpwrlimit},
};
```

- Edit the command usage function that defines the instructions printed when the help command is issued:

```
static void dump_wlan_get_txpwrlimit_usage()
{
    PRINTF("Usage:\r\n");
    PRINTF("wlan-get-txpwrlimit <subband> \r\n");
    PRINTF("\r\n");
    PRINTF("\t Where subband is: \r\n");
    PRINTF("\t 0x00 2G subband (2.4G: channel  1-14)\r\n");
  #ifdef CONFIG_5GHz_SUPPORT
    PRINTF("\t 0x10 5G subband0 (5G: channel 36,40,44,48,\r\n");
    PRINTF("\t                                52,56,60,64)\r\n");
    PRINTF("\t 0x11 5G subband1 (5G: channel 100,104,108,112,\r\n");
    PRINTF("\t                                116,120,124,128,\r\n");
    PRINTF("\t                                132,136,140,144)\r\n");
    PRINTF("\t 0x12 5G subband2 (5G: channel 149,153,157,161,165,172)\r\n");
    PRINTF("\t 0x13 5G subband3 (5G: channel 183,184,185,187,188,\r\n");
    PRINTF("\t                                189, 192,196;\r\n");
    PRINTF("\t                 5G: channel 7,8,11,12,16,34)\r\n");
  #endif
}
```

- Define the handler for the CLI command `wlan-get-txpwrlimit` to get TX power limit:

```
static void test_wlan_get_txpwrlimit(int argc, char **argv)
 {
   wifi_SubBand_t subband;
   wlan_txpwrlimit_t txpwrlimit;
   int i, j;
   if (argc != 2)
   {
     dump_wlan_get_txpwrlimit_usage();
     return;
   }
   subband = (wifi_SubBand_t)strtol(argv[1], NULL, 16);
   if (subband != SubBand_2_4_GHz
#ifdef CONFIG_5GHz_SUPPORT
   && subband != SubBand_5_GHz_0 && subband != SubBand_5_GHz_1 &&
   subband != SubBand_5_GHz_2 &&
   subband != SubBand_5_GHz_3
#endif
   )
   {
     dump_wlan_get_txpwrlimit_usage();
     return;
   }
   int rv = wlan_get_txpwrlimit(subband, &txpwrlimit);
   if (rv != WM_SUCCESS)
      PRINTF("Unable to get TX PWR Limit configuration\r\n");
   else
   {
     PRINTF("Get txpwrlimit: sub_band=%x \r\n", txpwrlimit.subband);
     for (i = 0; i < 1; i++)
     {
       PRINTF("StartFreq: %d\r\n",
        txpwrlimit.txpwrlimit_config[i].chan_desc.start_freq);
       PRINTF("ChanWidth: %d\r\n",
        txpwrlimit.txpwrlimit_config[i].chan_desc.chan_width);
       PRINTF("ChanNum: %d\r\n",
        txpwrlimit.txpwrlimit_config[i].chan_desc.chan_num);
       PRINTF("Pwr:");
       for (j = 0; j < txpwrlimit.txpwrlimit_config[i].num_mod_grps; j++)
       {
        if (j == (txpwrlimit.txpwrlimit_config[i].num_mod_grps - 1))
          PRINTF("%d,%d",
          txpwrlimit.txpwrlimit_config[i].txpwrlimit_entry[j].mod_group,
          txpwrlimit.txpwrlimit_config[i].txpwrlimit_entry[j].tx_power);
        else
          PRINTF("%d,%d,",
          txpwrlimit.txpwrlimit_config[i].txpwrlimit_entry[j].mod_group,
          txpwrlimit.txpwrlimit_config[i].txpwrlimit_entry[j].tx_power);
       }
       PRINTF("\r\n");
     }
   }
}
```

- Rebuild the SDK and execute the command `wlan-get-txpwrlimit`.

The three code samples below show examples of `wlan-get-txpwrlimit` command outputs.

```
# wlan-get-txpwrlimit 00
Get txpwrlimit:
sub_band=0
StartFreq: 2407
ChanWidth: 20
ChanNum: 1
Pwr:0,19,1,15,2,15,3,15,4,13,5,13,6,13,7,0,8,0,9,0,10,13,11,0,12,0,13,0,
14,0,15,0,16,13,17,0,18,0,19,0
```

```
# wlan-get-txpwrlimit 10
Get txpwrlimit: sub_band=10
StartFreq: 5000
ChanWidth: 20
ChanNum:   36
Pwr:1,20,2,20,3,20,4,20,5,20,6,19,7,0,8,0,9,0,10,16,11,0,12,0,13,0,14,0,
15,0,16,16,17,0,18,0,19,0
```

```
# wlan-get-txpwrlimit 12
Get txpwrlimit: sub_band=12
StartFreq: 5000
ChanWidth: 20
ChanNum:   149
Pwr:1,20,2,20,3,18,4,21,5,21,6,17,7,0,8,0,9,0,10,12,11,0,12,0,13,0,14,0,
15,0,16,10,17,0,18,0,19,0
```

4 Channel list and scan type

The command-line interface is provided in *wifi_cert* sample application to set/get the channel list and scan type. The *wifi_cert* application is enabled for RW61x platform.

The default channel list and scan configuration are described in Table 4.

Table 4. Default channel list and scan configuration

Bandwidth (GHz)	Channels	Scan
2.4	1-11	Active
2.4	12-14	Passive
5	36-48	Active
5	52-64	Passive
5	100-144	Passive
5	149-165	Passive

In passive scan, the client radio listens on each channel for beacons sent periodically by an AP. Passive scan usually lasts longer than active scan. During an active scan, the client radio transmits a probe request and listens for a probe response from an AP.

The channel list and scan type are set using `wlan_set_chanlist` API. This function uses `chanlist_2g_cfg` and `chanlist_5g_cfg` data structures, both defined in *<sdk_root>/middleware/wifi/wlcmgr/wlan_txpwrlimit_cfg_WW.h*.

`chanlist_2g_cfg` is used for 2.4 GHz band (see below) and `chanlist_5g_cfg` is used for 5 GHz.

```
static wlan_chanlist_t chanlist_2g_cfg = {
    .num_chans = 2,
    .chan_info[0] =
        {
            .chan_num                    = 1,
            .chan_freq                   = 2412,
            .passive_scan_or_radar_detect = false,
        },
    .chan_info[1] =
        {
            .chan_num                    = 12,
            .chan_freq                   = 2467,
            .passive_scan_or_radar_detect = true,
        },
};
```

The parameters used in those structures are:

Parameter	Description
num_chans	Number of channels showed in the list. The number must be the same as the number of chan_info elements.
chan_num	Number of the channel
chan_freq	Frequency of the channel
passive_scan_or_radar_detect	Informs the driver whether the channel is a passive channel (Band B/G) or a DFS channel (Band A) for which radar detection is run. If set to "True", only passive scanning is run for the channel. True: Passive Scan/DFS channel False: Active scanning

The following code sample shows chanlist_5g_cfg data structure (for 5 GHz band):

```
static wlan_chanlist_t chanlist_5g_cfg = {
    .num_chans = 2,
    .chan_info[0] =
        {
            .chan_num                    = 36,
            .chan_freq                   = 5180,
            .passive_scan_or_radar_detect = false,
        },
    .chan_info[1] =
        {
            .chan_num                    = 100,
            .chan_freq                   = 5500,
            .passive_scan_or_radar_detect = true,
        },
};
```

Two channel scan configurations are defined (num_chans parameter is set to 2):

- channel 36, frequency 5180, using active scan
- channel 100, frequency 5500, using passive scan

There are two ways to set/get the scan type:

- using the API
- using the command line

The following sections describe both methods.

4.1 Customization using API calls

To run `wlan_set_chanlist` command, use the driver APIs with the required arguments.

Syntax: `int wlan_set_chanlist(wlan_chanlist_t *chanlist)`

Where:

Parameter	Description
`[in] chanlist`	A pointer to `wlan_chanlist_t` structure to supply the channel list configuration. This structure is explained below.

The `wlan_set_chanlist` command is defined in the file *<sdk_root>/middleware/wifi/wlcmgr/ wlan.c*.

Returns:

WM_SUCCESS: the call is successful

WM_FAIL: the call failed

The following is an example of use of `wlan_set_chanlist`; the function takes the aforementioned structures `wlan_chanlist_t` as input parameter.

For 2.4 GHz:

```
ret = wlan_set_chanlist(&chanlist_2g_cfg);
if (ret != WM_SUCCESS)
    PRINTF("Cannot set Channel List 2G\r\n");
else
    PRINTF("Successfully set 2G Channel List\r\n");
```

For 5 GHz:

```
ret = wlan_set_chanlist(&chanlist_5g_cfg);
if (ret != WM_SUCCESS)
    PRINTF("Cannot set Channel List 5G\r\n");
else
    PRINTF("Successfully set 5G Channel List\r\n");
```

As shown in the sample code, the implementation provides a return status for the function to check whether the function call is successful or not.

To run the get operation `wlan_get_chanlist`, use the driver APIs with the required arguments.

Syntax: `int wlan_get_chanlist(wlan_chanlist_t *chanlist)`

Where:

Parameter	Description
`[out] chanlist`	A pointer to `wlan_chanlist_t` structure to get the channel list configuration.

The `wlan_get_chanlist` command is defined in file located in *<sdk_root>/middleware/wifi/wlcmgr/wlan.c*.

Returns:

WM_SUCCESS: the call is successful

WM_FAIL: the call failed

Note: *The channels that are configured using this API should be from the list of channels under World Wide Safe Mode (WWSM).*

The following is an example of use of `wlan_get_chanlist;`. The function takes the aforementioned `wlan_chanlist_t` structure as output parameter.

```
wlan_chanlist_t chanlist;
memset(&chanlist, 0x00, sizeof(wlan_chanlist_t));
int rv = wlan_get_chanlist(&chanlist);
if (rv != WM_SUCCESS)
   PRINTF(Ünable to get channel list configuration\r\n”);
else {
   int i;

   PRINTF("-----------------------------------------------------\r\n");
   PRINTF("Number of channels configured: %d\r\n",chanlist.num_chans);
   PRINTF("\r\n");
   for (i = 0; i < chanlist.num_chans; i++){
      PRINTF("ChanNum: %d\t", chanlist.chan_info[i].chan_num);
      PRINTF("ChanFreq: %d\t", chanlist.chan_info[i].chan_freq);
      PRINTF("%s",chanlist.chan_info[i].passive_scan_or_radar_detect ?
      "Passive" : "Active");
      PRINTF("\r\n");
   }
}
```

As shown in the sample code, the implementation provides a return status for the function to check whether the function call is successful or not.

4.2 Customization using the command-line interface

By default, the current implementation does not include the CLI commands to set the scan type. Follow the steps below to add the feature.

- Edit the file: *<sdk root>/middleware/wifi/wlcmgr/wlan_enhanced_tests.c*
- Add the following command:

```
{"wlan-set-chanlist", NULL, test_wlan_set_chanlist},
```

The example below shows the added line in **bold**.

```
static struct cli_command tests[] = {
    .
    .
    .
    {"wlan-set-chanlist", NULL, test_wlan_set_chanlist},
};
```

- Define the channel list structure with the required values for 2.4 GHz and 5 GHz, as shown below

For 2.4 GHz

```
static wlan_chanlist_t chanlist_2g_cfg = {
    .num_chans = 2,
    .chan_info[0] =
        {
            .chan_num                    = 1,
            .chan_freq                   = 2412,
            .passive_scan_or_radar_detect = false,
        },
    .chan_info[1] =
        {
            .chan_num                    = 12,
            .chan_freq                   = 2467,
            .passive_scan_or_radar_detect = true,
        },
};
```

For 5 GHz

```
#ifdef CONFIG_5GHz_SUPPORT
static wlan_chanlist_t chanlist_5g_cfg = {
    .num_chans = 2,
    .chan_info[0] =
        {
            .chan_num                    = 36,
            .chan_freq                   = 5180,
            .passive_scan_or_radar_detect = false,
        },
    .chan_info[1] =
        {
            .chan_num                    = 100,
            .chan_freq                   = 5500,
            .passive_scan_or_radar_detect = true,
        },
};
#endif
```

- Define the handler function that is triggered when the user executes the command `wlan-set-chanlist`

```
static void test_wlan_set_chanlist(int argc, char **argv)
{
   int ret = WM_SUCCESS;

   ret = wlan_set_chanlist(&chanlist_2g_cfg);
   if (ret != WM_SUCCESS)
      PRINTF("Cannot set Channel List 2G\r\n");
   else
      PRINTF("Successfully set 2G Channel List\r\n");
#ifdef CONFIG_5GHz_SUPPORT
   ret = wlan_set_chanlist(&chanlist_5g_cfg);
   if (ret != WM_SUCCESS)
      PRINTF("Cannot set Channel List 5G\r\n");
   else
      PRINTF("Successfully set 5G Channel List\r\n");
#endif
}
```

- Rebuild the SDK and run the command `wlan-set-chanlist` and read the output.

```
# wlan-set-chanlist
Successfully set 2G Channel List
Successfully set 5G Channel List
```

This code sample confirms that the channel list and scan information have been set.

To **verify** the channel list setting, create the `wlan-get-chanlist` command as detailed below.

- Edit the file *<sdk_root>/middleware/wifi/wlcmgr/ wlan_enhanced_tests.c*
- Find the tests structure and add the following line:

```
{"wlan-get-chanlist", NULL, test_wlan_get_chanlist},
```

The example below shows the added line in bold.

```
static struct cli_command tests[] = {
   .
   .
   .
   {"wlan-get-chanlist", NULL, test_wlan_get_chanlist},
};
```

- Define the handler for `wlan_get_channel_list` command

```
static void test_wlan_get_chanlist(int argc, char **argv)
{
   wlan_chanlist_t chanlist;
   memset(&chanlist, 0x00, sizeof(wlan_chanlist_t));
   int rv = wlan_get_chanlist(&chanlist);
   if (rv != WM_SUCCESS)
     PRINTF("Unable to get channel list configuration\r\n");
   else
     {
       int i;
       PRINTF("----------------------------------------------------------------\r\n");
       PRINTF("Number of channels configured: %d\r\n", chanlist.num_chans);
       PRINTF("\r\n");
       for (i = 0; i < chanlist.num_chans; i++){
         PRINTF("ChanNum: %d\t", chanlist.chan_info[i].chan_num);
         PRINTF("ChanFreq: %d\t", chanlist.chan_info[i].chan_freq);
         PRINTF("%s",chanlist.chan_info[i].passive_scan_or_radar_detect ?
         "Passive" : "Active");
         PRINTF("\r\n");
     }
}
```

- Rebuild the SDK and run the command `wlan-get-chanlist`. The output is similar to what is shown below.

```
# wlan-get-chanlist
----------------------------------------------------------------
Number of channels configured: 4
ChanNum: 1      ChanFreq: 2412  Active
ChanNum: 12     ChanFreq: 2467  Passive
ChanNum: 36     ChanFreq: 5180  Active
ChanNum: 100    ChanFreq: 5500  Passive
```

5 Acronyms and abbreviations

Table 5. Acronyms and abbreviations

Acronyms	Definition
AP	Access Point
CLI	Command Line Interface
CRDA	Central Regulatory Domain Agent
EEPROM	Electrically Erasable Programmable Read-Only Memory
FW	Firmware
IE	Information Element
OTP	One Time Programmable
TRPC	Transient receptor potential, canonical
WLAN	Wireless Local Area Network

6 Legal information

6.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

6.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

6.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Tables

Contents

For more information, please visit: http://www.nxp.com

Date of release: 10 August 2022
Document identifier: AN12918