



Hi3861 V100 / Hi3861L V100 Mesh Software

Development Guide

Issue	01
Date	2020-04-30

Copyright © HiSilicon (Shanghai) Technologies Co., Ltd. 2020. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of HiSilicon (Shanghai) Technologies Co., Ltd.

Trademarks and Permissions



HISILICON, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided “AS IS” without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

HiSilicon (Shanghai) Technologies Co., Ltd.

Address: New R&D Center, 49 Wuhe Road,
Bantian, Longgang District,
Shenzhen 518129 P. R. China

Website: <https://www.hisilicon.com/en/>

Email: support@hisilicon.com



About This Document

Purpose

This document describes the Wi-Fi mesh software development kit (SDK) of Hi3861 V100 and Hi3861L V100, including the SDK architecture, API functions, and usage.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3861	V100
Hi3861L	V100



Intended Audience

The document is intended for:




- Technical support engineers
- Software development engineers

Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
	Indicates a hazard with a high level of risk which, if not avoided, will result in death or serious injury.
	Indicates a hazard with a medium level of risk which, if not avoided, could result in death or serious injury.



Symbol	Description
 CAUTION	Indicates a hazard with a low level of risk which, if not avoided, could result in minor or moderate injury.
 NOTICE	Indicates a potentially hazardous situation which, if not avoided, could result in equipment damage, data loss, performance deterioration, or unanticipated results. NOTICE is used to address practices not related to personal injury.
 NOTE	Supplements the important information in the main text. NOTE is used to address information not related to personal injury, equipment damage, and environment deterioration.

Change History

Issue	Date	Change Description
01	2020-04-30	This issue is the first official release.
00B01	2020-04-10	This issue is the first draft release.



Contents

About This Document.....	i
1 Introduction.....	1
1.1 Background.....	1
1.2 Features.....	1
1.3 SDK Architecture.....	2
1.4 Topology and Node Roles.....	2
2 Original APIs.....	4
2.1 Layer 2 APIs.....	4
2.1.1 MBR/MG Node.....	4
2.1.1.1 Overview.....	4
2.1.1.2 Node Starting.....	4
2.1.1.3 Scanning.....	6
2.1.1.4 Connection.....	7
2.1.2 Mesh STA Node.....	9
2.1.2.1 Overview.....	9
2.1.2.2 Node Starting.....	10
2.1.2.3 Scanning.....	11
2.2 Layer 3 Network Access APIs.....	12
2.2.1 Election.....	12
2.2.1.1 Starting Election.....	12
2.2.1.2 Notifying a Node of the Network Access Role.....	13
2.2.1.3 Stopping Election.....	14
2.2.1.4 Sample Code.....	14
2.2.2 Setting the Vendor OUI.....	15
3 Automatic Networking.....	16
3.1 Overview.....	16
3.2 Development Process.....	17
3.3 Precautions.....	18
3.4 Programming Example.....	18



1 Introduction

[1.1 Background](#)

[1.2 Features](#)

[1.3 SDK Architecture](#)

[1.4 Topology and Node Roles](#)

1.1 Background

The wireless mesh network is a revolutionary technology innovation for the traditional wireless local area network (WLAN). From the topology perspective, the wireless mesh network is a many-to-many network. It enables large-scale high-speed communication through multi-hop that is impossible using traditional WLAN, in addition to features such as self-networking and self-healing. It extends the application scope of WLANs from hotspots to hot zones, with reduced dependency on wired networks. A wireless mesh network is a multi-hop network. Different from a traditional single-hop network, it has access points (APs) that provide user access and forward wireless data. Multiple APs form a mesh structure. Signals are routed from one AP to another in the network, and then transmitted to the wired network through the AP connected to the fixed line.

1.2 Features

The Wi-Fi mesh solution of Hi3861 V100/Hi3861L V100 is a three-layer routing (Route-Over) solution for IoT. It supports the mesh topology with low-power nodes and general routing nodes and provides the following features:

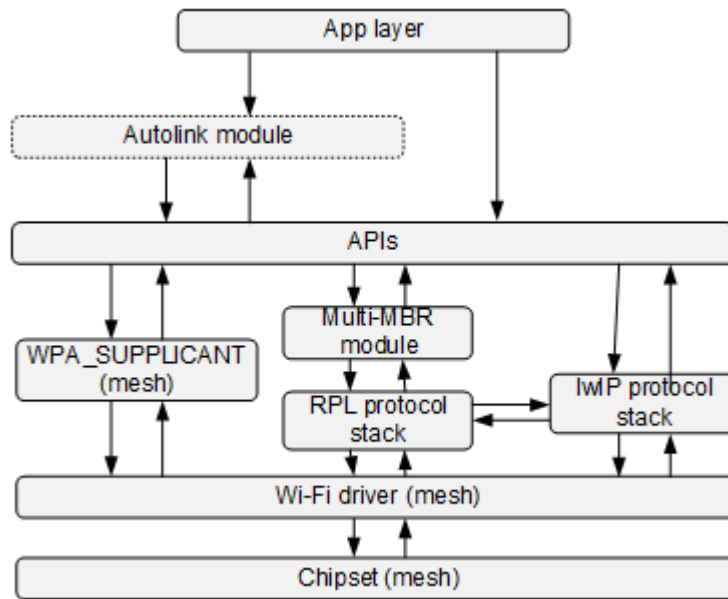
- Automatic networking
- Fast route switching
- IPv6 supported
- Group control
- Encryption
- Energy saving (for leaf nodes only)

- Multiple portals
- Loop prevention, detection, and repair
- Clock synchronization
- Beacon conflict detection

1.3 SDK Architecture

Figure 1-1 shows the software architecture of the mesh SDK.

Figure 1-1 Mesh SDK software architecture



The following describes the software architecture of the mesh SDK:

- App layer: secondary development with APIs
- Autolink module: developed based on the original APIs provided by the SDK
- APIs: standard interfaces provided by the mesh SDK
- WPA_SUPPLICANT: Wi-Fi management module, which provides scanning management and association state machine related to mesh
- Multi-MBR module: provides functions such as mesh MBR arbitration, election, and multi-MBR synchronization
- RPL protocol stack: RFC6550, short for IPv6 Routing Protocol for Low-Power and Lossy Networks, is a mesh core routing protocol
- IwIP protocol stack: network protocol stack
- Wi-Fi driver: 802.11 protocol implementation module, which provides mesh-related functions such as link layer scanning and association
- Chipset: chipset, including the MAC, PHY, and RF modules

1.4 Topology and Node Roles

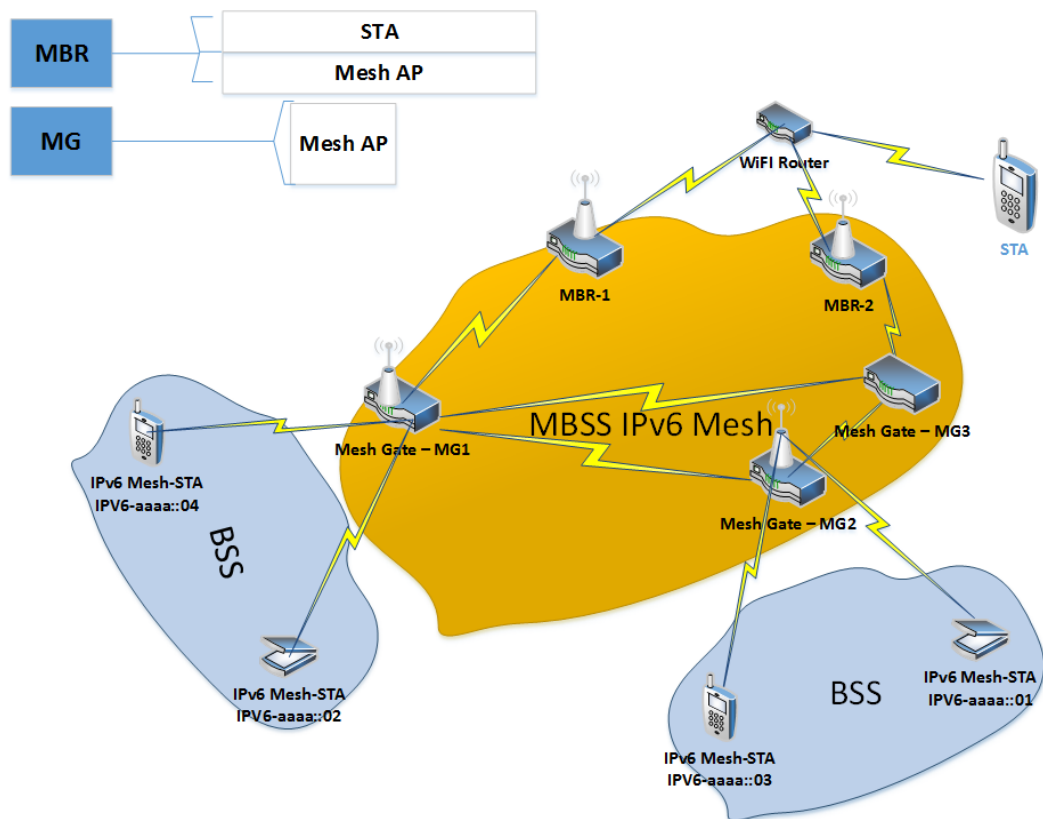
This mesh networking solution has three node roles:

- **Mesh border router (MBR)**
An MBR is a node that connects a mesh network to an external network. It consists of an STA that is directly connected to a router and mesh APs that participate in mesh backbone networking.
- **Mesh gate (MG)**
An MG also participates in mesh backbone networking. It can be directly connected to other mesh APs or MBRs, or be searched and connected by mesh STAs to form a basic service set (BSS) network. Serving as a bridge between the BSS and the mesh backbone network, an MG is applicable to nodes that have power consumption and latency requirements.
- **Mesh STA**
A mesh STA uses the 802.11 association process to associate with the MG and access the network by obtaining the IP address through the MG. Low power consumption is supported.

NOTE

The MBR and MG belong to the mesh backbone network. As a low-power node, a mesh STA only functions as a leaf node and does not participate in mesh route construction or forwarding.

Figure 1-2 Mesh topology





2 Original APIs

[2.1 Layer 2 APIs](#)

[2.2 Layer 3 Network Access APIs](#)

2.1 Layer 2 APIs

2.1.1 MBR/MG Node

2.1.1.1 Overview

MBR and MG are mesh backbone network nodes that form a mesh backbone network and have routing tables. They are applicable to devices that emphasize more on reduced latency than power consumption.

NOTE

This section describes only the startup, scanning, and association processes of the MBR and MG nodes on L2. On a mesh network, an MBR node must be started before the MG and MSTA nodes are mounted to the mesh network. After the L2 node is started successfully, the L3 network can be set up. For details about the how to set up the L3 network, see the *Hi3861 V100/Hi3861L V100 RPL Development Guide*.

2.1.1.2 Node Starting

Application Scenario

A mesh node needs to set up a mesh network or join an existing mesh network as the MBR/MG role.

Function

[Table 2-1](#) describes the MBR/MG node startup APIs provided by the mesh SDK.



Table 2-1 Description of the MBR/MG node startup APIs

API Name	Description
hi_wifi_mesh_disconnect	Disconnects the user with the specified MAC address from the mesh network.
hi_wifi_mesh_start	Starts the mesh interface.
hi_wifi_mesh_stop	Stops the mesh interface.
hi_wifi_protocol_chn_switch_enable	Enables channel switching on the entire mesh network.
hi_wifi_protocol_chn_switch	Sets channel switching parameters for the entire mesh network.
hi_wifi_get_mesh_node_info	Queries mesh node information.

Development Process

The typical process of starting an MBR node is as follows:

- Step 1** Start the co-existing STA by calling **hi_wifi_sta_start**.
- Step 2** Start the co-existing mesh AP by calling **hi_wifi_mesh_start**.
- End

The typical process of starting an MG node is as follows:

- Step 1** Start the mesh AP by calling **hi_wifi_mesh_start**.
- End

Code Sample

Sample 1: starting an MBR (STA+mesh AP).

```
hi_char g_meshid[12] = {"mesh"};
hi_s32 g_router_chan = 11;
hi_s32 hi_mbr_start(hi_s32 argc, const hi_char* argv[])
{
    hi_s32 ret;
    hi_s32 len_sta;
    hi_s32 len_mesh;
    hi_char sta_ifname[WIFI_IFNAME_MAX_SIZE + 1] = {0};
    hi_char mesh_ifname[WIFI_IFNAME_MAX_SIZE + 1] = {0};
    errno_t rc;
    hi_wifi_mesh_config wpa_mesh_start = {0};

    ret = hi_wifi_sta_start(sta_ifname, &len_sta);
    if (ret != HISI_OK) {
        printf("hi_wifi_sta_start fail!\n");
    }
    hi_wifi_register_event_callback(hi_wifi_wpa_event_cb);
    ...
    ...
    wpa_mesh_start.auth = HI_WIFI_SECURITY_OPEN;
```



```
wpa_mesh_start.channel = (hi_uchar)g_router_chan;
memcpy_s(wpa_mesh_start.ssid, HI_WIFI_MAX_SSID_LEN + 1, g_meshid, strlen(g_meshid)
+ 1);

ret = hi_wifi_mesh_start(&wpa_mesh_start, mesh_ifname, &len_mesh);
if (ret != HISI_OK) {
    printf("mesh start fail!\n");
}
printf("mesh start succ!\n");
}
```

Sample 2: starting an MG (mesh AP).

```
hi_char g_meshid[12] = {"mesh"};
hi_s32 g_router_chan = 11;
hi_s32 hi_mg_start(hi_s32 argc, const hi_char* argv[])
{
    hi_s32 ret;
    hi_s32 len_mesh;
    hi_char mesh_ifname[WIFI_IFNAME_MAX_SIZE + 1] = {0};
    errno_t rc;
    hi_wifi_mesh_config wpa_mesh_start = {0};

    ...
    ...
    wpa_mesh_start.auth = HI_WIFI_SECURITY_OPEN;
    wpa_mesh_start.channel = (hi_uchar)g_router_chan;
    memcpy_s(wpa_mesh_start.ssid, HI_WIFI_MAX_SSID_LEN + 1, g_meshid, strlen(g_meshid)
+ 1);

    ret = hi_wifi_mesh_start(&wpa_mesh_start, mesh_ifname, &len_mesh);
    if (ret != HISI_OK) {
        printf("mesh start fail!\n");
    }
    hi_wifi_register_event_callback(hi_wifi_wpa_event_cb);
    printf("mesh start succ!\n");
}
```

2.1.1.3 Scanning

Application Scenario

An MBR/MG node scans for other available mesh nodes in the current node range.

Function

[Table 2-2](#) describes the MBR/MG node scanning APIs provided by the mesh SDK.

Table 2-2 Description of the MBR/MG node scanning APIs

API Name	Description
hi_wifi_mesh_scan	Enables mesh AP scanning.
hi_wifi_mesh_advance_scan	Enables advanced scanning for a mesh AP.
hi_wifi_mesh_scan_results	Obtains the network scanning result of a mesh AP.



Development Process

The typical scanning process of an MBR node is as follows:

Step 1 Enable the STA to initiate scanning by calling **hi_wifi_sta_scan**.

Step 2 View the STA scanning result by calling **hi_wifi_sta_scan_results**.

----End

The typical scanning process of an MG node is as follows:

Step 1 Initiate scanning by calling **hi_wifi_mesh_scan** or **hi_wifi_mesh_advance_scan** (special scanning parameters: SSID, BSSID, SSID length, channel, and whether prefix scanning is used)

Step 2 View the scanning result of a mesh AP by calling **hi_wifi_mesh_scan_results**.

----End

Code Sample

Sample: scanning a mesh AP and querying the scanning result

```
hi_void hi_mr_scan(hi_void)
{
    ...
    pst_results = malloc(sizeof(hi_wifi_mesh_scan_result_info) * WIFI_SCAN_AP_LIMIT);
    if (pst_results == HI_NULL) {
        printf("Alloc mem fail!\n");
    }
    ret = hi_wifi_mesh_scan(HI_NULL);
    if (ret != HISI_OK) {
        printf("fail to scan!\r\n");
    }
    memset_s(pst_results, (sizeof(hi_wifi_mesh_scan_result_info) * WIFI_SCAN_AP_LIMIT),
        0, (sizeof(hi_wifi_mesh_scan_result_info) * WIFI_SCAN_AP_LIMIT));
    if (hi_wifi_mesh_ap_scan_results(pst_results, &num) != HISI_OK) {
        printf("Fail to get mesh scan results!\r\n");
    }
    printf("Scan Result num:%d\n", num);
    for (ul_loop = 0; (ul_loop < num) && (ul_loop < WIFI_SCAN_AP_LIMIT); ul_loop++) {
        printf("Scan Result:%s, %d\n", pst_results[i].ssid, pst_results[i].rssi);
    }
    free(pst_results);
}
```

2.1.1.4 Connection

Application Scenario

An MBR/MG node is associated with the specified target mesh node.

Function

Table 2-3 describes the MBR/MG node connection APIs provided by the mesh SDK.



Table 2-3 Description of the MBR/MG node connection APIs

API Name	Description
hi_wifi_mesh_connect	Specifies the MAC address of a mesh node.
hi_wifi_mesh_set_accept_peer	Specifies whether a mesh node supports mesh peer connections.
hi_wifi_mesh_set_accept_sta	Specifies whether a mesh node supports mesh STA connections.
hi_wifi_mesh_get_connected_peer	Obtains information about the connected mesh node.
hi_wifi_add_usr_app_ie	Adds the user IE field to the mesh management frame.
hi_wifi_delete_usr_app_ie	Deletes the user IE field from the mesh management frame.
hi_wifi_mesh_disconnect	Disassociates a specified node from a mesh node.

Development Process

The typical process of an associated node is as follows:

- Step 1** Set the mesh AP to receive association from the peer node by calling **hi_wifi_mesh_set_accept_peer**.
- Step 2** Set the mesh AP to receive association from a mesh STA by calling **hi_wifi_mesh_set_accept_sta**.
- Step 3** (Optional) Add the user-defined IE field to the mesh AP by calling **hi_wifi_add_usr_app_ie**.
- Step 4** Obtain information about the node associated with the mesh AP by calling **hi_wifi_mesh_get_connected_peer**.

----End

The typical process of associating a node is as follows:

- Step 1** (Optional) Add the user-defined IE field to the mesh AP by calling **hi_wifi_add_usr_app_ie**.
- Step 2** Initiate an association by calling **hi_wifi_mesh_connect**.
- Step 3** Obtain information about the node associated with a mesh node by calling **hi_wifi_mesh_get_connected_peer**.

----End

The typical process of disassociating a node is as follows:



Step 1 Disassociate from a mesh node from the peer node by calling **hi_wifi_mesh_disconnect**.

----End

Code Sample

Sample 1: connecting to a mesh AP and querying the number of users

```
hi_void hi_mr_connect(hi_void)
{
    hi_u8 peer_num;
    hi_s32 ret;
    hi_u8 *mac;
    ...
    ret = hi_wifi_mesh_disconnect(mac, 6);
    if (ret != HISI_OK) {
        printf("mesh_disconnect failed.\n");
    }
    ...
    hi_wifi_mesh_peer_info *peer_list = malloc(sizeof(hi_wifi_mesh_peer_info) * peer_num);
    if (peer_list == NULL) {
        printf("malloc res fail!\n");
    }
    ret = hi_wifi_mesh_get_connected_peer(peer_list, &peer_num);
    if (ret != HISI_OK) {
        printf("get connect peer fail!\n");
    }
    printf("Get connected peer succ,peer num[%d]\n", peer_num);
}
```

Sample 2: disconnecting from a mesh AP

```
hi_void hi_mr_disconnect(hi_void)
{
    hi_s32 ret;
    hi_u8 *mac;
    ...
    ret = hi_wifi_mesh_disconnect(mac, 6);
    if (ret != HISI_OK) {
        printf("mesh_disconnect failed.\n");
    }
    ...
}
```

2.1.2 Mesh STA Node

2.1.2.1 Overview

A mesh STA is an STA with the mesh capability that can join a mesh network as an end node. It is essentially the same as a common STA, but usually applies to a device that emphasizes more on energy saving.

NOTE

This section describes only the APIs added or modified for common STAs with the mesh capability. For details about other STA APIs, see *Hi3861 V100/Hi3861 LV100 Wi-Fi Software Development Guide*.



2.1.2.2 Node Starting

Application Scenario

A mesh node needs to set up a mesh network or join an existing mesh network as the mesh STA role.

Function

Table 2-4 describes the MBR/MG node startup API provided by the mesh SDK.

Table 2-4 Description of the MBR/MG node startup API

API Name	Description
hi_wifi_set_mesh_sta	Enables a common STA to connect to a mesh network.

Development Process

The typical process of starting a mesh STA is as follows:

Step 1 Enable a common STA to support mesh networking by calling **hi_wifi_set_mesh_sta**.

Step 2 Start the STA by calling **hi_wifi_sta_start**.

----End

Code Sample

Sample: enabling a mesh STA

```
hi_void hi_msta_start(hi_void)
{
    hi_s32 ret;
    hi_s32 len = 0;
    hi_char ifname[WIFI_IFNAME_MAX_SIZE + 1] = {0};

    ret = hi_wifi_set_mesh_sta(1);
    if (ret != HISI_OK) {
        printf("Set mesh sta flag fail!\n");
    }

    ret = hi_wifi_sta_start(ifname, &len);
    if (ret != HISI_OK) {
        printf("Mesh sta start fail!\n");
    }
    printf("Mesh sta start succ!\n");
}
```



2.1.2.3 Scanning

Application Scenario

A mesh STA node scans for other available mesh nodes in the current node range.

Function

[Table 2-5](#) describes the mesh STA node scanning APIs provided by the mesh SDK.

Table 2-5 Description of the mesh STA node scanning APIs

API Name	Description
hi_wifi_mesh_sta_scan	Starts mesh STA scanning.
hi_wifi_mesh_sta_advance_scan	Starts special mesh STA scanning with restrictions.
hi_wifi_mesh_sta_scan_results	Views the scanning result of the mesh STA.

Development Process

The typical process of scanning a mesh STA is as follows:

- Step 1** Start mesh STA scanning by calling **hi_wifi_mesh_sta_scan** or **hi_wifi_mesh_sta_advance_scan** (restricted scanning).
 - Step 2** View the scanning result of a mesh STA by calling **hi_wifi_mesh_sta_scan_results**.
- End

Code Sample

Sample 1: common scanning

```
hi_void hi_msta_scan(hi_void)
{
    hi_s32 ret;
    hi_wifi_mesh_scan_result_info *pst_results;
    hi_u32 num = WIFI_SCAN_AP_LIMIT;
    ...
    ret = hi_wifi_mesh_sta_scan();
    if (ret != HISI_OK) {
        printf("Scan fail!\n");
    }

    pst_results = malloc(sizeof(hi_wifi_mesh_scan_result_info) * WIFI_SCAN_AP_LIMIT);
    if (pst_results == HI_NULL) {
        printf("Alloc mem fail!\n");
    }

    if (hi_wifi_mesh_sta_scan_results(pst_results, &num) != HISI_OK) {
        free(pst_results);
        printf("Get scan result fail!\n");
    }
}
```




```
}  
  
printf("Get scan result succ!\n");  
free(pst_results);  
return;  
}
```

Sample 2: special scanning (with a specified channel)

```
hi_void channel_mesh_scan(hi_void)  
{  
    int ret;  
    hi_wifi_scan_params scanParams = {0};  
  
    scanParams.channel = 11; /*Specify scanning on channel 11*/  
    ret = hi_wifi_mesh_advance_scan(&scanParams);  
    if (ret != HISI_OK) {  
        printf("channel_mesh_scan fail.\n");  
    }  
}
```

2.2 Layer 3 Network Access APIs

2.2.1 Election

NOTE

- A mesh network has only one MBR, and not all nodes can function as MBRs. Therefore, election is required. When a device triggers election, the device sends an election message to all devices associated with the same router. The election message carries the organizationally unique identifier (OUI) of the vendor. For the election among devices of the same vendor, if the MBR already exists, the new node can only serve as the MG.
- After the election algorithm is started, the election module assigns a proper role (MBR or MG) to the node based on the current network status.

2.2.1.1 Starting Election

After a node is associated with a Wi-Fi router, the node determines whether the RSSI reaches the threshold. If the RSSI reaches the threshold, the node calls the election API to participate in the election. If no other node is elected with the MBR node in the first accessed mesh network, the MBR node is used by default.

Prototype	hi_s32 mesh_election_start(mbr_election_info* info);
Description	When a device is qualified to function as an MBR (has a fixed location, supplied by a non-battery power supply, and emphasizes less on low power consumption), but whether the number of MBRs on the network reaches the upper limit (8) is undetermined, you can execute this function to determine whether the device can be used as one of the MBRs.



Inputs	info : election information struct variable. The members are as follows: <ul style="list-style-type: none">• call_back: callback function after the election result is generated. The argument of the callback function is of the node type.• router_rssi: RSSI value of the Wi-Fi router
Outputs	None
Returns	<ul style="list-style-type: none">• HISI_OK: success• HISI_FAIL: failure
Note	<ul style="list-style-type: none">• The function is executed after the associated Wi-Fi router is determined and the IP address is obtained. Otherwise, the current node does not participate in the election competing with other nodes.• After the election is complete, the module on the layer 3 network calls the entry callback function. The argument is of the node type.• If the node type is MG, the connection with the Wi-Fi router needs to be removed, and the current node reconnects to the MBR as MG.• The election function is non-blocking. After the function is executed, the election process is not complete. After the election process is complete, the callback function input by the func parameter is called. The entry of the callback function is the election result. You can select the role of a node on a mesh network based on the election result.

2.2.1.2 Notifying a Node of the Network Access Role

After a node is elected, this function can be called to notify the election module of the node role in the mesh network.

Joining a Mesh Network

Prototype	hi_s32 mesh_election_notify_role(uint8_t mode)
Description	After determining the role of the device that is expected to join in the mesh network, the device notifies the election module of the node role.
Inputs	mode : access type, either MBR or MG
Outputs	None
Returns	<ul style="list-style-type: none">• HISI_OK: success• HISI_FAIL: failure



Note	This operation must be performed after the Wi-Fi router is associated.
-------------	--

2.2.1.3 Stopping Election

If a node determines to access the network as a non-MBR role, the node may stop election and free resources used by the election module.

Prototype	hi_s32 mesh_election_stop(hi_void)
Description	Stops node election.
Inputs	None
Outputs	None
Returns	<ul style="list-style-type: none">• HISI_OK: success• HISI_FAIL: failure
Note	If the MBR identity needs to be retained, do not call this API.

2.2.1.4 Sample Code

Sample: setting the OUI and starting node election

```
hi_void election_call(const hi_u8 mode)
{
    if (mode == HI_MBR) { // MBR
        mesh_election_notify_role(HI_MBR);
    }
    if (mode == HI_MG) { // MG
        printf("election result is mg");
    }
    return;
}

hi_void demo_join_mesh(hi_void)
{
    //Associate the Wi-Fi router first.
    mbr_election_info info = {0};
    info.call_back = hi_mesh_election_callback;
    info.router_rssi = g_connected_router_rssi;
    mesh_election_start(&info);
    return;
}

hi_void mbr_main_thread(hi_void) /*Device startup entry*/
{
    hi_u8 oui[OUI_LENGTH] = {1, 2, 3};
    hi_mesh_set_oui(oui, OUI_LENGTH);
    demo_join_mesh();
    return;
}
```



2.2.2 Setting the Vendor OUI

This API can be used to set the vendor OUI to which a product belongs. Devices with different OUIs cannot be deployed on the same mesh network.

Prototype	hi_s8 hi_mesh_set_oui(const hi_u8 *oui, hi_u8 oui_len)
Description	Sets the vendor OUI.
Inputs	oui : 3-byte OUI oui_len : OUI length, fixed at 3 bytes
Outputs	None
Returns	<ul style="list-style-type: none">• HISI_OK: success• HISI_FAIL: failure
Note	This API can be called only before node networking. Otherwise, the networking result may be incorrect.



3 Automatic Networking

[3.1 Overview](#)

[3.2 Development Process](#)

[3.3 Precautions](#)

[3.4 Programming Example](#)

3.1 Overview

The Autolink module for automatic networking is developed based on standard mesh APIs in the mesh SDK and is used for node role arbitration. The Autolink module provides the following functions:

- Node starting
 - User-specified: MBR, MG, and mesh STA
 - Automatically elected: mesh-auto (only for MBR and MG; a mesh STA does not support automatic networking election.)
- Node role arbitration
- Node association selection on L2 (association selection algorithm)
- Node association
 - Directly connected to a router
 - Directly connected to the MG
 - Constructing a mesh backbone network
- Mesh STA Roaming
- Troubleshooting
 - Router exception
 - MBR abnormal/offline issues
 - MG intermediate node exception

The APIs provided by this module are used to set up a mesh network for communication.



NOTE

This module can be directly used to set up a mesh network or used as the sample code for calling the original APIs provided by the mesh SDK.

3.2 Development Process

Function

Table 3-1 describes the Autolink function APIs provided by the mesh SDK.

Table 3-1 Description of the Autolink function APIs

API Name	Description
hi_mesh_start_autolink	Starts the Autolink module.
hi_mesh_exit_autolink	Exits the Autolink module.
hi_mesh_autolink_set_router_rssi_thres hold	Specifies the RSSI threshold of the router associated with the MBR.
hi_mesh_autolink_get_router_rssi_thres hold	Obtains the RSSI threshold of the router associated with the MBR.
hi_mesh_autolink_set_bw	Sets the bandwidth of the Autolink module.
hi_mesh_autolink_get_bw	Obtains the bandwidth of the Autolink module.
hi_mesh_autolink_register_event_callb ack	Registers the callback function of the Autolink module.

Development Process

The typical process of building a mesh network by using the APIs of the Autolink module is as follows:

Step 1 Set the automatic networking parameters based on the parameters of the access router and enable the automatic mesh network by calling **hi_mesh_start_autolink**.

Step 2 Register the callback function with the Autolink module by calling **hi_mesh_autolink_register_event_callback**, receive the role callback function, and obtain the node interface.

----End

The typical process of exiting the Autolink module is as follows:

Step 1 Exit the Autolink module by calling **hi_mesh_exit_autolink**.

----End



Returns

Table 3-2 describes the error codes returned by the mesh Autolink module.

Table 3-2 Return values of the Autolink module

No.	Return Value	Actual Value	Description
1	HISI_OK	0	Operation succeeded.
2	HISI_FAIL	-1	Operation failed.

3.3 Precautions

- **hi_mesh_start_autolink** is a non-blocking API. After the command is successfully delivered, the node role is returned through the event callback after a period of time.
- The Autolink module can be started by calling **hi_mesh_start_autolink** repeatedly. Each time this API is called, the system checks whether a process exists. If a process exists, the system stops the previous process and restarts the Autolink process.
- Ensure that no service VAP is started when the automatic networking API is called. If a service VAP is started, the corresponding stopping API must be called to stop the service VAP before the automatic networking API is called.
- When the Autolink module is enabled, the node role can be set to **HI_MESH_USRCONFIG_MBR**, **HI_MESH_USRCONFIG_MR**, **HI_MESH_USRCONFIG_MSTA**, or **HI_MESH_USRCONFIG_AUTO**. **HI_MESH_USRCONFIG_AUTO** can only elect the MBR or MR. To enable a node to function as a leaf node (mesh STA), specify the node as **HI_MESH_USRCONFIG_MSTA**.
- When setting up a mesh network, ensure that at least one node is started as **HI_MESH_USRCONFIG_MBR** or **HI_MESH_USRCONFIG_AUTO**.
- A timeout mechanism is configured for the Autolink module. If no valid mesh node role is returned within the timeout period, the **HI_MESH_AUTOLINK_ROUTER_MSTA** role is degraded to connect to the router when the timeout period (10 minutes) expires. To re-access the network, **hi_mesh_start_autolink** needs to be called again to start network access.
- Only **hi_mesh_exit_autolink** can be used to exit the Autolink module.

3.4 Programming Example

Starting Automatic Election

```
#include "hi_mesh_autolink_api.h"

hi_void mesh_autolink_cb(const hi_mesh_autolink_role_cb *role_cb)
{
```



```
dprintf("mesh role[%d]\n", role_cb->role);
dprintf("ifname_first[%s], len_first[%d]\n", role_cb->info.ifname_first, role_cb->info.len_first);
dprintf("ifname_second[%s], len_second[%d]\n", role_cb->info.ifname_second, role_cb->info.len_second);

return;
}

hi_void example_start_autolink(hi_void)
{
    /*****Mesh Auto Network*****/
    hi_mesh_autolink_config mesh_auto_config = {
        "meshnetwork", HI_MESH_AUTH, "123456789", HI_MESH_USRCONFIG_AUTO};

    hi_mesh_start_autolink(&mesh_auto_config);

    hi_mesh_autolink_register_event_callback(mesh_autolink_cb);
}
```

Result verification:

```
mesh role[0]
ifname_first[mesh0], len_first[5]
ifname_second[wlan0], len_second[5]
```

Starting a Specified MR

```
#include "hi_mesh_autolink_api.h"
hi_void mesh_autolink_cb(const hi_mesh_autolink_role_cb *role_cb)
{
    dprintf("mesh role[%d]\n", role_cb->role);
    dprintf("ifname_first[%s], len_first[%d]\n", role_cb->info.ifname_first, role_cb->info.len_first);
    dprintf("ifname_second[%s], len_second[%d]\n", role_cb->info.ifname_second, role_cb->info.len_second);

    return;
}

hi_void example_start_autolink(hi_void)
{
    /*****Mesh Auto Network*****/
    hi_mesh_autolink_config mesh_auto_config = {
        "meshnetwork", HI_MESH_AUTH, "123456789", HI_MESH_USRCONFIG_MR};

    hi_mesh_start_autolink(&mesh_auto_config);

    hi_mesh_autolink_register_event_callback(mesh_autolink_cb);
}
```

Result verification:

```
mesh role[1]
ifname_first[mesh0], len_first[5]
ifname_second[], len_second[0]
```