Tina Linux Wi-Fi 开发指南

版本号: 1.9
发布日期: 2024.12.04

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.05.20	AWA1381	初始版本。
1.1	2022.06.06	AWA1436	确定 wifimanager2.0 部分框架。
1.2	2023.02.10	AWA1436	wifimanager2.0 部分增加 p2p 相关内容。
1.3	2023.04.20	AWA1381	wifimanager 部分修订新框架。
1.4	2023.07.18	AWA1436	增加 wifimanager 扩展命令相关内容。
1.5	2023.11.04	AWA1436	增加 wifimanager 连接曾经连接过 ap 的相关内容。修正了 wifimanager 更新后的接口相关内容。
1.6	2023.12.12	AWA1436	增加适用 buildroot 编译方式相关说明
1.7	2024.10.28	AWA1436	修改章节排版更新 wifimanager 接口说明添加共存相关说明添加二次开发模型添加配网使用 demo 说明
1.8	2024.11.21	AWA1436	按摸版修改章节排版添加部分开发编程实例
1.9	2024.12.04	AWA2255	添加 ap 功能开发编程实例

版权所有 © 珠海全志科技股份有限公司。保留一切权利

目 录

1 前言 **1**
1.1 文档简介 1
1.2 目标读者 1
1.3 适用范围 1
1.4 文档约定 1
1.4.1 标志说明 1
1.4.2 地址与数据描述方法约定 2
1.5 相关术语介绍 2

2 wifimg 软件简介 **3**
2.1 Tina Wi-Fi 软件框架图 3
2.2 wifimg 代码框架图 4
2.3 wifimg 代码目录结构 4
2.4 wifimg 配置 7

3 wifimg 开发模型介绍 **9**
3.1 linux 和 xrlink 系统中开发模型 9
3.2 freertos 系统中开发模型 10

4 wifimg 开发介绍 **11**
4.1 初始化 11
4.1.1 接口概述 11
4.1.2 开发流程 11
4.1.3 编程实例 13
4.2 sta 功能开发 13
4.2.1 接口概述 13
4.2.2 开发流程 13
4.2.3 编程实例 15
4.3 ap 功能开发 17
4.3.1 接口概述 17
4.3.2 开发流程 17
4.3.3 编程实例 18
4.4 monitor 功能开发 20
4.4.1 接口概述 20
4.4.2 开发流程 20
4.4.3 编程实例 20
4.5 p2p 功能开发 20
4.5.1 接口概述 21
4.5.2 开发流程 21

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.5.3 编程实例 22
4.6 配网功能开发 22
4.6.1 接口概述 22
4.6.2 开发流程 23
4.6.3 编程实例 23
4.7 共存功能开发 25
4.7.1 接口概述 28
4.7.2 开发流程 28
4.7.3 编程实例 30
4.8 其他功能开发 33
4.8.1 接口概述 34
4.8.2 开发流程 34
4.8.3 编程实例 34

5 wifimg 测试工具介绍 35
5.1 测试工具目录结构 35
5.2 测试工具配置 35
5.3 测试工具支持的命令 36
5.3.1 sta 模式命令 36
5.3.2 ap 模式命令 38
5.3.3 monitor 模式命令 38
5.3.4 p2p 模式命令 39
5.3.5 其他命令 40
5.3.6 配网模式命令 41
5.3.6.1 softap 配网 42
5.3.6.2 ble 配网 44
5.3.6.3 soundwave 配网 46

6 附件 47
6.1 wifimg 结构体详细说明 47
6.2 wifimgAPI 详细说明 58

版权所有 © 珠海全志科技股份有限公司。保留一切权利

插　图

图 2-1 Tina-Wi-Fi-软件框架图 3
图 2-2 wifimager 框架图 4
图 2-3 libwifimg-v2.0 库文件调用关系图 6
图 2-4 Tina_wifimanager 配置 1 7
图 2-5 Tina_wifimanager 配置 2 8
图 2-6 Tina_wifimanager 配置 3 8
图 2-7 Tina_buildroot_wifimanager 配置 1 8
图 3-1 linux-xrlink 二次开发模型 10
图 4-1 模组支持列表说明 26
图 4-2 linux 共存模式配置 27
图 4-3 wifimg 模组名匹配图 27
图 4-4 xrlink 共存模式配置 28
图 4-5 实际支持模式 28
图 5-1 softap 配网 43
图 5-2 ble 配网 45
图 5-3 soundwave 配网 46

版权所有 © 珠海全志科技股份有限公司。保留一切权利

1 前言

1.1 文档简介

介绍 Tina Wi-Fi 软件管理框架，包括 Station、ap、monitor、p2p 模式以及 Wi-Fi 常见开发问题。

1.2 目标读者

适用 Tina 平台的广大客户。

1.3 适用范围

Allwinner 软件平台 Tina v4.0 版本及以上。

Allwinner 硬件平台 R/V/T/MR/H 系列。

AllwinnerOS 平台 Linux/Xrlink/Freertos 系列。

1.4 文档约定

1.4.1 标志说明

⚠ **注意**

- **提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。**

说明

为准确理解文中指令、正确实施操作而提供的补充或强调信息。

技巧

一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

1.4.2 地址与数据描述方法约定

本文档在描述地址、数据时遵循如下约定：

表 1-1: 地址与数据描述方法约定

符号	例子	说明
0x	0x0200，0x79	地址或数据以 16 进制表示。
0b	0b010，0b00 000 111	数据采用二进制表示 (寄存器描述除外)。
X	00X，XX1	数据描述中，X 代表 0 或 1。 例如，00X 代表 000 或 001；XX1 代表 001，011，101 或 111。

1.5 相关术语介绍

术语	解释说明
wifimanager	全志实现的一种用于屏蔽底层方便客户开发的中间件，文档会简称为 wifimg
os linux	wifimg 运行在标准的 linux 系统上，对接的是标准的 wpa_supplicant/hostapd
os xrlink	wifimg 运行在标准的 linux 系统上，但对接的是 xrlink 驱动 (xrlink 是全志实现的一种双栈 wifi 形式)
os freertos	wifimg 运行在 freertos 系统上。

ALLWINNER®

2 wifimg 软件简介

wifimg 主要用于 wifi 的连接管理，通信以及 wifi 的一些额外功能。支持 sta、ap、monitor、p2p 模式，并且集成了配网模式以及其他功能。屏蔽了底层系统的具体实现，能对接各种差异化系统例如 linux，xrlink，freertos 等，并对用户层提供一套统一的简单通用的接口。它提供了一个可给客户进行开发用的 libwifimg-v2.0 库，客户可以基于该 lib 库进行开发，把相关接口集成到客户自己的 wifi 应用中。它也提供了一个可以直接使用的 wifi 工具。

用户如果需要进行独自开发 (把对应的 wifi 功能集成到各自的应用里)，请重点查阅章节：wifimg 开发模型，wifimg 开发介绍，附件

用户如果不需要独自开发，可以直接使用 wifimg 提供的工具，仅需查看 wifimg 测试工具介绍。

2.1 Tina Wi-Fi 软件框架图

图 2-1: Tina-Wi-Fi-软件框架图

- wifimanager：全志为了方便客户使用 wifi 功能而开发的一套用于管理 wifi 连接的中间件。它支持 sta、ap、monitor、p2p 模式，并且集成了配网模式以及其他功能。屏蔽了底层系统的具体实现，能对接各种差异化系统平台例如 linux，xrlink(全志基于 linux 系统实现的一种双栈 wifi 驱动，用于 mcu 或内置 wifi 模组)，tian freertos 等。
- wpad：(wpa_supplicant + hostapd)，包含了开源的无线网络配置工具 wpa_supplicant，主要用来支持 WEP，WPA/WPA2/WPA3 无线协议和加密认证的。也包含了 hostapd 用户态用于 ap 和认证服务器的守护进程。主要用于 Tina linux 系统，标准的第 3 方开源工具。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

- 双栈驱动: 全志基于 Tina linux 系统实现的一种双栈 wifi 驱动主要用于 xr 系列 mcu wifi 模组或全志内置系列 wifi 模组。
- freertos 系统网络命令：tian freertos 系统提供的网络连接命令。

2.2 wifimg 代码框架图

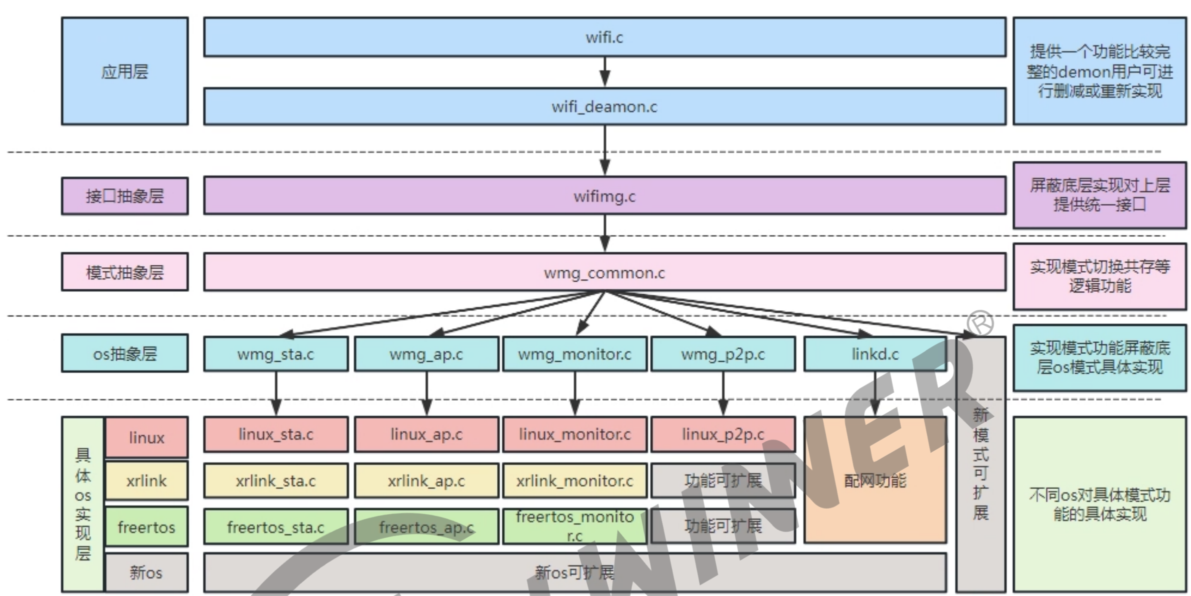


图 2-2: wifimager 框架图

- wifimanager: 兼容 linux，xrlink，freertos 等系统。支持 sta，ap，monitor 和 p2p 等模式，集成了 softap，ble，soundwave 等配网功能。提供了完善的 api 接口方便用户调用，同时提供了一个完整功能的测试工具，方便用户直接使用和测试。上图是 wifimg 的软件结构，整体分为 3 部分：应用层，lib 层 (接口抽象层，模式抽象层，os 抽象层)，os 具体实现层。
- 应用层：主要提供了一个完整功能的测试工具，方便用户直接使用。用户也可以不使用该 demo，直接调用 lib 库提供的 api 接口。把具体功能集成到自己的应用中。
- lib 层：包含了接口层，模式抽象层，os 抽象层，对上提供统一的 api 接口 (wifimg.h)，处理各种模式的逻辑以及共存功能，对下屏蔽 os 的具体模式功能。
- 具体 os 实现层：该层主要是不同系统对 wifi 功能的具体功能。

2.3 wifimg 代码目录结构

wifimanger 的主要目录结构如下：

```
├── app    //用于保存配网时使用的一些配网工具
├── core   //核心代码
│   ├── include  //存放核心代码相关头文件
│   │   └── wifimg.h  //对上提供的api接口头文件，集成开发需要包含的头文件
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
│   └── src //核心代码具体实现
│     ├── linkd.c //配网抽象层代码
│     ├── wifimg.c //对上提供的api接口实现代码
│     ├── wmg_common.c //模式切换共存逻辑处理层实现代码
│     ├── wmg_sta.c //sta模式抽象层代码
│     ├── wmg_ap.c //ap模式抽象层代码
│     ├── wmg_monitor.c //monitor模式抽象层代码
│     ├── wmg_p2p.c //p2p模式抽象层代码
│     ├── expand_cmd.c //扩展命令(与模组或系统特殊功能有关，例如获取或设置mac地址，设置特殊模组的ioct等)
│     └── os //具体os(linux,xrlink,freertos)模式功能实现代码
├── demo
│  │  └── wifi_test_tool //完整功能的测试工具
│  │     ├── wifi.c
│  │     └── wifi_daemon.c
│  └── ...... //开发参考用的其他demo
├── files //相关的配置文件
```

- app 目录：存放用于配网测试的 apk 安装包。
- core 目录：存放 wifimanger 核心代码的目录，会编译出 libwifimg-v2.0 库。
- demo 目录：wifi_test_tool 目录下存放的是完整功能的 wifi 测试工具，其他 demo 用于给用户参考开发用。
- files：存放相关的配置文件。

wifimanager 核心 lib 目录结构

wifimanager 核心代码路径如下：

wifimanager/core

主要目录和文件结构如下：

```
.
├── include
│   ├── linkd.h
│   ├── os
│   ├── wifi_log.h
│   ├── wifimg.h
│   ├── wmg_ap.h
│   ├── wmg_common.h
│   ├── wmg_monitor.h
│   ├── wmg_p2p.h
│   ├── expand_cmd.h
│   └── wmg_sta.h
└── src
  ├── linkd.c
  ├── log
  ├── os
  │  ├── linux(在非linux系统该目录不存在)
  │  ├── xrlink(在非xrlink系统该目录不存在)
  │  └── freertos(在非freertos系统该目录不存在)
  ├── wifimg.c
  ├── wmg_ap.c
  ├── wmg_common.c
  ├── wmg_monitor.c
  ├── wmg_p2p.c
  ├── expand_cmd.c
```

```
└── wmg_sta.c
```

- include: 保存核心代码相关头文件目录。
- src : 保存核心代码源码文件目录。
- src/log：打印相关代码。
- src/wifimg.c ： 用户接口对接口层文件（提供用户对接层 API 接口文件，对接 api 请查看 wifimg.h）。
- src/wmg_common.c：wifi 模式抽象层（各种模式的抽象层，处理切换和共存逻辑）。
- src/wmg_sta.c：station 模式抽象层。
- src/wmg_ap.c：ap 模式抽象层。
- src/wmg_monitor.c: monitor 模式抽象层。
- src/expand_cmd.c：特殊额外功能抽象层。
- src/os : 对应的系统模式实现层代码。
- src/os/linux：linux 平台 wifi 模式功能具体实现代码存放目录 (在非 linux 系统该目录不存在)。
- src/os/xrlink：xrlink 平台 wifi 模式功能具体实现代码存放目录 (在非 xrlink 系统该目录不存在)。
- src/os/freertos：freertos 平台 wifi 模式功能具体实现代码存放目录 (在非 freertos 系统该目录不存在)。

核心代码里各文件调用关系图如下。

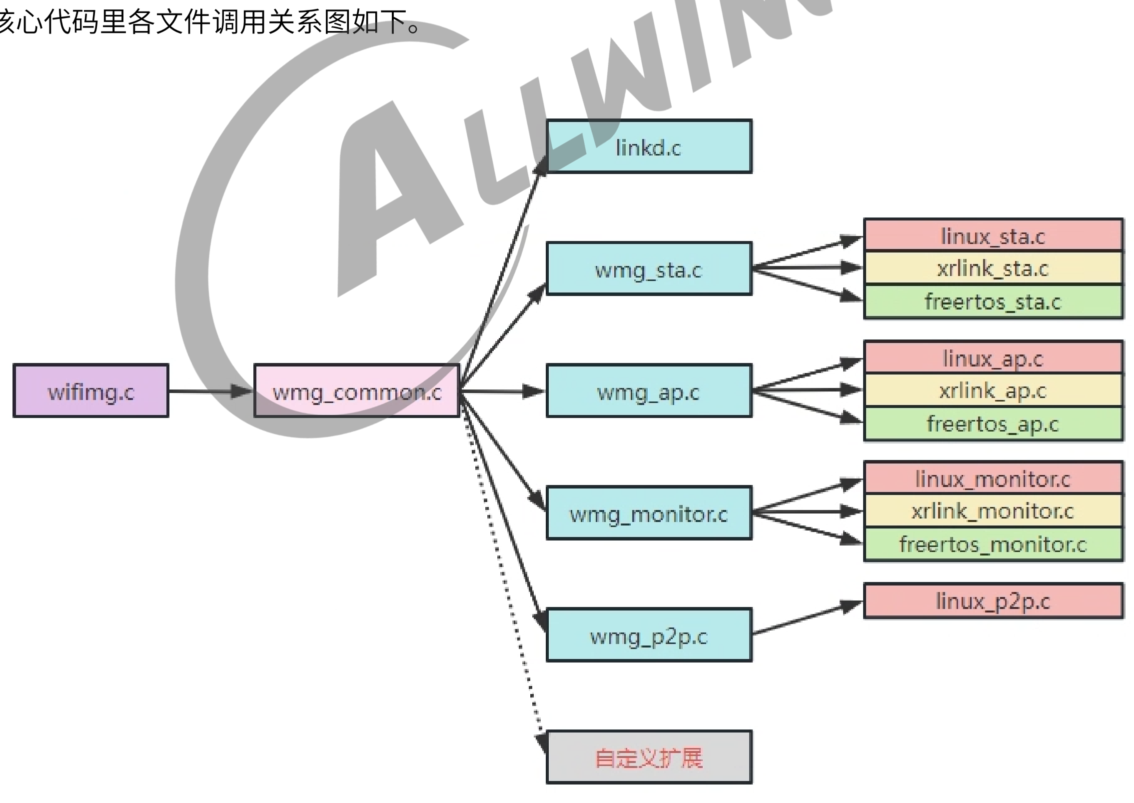


图 2-3: libwifimg-v2.0 库文件调用关系图

1. 用户会调用 wifimg.c 提供的接口函数。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

2.wifimg.c 的接口函数会调用到模式抽象层 wmg_common.c 里的函数。

3.wmg_common.c 里的函数会根据不同的模式调用到 wmg_sta.c(sta 模式抽象层)，wmg_ap.c(ap 模式抽象层)wmg_monitor.c(monitor 模式抽象层)wmg_p2p.c(p2p 模式抽象层) 里的函数。

4.wmg_sta.c(sta 模式抽象层) 会根据不同的平台调用到 linux_sta.c(linux 平台具体实现文件)，xrlink_sta.c(xrlink 平台具体实现文件)，freertos_sta.c(rtos 平台具体实现文件)。

5.wmg_ap.c(ap 模式抽象层) 会根据不同的平台调用到 linux_ap.c(linux 平台具体实现文件)，xrlink_ap.c(xrlink 平台具体实现文件)，freertos_ap.c(rtos 平台具体实现文件)。

6.wmg_monitor.c(monitor 模式抽象层) 会根据不同的平台调用到 linux_monitor.c(linux 平台具体实现文件)，xrlink_monitor.c(xrlink 平台具体实现文件)，freertos_monitor.c(rtos 平台具体实现文件)。

7.wmg_p2p.c(p2p 模式抽象层) 会根据不同的平台调用到 linux_p2p.c(linux 平台具体实现文件)。

8. 自定义扩展 (expand_cmd.c) 与系统或模组特殊功能有关，例如设置或获取 mac 地址，特殊 ioct，某些模组特殊功能等。

wifimanager demo 目录结构

wifimanager demo 代码路径如下：

wifimanager/demo

该目录下包含了多个子目录

- demo/wifi_test_tool：该目录里包含了完整功能的 wifi 测试工具
- demo/其他目录：其他目录均未开发测试用 demo，不定时会更新。

2.4 wifimg 配置

openwrt 编译方式：wifimg 配置如下：

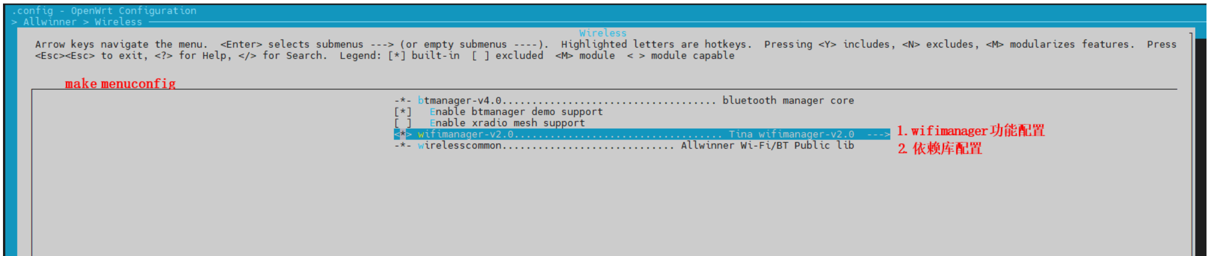


图 2-4: Tina_wifimanager 配置 1

```
.config - OpenWrt Configuration
> Allwinner > Wireless > wifimanager-v2.0.................................. Tina wifimanager-v2.0
                    wifimanager-v2.0.................................. Tina wifimanager-v2.0
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press
  <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

  make menuconfig
                    --- wifimanager-v2.0.................................. Tina wifimanager-v2.0     1. wifimanager功能配置
                          wifimanager-v2.0 Configuration  --->                                     2. 依赖库
                    <*>   wifimanager-v2.0-lib.......................... Tina wifimanager-v2.0 lib 3. 应用demo
                    <*>     wifimanager-v2.0-demo..................... Tina wifimanager-v2.0 app demo
```

图 2-5: Tina_wifimanager 配置 2

```
.config - OpenWrt Configuration
> Allwinner > Wireless > wifimanager-v2.0.................................. Tina wifimanager-v2.0 > Wifimanager-v2.0 Configuration
                    Wifimanager-v2.0 Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press
  <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

  make menuconfig
                    [*] CONFIG_WMG_PROTOCOL_SOFTAP                                1. 配网相关
                    [*] CONFIG_WMG_PROTOCOL_BLE
                    [ ] CONFIG_WMG_PROTOCOL_XCONFIG
                    [ ] CONFIG_WMG_PROTOCOL_SOUNDWAVE
                        Wifimanager support platform (LINUX PLATFORM OS)  --->    2. 系统相关
                    [ ] Wifimanager unregister callback function
                    [*] Wifimanager support send expand cmd                       3. 扩展命令相关
                        Wifimanager support mode Configuration  --->              4. 功能相关，如sta,ap,p2p等
                        Wifimanager set default debug level (default debug level: to info)  --->   5. 调试等级配置
```

图 2-6: Tina_wifimanager 配置 3

buildroot 编译方式：wifig 配置如下：

```
-> Target packages > allwinner platform private package select > wireless
                    wireless
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted
  letters are hotkeys.  Pressing <Y> selectes a feature, while <N> excludes a feature.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] feature is selected  [ ] feature is excluded
 build.sh buildroot menuconfig
              [*] wifimanager-v2.0
              [*]   wifimanager-v2.0-lib
              [*]     Tina wifimanager-v2.0-demo
              -*- wireless_common
              [*] btmanager-core
              [*]   Enable btmanager demo support
                  firmware  --->
```

图 2-7: Tina_buildroot_wifimanager 配置 1

说明

1.buildroot 编译方式暂不支持定制化配置，后续会支持。

2.buildroot 编译方式配置完后需要执行 build.sh buildroot_saveconfig 命令把配置信息保存起来

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3 wifimg 开发模型介绍

wifimg 支持的是库内部线程同步和互斥，但不支持进程间的同步和互斥，因此在开发模型中要稍微注意。

3.1 linux 和 xrlink 系统中开发模型

os linux 和 os xrlink 方式 wifi 都是基于 linux 系统环境开发的，linux 系统有真正意义上的进程，但 wifimg 底层是不支持进程间信息同步的因此在 linux 系统上使用 wifimg 进行开发时需要注意开发模型。在 linux 上开发模型有 2 个关键字，常驻和单一进程。

错误的二次开发模型：

1. 瞬时性模型

(1) 该模型的特点就是开发的程序只调用了 libwifimg-v2.0 库中的某个接口后就退出了。例如调用连接接口后该进程就退出了，然后又起来一个进程调用 libwifimg-v2.0 库的获取信息接口后又退出。

(2) 该模型只能使用单一独立的功能，不能使用延续性的功能。例如该模型下启动 A 进程进行扫描然后退出，启动 B 进程连接后退出，这种非延续性功能是可以使用的。但启动 B 进程连接后退出，再启动 C 进程去读取连接信息这时获取到的连接信息是无效的。因此该模型是不建议的。

2. 多常驻进程操作模型

(1) 该模型的特点就是系统中存在 2 个以上的进程集成了 libwifimg-v2.0 库，例如进程 A 和进程 B 都集成 libwifimg-v2.0 库，然后 A 进程进行了扫描，B 进程进行了连接。这种模型因为同时实例了 2 个 libwifimg-v2.0 库，因此会存在同时操作模组的情况 (因为 libwifimg-v2.0 库不支持多进程互斥操作)。但 libwifimg-v2.0 库是支持多线程操作的，例如进程 A 中存在 A1 和 A2 线程，A1A2 线程同时调用 libwifimg-v2.0 库的接口是允许的。

正确的二次开发模型：

1. 常驻单一进程 (进程中可支持多线程) 模型：

(1) 该模型的特点是操作管理 wifi 的进程是常驻于系统中的，并且有且仅有该进程操作 libwifimg-v2.0 库。

(2) 该模型中常驻进程 A 是允许内部的 A1A2 线程同时操作 libwifimg-v2.0 库的。

ALLWINNER®

(3) 该模型中若多个进程需要获取 wifi 信息，那么只能有 1 个进程对 libwifimg-v2.0 库进行直接操作，其他进程与该进程通信但不直接操作 libwifimg-v2.0 库的实例

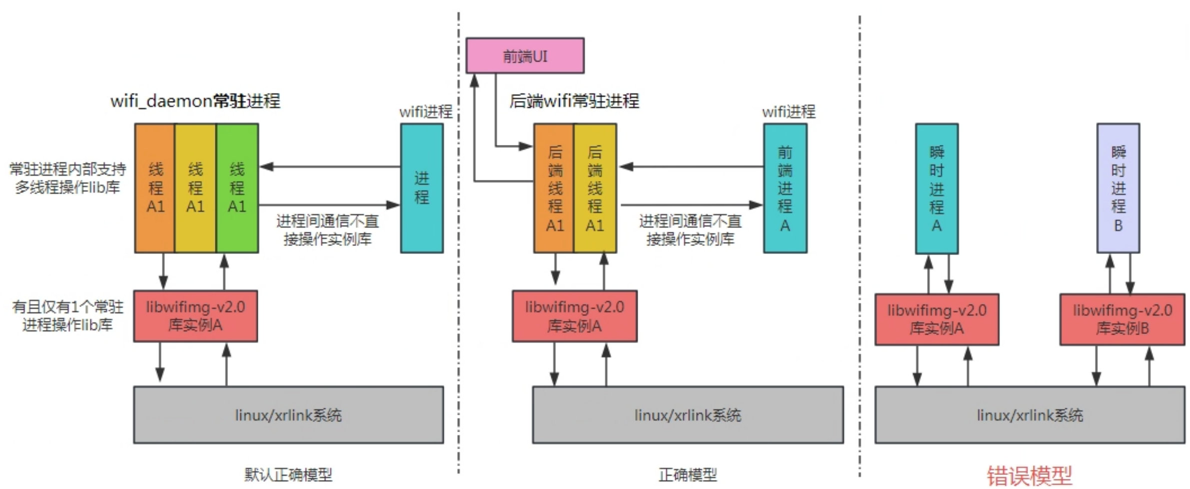


图 3-1: linux-xrlink 二次开发模型

3.2 freertos 系统中开发模型

freertos 系统中没有真正意义上的进程概念，因此 freertos 系统的开发不需要考虑什么开发模组，直接对接 wifimg 的 API 即可。

4 wifimg 开发介绍

下面提供 wifimg 各功能的开发流程 (简单介绍 api 的调用流程)。更详细的调用流程可以参考 wifi_deamon.c 文件里的编写流程。也可参考 demo 目录下的其他 demo

4.1 初始化

1. 在调用 libwifimg-v2.0 库中的任何 api 前都必须调用 wifimanager_init 函数。整个使用周期内 wifimanager_init 函数只需调用 1 次。调用 wifimager_deinit 函数后必须重新调用 wifimanager_init 函数。

2. 不同模式的功能接口需要打开某种模式后才能调用，例如 sta 模式的 connect 接口需要打开 sta 模式后才能调用。

4.1.1 接口概述

接口名称	描述
wifimanager_init	初始化 wifimg，整个 wifimg 使用周期中只需要调用 1 次
wifimanager_deinit	反初始化 wifimg，调用该函数后 wifimg 会整个退出
wifi_register_msg_cb	注册回调函数 (**注意**该回调函数是实时回调，因此切记不能做阻塞或耗时操作)
wifi_on	打开某种模式 (station/ap/monitor/p2p)
wifi_off	关闭某种模式 (WIFI_MODE_UNKNOWN)
wmg_get_debug_level	获取 debug 信息的打印等级 (需包含头文件 wifi_log.h)
wmg_set_debug_level	设置 debug 信息的打印等级 (需包含头文件 wifi_log.h)

4.1.2 开发流程

打开 sta(其他模式同理)

此操作示例实现初始 wifimg 进行一下平台性操作并打开 sta 模式，关闭 sta 模式并反初始化 wifimg

步骤 1 初始化 wifimg

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
wifimanager_init();
```

说明

初始化 wifimg，整个 wifimg 使用周期中只需要调用 1 次

步骤 2 注册回调函数

```
wifi_register_msg_cb(wifi_msg_cb, NULL)
```

说明

(1) 该回调函数是实时回调，因此切记不能做阻塞或耗时操作
(2)wifi_msg_cb:wifi_msg_cb_t 型回调函数，若填 NULL 相当于反注册回调函数 (取消先前注册的回调函数)
(3)NULL: 扩展或特殊用途用，用户可以先不关心，填 NULL 即可

步骤 3 获取当前打印等级 (**可选步骤**，特殊功能调试使用，实际使用场景用户不一定需要调用)

```
wmg_get_debug_level();
```

说明

(1)WMG_MSG_ERROR(1): 错误打印等级，最高打印等级
(2)WMG_MSG_WARNING(2): 警告打印等级
(3)WMG_MSG_INFO(3): 默认打印等级
(4)WMG_MSG_DEBUG(4):debug 打印等级
(5)WMG_MSG_MSGDUMP(5): 通信数据打印等级 (部分系统支持)
(6)WMG_MSG_EXCESSIVE(6): 第 3 方调试打印等级 1(部分系统支持)
(7)WMG_MSG_EXCESSIVE_MID(7): 第 3 方调试打印等级 2(部分系统支持)
(8)WMG_MSG_EXCESSIVE_MAX(8): 第 3 方调试打印等级 3(部分系统支持)

步骤 4 设置打印等级 (**可选步骤**，特殊功能调试使用，实际使用场景用户不一定需要调用)

```
wmg_set_debug_level(wmg_log_level_t level);
```

说明

设置打印等级描述如步骤 2，用户 debug 时一般设置到 4 或 5 打印等级即可，过高打印等级部分系统不支持

步骤 5 打开 sta 模式

```
wifi_on(WIFI_STATION);
```

说明

同理可以打开其他模式 (WIFI_STATION/WIFI_AP/WIFI_MONITOR/WIFI_P2P)
若打开其他模式后当前模式会隐式关闭，例如：打开 sta 模式后调用该接口打开 ap 模式，此时 sta 模式会被隐式关闭若需要再次使用 sta 模式功能需要再次打开 sta 模式。

步骤 6 关闭 sta 模式

```
wifi_off(WIFI_MODE_UNKNOWN);
```

说明

显式关闭当前模式

步骤 7 反化 wifimg

```
wifimanager_deinit();
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

📖 说明

反初始化后若需要再次使用 wifimg 需重新初始化 wifimg

4.1.3 编程实例

该接口功能是初始化功能，下述编程实例都包含，不单独提供编程实例。

4.2 sta 功能开发

4.2.1 接口概述

接口名称	描述
wifi_get_scan_results	扫描并获取扫描结果
wifi_sta_connect	连接某个 ap
wifi_sta_disconnect	断开与 ap 的连接
wifi_sta_auto_reconnect	启动或关闭自动连上某个 ap 功能
wifi_sta_auto_connect	指定连接某个曾经连接过的 ap(不需要填 password)
wifi_sta_get_info	获取当前 station 模式状态的一些信息 (包括连接上的 ap 的 ssid，bssid 等)
wifi_sta_list_networks	在 sta 模式下获取已保存的 ap 列表信息
wifi_sta_remove_networks	在 sta 模式下移除某个或全部已保存的 ap 的信息

4.2.2 开发流程

连接上某个 ap 热点

此操作示例实现扫描当前环境，并连接上某个指定 ap

步骤 1 打开 sta 模式

```
wifi_on(WIFI_STATION);
```

步骤 2 扫描当前环境存在哪些 ap(**可选步骤**，实际使用场景不一定需要进行扫描)

```
wifi_get_scan_results(scan_res, ssid/NULL, &bss_num, RES_LEN);
```

📖 说明

(1)wifi_scan_result_t scan_res: 该参数需要用户申请内存，主要是用于保存扫描结果
(2)char *ssid: 需要扫描的隐藏 ssid，若不需要则填 NULL(当前仅 freertos 系统支持，其他系统暂不支持)
(3)bss_num: 该参数传到库底层，系统会记录扫描到了多少条结果并更新到该参数中
(4)RES_LEN: 该参数主要是表明 scan_res 数组的大小 (条数)

步骤 3 设置自动重连功能 (**可选步骤**，默认是 enable，实际使用场景用户不一定需要调用)

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
wifi_sta_auto_reconnect(true/false);
```

说明

自动重连功能在底层有 2 种作用
(1) 开启自动连接功能后，如果连接上了某个 ap，该 ap 的信息会保存在系统中，下次系统启动后会尝试去连接已连接过的 ap(依赖于 wap_supplicant 实现)
(2) 开启自动连接功能后，如果已经连上了某个 ap，但某些原因与该 ap 断开了连接，系统会尝试继续与该 ap 进行连接 (依赖于 wpa_supplcant 实现)
(3) 该功能目前仅在 linux 系统方案有实现，其他系统方案暂未实现

步骤 4 使用系统保存的密码信息连接指定的 ap(**可选步骤**，实际使用场景用户不一定需要调用)

```
wifi_sta_auto_connect(char *ssid);
```

说明

(1) 该接口与普通连接接口的区别在于该接口只需要填写 ssid，底层逻辑会判断你输入的 ssid 是否曾经连接过, 如果连接过，使用保存的配置 (加密方式，密码等) 去连接该 ap，若未连接过则返回错误 (依赖于 wpa_supplicant)
(2) 该功能目前仅在 linux 系统方案有实现，其他系统方案暂未实现。

步骤 5 连接指定的 ap

```
wifi_sta_connect(&cn_para);
```

说明

(1)wifi_sta_cn_para_t cn_para 结构体不同参数有不同效果
(2)cn_para.ssid: 指定 ap 的 ssid 必须设置
(3)cn_para.password: 指定 ap 的 password 必须设置
(4)cn_para.bssid: 可选项 (当环境中存在同名 ssid 时配置该参数可连接特定的同名 ap)
(5)cn_para.sec: 指定 ap 的加密方式必须设置，可用步骤 2 的扫描函数获取，或者设置为 WIFI_SEC_UNKNOWN
WIFI_SEC_UNKNOWN：设置该参数后底层逻辑也是通过扫描获取指定 ap 的加密方式
(6)cn_para.fast_connect: 可选项仅 freertos 系统会使用到，客户可不用理会

步骤 6 列出所有已保存的 ap 信息 (**可选步骤**，特殊功能，实际使用场景用户不一定需要调用)

```
wifi_sta_list_networks(&sta_list_networks);
```

说明

(1)sta_list_networks 参数是 wifi_sta_list_t 类型的结构体，需要用户申请相关的内存，其中 list_nod 字段用于保存列表信息，list_num 字段用于告诉系统列表空间的大小，sys_list_num 字段是返回值，调用函数后系统实际列表项数目会保存在该字段。
(2) 该功能 freertos 系统方案暂未实现，xrlink 系统方案中因仅能保存一个连接信息因此调用该接口也仅能显示 1 个 ap 信息。

步骤 7 移除某个或全部已保存的 ap 信息 (**可选步骤**，特殊功能，实际使用场景用户不一定需要调用)

```
wifi_sta_remove_networks(NULL/ssid);
```

说明

(1) 该函数要配合 wifi_sta_list_networks 函数来使用，调用 wifi_sta_list_networks 函数列出已保存的 ap 信息，再根据信息来移除不需要的 ap。如果 wifi_sta_remove_networks 传入的参数是一个具体的 ssid，那么仅仅移除该 ssid 的信息，如果传入的参数是 NULL，那么把所有的已保存的 ap 信息都删掉。
(2) 该功能 freertos 系统方案暂未实现，xrlink 系统方案中因仅能保存一个连接信息因此调用该接口也仅能删除唯一的一个 ap 信息。

步骤 8 列出当前连接的 ap 的信息 (**可选步骤**，特殊功能，实际使用场景用户不一定需要调用)

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
wifi_sta_get_info(&wifi_sta_info);
```

说明

(1) 该函数主要是获取当前连接的 ap 的信息，因此需要连接上某个 ap 后才能调用，调用时用户需要自己申请一个 wifi_sta_info 结构体。
(2) 可用获取连接 ap 的信道，信号强度，bssid，ssid，加密方式等。

步骤 9 断开与 ap 的连接

```
wifi_sta_disconnect();
```

4.2.3 编程实例

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/socket.h>
4 #include <sys/un.h>
5 #include <unistd.h>
6 #include <string.h>
7 #include "wifimg.h"
8 #include <wifi_log.h>
9
10 #ifndef UNREGISTER_CB
11 //Callback functions cannot do high-load and blocking actions
12 void wifi_msg_cb(wifi_msg_data_t *msg)
13 {
14   if(msg->id == WIFI_MSG_ID_STA_STATE_CHANGE) {
15     WMG_DEBUG("get sta state:(%d) ", msg->data.state);
16     switch(msg->data.state) {
17       case WIFI_STA_CONNECTING:
18       WMG_INFO("Connecting......\n");
19       break;
20       case WIFI_STA_CONNECTED:
21       WMG_INFO("Connected to the AP\n");
22       break;
23       case WIFI_STA_OBTAINING_IP:
24       WMG_INFO("Obtaining ip address......\n");
25       break;
26       case WIFI_STA_NET_CONNECTED:
27       WMG_DEBUG("Successful net connected\n");
28       break;
29       case WIFI_STA_DISCONNECTED:
30       WMG_INFO("Disconnected\n");
31       break;
32     }
33   }
34 }
35 #endif
36
37 void handler(int sig)
38 {
39   WMG_INFO("get a sig, num is %d\n",sig);
40   if(sig == 2){
41     wifi_off(WIFI_MODE_UNKNOWN);
42     wifimanager_deinit();
43     WMG_INFO("Exit sta mode simple demo\n");
```

```
44      exit(0);
45   }
46 }
47
48 int main(int argc, char *argv[])
49 {
50   pid_t pc;
51   wifi_sta_cn_para_t cn_para;
52   int ret_exit = -1;
53
54   pc = fork();
55   if(pc < 0) {
56      WMG_ERROR("fork error\n");
57      exit(-1);
58   } else if (pc>0) {
59      exit(0);
60   }
61
62   signal(2,handler);
63
64   /* wifimanager init */
65   wifimanager_init();
66
67   /* open sta mode */
68   if(!wifi_on(WIFI_STATION)) {
69 #ifndef UNREGISTER_CB
70      if(wifi_register_msg_cb(wifi_msg_cb, NULL)) {
71         WMG_ERROR("register msg cb failed\n");
72      }
73 #endif
74   } else {
75      WMG_ERROR("sta mode simple demo: open sta mode failed, exit now\n");
76      goto exit;
77   }
78
79   /* connect wifi */
80   memset(&cn_para, 0, sizeof(wifi_sta_cn_para_t));
81   cn_para.ssid = argv[1];
82   cn_para.password = argv[2];
83   cn_para.sec = WIFI_SEC_UNKNOWN;
84   if(wifi_sta_connect(&cn_para) == 0) {
85      WMG_INFO("==Wi-Fi ssid: %s password: %s sec: %d connect successful==\n",
86            cn_para.ssid, cn_para.password, cn_para.sec);
87   } else {
88      WMG_ERROR("==Wi-Fi connect failed==\n");
89      goto exit;
90   }
91
92   while(1) {
93      sleep(1);
94   }
95
96   ret_exit = 0;
97
98 exit:
99   wifimanager_deinit();
100   WMG_INFO("Exit sta mode simple demo\n");
101   exit(ret_exit);
102 }
```

sta 模式连接 demo 代码可参考 SDK 中

wifimanager/demo/wifi_sta/wifi_sta.c 文件

4.3 ap 功能开发

4.3.1 接口概述

接口名称	描述
wifi_ap_enable	启动 ap 热点
wifi_ap_get_config	获取 ap 相关信息
wifi_ap_disable	关闭 ap 热点

4.3.2 开发流程

此操作示例实现启动一个 ap 热点

步骤 1 打开 ap 模式

```
wifi_on(WIFI_AP);
```

步骤 2 启动 ap 热点

```
wifi_ap_enable(&ap_config);
```

说明

(1)wifi_ap_config_t ap_config: 结构体不同参数有不同效果启动 ap 热点时只需填下面参数
(2)ap_config.ssid: 启动 ap 热点的 ssid
(3)ap_config.psk: 启动 ap 热点的密码，若启动无密码 ap，填 NULL 即可
(4)ap_config.sec: 启动 ap 热点的加密方式 (支持 wpa/wpa2)
(5)ap_config.channel: 启动 ap 热点的信道 (当前仅支持启动 2.4G 信道，暂不支持启动 5G 信道)
(6)ap_config.ip_addr: 启动 ap 热点的 ip 网段 (可选项，不填系统会设置一个默认值 192.168.5)

步骤 3 获取 ap 热点信息 (**可选步骤**，特殊功能，实际使用场景用户不一定需要调用)

```
wifi_ap_get_config(&ap_config);
```

说明

(1)wifi_ap_config_t ap_config 结构体获取信息时，不同字段包含不同的信息
(2)ssid，psk，sec，channel，ip_addr: 启动 ap 热点时的信息
(3)ap_config.sta_num: 启动 ap 热点后，当前有几个 sta 已连接上
(4)ap_config.dev_list[STA_MAX_NUM]: 已连接上的 sta 的简单信息

步骤 4 关闭 ap 热点

```
wifi_ap_disable(void);
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.3.3 编程实例

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <string.h>
#include "wifimg.h"
#include <wifi_log.h>

#ifndef UNREGISTER_CB
//Callback functions cannot do high-load and blocking actions
void wifi_msg_cb(wifi_msg_data_t *msg)
{
  if(msg->id == WIFI_MSG_ID_AP_STATE_CHANGE) {
    WMG_DEBUG("get ap state:(%d) ", msg->data.ap_state);
    switch(msg->data.ap_state) {
      case WIFI_AP_DISABLE:
      WMG_INFO("ap disable\n");
      break;
      case WIFI_AP_ENABLE:
      WMG_INFO("ap enable\n");
      break;
      case WIFI_AP_STA_CONNECTE:
      WMG_INFO("one sta connected\n");
      break;
    }
  }
}
#endif

void handler(int sig)
{
  WMG_INFO("get a sig, num is %d\n",sig);
  if(sig == 2){
    wifi_off(WIFI_MODE_UNKNOWN);
    wifimanager_deinit();
    WMG_INFO("Exit ap mode simple demo\n");
    exit(0);
  }
}

int main(int argc, char *argv[])
{
  int ret_exit = -1;
  pid_t pc;

  char ssid_buf[SSID_MAX_LEN + 1] = "allwinner-ap";
  char psk_buf[PSK_MAX_LEN + 1] = "Aa123456";
  wifi_ap_config_t ap_config;
  memset(&ap_config, 0, sizeof(wifi_ap_config_t));
  ap_config.ssid = ssid_buf;
  ap_config.psk = psk_buf;
  ap_config.sec = WIFI_SEC_WPA2_PSK;
  ap_config.channel = 6;

  pc = fork();
  if(pc < 0) {
```

```
        WMG_ERROR("fork error\n");
        exit(-1);
    } else if (pc>0) {
        exit(0);
    }

    signal(2,handler);

    /* wifimanager init */
    wifimanager_init();

    /* for debug */
    //wmg_set_debug_level(WMG_MSG_DEBUG);

    /* register callback function */
    if(wifi_register_msg_cb(wifi_msg_cb, NULL)) {
        WMG_ERROR("register msg cb failed\n");
    }

    if(argc > 1){
        memset(ap_config.ssid, 0, SSID_MAX_LEN + 1);
        strncpy(ap_config.ssid, argv[1], SSID_MAX_LEN);
    }
    if(argc > 2){
        memset(ap_config.psk, 0, PSK_MAX_LEN + 1);
        strncpy(ap_config.psk, argv[2], SSID_MAX_LEN);
    }

    /* open ap mode */
    if(wifi_on(WIFI_AP) == 0) {
        if(wifi_ap_enable(&ap_config) == 0) {
            WMG_INFO("ap enable success, ssid=%s, psk=%s, sec=%d, channel=%d\n", ap_config.ssid,
                    ap_config.psk, ap_config.sec, ap_config.channel);
        } else {
            WMG_ERROR("ap enable failed\n");
            wifi_off(WIFI_MODE_UNKNOWN);
            goto exit;
        }
    } else {
        WMG_ERROR("wifi on ap failed\n");
        wifi_off(WIFI_MODE_UNKNOWN);
        goto exit;
    }

    while(1) {
        sleep(1);
    }

    ret_exit = 0;
    wifi_off(WIFI_MODE_UNKNOWN);

exit:
    wifimanager_deinit();
    WMG_INFO("Exit ap simple demo\n");
    exit(ret_exit);
}
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.4 monitor 功能开发

该模式暂不支持共存模式，该模式当前仅在 os xrlink 实现其他系统平台暂不支持该模式

该模式需要注册回调函数

4.4.1 接口概述

接口名称	描述
wifi_monitor_enable	启动 monitor 功能监听某个信道
wifi_monitor_set_channel	切换监听的信道
wifi_monitor_disable	关闭 monitor 监听功能

4.4.2 开发流程

此操作示例实现启动 monitor 模式监听某个信道

步骤 1 打开 monitor 模式

```
wifi_on(WIFI_MONITOR);
```

步骤 2 启动 monitor 功能监听某个信道

```
wifi_monitor_enable(channel);
```

步骤 3 切换监听的信道 (**可选步骤**，特殊功能，实际使用场景用户不一定需要调用)

```
wifi_monitor_set_channel(channel);
```

步骤 4 关闭 monitor 监听功能

```
wifi_monitor_enable(channel);
```

4.4.3 编程实例

暂未提供编译实例，后续会更新

4.5 p2p 功能开发

请注意该模式当前仅在 os linux 平台实现其他系统平台暂不支持该模式，该模式也依赖 wpa_supplicant 版本。

4.5.1 接口概述

接口名称	描述
wifi_p2p_enable	启动 p2p 功能
wifi_p2p_find	启动 p2p 寻找功能
wifi_p2p_connect	连接 p2p 设备
wifi_p2p_get_info	获取 p2p 设备信息
wifi_p2p_disconnect	断开 p2p 设备
wifi_p2p_disable	关闭 p2p 功能

4.5.2 开发流程

此操作示例实现启动 p2p 功能，并进行 p2p 设备连接

步骤 1 打开 p2p 模式

```
wifi_on(WIFI_P2P);
```

步骤 2 启动 p2p 功能

```
wifi_p2p_enable(&p2p_config);
```

说明

(1)wifi_p2p_config_t p2p_config: 结构体不同参数有不同效果
(2)p2p_config.name: 启动 p2p 设备时的设备名
(3)p2p_config.time: 启动 p2p 设备是启动多久扫描时间
(4)p2p_config.p2p_go_intent: 协商 gc/go 值 0~15
(5)p2p_config.auto_connect: 扫描时若有 p2p 设备请求连接时是否自动连接

步骤 3 扫描周围 p2p 设备

```
wifi_p2p_find(&p2p_peers, find_second);
```

步骤 4 连接某个 p2p 设备

```
wifi_p2p_connect(p2p_mac_addr);
```

说明

(1) 必须是步骤 2 扫描出来的 p2p_mac_addr
(2) 连接时对端设备也必须处于 find 状态否则会连接失败

步骤 5 获取 p2p 设备信息 (**可选步骤**，特殊功能，实际使用场景用户不一定需要调用)

```
wifi_p2p_get_info(&p2p_info);
```

说明

(1)wifi_p2p_info_t p2p_info: 结构体获取信息时，不同字段包含不同的信息
(2)p2p_info.mode:0: 当前是未连接的 p2p 设备 1: 当前已连接协商为 go 2: 当前已连接协商为 gc

ALLWINNER®

步骤 6 与对端 p2p 设备断开

```
wifi_p2p_disconnect(uint8_t *p2p_mac_addr);
```

步骤 7 关闭 p2p 功能

```
wifi_p2p_disable(void);
```

4.5.3 编程实例

暂未提供编译实例，后续会更新

4.6 配网功能开发

目前 wifimg 支持 sofap，BLE，声波配网和二维码配网模式。

所有的配网模式包含 2 个主要步骤。

1. 获取连接 ap 的 ssid 和 psk。

2. 根据获取到的信息进行联网。

每种配网模式的第二部分的实现都是一样的，调用 libwifimg-v2.0 里提供的 API 进行网络连接。

差异化部分只有获取要连接 ap 的 ssid 和 psk 部分。

- sofap 配网模式是通过开启一个 ap 热点，用户通过手机 app 连上该 ap 热点后把配网信息传送到小机端。
- 蓝牙配网模式是启动蓝牙后，用户通过手机 app 把配网信息通过蓝牙传送到小机端。
- 声波配网模式是启动 mic 后，用户通过手机 app 发送一段带有配网信息的声波给小机端。
- 二维码配网模式是启动摄像头，通过扫描二维码获取配网信息后进行连接。

当前配网功能暂未开放对第 3 方配网方式的集成开发，后续会支持第 3 方集成开发功能。

4.6.1 接口概述

接口名称	描述
wifi_linkd_protocol	启动某种配网模式获取配网信息 (ssid,psk)

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.6.2 开发流程

步骤 1 启动某种配网模式获取配网信息

```
wifi_linkd_protocol(linkd_mode, linkd_params, time, &linkd_result);
```

说明

(1)linkd_mode: 启动什么配网模式 (不同方案不同配网模式不完全支持)

- **WMG_LINKD_MODE_BLE：蓝牙配网模式，依赖方案是否支持蓝牙**
- **WMG_LINKD_MODE_SOFTAP：ap 配网模式**
- **WMG_LINKD_MODE_SOUNDWAVE：声波配网模式，依赖方案硬件是否支持音频**
- **WMG_LINKD_MODE_QRCODE：二维码配网模式，依赖方案硬件是否支持摄像头**

(2)linkd_params: 不同配网模式的参数

- **蓝牙配网模式：参数无效，填 NULL 即可**
- **ap 配网模式：参数有效，可配置 ap 热点相关参数，请参考编程实例**
- **声波配网模式：参数无效，填 NULL 即可**
- **二维码配网模式：参数无效，填 NULL 即可**

(3)time: 配网等待时间，填 0 会使用默认配网等待时间 120 秒
(4)linkd_result: 当配网接口执行成功后该参数保存配网信息

4.6.3 编程实例

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/socket.h>
4 #include <sys/un.h>
5 #include <unistd.h>
6 #include <string.h>
7 #include "wifimg.h"
8 #include <wifi_log.h>
9
10 #ifndef UNREGISTER_CB
11 //Callback functions cannot do high-load and blocking actions
12 void wifi_msg_cb(wifi_msg_data_t *msg)
13 {
14   if(msg->id == WIFI_MSG_ID_STA_STATE_CHANGE) {
15     WMG_DEBUG("get sta state:(%d) ", msg->data.state);
16     switch(msg->data.state) {
17       case WIFI_STA_CONNECTING:
18       WMG_INFO("Connecting......\n");
19       break;
20       case WIFI_STA_CONNECTED:
21       WMG_INFO("Connected to the AP\n");
22       break;
23       case WIFI_STA_OBTAINING_IP:
24       WMG_INFO("Obtaining ip address......\n");
25       break;
26       case WIFI_STA_NET_CONNECTED:
27       WMG_DEBUG("Successful net connected\n");
28       break;
29       case WIFI_STA_DISCONNECTED:
30       WMG_INFO("Disconnected\n");
31       break;
```

```
32         }
33     }
34 }
35 #endif
36
37 void handler(int sig)
38 {
39     WMG_INFO("get a sig, num is %d\n",sig);
40     if(sig == 2){
41         wifi_off(WIFI_MODE_UNKNOWN);
42         wifimanager_deinit();
43         WMG_INFO("Exit net config simple demo\n");
44         exit(0);
45     }
46 }
47
48 int main(int argc, char *argv[])
49 {
50     int ret_exit = -1;
51     pid_t pc;
52     wifi_sta_cn_para_t cn_para;
53     memset(&cn_para, 0, sizeof(wifi_sta_cn_para_t));
54     char ssid_buf[SSID_MAX_LEN + 1] = {0};
55     char psk_buf[PSK_MAX_LEN + 1] = {0};
56     wifi_linkd_result_t linkd_result;
57     void *linkd_params = NULL;
58     wifi_linkd_mode_t linkd_mode = WMG_LINKD_MODE_NONE;
59
60     linkd_result.ssid = ssid_buf;
61     linkd_result.psk = psk_buf;
62
63     wifi_ap_config_t ap_config;
64     char ap_ssid_buf[SSID_MAX_LEN + 1] = "Aw-config-net-Test";
65     char ap_psk_buf[PSK_MAX_LEN + 1] = "Aa123456";
66
67     pc = fork();
68     if(pc < 0) {
69         WMG_ERROR("fork error\n");
70         exit(-1);
71     } else if (pc>0) {
72         exit(0);
73     }
74
75     signal(2,handler);
76
77     /* wifimanager init */
78     wifimanager_init();
79
80     /* get network information */
81     if(!strncmp(argv[1], "ble", 3)) {
82         linkd_mode = WMG_LINKD_MODE_BLE;
83     } else if(!strncmp(argv[1], "softap", 6)) {
84         ap_config.ssid = ap_ssid_buf;
85         ap_config.psk = ap_psk_buf;
86         ap_config.sec = WIFI_SEC_WPA2_PSK;
87         ap_config.channel = 6;
88         linkd_params = &ap_config;
89         linkd_mode = WMG_LINKD_MODE_SOFTAP;
90     } else {
91         WMG_ERROR("Unsupport net config\n");
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
92        goto exit;
93    }
94
95    if(wifi_linkd_protocol(linkd_mode, linkd_params, 120, &linkd_result) != MG_STATUS_SUCCESS){
96        WMG_ERROR("config net get result failed\n");
97        goto exit;
98    } else {
99        cn_para.ssid = ssid_buf;
100       cn_para.password = psk_buf;
101   }
102
103   /* open sta mode */
104   if(!wifi_on(WIFI_STATION)) {
105 #ifndef UNREGISTER_CB
106       if(wifi_register_msg_cb(wifi_msg_cb, NULL)) {
107           WMG_ERROR("register msg cb failed\n");
108       }
109 #endif
110   } else {
111       WMG_ERROR("net config simple demo: open sta mode failed, exit now\n");
112       goto exit;
113   }
114
115   /* connect wifi */
116   cn_para.sec = WIFI_SEC_UNKNOWN;
117   if(wifi_sta_connect(&cn_para) == 0) {
118       WMG_INFO("==Wi-Fi ssid: %s password: %s sec: %d connect successful==\n",
119           cn_para.ssid, cn_para.password, cn_para.sec);
120   } else {
121       WMG_ERROR("==Wi-Fi connect failed==\n");
122       goto exit;
123   }
124
125   while(1) {
126       sleep(1);
127   }
128
129   ret_exit = 0;
130
131 exit:
132   wifimanager_deinit();
133   WMG_INFO("Exit net config simple demo\n");
134   exit(ret_exit);
135 }
```

配网模式 demo 代码可参考 SDK 中

wifimanager/demo/wifi_net_config/wifi_net_config.c 文件

4.7 共存功能开发

wifimg 共存模式实现逻辑

1.wifimg 支持部分共存模式 (sta+ap，sta+p2p)，但整个系统中是否支持共存模式关键取决于模组而非 wifimg，只有模组支持共存模式后 (是否支持请咨询模组原厂)wifimg 的共存模式才能有效，

版权所有 © 珠海全志科技股份有限公司。保留一切权利

否则 wifimg 的共存模式开启后也无法使用。

2. 共存模式只影响模式的开启与关闭逻辑，不影响各模式中的功能 (但可能会影响部分性能如吞吐)。例如单独模式中 sta 支持扫描和连接，在共存模式下 sta 模式依旧支持扫描和连接。

3. 目前只有 os linux 或 os xrlink 系统支持部分共存模式，freertos 暂未支持共存模式。

4. 单独模式与共存模式的区别在于底层逻辑的处理，处理逻辑如下：

(1) 单独模式下 sta 切换到 ap 模式时 lib 库会先把 sta 模式关闭，然后再启动 ap 模式。因此此时 sta 模式的功能 (例如扫描和连接) 是无法使用的。

(2) 共存模式下 sta 切换到 ap 模式时 lib 库不会把 sta 模式关闭，因此此时 sta 模式的功能 (例如扫描和连接) 是可以使用的。但在共存模式下如果不使用模组的某种模式了必须显式关闭该模式。例如切到 ap 模式后如果不想使用 sta 模式，需要显式调用关闭接口关闭 sta 模式。

os linux 系统共存模式开发

os linux 系统下的 wifimg 支持用户添加第 3 方模组的共存模式，因此配置相对复杂一点 (但请注意即使 wifimg 支持了共存，但若该模组驱动不支持也没用)。

1. 模组模式支持列表：主要用于描述该模组支持哪些模式 (单独模式或共存模式)

2.menuconfig 共存配置：该配置主要是控制编译行为 (若不把共存配置打开，共存代码不会编译进 wifimg 中，运行时就无法使用共存功能)

3. 动态修改 wifimg.config：动态修改小机端这个配置能动态修改 wifimg 对单独模式和共存模式的底层处理逻辑。

添加模组模式支持列表

模组模式支持列表文件在路径：wifimanager/core/src/os/linux/linux_support_list.c

```
174 >--->---.module_mode_support_list = LINUX_MODE_SUPPORT_LIST(STA_ALONE, AP_ALONE),
175 >--->---.wpad_indep_bitmap = 0,
176 >---},
177 #endif
178
179 #ifdef LINUX_VERSION_5_10
180 >---{
181 >--->---.module_name = "AIC8800-5.10-NORMAL",
182 >--->---.module_mode_support_list = LINUX_MODE_SUPPORT_LIST(STA_ALONE, AP_ALONE),
183 >--->---.wpad_indep_bitmap = 0,
184 >---},
185 #endif
186
187 #ifdef LINUX_VERSION_5_15
188 >---{
189 >--->---.module_name = "AIC8800-5.15-NORMAL",
190 >--->---.module_mode_support_list = LINUX_MODE_SUPPORT_LIST(STA_ALONE, AP_ALONE),
191 >--->---.wpad_indep_bitmap = 0,
192 >---},
193
194 >---{
195 >--->---.module_name = "AIC8800-5.15-SA",
196 >--->---.module_mode_support_list = LINUX_MODE_SUPPORT_LIST(STA_ALONE, AP_ALONE, STA_AP_COEXIST),
197 >--->---.wpad_indep_bitmap = WPA_HOSTAPD_COEXIST,
198 >---},
199 #endif
200
201 #ifdef LINUX_VERSION_6_6
202 >---{
203 >--->---.module_name = "AIC8800-6.6-NORMAL",
204 >--->---.module_mode_support_list = LINUX_MODE_SUPPORT_LIST(STA_ALONE, AP_ALONE),
205 >--->---.wpad_indep_bitmap = 0,
NORMAL  BR: ipc-v821-tina-v1.0rc4 | linux_support_list.c |      unix | utf-8 | c  68%  LN 200:1
```


图 4-1: 模组支持列表说明

目前 wifimg 添加了 4.9，5.4，5.10，5.15，6.6 版本内核的 819s，829，aic8800 模组支持列表，若需要添加其他第 3 方模组请参考该文件添加。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

选上 menuconfig 配置

linux 系统执行 make menuconfig 选上对应的共存模式即可，如下图配置即可：

```
.config - OpenWrt Configuration
> Allwinner > Wireless > wifimanager-v2.0................................ Tina wifimanager-v2.0 > Wifimanager-v2.0 Configuration > Wifimanager support mode Configuration
                                Wifimanager support mode Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M>
  modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                    [*] Wifimanager support sta mode enable
                    [*] Wifimanager support ap mode enable
                    [ ] Wifimanager support monitor mode enable
                    [*] Wifimanager support p2p mode enable
                    [ ] Wifimanager support sta + ap coexist mode enable
                    [ ] Wifimanager support sta + p2p coexist mode enable (NEW)

              <Select>    < Exit >    < Help >    < Save >    < Load >
```

图 4-2: linux 共存模式配置

动态修改 wifimg.config

```
#set wifi module name
#wifi_module_name=UNKNOWN

##############################################################################
##                                                                          ##
## The following directory paths are all relative paths (Relative to current file) ##
##                                                                          ##
##############################################################################

#############################
##      linux os use       ##
#############################
wpa_sta_iface=wlan0
wpa_sta_config_file=/wpa_supplicant/wpa_supplicant.conf
wpa_sta_iface_path=/wpa_supplicant/sockets
wpa_sta_auto_reconnect=enable

wpa_p2p_process=p2p_supplicant
wpa_p2p_iface=p2p-dev-wlan0
wpa_p2p_config_file=/wpa_supplicant/wpa_supplicant_p2p.conf
wpa_p2p_iface_path=/wpa_supplicant/sockets
wpa_p2p_listen_time=10
#Configure the network segment of p2p GO mode, If not configured, the default is 192.168.6
#p2p_net_segment=192.168.6

#If this name conflicts with wpa_sta_iface in coexistence mode, the name will be invalid
hapd_ap_iface=wlan0
hapd_config_file=/hostapd/hostapd.conf
hapd_ap_iface_path=/hostapd/sockets

wpa_iface_path=/wpa_supplicant/sockets
hapd_iface_path=/hostapd/sockets

## sta mode need get ip
## ip_need=0: supports obtaining any ip mode
## ip_need=1: support single get ipv4 mode
## ip_need=2: support single get ipv6 mode
## ip_need=3: support support must obtain both ipv4
#ip_need=0
#############################

#############################
##      xrlink os use      ##
#############################
xrlink_auto_reconnect=enable
NORMAL  BR: ipc-v821-tina-v1.0rc4 |  files/wifimg.config          unix | utf-8 | conf   2%  LN   1:21
```

图 4-3: wifimg 模组名匹配图

系统运行时修改小机端/etc/wifi/wifimg.conf 配置文件可以动态修改底层共存运行逻辑。修改 wifi_module_neme 匹配到对应的支持列表即可

os xrlink 系统共存模式开发

xrlink 系统目前仅支持全志系列 mcu wifi 模组或内置 wifi 模组，因此共存模式配置简单。

xrlink 系统执行 make menuconfig 选上对应的共存模式即可，如下图配置即可：

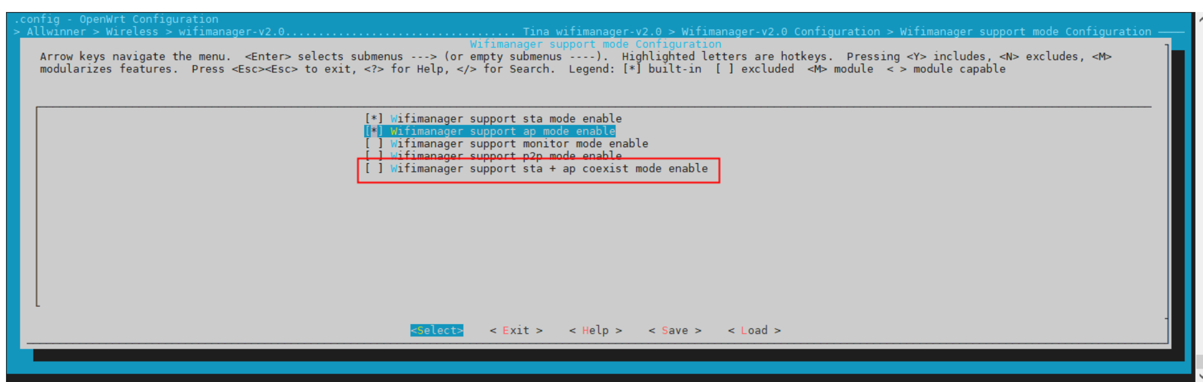


图 4-4: xrlink 共存模式配置

当前 xrlink 系统仅支持 sta + ap 模式共存。

freertos 系统共存模式开发

freertos 系统暂不支持共存模式开发

查看当前系统支持的模式

系统启动，调用了 wifi_init 接口后会打印相关模式支持信息

当前系统实际支持模式是由 wifimanger 支持模式 (编译链接)+ 模组支持模式的交集如下图：

```
WINF: ****************************************
WINF: * Copyright (c) 2019-2025 Allwinner Technology Co., Ltd. ALL rights reserved
WINF: * version: 2.0.8.5 20240105 10:00
WINF: * module name: AIC8800-5.15-SA (M)
WINF: * wifimg support mode:(sta | ap | --- | --- | sta-ap | --------)
WINF: * module support mode:(sta | ap | --- | --- | sta-ap | --------)
WINF: * actual support mode:(sta | ap | --- | --- | sta-ap | --------)
WINF: ****************************************
```

模组模式列表匹配名

wifimanager编译了哪些模式

模组支持哪些模式

实际运行时逻辑

图 4-5: 实际支持模式

4.7.1 接口概述

共存模式相对于单独模式来说没有新增任何接口，区别仅仅在于底层逻辑的处理上

4.7.2 开发流程

打开 sta 模式连接上某个 ap 热点后再启动 ap 模式启动一个 ap 热点

此操作示例实现打开 sta 模式连接上某个 ap 热点后再启动 ap 模式启动一个 ap 热点，然后关闭对应的模式

步骤 1 共存模式下打开 sta 模式

```
wifi_on(WIFI_STATION);
```

说明

(1) 启动 sta 模式后能操作 sta 模式相关功能例如扫描连接等，sta 功能操作与单独模式没有区别

步骤 2 共存模式下连接指定 ap 热点

```
wifi_sta_connect(&cn_para);
```

说明

(1) 连接参数与单独模式没有区别，但需注意连接的路由信道会影响共存模式下后续启动 ap 的性能或模组底层逻辑。
(2) 启动 sta 模式连接 ap 后就能确定当前 sta 模式的信道，该信道值在不同模组下的共存模式会有差异 (与模组底层实现逻辑有关)
(3)sta 模式信道由所连接的 ap 决定，例如路由 A 信道是 4 路由 B 信道是 9，连接路由 A 后信道就是 4，连接路由 B 后信道就是 9

步骤 3 共存模式下打开 ap 模式

```
wifi_on(WIFI_AP);
```

说明

(1) 共存模式下启动 AP 模式后可进行 ap 相关功能例如启动 ap 热点等，ap 功能操作与单独模式没有区别

步骤 4 启动 ap 热点

```
wifi_ap_enable(&ap_config);
```

说明

(1) 共存模式下启动 AP 模式是需要设置信道参数的，信道参数会影响吞吐性能以及模组底层实现逻辑
(2) 某些模组在共存模式下是不支持 sta 模式和 ap 模式不同信道的，底层逻辑会把它配置同信道 (在此情况下 ap 模式信道配置参数可能无效)
(3) 某些模组是支持共存模式下 sta 模式和 ap 模式不同信道的，底层逻辑会在不同信道中切换但会影响吞吐
(4) 共存模式下 sta 和 ap 的信道问题是底层模组逻辑决定的与 wifimg 无关，如何配置交由应用策略处理，原则上建议信道相同保证吞吐

步骤 5 共存模式下关闭 sta/ap 模式

```
wifi_off(WIFI_STATION/WIFI_AP/WIFI_MODE_UNKNOWN);
```

说明

(1) 若共存模式下已同时打开 sta 和 ap 模式输入参数不同底层逻辑处理会不同
(2)WIFI_STATION: 底层逻辑会关闭 sta 模式，此时 sta 模式功能不能使用，ap 模式功能依旧可使用
(3)WIFI_AP: 底层逻辑会关闭 ap 模式，此时 ap 模式功能不能使用，sta 模式功能依旧可以使用
(4)WIFI_MODE_UNKNOWN: 底层逻辑会关闭当前所有打开的模式

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.7.3 编程实例

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/socket.h>
4 #include <sys/un.h>
5 #include <unistd.h>
6 #include <string.h>
7 #include "wifimg.h"
8 #include <wifi_log.h>
9
10 #define BAND_NOME    0
11 #define BAND_2_4G    1
12 #define BAND_5G      2
13
14 static uint32_t freq_to_channel(uint32_t freq)
15 {
16     int band;
17     uint32_t channel = 0;
18     if((freq >= 5180) && (freq <= 5825)){
19         band = BAND_5G;
20     } else if((freq >= 2407) && (freq <= 2484)){
21         band = BAND_2_4G;
22     } else {
23         band = BAND_NOME;
24     }
25     switch (band) {
26     case BAND_2_4G:
27         if(freq == 2484) {
28             channel = 14;
29         } else if(freq == 2407) {
30             channel = 0;
31         } else if((freq <= 2472) && (freq > 2407)){
32             if(((freq - 2407) % 5) == 0) {
33                 channel = ((freq - 2407) / 5);
34             } else {
35                 channel = 1000;
36             }
37         } else {
38             channel = 1000;
39         }
40         break;
41     case BAND_5G:
42         if(((freq - 5180) % 5) == 0) {
43             channel = ((freq - 5180) / 5) + 36;
44         } else {
45             channel = 1000;
46         }
47         break;
48     case BAND_NOME:
49     default:
50         channel = 1000;
51         break;
52     }
53     return channel;
54 }
55
56 static uint8_t get_sta_connect_channel()
57 {
```

```
58   uint8_t channel = 1000;
59   wifi_sta_info_t wifi_sta_info;
60   if(wifi_sta_get_info(&wifi_sta_info) == WMG_STATUS_SUCCESS) {
61     channel = freq_to_channel(wifi_sta_info.freq);
62   }
63   return channel;
64 }
65
66 #ifndef UNREGISTER_CB
67 //Callback functions cannot do high-load and blocking actions
68 void wifi_msg_cb(wifi_msg_data_t *msg)
69 {
70   if(msg->id == WIFI_MSG_ID_STA_STATE_CHANGE) {
71     WMG_DEBUG("get sta state:(%d) ", msg->data.state);
72     switch(msg->data.state) {
73       case WIFI_STA_CONNECTING:
74       WMG_INFO("Connecting......\n");
75       break;
76       case WIFI_STA_CONNECTED:
77       WMG_INFO("Connected to the AP\n");
78       break;
79       case WIFI_STA_OBTAINING_IP:
80       WMG_INFO("Obtaining ip address......\n");
81       break;
82       case WIFI_STA_NET_CONNECTED:
83       WMG_DEBUG("Successful net connected\n");
84       break;
85       case WIFI_STA_DISCONNECTED:
86       WMG_INFO("Disconnected\n");
87       break;
88     }
89   }
90 }
91 #endif
92
93 void handler(int sig)
94 {
95   WMG_INFO("get a sig, num is %d\n",sig);
96   if(sig == 2){
97     wifi_off(WIFI_MODE_UNKNOWN);
98     wifimanager_deinit();
99     WMG_INFO("Exit sta mode simple demo\n");
100     exit(0);
101   }
102 }
103
104 int main(int argc, char *argv[])
105 {
106   int ret_exit = -1;
107   pid_t pc;
108   wifi_sta_cn_para_t cn_para;
109
110   char ssid_buf[SSID_MAX_LEN + 1] = "allwinner-sta-ap-coexist";
111   char psk_buf[PSK_MAX_LEN + 1] = "Aa123456";
112   wifi_ap_config_t ap_config;
113   memset(&ap_config, 0, sizeof(wifi_ap_config_t));
114   ap_config.ssid = ssid_buf;
115   ap_config.psk = psk_buf;
116   ap_config.sec = WIFI_SEC_WPA2_PSK;
117
```

```
118    pc = fork();
119    if(pc < 0) {
120        WMG_ERROR("fork error\n");
121        exit(-1);
122    } else if (pc>0) {
123        exit(0);
124    }
125
126    signal(2,handler);
127
128    /* wifimanager init */
129    wifimanager_init();
130
131    /* open sta mode */
132    if(!wifi_on(WIFI_STATION)) {
133 #ifndef UNREGISTER_CB
134        if(wifi_register_msg_cb(wifi_msg_cb, NULL)) {
135            WMG_ERROR("register msg cb failed\n");
136        }
137 #endif
138    } else {
139        WMG_ERROR("sta mode simple demo: open sta mode failed, exit now\n");
140        goto exit;
141    }
142
143    /* connect wifi */
144    memset(&cn_para, 0, sizeof(wifi_sta_cn_para_t));
145    cn_para.ssid = argv[1];
146    cn_para.password = argv[2];
147    cn_para.sec = WIFI_SEC_UNKNOWN;
148    if(wifi_sta_connect(&cn_para) == 0) {
149        WMG_INFO("==Wi-Fi ssid: %s password: %s sec: %d connect successful==\n",
150                cn_para.ssid, cn_para.password, cn_para.sec);
151    } else {
152        WMG_ERROR("==Wi-Fi connect failed==\n");
153        return -1;
154    }
155
156    /* Get the channel after the STA mode connection is successful */
157    ap_config.channel = get_sta_connect_channel();
158    if(ap_config.channel != 1000) {
159        WMG_INFO("wifi connect ap channel %d\n", ap_config.channel);
160    } else {
161        wifi_off(WIFI_MODE_UNKNOWN);
162        goto exit;
163    }
164
165    /* coexistence mode opens ap */
166    if(wifi_on(WIFI_AP) == 0) {
167        if(wifi_ap_enable(&ap_config) == 0) {
168            WMG_INFO("ap enable success, ssid=%s, psk=%s, sec=%d, channel=%d\n", ap_config.ssid,
169                    ap_config.psk, ap_config.sec, ap_config.channel);
170        } else {
171            WMG_ERROR("ap enable failed\n");
172            wifi_off(WIFI_MODE_UNKNOWN);
173            goto exit;
174        }
175    } else {
176        WMG_ERROR("wifi on ap failed\n");
177        wifi_off(WIFI_MODE_UNKNOWN);
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
178        goto exit;
179    }
180
181    while(1) {
182        sleep(1);
183    }
184
185    ret_exit = 0;
186    wifi_off(WIFI_MODE_UNKNOWN);
187
188 exit:
189    wifimanager_deinit();
190    WMG_INFO("Exit sta ap coexist simple demo\n");
191    exit(ret_exit);
192 }
```

共存模式 demo 代码可参考 SDK 中

wifimanager/demo/wifi_sta_ap_coexist/wifi_sta_ap_coexist.c 文件

4.8 其他功能开发

wifimg 功能开发

此部分功能与网络或 wifimg 功能有关，非传统意义上的 wifi 功能例如设置获取 mac 地址，获取 wifimg 的状态等。

wifimg 扩展模式

wifimanager 集成了扩展模式，扩展模式主要用于扩展 wifi 差异性功能。这些功能与模组或系统有强相关，非通用性。

例如 A 模组支持使用 ioct 或设备节点控制唤醒功能，但 B 模组没有这些功能，A 模组的这些差异性功能就会被集成到扩展模式里。

扩展功能没有通用性，因此不在此处进行简述，需要客户自行查看源码，同样客户也可以根据模组的特性在此处添加模组差异性功能。

core/src/expand_cmd.c 文件是扩展命令屏蔽层，该文件 (层) 屏蔽了系统性差异功能。

core/src/os/linux/expand/ –> linux 系统差异性功能具体实现代码 (非通用性，客户自行查阅以及增删)。

core/src/os/xrlink/expand/ –> xrlink 系统差异性功能具体实现代码 (非通用性，客户自行查阅以及增删)。

core/src/os/freertos/expand/ –> freertos 系统差异性功能具体实现代码 (非通用性，客户自行查阅以及增删)

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.8.1 接口概述

接口名称	描述
wifi_get_mac	获取 mac 地址
wifi_set_mac	设置 mac 地址
wifi_send_expand_cmd	设置扩展命令

4.8.2 开发流程

步骤 1 获取 mac 地址

```
wifi_get_mac(ifname, mac_addr);
```

说明

(1) 用户需要申请空间 mac_addr 来保存获取到的 mac 地址
(2)ifname 是网卡名，如果传入的网卡名为 NULL，那么底层逻辑默认获取 wlan0 网卡的 mac 地址

步骤 2 设置 mac 地址

```
wifi_set_mac(ifname, mac_addr);
```

说明

(1)fname 是网卡名，如果传入的网卡名为 NULL，那么底层逻辑默认设置 wlan0 网卡的 mac 地址
(2) 设置网卡的 mac 地址是临时性的，重启后设置的 mac 地址会失效

步骤 3 发送扩展命令

```
wifi_send_expand_cmd(char *expand_cmd, void *expand_params, void *expand_cb);
```

说明

(1)expand_cmd: 发送的扩展命令，与系统和模组功能强相关，请查看源码
(2)expand_params: 发送扩展命令的参数，与系统和模组功能强相关，请查看源码
(3)expand_cb: 扩展命令的返回信息，与系统和模组功能强相关，请查看源码

4.8.3 编程实例

该部分开发不提供编程实例

ALLWINNER®

5 wifimg 测试工具介绍

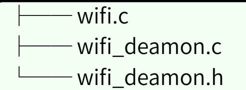
wifimg 同样提供了一个完整网络功能的测试工具，用户可以直接使用该测试工具来进行网络配置，连接等。也可以参考该 demo 对核心代码 API 的使用，把对应的功能集成到用户自己的应用中去。

5.1 测试工具目录结构

wifimg 测试工具源码在 SDK 里的路径如下：

wifimanager/demo/wifi_test_tool

wifi_test_tool 的主要目录结构如下：



```
├── wifi.c
├── wifi_deamon.c
└── wifi_deamon.h
```

- wifi.c：wifi 组件主要是提供了一些简单易用的命令行命令给用户。用户可以直接使用这些简单的命令行命令即可连接网络和配网。
- wifi_deamon.c：wifi_deamon 常驻后台进程，linux/xrlink 下的实现，wifi.c 文件通过解析用户输入的命令后通过 socket 接口发送给后台组件 wifi_deamon，由 wifi_deamon 组件实现真正的网络功能。freertos 下的实现，直接采用函数调用的方式。
- wifi_deamon.h：wifi_deamon 头文件。

5.2 测试工具配置

1. 根目录下执行 make menuconfig(openwrt 编译方式)/buildroot.sh buildroot_menuconfig(buildroot 编译方式) 2. 如下把对应的配置项选上即可。

```
openwrt编译方式
> Allwinner > Wireless > wifimanager-v2.0
 <*> wifimanager-v2.0-demo
     Wifimanager demo choice (DEMO_TEST_TOOL)

buildroot编译方式
> Target packages > allwinner platform private package select > wireless
 [*] wifimanager-v2.0
 [*]  wifimanager-v2.0-lib
 [*]   Tina wifimanager-v2.0-demo
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5.3 测试工具支持的命令

1. 命令行下执行 wifi -h 可以查看 wifi 组件支持的所有命令 (根据配置的不同以及系统的不同支持的命令种类会不同)。

2. 下面根据不同的模式对支持的命令进行分类说明。

3. 不需要单独开发的用户可以直接使用下面的命令进行 wifi 功能调试。

4. 扩展命令比较特殊与模组和系统平台有强相关，非通用性命令，此处不进行描述。

5.3.1 sta 模式命令

命令名称	wifi -o sta
参数说明	-o：以某种模式启动 wifimanager。 sta: 以 sta 模式启动 wifimanager。
功能描述	该命令用于打开 sta 模式。
备注：	使用所有与 sta 模式相关的命令前必须执行该命令。

命令名称	wifi -s
参数说明	-s：sta 模式下进行 ap 扫描。
功能描述	该命令用于在 sta 模式扫描 ap。
备注：	执行该命令前，需要执行 wifi -o sta。

命令名称	wifi -c ssid [passwd]
参数说明	-c：sta 模式下连接到某个特定的 ap。 ssid: 要连接的特定 ap 的 ssid。 passwd: 可选参数，要连接的 ap 的 passwd。
功能描述	该命令用于在 sta 模式下连接到某个 ap。
备注：	若参数 passwd 没有进行设置，那么认为连接的 ap 无加密，执行该命令前，需要执行 wifi -o sta。

命令名称	wifi -t ssid
参数说明	-t：sta 模式下尝试连接到某个曾经连接过的特定 ap。 ssid: 要连接的特定 ap 的 ssid(无需带密码，非无密码 ssid)。
功能描述	该命令用于在 sta 模式下连接到某个曾经连接过的特定 ap。

ALLWINNER®

命令名称	wifi -t ssid
备注：	无需带密码 (密码加密方式使用已保存的配置)，非无密码 ssid，该命令目前只在 linux 系统平台支持。

命令名称	wifi -d
参数说明	-d：断开与 ap 的连接。
功能描述	该命令用于在 sta 模式下断开与 ap 的连接。
备注：	需要已连接某个 ap 后执行才有效。

命令名称	wifi -a [enable/disable]
参数说明	-a：自动重连。 enable: 使能自动重连功能。 disable: 关闭自动重连功能。
功能描述	该命令主要用于使能或关闭自动重连的功能。
备注：	(1). 当打开自动重连功能后，如果此时有连接成功的 ap，会把该 ap 信息保存到系统中，并在下次系统起来后进行重连。 (2). 当打开自动重连功能后，如果此时已成功连接某个 ap，但某些原因，导致了当前连接断开，此时后台会不断对这个 ap 进行连接。

命令名称	wifi -l [all]
参数说明	-l：sta 模式下列出连接的 ap 的一些参数。 all: 可选参数 all，根据是否有 all 会列出不同的信息。
功能描述	该命令主要用于在 sta 模式下列出一些 sta 的参数。
备注：	(1). 没有参数 all 时列出的信息是当前已连接的 ap 的信息。 (2). 带参数 all 列出保存在系统的已连接过的 ap 信息，该命令主要用于配合 wifi -r 的命令使用。

命令名称	wifi -r [ssid/all]
参数说明	-r：sta 模式下移除 ap。 ssid: 可选参数移除某个特定 ap 的信息。 all: 可选参数移除所有保存的 ap 信息。
功能描述	该命令主要用于移除在系统中保存的 ap 的信息。
备注：	该命令主要用于配合 wifi -l 的命令使用，先使用 wifi -l 获取当前系统已保存的 ap 信息，再使用 wifi -r 命令移除想要移除的 ap 信息。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5.3.2 ap 模式命令

命令名称	wifi -o ap [ssid] [passwd]
参数说明	-o：以某种模式启动 wifimanager。 ap: 以 ap 模式启动 wifimanger。 ssid: 可选参数，设定启动的 ap 的 ssid。 passwd: 可选参数，设定启动的 ap 的 passwd。
功能描述	该命令用于打开 ap 模式，并创建 ap 热点。
备注：	该命令用于打开一个 ap 热点，根据输入的参数不同，启动的 ap 热点配置也不同。 (1). 如果可选参数 ssid 和 passwd 都没有填写，那么会使用默认配置启动一个 ap。 默认 ap 的 ssid 为: allwinner-ap。 默认 ap 的 passwd 为: Aa123456。 (2). 如果只配置了可选参数 ssid，那么会启动一个名称为 ssid 但无密码的 ap。 (3). 如果 ssid 和 passwd 都配置了，那么会根据配置启动 ap。

命令名称	wifi -s ap
参数说明	-s：ap 模式下扫描周围热点 (仅 rtos 系统支持，linux 系统不支持)。
功能描述	该命令主要用于在 ap 模式下扫描周围 ap 热点
备注：	无。

命令名称	wifi -l
参数说明	-l：ap 模式下列出连接到 ap 热点的 sta 信息。
功能描述	该命令主要用于在 ap 模式下列出连接到当前 ap 热点的 sta 信息以及 ap 热点的配置信息。
备注：	无。

命令名称	wifi -d ap
参数说明	-d：断开某种模式。 ap：关闭 ap 模式
功能描述	该命令主要用于在 ap 模式下关闭 ap。
备注：	无。

5.3.3 monitor 模式命令

版权所有 © 珠海全志科技股份有限公司。保留一切权利

命令名称	wifi -o monitor
参数说明	-o：以某种模式启动 wifimanager。 monitor: 以 monitor 模式启动 wifimanager。
功能描述	该命令用于打开 monitor 模式。
备注：	启动 monitor 模式后，需要注册回调函数，否则收不到监控的数据包。

5.3.4 p2p 模式命令

命令名称	wifi -o p2p [N:name] [I:intent] [T:time] [A:enable]
参数说明	-o：以某种模式启动 wifimanager。 p2p: 以 p2p 模式启动 wifimanager。N:name: 可选参数，是否设置 p2p 设备的设备名。I：intent：intent 值，影响 go/gc 模式的协助值。T: time：启动 p2p 模式后监听多少秒。A: enable/disable：是否开启被动连接
功能描述	该命令用于打开 p2p 模式。
备注：	注意可选参数的格式。

命令名称	wifi -s p2p
参数说明	-s：以某种模式进行扫描。 p2p: 以 p2p 模式进行扫描。
功能描述	该命令用于 p2p 模式时进行 p2p 设备扫描。
备注：	无。

命令名称	wifi -C macaddr
参数说明	-C：连接一个 p2p 设备。 macaddr: 要连接的 p2p 设备的 mac 地址。
功能描述	该命令用于连接一个 p2p 色板。
备注：	命令参数-C 是大写的 C。

命令名称	wifi -l p2p
参数说明	-l：列举信息。 p2p：列举已连接的 p2p 设备信息。
功能描述	该命令用于列举 p2p 模式时已连接的 p2p 设备的一些信息。
备注：	无。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

命令名称	wifi -d p2p
参数说明	-d：断开某些模式。 p2p: 断开 p2p 模式。
功能描述	该命令用于 p2p 模式时断开已连接的 p2p 设备。
备注：	无。

5.3.5 其他命令

命令名称	wifi -f
参数说明	-f：关闭 wifimanger。
功能描述	该命令用于关闭 wifimanger。
备注：	所有模式中调用 wifi -f 命令都会关闭 wifimanger，调用 wifi -f 命令后需要重新执行 wifi -o XXX 命令重新启动动 wifimager 才可以继续使用网络功能。

命令名称	wifi -D [error/warn/info/debug/dump/exce/open/close]
参数说明	-D：设置打印等级。 erro: 设置打印等级为 erro。 warn: 设置打印等级为 warn。 info: 设置打印等级为 info。 debug: 设置打印等级为 debug。 dump: 设置打印等级为 dump。 exce: 设置打印等级为 exce。 open：增加额外打印信息，包括时间、文件名、函数名、行号。 close：关闭额外的打印信息。
功能描述	该命令主要是设置打印等级。
备注：	打印 erro 为最低，exce 为最高，erro 打印等级只打印错误信息，warn 打印等级会额外多打印 warn 信息，info 打印等级是正常打印等级 (默认开启的打印等级)，debug 打印等下会打印出 debug 信息，dump 等级会把一些通信数据打印出来，exce 会把一些执行命令的信息打印出来。open 和 close 用于控制是否增加额外的打印信息 (例如文件名，函数名，行号等)。

命令名称	wifi -g
参数说明	-g：获取 mac 地址信息。
功能描述	该命令主要用于获取当前系统的 mac 地址信息。
备注：	需在执行该命令前执行 wifi -o XXX 命令。

ALLWINNER®

命令名称	wifi -m [macaddr]
参数说明	-m：设置 mac 地址。 macaddr: 要设置的 mac 地址。
功能描述	该命令主要用于设置当前系统的 mac 地址。
备注:	需在执行该命令前执行 wifi -o XXX 命令。

命令名称	wifi -e [linux: XXX]/[xrlink: XXX]/[freertos: XXX]
参数说明	-e：支持的扩展命令 (非通用性，不同模组支持的命令不一样)。 linux:XXX(linux 系统下不同模组支持的扩展命令)。xrlink:XXX(xrlink 系统下不同模组支持的扩展命令)。freertos:XXX(freertos 系统下不同模组支持的扩展命令)。
功能描述	该命令主要用于使用模组特殊功能。
备注:	不同模组支持和实现不一样，具体需要看代码实现。

命令名称	wifi -i
参数说明	-i：打印当前 wifimanager 的一些状态信息。
功能描述	该命令主要用于打印当前 wifimanager 的一些状态信息，包括支持什么模式，当前是什么模式，当前模式的状态等。
备注:	无。

命令名称	wifi -h
参数说明	-h：打印说明。
功能描述	该命令主要打印 wifi 组件支持的命令。
备注:	无。

5.3.6 配网模式命令

配网模式需要手机软件配合，下面描述如何使用手机 app 进行配网

相关手机 app 在 SDK 里 wifimanager/app 目录里可以获取到，目前只提供安卓版本

命令名称	wifi -p [softap/ble/soundwave]
参数说明	-p：以某种方式进行配网。 softap: 以 softap 模式进行配网。 ble: 以 ble 模式进行配网。 soundwave: 以 soundwave 模式进行配网。
功能描述	该命令主要用于打开某种配网模式进行配网。

命令名称	wifi -p [softap/ble/soundwave]
备注:	无

5.3.6.1 softap 配网

用例名称	softap 配网
功能说明	使用 softap 配网模式进行配网
前置条件	手机安装 softap 配网工具 (ckySoftAPDemo)
操作步骤	1. 执行：小机端执行命令 wifi -p softap 2. 手机端按附件插图操作 ckySoftAPDemo 工具发送 ssid 和 psk 给样机

手机操作附件插图：

Aw-config-net-Test
密码Aa123456
1. 手机进行扫描并连接上该热点
下午9:26
ckySoftAPDemo
Target IP: 255.255.255.255
Target Port: 8066
Locat Port: 8066
Locat Port: 0.0.0.0
SEND
STOP
2. 打开软件并填写要连接的ssid和password
SSID: 12345678
Password: 12345678
WPA-PSK
3. 点击发送
一键连接

图 5-1: softap 配网

5.3.6.2 ble 配网

用例名称	ble 配网
功能说明	使用 ble 配网模式进行配网
前置条件	手机安装 ble 配网工具 (Blink)
操作步骤	1. 执行：小机端执行命令 wifi -p ble 2. 手机端按附件插图操作 Blink 工具发送 ssid 和 psk 给样机

手机操作附件插图：

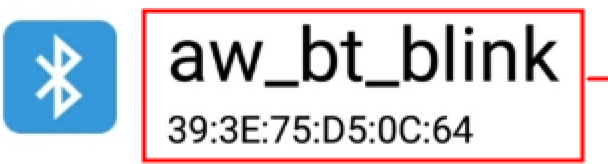


图 5-2: ble 配网

5.3.6.3 soundwave 配网

用例名称	soundwave 配网
功能说明	使用 soundwave 配网模式进行配网
前置条件	手机安装 soundwave 配网工具 (SoundAuthenticationTest)
操作步骤	1. 执行：小机端执行命令 wifi -p soundwave 2. 手机端按附件插图操作 SoundAuthenticationTest 工具发送 ssid 和 psk 给样机

手机操作附件插图：

图 5-3: soundwave 配网

6 附件

6.1 wifimg 结构体详细说明

说明

该章节主要用于描述核心代码中使用到的一些关键的结构体。不需要单独阅读该章节，该章节属于查询性质，当在其他章节中查看到需要查询的结构体时再查询该章节即可。该章节的关键结构体都在下面文件里定义。

wifimanager/core/include/wifimg.h

wmg_status_t 结构体

结构体描述：该结构体主要用于定义 wifimanger 各函数执行后的返回值。

```
typedef enum {
   WMG_STATUS_SUCCESS = 0,
   WMG_STATUS_FAIL = -1,
   WMG_STATUS_NOT_READY = -2,
   WMG_STATUS_NOMEM = -3,
   WMG_STATUS_BUSY = -4,
   WMG_STATUS_UNSUPPORTED = -5,
   WMG_STATUS_INVALID = -6,
   WMG_STATUS_TIMEOUT = -7,
   WMG_STATUS_UNHANDLED = -8,
} wmg_status_t;
```

- WMG_STATUS_SUCCESS：函数执行成功。
- WMG_STATUS_FAIL：函数执行失败。
- WMG_STATUS_NOT_READY：函数没有准备好。
- WMG_STATUS_NOMEM：函数无法申请到需要的内存。
- WMG_STATUS_BUSY：函数处于忙状态。
- WMG_STATUS_UNSUPPORTED：函数不支持该功能。
- WMG_STATUS_INVALID：函数收到无效数据。
- WMG_STATUS_TIMEOUT：函数执行超时。
- WMG_STATUS_UNHANDLED：函数不处理该次调用。

wifi_mode_t 结构体

结构体描述：该结构体主要用于定义 wifimanger 支持的模式。

```
typedef enum {
   WIFI_MODE_UNKNOWN = 0b0,
   WIFI_STATION = 0b1,
   WIFI_AP = 0b10,
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
    WIFI_STATION_AP = 0b11,
    WIFI_MONITOR = 0b100,
    WIFI_P2P = 0b1000,
    WIFI_STATION_P2P = 0b1001,
} wifi_mode_t;
```

- WIFI_STATION：station 模式。
- WIFI_AP：ap 模式。
- WIFI_MONITOR：monitor 模式 (该模式需要看方案和模组是否支持)。
- WIFI_P2P：p2p 模式 (该模式需要看方案和模组是否支持)。
- WIFI_STATION_AP：station + ap 共存模式 (该模式需要看方案和模组是否支持)。
- WIFI_STATION_P2P：station + p2p 共存模式 (该模式需要看方案和模组是否支持)。
- WIFI_MODE_UNKNOWN：未定义模式。

wifi_dev_status_t 结构体

结构体描述：该结构体主要用于定义 wifimanger 对网卡设备状态的识别。

```
typedef enum {
    WLAN_STATUS_DOWN,
    WLAN_STATUS_UP,
} wifi_dev_status_t;
```

- WLAN_STATUS_DOWN：网卡设备关闭。
- WLAN_STATUS_UP：网卡设备启动。

wifi_msg_id_t 结构体

结构体描述：该结构体主要用于定义 wifimanger 收到的回调消息的类型。

```
typedef enum {
    WIFI_MSG_ID_DEV_STATUS,
    WIFI_MSG_ID_STA_CN_EVENT,
    WIFI_MSG_ID_STA_STATE_CHANGE,
    WIFI_MSG_ID_AP_CN_EVENT,
    WIFI_MSG_ID_P2P_CN_EVENT,
    WIFI_MSG_ID_P2P_STATE_CHANGE,
    WIFI_MSG_ID_MONITOR,
    WIFI_MSG_ID_MAX,
} wifi_msg_id_t;
```

- WIFI_MSG_ID_DEV_STATUS：设备状态发生了改变的消息。
- WIFI_MSG_ID_STA_CN_EVENT：sta 模式在连接过程中事件发生改变的消息。
- WIFI_MSG_ID_STA_STATE_CHANGE：sta 模式状态发生改变的消息。
- WIFI_MSG_ID_AP_CN_EVENT：ap 模式在连接过程中事件发生改变的消息。
- WIFI_MSG_ID_P2P_CN_EVENT：p2p 模式在连接过程中事件发生改变的消息。
- WIFI_MSG_ID_P2P_STATE_CHANGE：p2p 模式状态发生改变的消息。
- WIFI_MSG_ID_MONITOR：monitor 模式的消息。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

- WIFI_MSG_ID_MAX：无意义消息类型，界限结构体用。

wifi_secure_t 结构体

结构体描述：该结构体主要用于定义 wifimanger 的加密方式。

```
typedef enum {
    WIFI_SEC_NONE = 0b0,
    WIFI_SEC_WEP = 0b1,
    WIFI_SEC_WPA_PSK = 0b10,
    WIFI_SEC_WPA2_PSK = 0b100,
    WIFI_SEC_WPA2_PSK_SHA256 = 0b1000,
    WIFI_SEC_WPA3_PSK = 0b10000,
    WIFI_SEC_EAP = 0b100000,
} wifi_secure_t;
```

- WIFI_SEC_NONE：无加密。
- WIFI_SEC_WEP：wep 加密方式。
- WIFI_SEC_WPA_PSK：wpa 加密方式。
- WIFI_SEC_WPA2_PSK：wpa2 加密方式。
- WIFI_SEC_WPA2_PSK_SHA256 ：wpa2 保护帧方式。
- WIFI_SEC_WPA3_PSK：wpa3 加密方式。
- WIFI_SEC_EAP ：扩展认证协议。

wifi_sta_state_t 结构体

结构体描述：该结构体主要用于定义 wifimanger 的 station 模式状态。

```
typedef enum {
    WIFI_STA_IDLE,
    WIFI_STA_CONNECTING,
    WIFI_STA_CONNECTED,
    WIFI_STA_OBTAINING_IP,
    WIFI_STA_NET_CONNECTED,
    WIFI_STA_DHCP_TIMEOUT,
    WIFI_STA_DISCONNECTING,
    WIFI_STA_DISCONNECTED,
    WIFI_STA_BUSY_TIMEOUT,
} wifi_sta_state_t;
```

- WIFI_STA_IDLE：station 模式处于空闲状态。
- WIFI_STA_CONNECTING：station 模式处于正在连接 ap 状态。
- WIFI_STA_CONNECTED：station 模式处于已连接上 ap 状态。
- WIFI_STA_OBTAINING_IP：station 模式处于正在获取 IP 状态。
- WIFI_STA_NET_CONNECTED：station 模式处于网络连接已完成状态。
- WIFI_STA_DHCP_TIMEOUT：station 模式处于 DHCP 超时状态。
- WIFI_STA_DISCONNECTING：station 模式处于正在取消连接状态。
- WIFI_STA_DISCONNECTED：station 模式处于已取消连接状态。
- WIFI_STA_BUSY_TIMEOUT：station 模式设备忙超时状态。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

wifi_sta_event_t 结构体

结构体描述：该结构体主要用于定义 wifimanger 的 station 模式在连接 ap 过程中可能会出现的事件。

```
typedef enum {
    WIFI_DISCONNECTED,
    WIFI_SCAN_STARTED,
    WIFI_SCAN_SUCCESS,
    WIFI_SCAN_FAILED,
    WIFI_SCAN_RESULTS,
    WIFI_NETWORK_NOT_FOUND,
    WIFI_PASSWORD_INCORRECT,
    WIFI_AUTHENTIACATION,
    WIFI_AUTH_REJECT,
    WIFI_AUTH_TIMEOUT,
    WIFI_AUTH_FAILED,
    WIFI_DEAUTH,
    WIFI_ASSOCIATING,
    WIFI_ASSOC_REJECT,
    WIFI_ASSOCIATED,
    WIFI_ASSOC_TIMEOUT,
    WIFI_ASSOC_FAILED,
    WIFI_DISASSOC,
    WIFI_4WAY_HANDSHAKE,
    WIFI_4WAY_HANDSHAKE_FAILED,
    WIFI_GROUNP_HANDSHAKE,
    WIFI_GROUNP_HANDSHAKE_DONE,
    WIFI_CONNECTED,
    WIFI_CONNECT_TIMEOUT,
    WIFI_CONNECT_FAILED,
    WIFI_CONNECTION_LOSS,
    WIFI_DEV_HANG,
    WIFI_DHCP_START,
    WIFI_DHCP_TIMEOUT,
    WIFI_DHCP_SUCCESS,
    WIFI_SAE_COMMIT_FAILED,
    WIFI_SAE_CONFIRM_FAILED,
    WIFI_NETWORK_UP,
    WIFI_NETWORK_DOWN,
    WIFI_TERMINATING,
    WIFI_UNKNOWN,
} wifi_sta_event_t;
```

- WIFI_DISCONNECTED：已取消连接。
- WIFI_SCAN_STARTED：扫描开始。
- WIFI_SCAN_SUCCESS：扫描成功。
- WIFI_SCAN_FAILED：扫描失败。
- WIFI_SCAN_RESULTS：获取到扫描结果。
- WIFI_NETWORK_NOT_FOUND：没有找到对应的 network。
- WIFI_PASSWORD_INCORRECT：密码不正确。
- WIFI_AUTHENTIACATION：认证。
- WIFI_AUTH_REJECT：认证被拒绝。
- WIFI_AUTH_TIMEOUT：认证超时。
- WIFI_AUTH_FAILED：认证失败。

- WIFI_DEAUTH：取消认证。
- WIFI_ASSOCIATING：关联。
- WIFI_ASSOC_REJECT：关联被拒绝。
- WIFI_ASSOCIATED：关联完成。
- WIFI_ASSOC_TIMEOUT：关联超时。
- WIFI_ASSOC_FAILED：关联失败。
- WIFI_DISASSOC：取消关联。
- WIFI_4WAY_HANDSHAKE：4 次握手。
- WIFI_4WAY_HANDSHAKE_FAILED：4 次握手失败。
- WIFI_GROUNP_HANDSHAKE：交换组密钥。
- WIFI_GROUNP_HANDSHAKE_DONE：交换组密钥完成。
- WIFI_CONNECTED：连接完成。
- WIFI_CONNECT_TIMEOUT：连接超时。
- WIFI_CONNECT_FAILED：连接失败。
- WIFI_CONNECTION_LOSS：连接丢失。
- WIFI_DEV_HANG：底层设备异常。
- WIFI_DHCP_START：DHCP 开始。
- WIFI_DHCP_TIMEOUT：DHCP 超时。
- WIFI_DHCP_SUCCESS：DHCP 成功。
- WIFI_SAE_COMMIT_FAILED：SAE 加密失败。
- WIFI_SAE_CONFIRM_FAILED：SAE 加密失败。
- WIFI_NETWORK_UP：rtos 系统网络启动。
- WIFI_NETWORK_DOWN：rtos 系统网络关闭。
- WIFI_TERMINATING：终止。
- WIFI_UNKNOWN：未知。

wifi_sta_info_t 结构体

结构体描述：该结构体主要用于定义 wifimanger 的 station 模式已经连接成功后的一些信息。

```
typedef struct {
    int id;
    int freq;
    int rssi;
    uint8_t bssid[6];
    char ssid[SSID_MAX_LEN + 1];
    uint8_t mac_addr[6];
    uint8_t ip_addr[4];
    uint8_t gw_addr[4];
    wifi_secure sec;
} wifi_sta_info_t;
```

- id：wpa_supplicant 里保存的 network id 号，某些系统不会使用到，用户可以不用关心。
- freq：频率 (指的是信道频率，2412 = channel 1)。
- rssi：信号强度。
- bssid[6]：连接的 ap 的 bssid。

- ssid[SSID_MAX_LEN]：连接的 ap 的 ssid。
- mac_addr[6]：本地 mac 地址。
- ip_addr[4]：本地 ip 地址。
- gw_addr[4]：网关地址。
- sec：加密方式。

wifi_sta_list_nod_t 结构体

结构体描述：该结构体主要用于定义 wifimanger station 模式时曾经连接过的一条 ap 的信息。

```
typedef struct {
    int id;
    char ssid[SSID_MAX_LEN + 1];
    uint8_t bssid[6];
    char flags[16];
} wifi_sta_list_nod_t;
```

- id：wpa_supplicant 里保存的 network id 号，某些系统不会使用到，用户可以不用关心。
- ssid：连接过的 ap 的 ssid。
- bssid：连接过的 ap 的 bssid。
- flags：一些状态码，用户可以不用关心。

wifi_sta_cn_para_t 结构体

结构体描述：该结构体主要用于描述 wifimanger station 模式时要连接的 ap 的配置信息，在连接某个特定 ap 时，用户需要填充这个结构体。

```
typedef struct {
    const char * ssid;
    const char * password;
    uint8_t bssid[6];
    wifi_secure_t sec;
    bool fast_connect;
} wifi_sta_cn_para_t;
```

- ssid：要连接的 ap 的 ssid。
- password：要连接的 ap 的 password。
- bssid：要连接的 ap 的 bssid(使用 bssid 连接方式使用，非 bssid 连接方式可为空，仅 freerto 系统支持，linux 系统暂未支持)。
- sec：要连接的 ap 的加密方式。
- fast_connect：该参数暂时没有作用，扩展用，用户可以不用关心。

wifi_scan_result_t 结构体

结构体描述：该结构体主要用于描述 wifimanger station 模式时扫描到的一条 ap 结果包含什么内容。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
typedef struct {
    uint8_t bssid[6];
    char ssid[SSID_MAX_LEN + 1];
    uint32_t freq;
    int rssi;
    wifi_secure_t key_mgmt;
    bool scan_action;
} wifi_scan_result_t;
```

- bssid：扫描到的 ap 的 bssid。
- ssid：扫描到的 ap 的 ssid。
- freq：扫描到的 ap 的频率 (指的是信道频率，2412 = channel 1)。
- rssi：扫描到的 ap 的信号强度。
- key_mgmt：扫描到的 ap 的加密方式。
- scan_action：重新扫描获取扫描结果或使用缓存扫描结果标志位。

wifi_ap_state_t 结构体

结构体描述：该结构体主要用于描述 wifimanger ap 模式时的状态。

```
typedef enum {
    WIFI_AP_DISABLE,
    WIFI_AP_ENABLE,
} wifi_ap_state_t;
```

- WIFI_AP_DISABLE：ap 模式处于非使能状态。
- WIFI_AP_ENABLE：ap 模式处于使能状态。

wifi_ap_event_t 结构体

结构体描述：该结构体主要用于描述 wifimanger ap 模式时会收到的事件。

```
typedef enum {
    WIFI_AP_ENABLED = 1,
    WIFI_AP_DISABLED,
    WIFI_AP_STA_DISCONNECTED,
    WIFI_AP_STA_CONNECTED,
    WIFI_AP_UNKNOWN,
} wifi_ap_event_t;
```

- WIFI_AP_ENABLED：ap 模式已使能。
- WIFI_AP_DISABLED：ap 模式未使能。
- WIFI_AP_STA_DISCONNECTED：ap 模式触发了有 sta 取消连接事件。
- WIFI_AP_STA_CONNECTED：ap 模式触发了有 sta 进行连接事件。
- WIFI_AP_UNKNOWN：ap 模式下未定义事件。

wifi_ap_config_t 结构体

结构体描述：该结构体主要用于定义 wifimanger ap 模式时开启的 ap 热点的配置信息。

```
typedef struct {
    char *ssid;
    char *psk;
    wifi_secure sec;
    uint8_t channel;
    int key_mgmt;
    uint8_t mac_addr[6];
    uint8_t ip_addr[4];
    uint8_t gw_addr[4];
    char *dev_list[STA_MAX_NUM];
    uint8_t sta_num;
} wifi_ap_config_t;
```

- ssid：要开启的 ap 热点的 ssid。
- psk：要开启的 ap 热点的密码。
- sec：要开启的 ap 热点的加密方式。
- channel：要开启的 ap 热点的信道。
- key_mgmt：加密类型。
- mac_addr：开启的 ap 地址 (信息获取非设置)。
- ip_addr：开启的 ap 的 ip 地址 (信息获取非设置)。
- gw_addr：开启的 ap 的网关 (信息获取非设置)。
- dev_list：连接到 ap 热点的 sta 设备 (信息获取非设置)。
- sta_num：连接到 ap 热点的 sta 设备的个数 (信息获取非设置)。

wifi_monitor_state_t 结构体

结构体描述：该结构体主要用于定义 wifimanger monitor 模式的状态。

```
typedef enum {
    WIFI_MONITOR_DISABLE,
    WIFI_MONITOR_ENABLE,
    WIFI_MONITOR_UNKNOWN,
} wifi_monitor_state_t;
```

- WIFI_MONITOR_DISABLE：使能状态的 monitor 模式。
- WIFI_MONITOR_ENABLE：使能状态的 monitor 模式。
- WIFI_MONITOR_UNKNOWN：未知的 monitor 模式

wifi_monitor_data_t 结构体

结构体描述：该结构体主要用于描述 wifimanger monitor 模式时收到的帧的内容。

```
typedef struct {
    uint8_t *data;
    uint32_t len;
    uint8_t channel;
    void *info;
} wifi_monitor_data_t;
```

- data：monitor 模式时收到的帧数据。

- len：monitor 模式时收到的帧的长度。
- channel：monitor 模式时从什么信道收到的帧。
- info：monitor 模式时收到帧的扩展信息 (目前暂时没有意义)。

wifi_p2p_config_t 结构体

结构体描述：该结构体主要用于描述 wifimanger p2p 模式启动时配置什么参数。

```
typedef struct {
    char *dev_name;
    int listen_time;
    int p2p_go_intent;
    bool auto_connect;
} wifi_p2p_config_t;
```

- dev_name：p2p 设备名。
- listen_time：启动 p2p 模式时进行监听多少秒。
- p2p_go_intent：启动 p2p 模式是 go intent 值设置多少 (0 ~ 15 会影响到 gc go 的协商)。
- auto_connect：是否支持被动连接。

wifi_p2p_peers_t 结构体

结构体描述：该结构体主要用于描述 wifimanger p2p 模式时扫描到的 p2p 设备。

```
typedef struct {
    char dev_name[P2P_DEV_NAME_MAX_LEN + 1];
    uint8_t mac_addr[6];
} wifi_p2p_peers_t;
```

- dev_name：找到的 p2p 设备名。
- mac_addr：找到的 p2p 设备的 mac 地址

wifi_p2p_info_t 结构体定义

结构体描述：该结构体主要用于描述 wifimanger p2p 模式连接成功后的信息。

```
typedef struct {
    uint8_t bssid[6];
    int mode;
    int freq;
    char ssid[SSID_MAX_LEN + 1];
} wifi_p2p_info_t;
```

- bssid：p2p 的 bssid。
- mode：连接成功后协商的模式 go/gc。
- freq：连接成功后的频率 (信道)。
- ssid：连接成功后的 ssid。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

wifi_p2p_state_t 结构体

结构体描述：该结构体主要用于描述 wifimanger p2p 模式时连接的状态。

```
typedef enum {
    WIFI_P2P_ENABLE,
    WIFI_P2P_DISABLE,
    WIFI_P2P_CONNECTD_GC,
    WIFI_P2P_CONNECTD_GO,
    WIFI_P2P_DISCONNECTD,
} wifi_p2p_state_t;
```

- WIFI_P2P_ENABLE：使能状态的 p2p 模式。
- WIFI_P2P_DISABLE：不使能状态的 p2p 模式。
- WIFI_P2P_CONNECTD_GC：连接状态，协商模式为 gc。
- WIFI_P2P_CONNECTD_GO：连接状态，协商模式未 go。
- WIFI_P2P_DISCONNECTD：未连接状态。

wifi_p2p_event_t 结构体

结构体描述：该结构体主要用于描述 wifimanger p2p 模式时连接是可能会触发的事件。

```
typedef enum {
    WIFI_P2P_DEV_FOUND,
    WIFI_P2P_DEV_LOST,
    WIFI_P2P_PBC_REQ,
    WIFI_P2P_GO_NEG_RQ,
    WIFI_P2P_GO_NEG_SUCCESS,
    WIFI_P2P_GO_NEG_FAILURE,
    WIFI_P2P_GROUP_FOR_SUCCESS,
    WIFI_P2P_GROUP_FOR_FAILURE,
    WIFI_P2P_GROUP_STARTED,
    WIFI_P2P_GROUP_REMOVED,
    WIFI_P2P_CROSS_CONNECT_ENABLE,
    WIFI_P2P_CROSS_CONNECT_DISABLE,
    WIFI_P2P_SCAN_STARTED,
    WIFI_P2P_AP_STA_DISCONNECTED,
    /*wifimanager self-defined state*/
    WIFI_P2P_SCAN_RESULTS,
    WIFI_P2P_GROUP_DHCP_DNS_FAILURE,
    WIFI_P2P_GROUP_DHCP_SUCCESS,
    WIFI_P2P_GROUP_DHCP_FAILURE,
    WIFI_P2P_GROUP_DNS_SUCCESS,
    WIFI_P2P_GROUP_DNS_FAILURE,
    WIFI_P2P_UNKNOWN,
} wifi_p2p_event_t;
```

- WIFI_P2P_DEV_FOUND：寻找 p2p 设备事件。
- WIFI_P2P_DEV_LOST：p2p 设备丢失事件。
- WIFI_P2P_PBC_REQ：以 pbc 方式连接请求事件。
- WIFI_P2P_GO_NEG_RQ：go 模式协商请求事件。
- WIFI_P2P_GO_NEG_SUCCESS：go 模式协商成功事件。
- WIFI_P2P_GO_NEG_FAILURE：go 模式协商失败事件。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

- WIFI_P2P_GROUP_FOR_SUCCESS：p2p 组创建成功事件。
- WIFI_P2P_GROUP_FOR_FAILURE：p2p 组创建失败事件。
- WIFI_P2P_GROUP_STARTED：p2p 组创建开始事件。
- WIFI_P2P_GROUP_REMOVED：p2p 移除组事件。
- WIFI_P2P_CROSS_CONNECT_ENABLE。
- WIFI_P2P_CROSS_CONNECT_DISABLE。
- WIFI_P2P_SCAN_STARTED：p2p 扫描开始事件
- WIFI_P2P_AP_STA_DISCONNECTED：go 角色时收到 gc 断开事件。
- /wifimanager self-defined state/ –> wifimanager 自定义事件。
- WIFI_P2P_SCAN_RESULTS：p2p 扫描完成事件。
- WIFI_P2P_GROUP_DHCP_DNS_FAILURE：p2p 组 dhcp/dns 启动失败事件。
- WIFI_P2P_GROUP_DHCP_SUCCESS：p2p dhcp 成功事件 (gc 模式下获取到了 ip 地址)。
- WIFI_P2P_GROUP_DHCP_FAILURE：p2p dhcp 失败事件。
- WIFI_P2P_GROUP_DNS_SUCCESS：p2p dns 服务启动成功事件 (go 模式下 dns 服务成功)。
- WIFI_P2P_GROUP_DNS_FAILURE：p2p dns 服务启动失败事件。
- WIFI_P2P_UNKNOWN：未知 p2p 事件。

wifi_msg_data_t 结构体

结构体描述：该结构体主要用于定义 wifimanger 可能会收到的回调的事件。

```
typedef struct {
    wifi_msg_id_t id;
    union {
        wifi_dev_status_t d_status;
        wifi_sta_event_t event;
        wifi_sta_state_t state;
        wifi_ap_event_t ap_event;
        wifi_ap_state_t ap_state;
        wifi_monitor_state_t mon_state;
        wifi_monitor_data_t *frame;
        wifi_p2p_event_t p2p_event;
        wifi_p2p_state_t p2p_state;
    } data;
} wifi_msg_data_t;
```

- id：回调事件的数据类型，根据这个 id 好确定 data 里的数据是什么类型的。data：数据类型
- d_status：回调事件数据类型是设备状态变化信息。
- even：回调事件数据类型是 sta 模式连接 ap 过程中状态变化信息。
- state：回调事件数据类型是 sta 模式状态信息。
- ap_event：回调事件数据类型是 ap 模式连接过程中状态变化信息。
- ap_state：回调事件数据类型是 ap 模式状态信息。
- mon_state：回调事件数据类型是 monitor 模式状态信息。
- p2p_event：回调事件数据类型是 p2p 模式连接过程中状态变化信息。
- p2p_state：回调事件数据类型是 p2p 模式状态信息。
- frame：回调事件数据类型是 monitor 模式收到的数据帧。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

6.2 wifimgAPI 详细说明

该章节主要介绍 lib 中各 API(需要 2 次开发的人员重点关注和查阅该章节)。

该章节的所有 API 对应的是 wifimanager 中 wifimg.h 头文件。

平台相关 API

wifi_init

函数原型	wmg_status_t wifi_init(void)
参数说明	无。
返回说明	WMG_STATUS_SUCCESS(0)：wifimanager 初始化成功； 非 WMG_STATUS_SUCCESS(非 0)：wifimanager 初始化失败。
功能描述	初始化 wifimanager。
备注：	想要使用 wifimanager 的功能前，必须调用该函数进行 wifimanager 初始化(仅需调用 1 次即可)， 反初始化函数调用后需重新调用该函数。

wifi_deinit

函数原型	wmg_status_t wifi_deinit(void)
参数说明	无。
返回说明	WMG_STATUS_SUCCESS(0)：wifimanager 反初始化成功； 非 WMG_STATUS_SUCCESS(非 0)：wifimanager 反初始化失败。
功能描述	反初始化 wifimanager。
备注：	

wifi_on

函数原型	wmg_status_t wifi_on(wifi_mode_t mode)
参数说明	mode：需要打开的模式（WIFI_STATION / WIFI_AP / WIFI_MONITOR / WIFI_P2P）。
返回说明	WMG_STATUS_SUCCESS(0)：打开模式成功； 非 WMG_STATUS_SUCCESS(非 0)：打开模式失败。
功能描述	打开某种 WiFi 模式。
备注：	想要使用 wifimanager 的某种模式前，必须调用该函数打开某种模式。

wifi_off

版权所有 © 珠海全志科技股份有限公司。保留一切权利

函数原型	wmg_status_t wifi_off(wifi_mode_t mode);
参数说明	mode：需要关闭的模式 (共存模式会使用到，输入 WIFI_MODE_UNKNOWN 时关闭所有模式)
返回说明	WMG_STATUS_SUCCESS(0)：关闭模式成功； 非 WMG_STATUS_SUCCESS(非 0)：关闭模式失败。
功能描述	关闭某种 WIFI 模式或关闭所有模式。
备注：	共存模式时：可以单独关闭某种模式，若所有模式均关闭会关闭 wifimanager 或输入 WIFI_MODE_UNKNOWN 显式关闭所有模式 非共存模式下：输入参数 WIFI_MODE_UNKNOWN 即可 (底层会自动进行模式切换)

sta 模式相关 API wifi_sta_connect

函数原型	wmg_status_t wifi_sta_connect(wifi_sta_cn_para_t * cn_para)
参数说明	cn_para(需要连接的 ap 的信息，结构体成员具体说明请查看 wifi_sta_cn_para_t 结构体说明)。
返回说明	WMG_STATUS_SUCCESS(0)：连接成功； 非 WMG_STATUS_SUCCESS(非 0)：连接失败。
功能描述	连接到某个 ap。
备注：	无。

wifi_sta_disconnect

函数原型	wmg_status_t wifi_sta_disconnect(void)
参数说明	无。
返回说明	WMG_STATUS_SUCCESS(0)：断开连接成功； 非 WMG_STATUS_SUCCESS(非 0)：断开连接失败。
功能描述	断开与 ap 的连接。
备注：	只有在已连接上某个 ap 后调用该接口才有用，否则会返回失败。

wifi_sta_auto_reconnect

函数原型	wmg_status_t wifi_sta_auto_reconnect(wmg_bool_t enable)
参数说明	enable：打开或关闭该功能。
返回说明	WMG_STATUS_SUCCESS(0)：打开或关闭该功能成功； 非 WMG_STATUS_SUCCESS(非 0)：打开或关闭该功能失败。
功能描述	自动连上某个 ap。
备注：	该功能主要有 2 个附加作用 (该接口只有在 os 为 linux 时有效，os 为 xrlink/freertos 时暂未支持)

版权所有 © 珠海全志科技股份有限公司。保留一切权利

函数原型	wmg_status_t wifi_sta_auto_reconnect(wmg_bool_t enable)
	(1). 当系统有保存已连接过的 ap 信息时，调用该接口后会自动尝试去连接已连接过的 ap。 (2). 当已连接上某个 ap 后，因某些原因导致了与 ap 断开，打开了该功能会尝试继续连接该 ap。 (3). 该功能生效的前提是曾经连接过某个 ap

wifi_sta_auto_connect

函数原型	wmg_status_t wifi_sta_auto_connect(char *ssid);
参数说明	ssid：要连接曾经连接过的特定 ap。
返回说明	WMG_STATUS_SUCCESS(0)：尝试自动连接成功； 非 WMG_STATUS_SUCCESS(非 0)：尝试自动连接失败； WMG_STATUS_UNSUPPORTED：未曾连接过该 ap；
功能描述	指定连接某个曾经连接过的 ap。
备注：	(该接口只有在 os 为 linux 时有效，os 为 xrlink/freertos 时暂未支持) (1). 当系统有保存已连接过的特定 ap 信息时，调用该接口后会自动使用保存的信息去连接该指定的 ap

wifi_sta_get_info

函数原型	wmg_status_t wifi_sta_get_info(wifi_sta_info_t * sta_info)
参数说明	sta_info (能获取的 station 模式的信息，结构体成员具体说明请查看 wifi_sta_info_t 结构体说明)。
返回说明	WMG_STATUS_SUCCESS(0)：获取信息成功； 非 WMG_STATUS_SUCCESS(非 0)：获取信息失败。
功能描述	获取当前 station 模式状态的一些信息 (包括连接上的 ap 的 ssid，bssid 等)。
备注：	无。

wifi_sta_list_networks

函数原型	wmg_status_t wifi_sta_list_networks(wifi_sta_list_t * sta_list);
参数说明	sta_list(保存列表信息，结构体成员具体说明请查看 wifi_sta_list_t 结构体说明)。
返回说明	WMG_STATUS_SUCCESS(0)：获取保存的列表信息成功； 非 WMG_STATUS_SUCCESS(非 0)：获取保存的列表信息失败。
功能描述	在 sta 模式下获取保存 ap 列表信息。
备注：	(该接口只有在 os 为 linux/xrlink 时有效，os 为 freertos 时暂未支持)

版权所有 © 珠海全志科技股份有限公司。保留一切权利

wifi_sta_remove_networks

函数原型	wmg_status_t wifi_sta_remove_networks(char * ssid)
参数说明	ssid：要移除的 ap 的 ssid；NULL：输入 NULL 时把所有保存的列表信息均删除
返回说明	WMG_STATUS_SUCCESS(0)：删除某个或所有列表成功； 非 WMG_STATUS_SUCCESS(非 0)：删除某个或所有失败。
功能描述	在 sta 模式下移除某个或全部已保存的 ap 的信息。
备注：	(该接口只有在 os 为 linux/xrlink 时有效，os 为 freertos 时暂未支持) 该接口主要配合 wifi_sta_list_networks 接口使用。

ap 模式模式相关 API wifi_ap_enable

函数原型	wmg_status_t wifi_ap_enable(wifi_ap_config_t * ap_config)
参数说明	ap_config(使能 ap 热点时的配置参数，结构体成员具体说明请查看 wifi_ap_config_t 结构体说明)。
返回说明	WMG_STATUS_SUCCESS(0)：ap 热点使能成功； 非 WMG_STATUS_SUCCESS(非 0)：失败。
功能描述	在 ap 模式下启动 ap 热点功能。
备注：	根据 wifi_ap_config_t 的设置不同，启动的热点会不同。

wifi_ap_disable

函数原型	wmg_status_t wifi_ap_disable(void)
参数说明	无。
返回说明	WMG_STATUS_SUCCESS(0)：关闭 ap 热点成功； 非 WMG_STATUS_SUCCESS(非 0)：关闭 ap 热点失败。
功能描述	在 ap 模式下关闭 ap 热点功能。
备注：	无。

wifi_ap_get_config

函数原型	wifi_ap_get_config(wifi_ap_config_t * ap_config)
参数说明	ap_config 获取 (ap 热点的配置参数，结构体成员具体说明请查看 wifi_ap_config_t 结构体说明)。
返回说明	WMG_STATUS_SUCCESS(0)：获取配置参数成功； 非 WMG_STATUS_SUCCESS(非 0)：获取配置参数失败。
功能描述	在 ap 模式下获取已使能的 ap 热点的配置信息。
备注：	该接口主要是用于获取当前启动的 ap 热点的配置信息。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

monitor 模式相关 API wifi_monitor_enable

函数原型	wmg_status_t wifi_monitor_enable(uint8_t channel)
参数说明	channel：使能 monitor 模式时需要监听的信道。
返回说明	WMG_STATUS_SUCCESS(0)：监听 monitor 模式某信道成功； 非 WMG_STATUS_SUCCESS(非 0)：监听 monitor 模式某信道失败。
功能描述	在 monitor 模式下使能监听某信道功能。
备注：	无。

wifi_monitor_set_channel

函数原型	wmg_status_t wifi_monitor_set_channel(uint8_t channel)
参数说明	channel：要修改监听的信道。
返回说明	WMG_STATUS_SUCCESS(0)：修改监听的信道成功； 非 WMG_STATUS_SUCCESS(非 0)：修改监听的信道失败。
功能描述	在 monitor 模式下动态切换要监听信道。
备注：	无。

wifi_monitor_disable

函数原型	wmg_status_t wifi_monitor_disable(void)
参数说明	无。
返回说明	WMG_STATUS_SUCCESS(0)：关闭 monitor 模式监听功能成功； 非 WMG_STATUS_SUCCESS(非 0)：关闭 monitor 模式监听功能失败。
功能描述	在 monitor 模式下关闭 monitor 的监听功能。
备注：	无。

p2p 模式相关 API wifi_p2p_enable

函数原型	wmg_status_t wifi_p2p_enable(wifi_p2p_config_t *p2p_config)
参数说明	p2p_config(使能 p2p 时的配置参数，结构体成员具体说明请查看 wifi_p2p_config_t 结构体说明)。
返回说明	WMG_STATUS_SUCCESS(0)：使用 p2p 模式功能成功； 非 WMG_STATUS_SUCCESS(非 0)：使用 p2p 模式功能失败。
功能描述	在 p2p 模式下启动 p2p 功能。
备注：	无。

wifi_p2p_disable

版权所有 © 珠海全志科技股份有限公司。保留一切权利

函数原型	wmg_status_t wifi_p2p_disable(void);
参数说明	无。
返回说明	WMG_STATUS_SUCCESS(0)：关闭 p2p 模式功能成功； 非 WMG_STATUS_SUCCESS(非 0)：关闭 p2p 模式功能失败。
功能描述	在 p2p 模式下关闭 p2p 功能。
备注：	无。

wifi_p2p_find

函数原型	wmg_status_t wifi_p2p_find(wifi_p2p_peers_t *p2p_peers, uint8_t find_second);
参数说明	p2p_peers(扫描到的设备会通过改参数返回) find_second(要扫描多久)
返回说明	WMG_STATUS_SUCCESS(0)：p2p 模式扫描成功； 非 WMG_STATUS_SUCCESS(非 0)：p2p 扫描失败。
功能描述	p2p 模式下获取扫描结果。
备注：	在执行扫描的过程中也需要对端发起扫描，否则扫描不到对端设备。

wifi_p2p_connect

函数原型	wmg_status_t wifi_p2p_connect(uint8_t *p2p_mac_addr);
参数说明	p2p_mac_addr：需要连接的 p2p 设备的 mac 地址。
返回说明	WMG_STATUS_SUCCESS(0)：连接成功； 非 WMG_STATUS_SUCCESS(非 0)：失败。
功能描述	在 p2p 模式下以 pbc 的方式连接另外一个 p2p 设备。
备注：	连接的过程中需要对方同意连接，否则会连接失败。

wifi_p2p_disconnect

函数原型	wmg_status_t wifi_p2p_disconnect(uint8_t *p2p_mac_addr);
参数说明	p2p_mac_addr：go 模式下需要断开的 p2p 设备的 mac 地址。 NULL：go 模式下断开所有 p2p 设备的连接，gc 模式下断开与 go 的连接
返回说明	WMG_STATUS_SUCCESS(0)：断开连接成功； 非 WMG_STATUS_SUCCESS(非 0)：断开连接失败。
功能描述	在 p2p 模式断开某个已连接的 p2p 设备。
备注：	go 模式下断开某个 p2p 设备功能暂未支持，仅预留接口，该函数请输入 NULL 参数即可

版权所有 © 珠海全志科技股份有限公司。保留一切权利

wifi_p2p_get_info

函数原型	wmg_status_t wifi_p2p_get_info(wifi_p2p_info_t *p2p_info)
参数说明	p2p_info(p2p 设备信息参数，结构体成员具体说明请查看 wifi_p2p_info_t 结构体说明)。
返回说明	WMG_STATUS_SUCCESS(0)：获取 p2p 设备信息成功； 非 WMG_STATUS_SUCCESS(非 0)：获取 p2p 设备信息失败。
功能描述	在 p2p 模式下获取 p2p 设备的某些信息。
备注：	无。

其他 API wifi_register_msg_cb

函数原型	wmg_status_t wifi_register_msg_cb(wifi_msg_cb_t msg_cb)
参数说明	msg_cb：要回调的回调函数。
返回说明	WMG_STATUS_SUCCESS(0)：设置回调函数成功； 非 WMG_STATUS_SUCCESS(非 0)：设置回调函数失败。
功能描述	在任意模式下注册回调函数。
备注：	该接口可以在任意模式下调用，但必须在调用 wifi_on 函数后才能调用，否则会注册回调函数失败。 该回调函数为实时回调函数，禁止在该函数中实现阻塞或者耗费长时间的操作。

wifi_set_scan_param

函数原型	wmg_status_t wifi_set_scan_param(wifi_scan_param_t * scan_param)
参数说明	无效。
返回说明	WMG_STATUS_SUCCESS(0)：成功； 非 WMG_STATUS_SUCCESS(非 0)：失败。
功能描述	无效。
备注：	预留接口，功能没有实现。

wifi_get_scan_results

函数原型	wmg_status_t wifi_get_scan_results(wifi_scan_result_t result, char ssid，uint32_t * bss_num, uint32_t arr_size)
参数说明	wifi_scan_result_t(扫描结果保存结构体，结构体具体说明请参考 libwifimg-v2.0 库关键结构体说明)。 ssid: 需要扫描的隐藏 ssid(不需要默认填 NULL 即可)。 bss_num: 扫描结果扫描到多少条。 arr_size：扫描的 buff 的大小。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

函数原型	wmg_status_t wifi_get_scan_results(wifi_scan_result_t result, char ssid, uint32_t * bss_num, uint32_t arr_size)
返回说明	WMG_STATUS_SUCCESS(0)：扫描成功； 非 WMG_STATUS_SUCCESS(非 0)：失败。
功能描述	sta 模式下获取扫描结果。
备注：	调用者需要根据具体情况申请内存。

wifi_set_mac

函数原型	wifi_set_mac(const char * ifname, uint8_t * mac_addr)
参数说明	ifname：要设置 mac 地址的网卡名。mac_addr：要设置 mac 地址。
返回说明	WMG_STATUS_SUCCESS(0)：设置 mac 地址成功； 非 WMG_STATUS_SUCCESS(非 0)：失败。
功能描述	任意模式下设置 mac 地址。
备注：	该接口可以在任意模式下调用，但必须在调用 wifi_on 函数后才能调用，否则会设置失败，该接口设置的 mac 地址仅临时性

wifi_get_mac

函数原型	wifi_get_mac(const char * ifname, uint8_t * mac_addr)
参数说明	ifname：要设置 mac 地址的网卡名。mac_addr: 获取到的 mac 地址。
返回说明	WMG_STATUS_SUCCESS(0)：获取 mac 地址成功； 非 WMG_STATUS_SUCCESS(非 0)：获取 mac 地址失败。
功能描述	任意模式下获取 mac 地址。
备注：	该接口可以在任意模式下调用，但必须在调用 wifi_on 函数后才能调用，否则会获取失败。

wifi_linkd_protocol

函数原型	wmg_status_t wifi_linkd_protocol(wifi_linkd_mode_t mode,void params, int second, wifi_linkd_result_tlinkd_result);
参数说明	mode：要使用哪种配网模式。 params: 配网模式的参数 (不同模式支持的特殊参数定义可能不一样)。 second：配网超时时间 (秒)。 linkd_result: 配网获取到的结果 (ssid 和 psk)。
返回说明	WMG_STATUS_SUCCESS(0)：配网成功，获取到配网信息； WMG_STATUS_SUCCESS(0)：配网失败，获取配网信息失败。
功能描述	执行配网功能并且获取配网结果。

函数原型	wmg_status_t wifi_linkd_protocol(wifi_linkd_mode_t mode,void params, int second, wifi_linkd_result_tlinkd_result);
备注：	调用该接口后以某种配网模式去获取配网结果 (ssid 和 psk)，配网模式不同系统支持情况不同。

wifi_get_wmg_state

函数原型	wmg_status_t wifi_get_wmg_state(wifi_wmg_state_t *wmg_state);
参数说明	wmg_state：获取到的 wifimanager 的状态信息。
返回说明	WMG_STATUS_SUCCESS(0)：获取 wifimanager 状态成功； WMG_STATUS_SUCCESS(0)：获取 wifimanager 状态失败。
功能描述	任意模式下获取 wifimanager 的状态信息。
备注：	该接口可以在任意模式下调用，用于获取 wifimanager 的状态信息，只有返回值为 0 时获取的状态信息才有效。

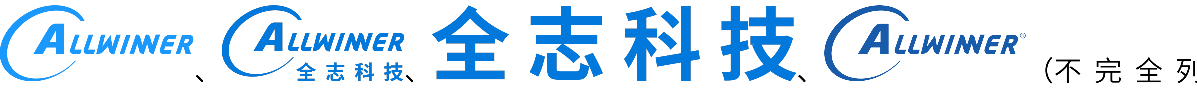
著作权声明

版权所有 ©2024 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明



（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。