



# SIM7600 系列\_Open Linux\_ 开发指南

LTE 模组

SIMCom Wireless Solutions Limited  
Building B, SIM Technology Building, No.633, Jinzhong Road  
Changning District, Shanghai P.R. China  
Tel: 86-21-31575100  
[support@simcom.com](mailto:support@simcom.com)  
[www.simcom.com](http://www.simcom.com)

名称:	SIM7600系列_Open Linux_开发指南
版本:	V2.00
类别:	应用文档
状态:	已发布

## 版权声明

本手册包含芯讯通无线科技（上海）有限公司（简称：芯讯通）的技术信息。除非经芯讯通书面许可，任何单位和个人不得擅自摘抄、复制本手册内容的部分或全部，并不得以任何形式传播，违反者将被追究法律责任。对技术信息涉及的专利、实用新型或者外观设计等知识产权，芯讯通保留一切权利。芯讯通有权在不通知的情况下随时更新本手册的具体内容。

本手册版权属于芯讯通，任何人未经我公司书面同意进行复制、引用或者修改本手册都将承担法律责任。

### 芯讯通无线科技(上海)有限公司

上海市长宁区金钟路 633 号晨讯科技大楼 B 座 6 楼

电话: 86-21-31575100

邮箱: [simcom@simcom.com](mailto:simcom@simcom.com)

官网: [www.simcom.com](http://www.simcom.com)

了解更多资料，请点击以下链接：

<http://cn.simcom.com/download/list-230-cn.html>

技术支持，请点击以下链接：

<http://cn.simcom.com/ask/index-cn.html> 或发送邮件至 [support@simcom.com](mailto:support@simcom.com)

版权所有 © 芯讯通无线科技(上海)有限公司 2020，保留一切权利。

## Version History

Version	Date	Owner	What is new
V2.00	2020.8.6		更新文档格式

## This document applies to the following products

本文档只适应于 SIM7600 系列型号

SIMCom  
Confidential

# 目录

Version History.....	3
This document applies to the following products.....	4
目录.....	5
缩略语.....	10
<b>1. SIM7600 开发平台概述.....</b>	<b>11</b>
1.1 系统概述.....	11
1.2 二次开发选型及功能.....	12
1.3 二次开发相关 PIN 脚定义.....	13
<b>2. 二次开发环境搭建.....</b>	<b>18</b>
2.1 安装 Windows 嵌入式编译器.....	18
2.1.1 2.1.1 安装 ARM GNU/LINUX.....	18
2.1.2 2.1.2 安装 Cygwin.....	20
2.2 配置 LINUX 编译环境.....	21
2.2.1 编译方法.....	21
2.2.2 ota 升级包制作命令.....	23
2.3 安装 Windows 驱动.....	24
2.4 调试下载工具.....	25
2.5 应用程序编译和运行.....	26
2.5.1 demo 应用.....	27
2.5.2 helloworld 应用.....	28
2.6 系统分区和文件保护.....	28
2.7 产线生产模式.....	29
2.7.1 单独下载 APP.....	29
2.7.2 下载编译出的 Images.....	30
<b>3. 编程指导.....</b>	<b>31</b>
3.1 系统基本 API.....	31
3.2 嵌入式 AT 收发.....	31
3.2.1 发送 AT 命令接口.....	31
3.3 UART.....	31
3.4 GPIO.....	33
3.5 ADC.....	35
3.6 I2C.....	35
3.6.1 写 I2C 接口.....	35
3.6.2 读 I2C 接口.....	36
3.7 SD Card/EMMC flash.....	36
3.7.1 分区.....	36
3.7.2 格式化.....	36

3.7.3	挂载.....	36
3.7.4	CFDISK 命令.....	37
3.8	UIM.....	38
3.8.1	查询 SIM 卡状态.....	38
3.8.2	查询 SIM 卡 ICCID.....	38
3.8.3	查询 SIM 卡 IMSI.....	39
3.9	SMS.....	39
3.9.1	短信初始化.....	39
3.9.2	接收短信格式设置.....	39
3.9.3	短信发送.....	39
3.9.4	回调函数处理短信接收.....	40
3.10	Voice Call.....	41
3.10.1	电话初始化.....	41
3.10.2	拨打电话.....	41
3.10.3	处理当前通话.....	41
3.10.4	得到指定通话的状态.....	42
3.10.5	得到所有的通话状态.....	42
3.10.6	回调函数.....	43
3.11	NAS.....	43
3.11.1	查询注册网络状态.....	43
3.11.2	查询信号.....	44
3.12	WDS.....	45
3.12.1	查询 APN.....	45
3.12.2	设置 APN.....	45
3.12.3	Data Call.....	46
3.12.4	初始化网络.....	46
3.12.5	建立数据链接.....	46
3.12.6	获取数据链接参数.....	46
3.12.7	释放数据链接.....	47
3.12.8	释放网络资源.....	47
3.12.9	Data Call 流程说.....	47
3.13	GNSS.....	48
3.13.1	初始化 GNSS.....	48
3.13.2	使能 XTRA.....	48
3.13.3	禁止 XTRA.....	49
3.13.4	GNSS 冷启动.....	49
3.13.5	GNSS 热启动.....	49
3.13.6	停止定位.....	49
3.13.7	回调函数输出简要位置信息.....	50
3.13.8	回调函数输出 NMEA 语句.....	50
3.14	WIFI.....	51
3.14.1	获取当前 WIFI 模式设置(for W58).....	51
3.14.2	设置 WIFI 模式(for W58).....	51
3.14.3	WIFI 开关.....	52
3.14.4	获取 WIFI 状态.....	52
3.14.5	设置 WIFI 热点名称.....	53

3.14.6	获取 WIFI 热点名称.....	53
3.14.7	设置 AP auth 类型, 加密模式, 密码.....	53
3.14.8	获取 AP auth 类型, 加密模式, 密码.....	54
3.14.9	设置 WIFI 广播开关.....	55
3.14.10	获取 WIFI 广播设定.....	55
3.14.11	获取 DHCP 设定.....	55
3.14.12	获取连接的客户端数.....	56
3.14.13	获取 STA 模式下的 IP 地址.....	56
3.14.14	获取 WIFI 物理地址.....	56
3.14.15	设置 STA 连接外部热点后获取到的 IP.....	57
3.14.16	设置 STA 连接外部 AP 的 SSID 和密码.....	57
3.14.17	获取 STA 设置的 SSID 和 密码.....	57
3.14.18	wifi sta 扫描可用热点.....	57
3.14.19	wifi sta 扫描可用热点 (显示信号强度).....	58
3.14.20	设置 cdma 模式拨号的用户名和密码.....	58
3.14.21	获取 cdma 模式拨号的用户名和密码.....	59
3.14.22	获取网络状态.....	59
3.14.23	恢复 wifi 设定.....	59
3.14.24	设置 AP 热点名称和 auth 类型, 加密模式, 密码.....	59
3.14.25	开启/关闭 STA 模式(for W58L).....	60
3.14.26	获取 STA 模式开启状态(for W58L).....	61
3.14.27	获取操作的结果码.....	61
3.14.28	STA-AP 功能使用说明.....	61
3.14.29	AP-AP 功能使用说明.....	62
3.15	SPI.....	63
3.16	3.17 USB OTG.....	63
3.17	蓝牙.....	63
3.17.1	蓝牙接口初始化.....	65
3.17.2	蓝牙开关.....	66
3.17.3	获取已配对列表.....	66
3.17.4	搜索蓝牙.....	66
3.17.5	蓝牙配对.....	66
3.17.6	配对确认请求.....	67
3.17.7	接受配对.....	67
3.17.8	配对结果.....	67
3.17.9	开启 SPP Server.....	68
3.17.10	查询对端蓝牙是否开启 SPP Server.....	68
3.17.11	发起 SPP 连接.....	68
3.17.12	断开 SPP 连接.....	68
3.17.13	非主动连接 SPP.....	69
3.17.14	非主动断开 SPP 连接.....	69
3.17.15	查询本地 SPP Server 状态.....	69
3.17.16	查询 SPP 连接状态.....	69
3.17.17	SPP 发送数据.....	69
3.17.18	SPP 接收数据.....	70
3.17.19	查询本地蓝牙名称和 MAC 地址.....	70

3.17.20	设置本地蓝牙名称.....	70
3.17.21	设置 PIN 码.....	71
3.17.22	读取 PIN 码.....	71
3.17.23	设置 BR/EDR inquiry scan 和 page scan.....	71
3.17.24	查询 BR/EDR inquiry scan 和 page scan 设定.....	71
3.17.25	GATT 注册.....	72
3.17.26	创建 database.....	72
3.17.27	创建 16 位 UUID 服务.....	72
3.17.28	创建 128 位 UUID 服务.....	72
3.17.29	创建 16 位特征.....	73
3.17.30	创建 128 位特征.....	73
3.17.31	创建描述符.....	73
3.17.32	将创建的服务添加到数据库.....	73
3.17.33	发送 notification.....	74
3.17.34	发送 indication.....	74
3.17.35	返回主机从本地读数据请求.....	74
3.17.36	返回主机从本地写数据请求.....	75
3.17.37	回调函数.....	75
3.18	ETH.....	76
3.18.1	网卡模式设置.....	76
3.18.2	网卡型号选择.....	77
3.18.3	驱动加载.....	77
3.18.4	驱动卸载.....	77
3.18.5	从 NV 中读取预设 MAC 地址.....	77
3.18.6	设置 MAC 地址.....	78
3.18.7	设置 IP.....	78
3.19	网络设置.....	78
3.20	网络访问方式.....	78
3.20.1	默认路由优先级预置.....	80
3.21	ALSA.....	80
3.21.1	设置内部扬声器的音量输出.....	80
3.21.2	获取内部扬声器的音量.....	81
3.21.3	设置 mic 增益.....	81
3.21.4	获取 mic 增益.....	81
3.21.5	切换语音通道.....	81
3.21.6	查询当前语音通道.....	82
3.21.7	启动录音.....	82
3.21.8	停止录音.....	82
3.21.9	查询录音状态.....	82
3.22	设备控制.....	83
3.22.1	进入 recovery 模式.....	83
3.22.2	adb 设置.....	83
3.23	DMS.....	83
3.23.1	初始化.....	83
3.23.2	获取 imei.....	84
3.23.3	获取 meid.....	84



3.23.4	获取固件版本识别码.....	84
3.23.5	设置 UE 工作模式.....	85
3.23.6	释放.....	85
4.客户版本维护.....		86

SIMCom  
Confidential

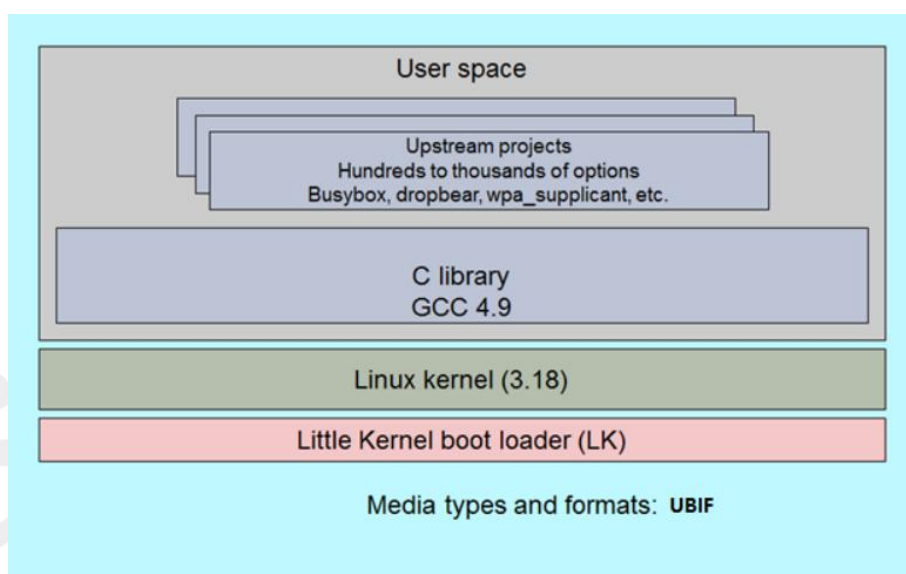
## 缩略语

AT	ATtention; the two-character abbreviation is used to start a command line to be sent from TE/DTE to TA/DCE
DCE	Data Communication Equipment; Data Circuit terminating Equipment
DCS	Digital Cellular Network
DTE	Data Terminal Equipment
DTMF	Dual Tone Multi-Frequency
EDGE	Enhanced Data GSM Environment
EGPRS	Enhanced General Packet Radio Service
GPIO	General-Purpose Input/Output
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HSDPA	High Speed Downlink Packet Access
HSUPA	High Speed Uplink Packet Access
I2C	Inter-Integrated Circuit
IMEI	International Mobile station Equipment Identity
IMSI	International Mobile Subscriber Identity
ME	Mobile Equipment
MO	Mobile-Originated
MS	Mobile Station
MT	Mobile-Terminated; Mobile Termination
PCS	Personal Communication System
PIN	Personal Identification Number
PUK	Personal Unlock Key
SIM	Subscriber Identity Module
SMS	Short Message Service
SMS-SC	Short Message Service-ServiceCenter
TA	Terminal Adaptor; e.g. a data card (equal to DCE)
TE	Terminal Equipment; e.g. a computer (equal to DTE)
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
USIM	Universal Subscriber Identity Module
WCDMA	Wideband Code Division Multiple Access
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
RTC	Real Time Clock
NAS	Network Access Service
WDS	Wireless Data Service
QMI	Qualcomm Messaging Interface

# 1.SIM7600 开发平台概述

## 1.1 系统概述

SIM7600 模块的开发平台是 Linux 系统，其框架图如下：



SIM7600 基于 ARM Cortex-A71.3GHz 中央处理器,运行 Linux 操作系统，内核版本是 3.18.20。文件系统采用 UBIFS 文件系统，Linux 管理的 ubi 文件系统包含三个逻辑分区：

-ubi0:rootfs :

rootfs 逻辑分区是只读的，存放 Linux 的 Code。

-ubi0:usrfs:

Usrfs 存放 Linux 的文件系统，一般二次开发用户的应用程序放在此分区。

-ubi0:cache fs

cache fs 此空间一般是用于 FOTA 升级。如果升级的时候 cache 空间不够，升级程序会通过删除未打开的文件来释放部分空间用于 FOTA 升级。所以安全起见，用户的数据最好不要放在 cache 分区。

SIM7600CE-T 的 3 个分区具体信息如下表所示：

Filesystem	Size	Used	Available	Use%	Mounted on
ubi0:rootfs	50.2M	36.1M	14.1M	72%	/
ubi0:usrfs	10.5M	364K	10.1M	3%	/data
Ubi0:cache fs	40.8M	24K	38.6M	0%	/cache

## 1.2 二次开发选型及功能

根据市场的需要，我们分别推出了几个支持二次开发的 4G 模块型号。客户可以根据自己的产品定义，来选择性价比最优的方案。具体支持的功能表如下：

	SIM7600CE-T	SIM7600CE-AT
Platform	MDM9x07	MDM9x07
Memory(bit)	2+2	4+2
Protocols	TCP/IP/IPV4/IPV6/Multi PDP/FTPS/HTTPS/DNS/COAP/MQTT	TCP/IP/IPV4/IPV6/Multi PDP/FTPS/HTTPS/DNS/COAP/MQTT
CALL	.	.
ECALL	.	.
SMS	.	.
TLS1.2	.	.
Audio		
Record/Play	.	.
TTS	.	.
DTMF	.	.
LBS	.	.
FOTA	.	.
Security		
Routing policy	4G,WIFI and Ethernet port	4G,WIFI and Ethernet port
NDIS/RNDIS	.	.
Bluetooth	BLE4.2（外接W58模块）	BLE4.2（外接W58模块）
WIFI	2.4G（外接W58模块）	2.4G（外接W58模块）
GNSS	GPS/GLONASS/BEIDOU	GPS/GLONASS/BEIDOU
SGMII	HSCI转LAN9730(100M)	车载以太网（1000M）
UART	2* High UARTs	3* High UARTs
USB2.0	.	.
OTG	.	.
HSIC	.	.
Audio PCM	1*PCM	1* PCM
Audio Analog	1input+1output (Common access with PCM)	1input+1output (Common access with PCM)
GPIO	At least 5*GPIO	At least 5*GPIO
SDIO	SDIO3.0（200MHz Max）	SDIO3.0（200MHz Max）
SD Card	SD3.0（128G Max）	SD3.0（128G Max）
SPI	Optional	1*SPI
I2C	1*I2C	1*I2C

### 1.3 二次开发相关 PIN 脚定义

SIM7600CE-T

PIN No.	PINNAME	SYS GPIO No.	Default Function	Function 1	Function2	Pull	Wakeup Interrupt
6	SPI_CLK	--	UART1_RTS	--	--	B-P D	
7	SPI_MISO	--	UART1_RX	--	--	B-P D	
8	SPI_MOSI	--	UART1_TX	--	--	B-P D	
9	SPI_CS	--	UART1_CTS	--	--	B-P D	
21	SD_CMD	--	SD Card	--	--	B-P D	
22	SD_DATA0	--	SD Card	--	--	B-P D	
23	SD_DATA1	--	SD Card	--	--	B-P D	
24	SD_DATA2	--	SD Card	--	--	B-P D	
25	SD_DATA3	--	SD Card	--	--	B-P D	
26	SD_CLK	--	SD Card	--	--	B-N P	
27	SDIO_DATA1	--	WLAN	--	--	B-P D	
28	SDIO_DATA2	--	WLAN	--	--	B-P D	
29	SDIO_CMD	--	WLAN	--	--	B-P D	
30	SDIO_DATA0	--	WLAN	--	--	B-P D	
31	SDIO_DATA3	--	WLAN	--	--	B-P D	
32	SDIO_CLK	--	WLAN	--	--	B-N P	
33	GPIO3	GPIO_1020	MIFI_POWER_EN	GPIO	MIFI_POWER_EN	B-P U	
34	GPIO6	GPIO_1023	MIFI_SLEEP_CLK	GPIO	MIFI_SLEEP_CLK	B-P D	

46	ADC2		ADC			--	
47	ADC1		ADC			B-P U	
48	SD_DET	GPIO _26	GPIO	GPIO	SD_DET	B-P D	
49	STATUS	GPIO _52	Status	GPIO	Status	B-P D	
50	GPIO43	GPIO _36	MIFI_COEX	GPIO	MIFI_COEX	B-P D	
51	NETLIGHT	GPIO _18	NETLIGHT	GPIO	NETLIGHT	B-P D	
52	GPIO41	GPIO _79	BT_PCM	GPIO	BT_PCM	B-P D	
54	FLIGHTMODE	GPIO _76	BT_PCM	GPIO	BT_PCM	B-P D	
66	RTS	--	UART2_RTS	--	--	B-P D	
67	CTS	--	UART2_CTS	--	--	B-P D	
68	RXD	--	UART2_RX	--	--	B-P D	
69	RI	GPIO _50	GPIO(RI)	--	--	B-P D	
70	DCD	GPIO _51	GPIO	GPIO	GPIO	B-P D	
71	TXD	--	UART2_TX	--	--	B-P D	
72	DTR	GPIO _74	GPIO(DTR)	--	--	B-P D	
73	PCM_OUT	--	PCM	--	--	B-P D	
74	PCM_IN	--	PCM	--	--	B-P D	
75	PCM_SYNC	--	PCM	--	--	B-P D	
76	PCM_CLK	--	PCM	--	--	B-P U	
86	COEX3	GPIO _78	BT_PCM	GPIO	BT_PCM	B-P D	
87	GPIO77	GPIO _77	BT_PCM	GPIO	BT_PCM	B-P D	

SIM7600CE-AT

Pin No.	PINNAME	SYS GPIO No.	Default Function	Function 1	Function2	Pull	Wakeup Interrupt
6	SPI_CLK	--	BT	UART2_RTS	BT	B-PD	
7	SPI_MISO	--	BT	UART2_RX	BT	B-PD	
8	SPI_MOSI	--	BT	UART2_TX	BT	B-PD	
9	SPI_CS	--	BT	UART2_CTS	BT	B-PD	
21	SD_CMD	--	SD Card	--	--	B-PD	
22	SD_DATA0	--	SD Card	--	--	B-PD	
23	SD_DATA1	--	SD Card	--	--	B-PD	
24	SD_DATA2	--	SD Card	--	--	B-PD	
25	SD_DATA3	--	SD Card	--	--	B-PD	
26	SD_CLK	--	SD Card	--	--	B-NP	
27	SDIO_DATA1	--	WLAN	--	--	B-PD	
28	SDIO_DATA2	--	WLAN	--	--	B-PD	
29	SDIO_CMD	--	WLAN	--	--	B-PD	
30	SDIO_DATA0	--	WLAN	--	--	B-PD	
31	SDIO_DATA3	--	WLAN	--	--	B-PD	
32	SDIO_CLK	--	WLAN	--	--	B-NP	
33	GPIO3	GPIO_1020	MIFI_POWER_EN	GPIO	MIFI_POWER_EN	B-PU	
34	GPIO6	GPIO_1023	MIFI_SLEEP_CLK	GPIO	MIFI_SLEEP_CLK	B-PD	
42	DGB_TXD	GPIO_8	Debug	UART3_TX	Debug	B-PD	
46	ADC2		ADC			--	
47	ADC1		ADC			B-PU	

48	SD_DET	GPIO_26	GPIO	GPIO	SD_DET	B-P D	*
49	STATUS	GPIO_52	Status	GPIO	Status	B-P D	*
50	GPIO43	GPIO_36	MIFI_COEX_RX	GPIO	MIFI_COEX	B-P D	
51	NETLIGHT	--	I2C_SDA	--	--	B-P D	
52	GPIO41	--	PCM_SYNC	--	--	B-P D	*
54	FLIGHTMODE	--	PCM_DIN	--	--	B-P D	
55	SCL	--	Debug	UART3_RTS	Debug	B-P D	*
56	SDA	--	Debug	UART3_CTS	Debug	B-P U	
66	RTS	--	UART1_RTS	--	--	B-P D	
67	CTS	--	UART1_CTS	--	--	B-P D	
68	RXD	--	UART1_RX	--	--	B-P D	
69	RI	GPIO_50	GPIO(RI)	--	--	B-P D	
70	DCD	GPIO_51	GPIO	--	--	B-P D	
71	TXD	--	UART1_TX	--	--	B-P D	
72	DTR	GPIO_74	GPIO(DTR)	--	--	B-P D	*
73	PCM_OUT	--	SPI_CS	UART4_CTS	SPI_CS	B-P D	
74	PCM_IN	--	SPI_MISO	UART4_RX	SPI_MISO	B-P D	
75	PCM_SYNC	--	SPI_CS	UART4_TX	SPI_CS	B-P D	
76	PCM_CLK	--	SPI_CLK	UART4_RTS	SPI_CLK	B-P U	
85	BOOT_CFG0	GPIO_37	MIFI_COEX_TX	GPIO	MIFI_COEX	B-P D	
86	COEX3	--	PCM_CLK	--	--	B-P D	
87	GPIO77	--	PCM_DOUT	--	--	B-P D	
106	DGB_RXD	--	Debug	UART3_RX	Debug	B-P	



						D	
108	EPHY_RST_N	--	SGMII_RST	--	--	B-P D	
109	EPHY_INT_N	--	SGMII_INT	--	--	B-P D	
112	SGMII_RX_P	--	SGMII_RX_P	--	--	B-P D	
113	SGMII_RX_M	--	SGMII_RX_M	--	--	B-P D	
114	SGMII_TX_M	--	SGMII_TX_M	--	--	B-P D	
115	SGMII_TX_P	--	SGMII_TX_P	--	--	B-P D	
116	SGMII_DATA	--	SGMII_DATA	--	--	B-P D	
117	SGMII_CLK	--	SGMII_CLK	--	--	B-P D	
119	GPIO_19	--	I2C_SCL	--	--	B-P D	

## 2. 二次开发环境搭建

二次开发有 Windows 和 Linux 两种环境，客户可以根据需要选择自己熟悉的环境来开发，如果只开发一个应用程序，用 Windows 会比较快捷，如果涉及到 Kernel 的改动等，必须使用 Linux 环境。

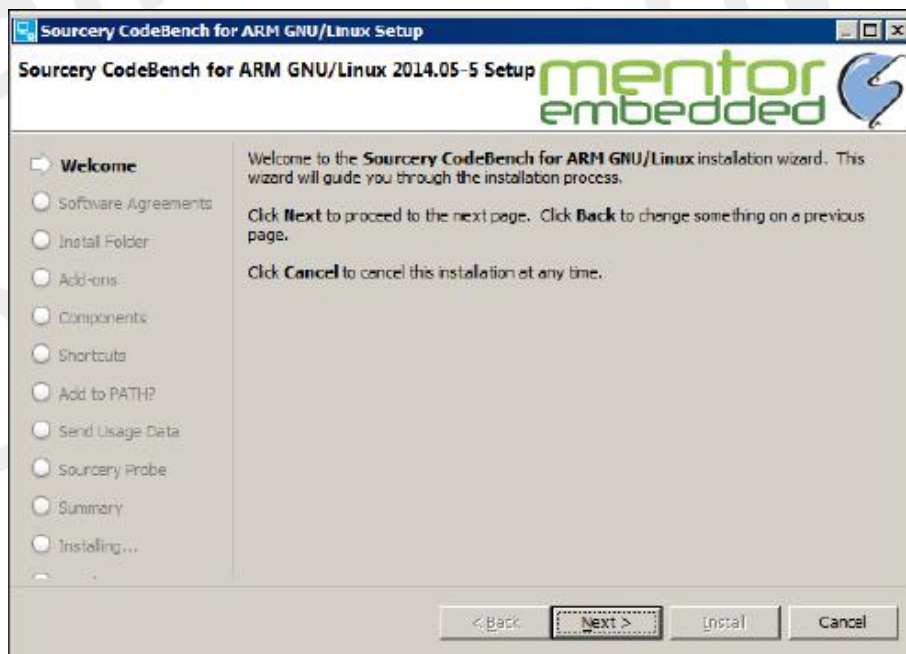
### 2.1 安装 Windows 嵌入式编译器

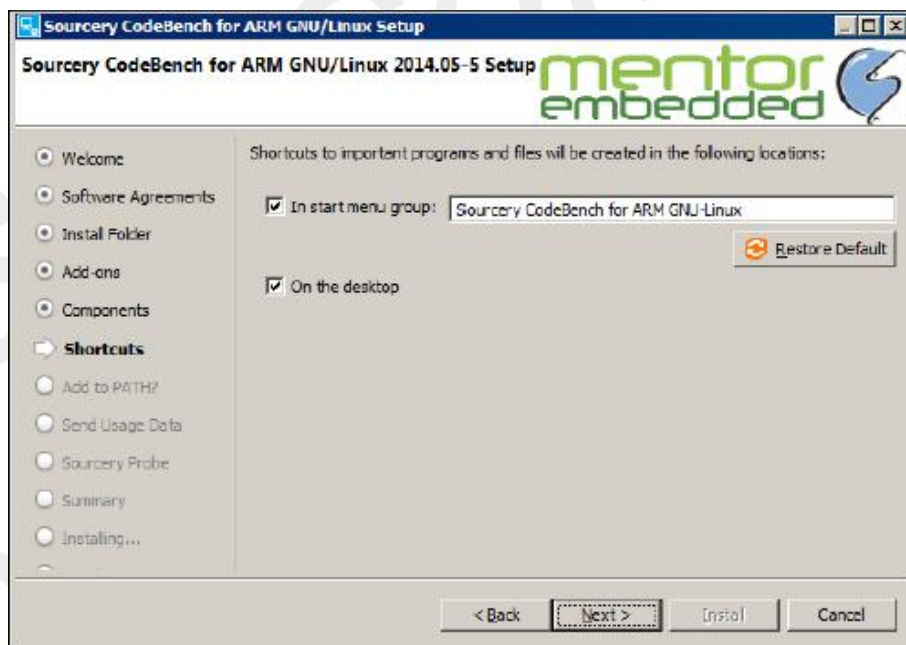
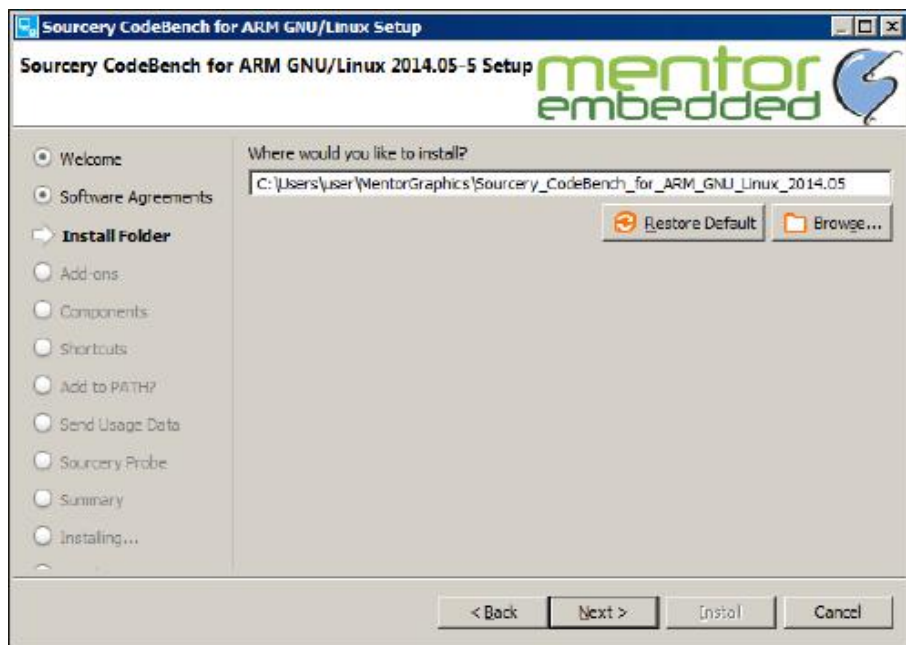
#### 2.1.1 2.1.1 安装 ARM GNU/LINUX

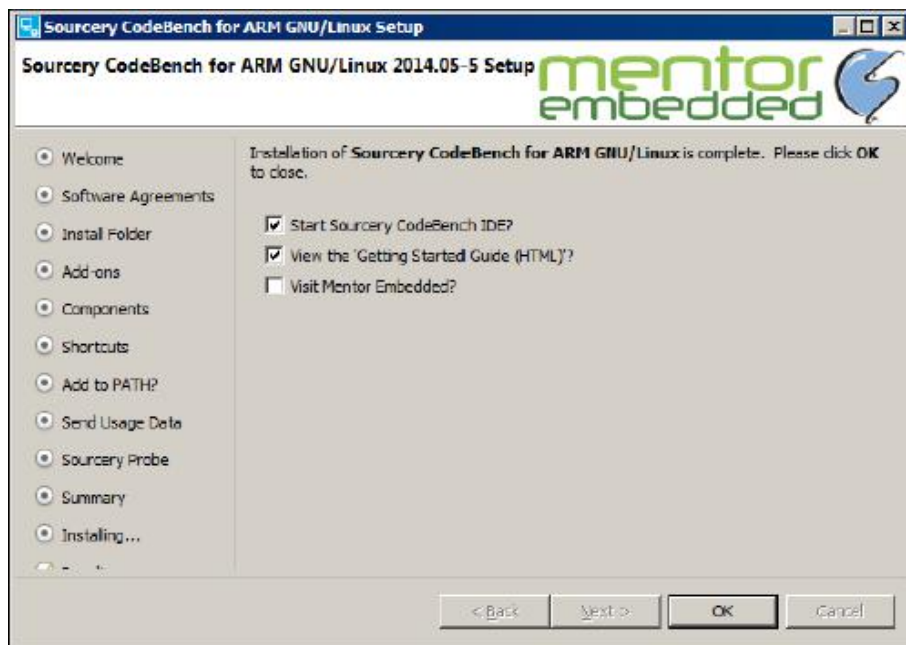
SIM7600 的开发其实是嵌入式 Linux 的开发，可以选择在 Windows 操作系统下搭建一个兼容 SIM7600 平台的 ARM-Linux 编译器用于编译应用程序。

在 Windows 系统下编译 SIM7600 的应用程序需要安装的工具 SourceryCodeBenchLiteARM GNU/Linux 编译器下载地址：[https://sourcery.mentor.com/public/gnu\\_toolchain/](https://sourcery.mentor.com/public/gnu_toolchain/)

安装流程：







### 2.1.2 2.1.2 安装 Cygwin

为了简化应用程序的编译和方便管理应用程序的代码工程，建议采用 `make` 的方式编译程序。在 Windows 环境下可以采用 Cygwin 的 `make` 程序。客户可以从 cygwin 的官方网站下载免费的 Cygwin。

安装成功后把 Cygwin/bin/的路径加到环境变量之中，如下图所示：



## 2.2 配置 LINUX 编译环境

推荐使用 Ubuntu12.04 64 位来做编译系统

### 2.2.1 编译方法

使用 `sudo tar` 命令解压 `sdk` 压缩包，如解压到当前目录命令如下

```
sudo tar xzf sim_open_sdk.tar.gz
```

之后在 `sim open sdk` 目录下初始化环境变量，命令如下

sourcesim\_crosscompile/sim-crosscompile-env-init

### 2.2.1.1 全部编译

在 `sim_open_sdk` 目录下执行 `make` 命令即可，编译完成后生成的所有镜像在 `sim_open_sdk/output` 目录下，如下

```
appsboot.mbn boot.img system.img
```

### 2.2.1.2 编译 bootloader

在 `sim_open_sdk` 目录下执行 `make about` 即可  
镜像在 `sim_open_sdk/output` 目录下，如下

```
appsboot.mbn
```

### 2.2.1.3 编译 kernel

- 1) 在 `sim_open_sdk` 目录下执行 `make kernel_menuconfig` 配置内核，之后会弹出图像化配置界面，如需要添加新的配置选项，在图像化配置界面选择相应的配置后，保存退出即可；如果不再添加新的配置，直接退出即可。配置完成后会在 `sim_kernel/build/` 目录下生成 `.config` 文件
- 2) 执行 `make kernel` 编译内核，生成的内核镜像在 `sim_open_sdk/output` 目录下

```
boot.img
```

### 2.2.1.4 生成 rootfs 文件系统镜像

在 `sim_open_sdk` 目录下执行 `make rootfs` 即可，生成的镜像在 `sim_open_sdk/output` 目录下

```
system.img
```

### 2.2.1.5 编译驱动模块

执行 `make kernel_module` 编译驱动模块，编译成功后自动安装在 `sim_rootfs` 相应的目录下，所以执行 `make rootfs` 重新生成 `rootfs` 文件系统镜像即可

### 2.2.1.6 编译 demo

执行 `make demo` 编译 demo 程序，生成的镜像在 `sim_open_sdk/output` 目录下

`demo_app`

### 2.2.1.7 清除生成的镜像

- 1) 全部清除执行 `make clean` 即可
- 2) 清除 bootloader 执行 `make about_clean`
- 3) 清除 kernel 执行 `make kernel_clean`
- 4) 清除 rootfs 执行 `make rootfs_clean`
- 5) 清除 demo 执行 `make demo_clean`

### 2.2.1.8 question

- 1) 如果在执行 `make kernel_menuconfig` 配置内核时出现如下错误  
error: ../scripts/kconfig/lxdialog/dialog.h: fatal error: curses.h: No such file or directory  
执行 `sudo apt-get install libncurses5-dev` 即可

## 2.2.2 ota 升级包制作命令

制作 ota 升级包需要同时提供当前版本包 `source` 和目标版本包 `target`，`source` 和 `target` 需要放入 `sim_open_sdk/sim_ota` 目录下

#### 1) target 生成

待应用程序准备完成，bootloader、kernel、system 全部编译完成，即 `output` 目录下已经生成 `appsboot.mbn`、`boot.img`、`system.img` 后，在 `sim_open_sdk` 目录下执行 `make ota` 命令，执行之后在 `sim_open_sdk/sim_ota` 目录下就可看到准备好的 `target`

#### 2) 客户准备好的 source 放入 `sim_open_sdk/sim_ota` 目录下，source 和 target 结构一样

#### 3) 执行 `sim_open_sdk` 目录下的 `ota.sh` 即可生成 ota 升级包，在 `sim_open_sdk/output` 目录下，如下

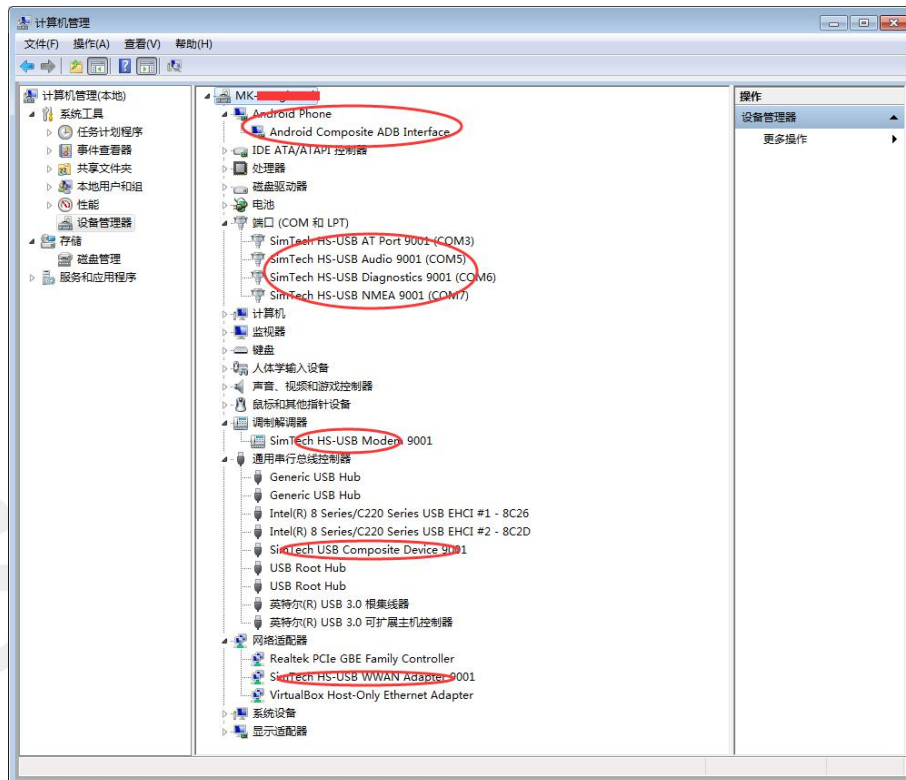
`update_ota2+1.zip` `update_ota.zip`

两个包差分方式不同，升级时使用 `update_ota2+1.zip`



## 2.3 安装 Windows 驱动

SIM7600 的 USB 接入电脑上，会出现一些 USB 虚拟设备需要安装驱动，请使用对应的 Windows 驱动安装包来安装，安装完后如下图：



每个设备对应功能如下图：

Interface number	windows name	Linux name	Interface description
0	SimTech HS-USB Diagnostics	USB serial	Diagnostic Interface
1	SimTech HS-USB NMEA	USB serial	GPS NMEA Interface
2	SimTech HS-USB AT Port	USB serial	AT port Interface
3	SimTech HS-USB Modem	USB serial	Modem port Interface
4	SimTech HS-USB Audio	USB serial	USB Audio Interface
5	SimTech HS-USB WWAN Adapter	USB Net	NDIS wwan interface
6	Android Composite ADB Interface	USB adb	Android add debug port

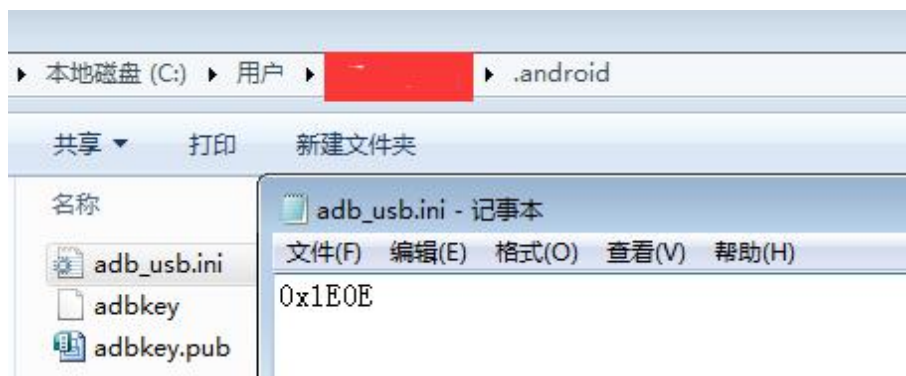
SIM7600 的 adb 设备默认是关闭的，需要通过 AT 命令设置打开，然后重启模块生效。

AT+CUSBADB=1 命令返回 OK 后，需要手动重启模块，从 Windows 的设备管理器里可以看到一个 Android Composite ADB Interface 设备，就说明 adb 设备可以用了。

注：

如果电脑第一次装 ADB 设备，需要在 C 盘，“用户”中的当前登录帐户下，在 .android 目录中增加 adb\_usb.ini 文件，且在文件中增加 SIM7600 的 VID 为 0x1E0E





通过 adb 工具可以在 Windows cmd 窗口里测试

```
c:\>adb kill-server

c:\>adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
0123456789ABCDEF    device
```

adb shell 可以进入到 SIM7600 的 Linux 系统的控制台。

用 alias ls='ls-color=null'来将系统的显示对齐，如下图：

```
E:\>adb shell
/ # alias ls='ls --color=null'
alias ls='ls --color=null'
/ # ls
ls
WEBSERVER  cache      firmware  media      sbin        system      var
bin         data       home      mnt         sdcard      target      www
boot        dev        lib        proc        share       tmp
build.prop  etc        linuxrc   run         sys         usr
/ #
```

## 2.4 调试下载工具

注意：

模块软件版本必须为二次开发版本才可按照下面的说明使用 **fastboot** 下载各分区，如果不是二次开发版本，则升级为二次开发版本之后再行下面的操作。通过 **at** 指令 **ati** 判断是否为二次开发版本，如果版本中包含关键字 **OL**，则为二次开发版本。

第一次下载版本，必须使用工具下载一个完整的二次开发版本，不能使用 **fastboot** 更新部分版本文件，否则可能导致某些功能不能正常使用。

无论应用程序还是编译出的 **image** 文件，都推荐用 **windows** 下载调试。

应用程序调试使用 **adb** 工具，**image** 下载使用 **fastboot** 工具。这两个工具可以从网上下载到或者使用我们提供的环境压缩包内的。

**adb** 工具可以把 **app** 程序 **push** 到模块内，**app** 在前台运行，可以将 **app** 所有 **log** 打印出来。和标准 **linux** 一样，用 **dmesg** 可以打印 **kernel** 的所有 **log**。

```
c:\>adb push helloworld /data/helloworld
629 KB/s (5800 bytes in 0.009s)

c:\>adb shell
/ # chmod a+x /data/helloworld
chmod a+x /data/helloworld
/ # ./data/helloworld
./data/helloworld
```

fastboot 工具可以下载 appsboot.mbn, modem.img, boot.img, system.img, recovery.img, recoveryfs.img. 用 AT 命令 at+bootldr 或者 adb 命令 adb reboot bootloader 进入 fastboot 模式，可以下载以上 image.

```
E:\>adb reboot bootloader

E:\>fastboot flash aboot E:\7600\LE11B01U01SIM76000L\appsboot.mbn
target reported max download size of 134217728 bytes
sending 'aboot' (384 KB)...
OKAY [ 0.017s]
writing 'aboot'...
OKAY [ 0.728s]
finished. total time: 0.748s

E:\>fastboot flash boot E:\7600\LE11B01U01SIM76000L\boot.img
target reported max download size of 134217728 bytes
sending 'boot' (5412 KB)...
OKAY [ 0.178s]
writing 'boot'...
OKAY [ 1.684s]
finished. total time: 1.867s

E:\>fastboot flash system E:\7600\LE11B01U01SIM76000L\system.img
target reported max download size of 134217728 bytes
sending 'system' (43008 KB)...
OKAY [ 1.362s]
writing 'system'...
OKAY [ 20.118s]
finished. total time: 21.485s

E:\>fastboot flash modem E:\7600\LE11B01U01SIM76000L\modem.img
target reported max download size of 134217728 bytes
sending 'modem' (37632 KB)...
OKAY [ 1.192s]
writing 'modem'...
OKAY [ 11.740s]
finished. total time: 12.936s

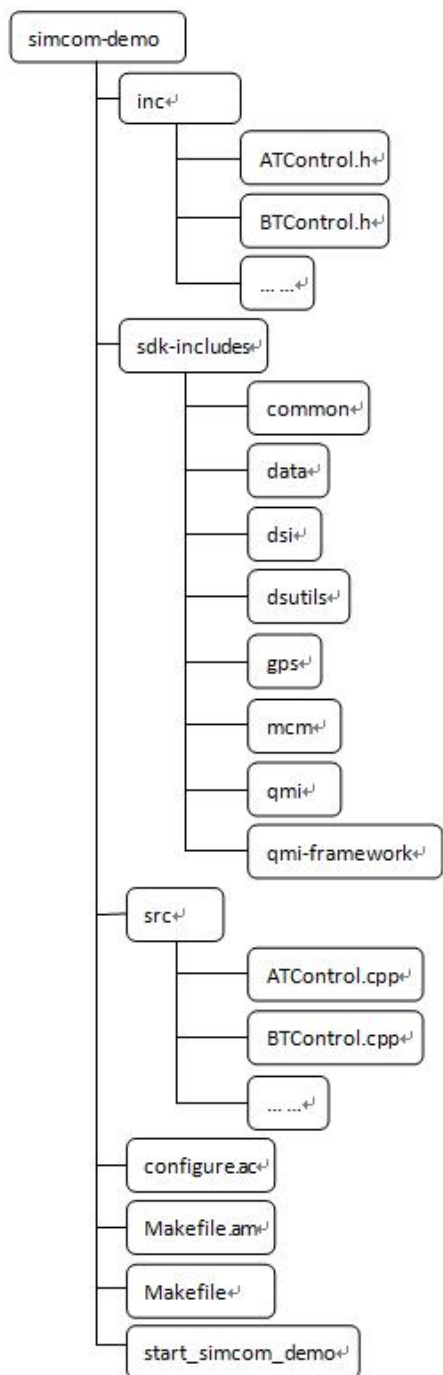
E:\>fastboot flash recovery E:\7600\LE11B01U01SIM76000L\recovery.img
target reported max download size of 134217728 bytes
sending 'recovery' (6006 KB)...
OKAY [ 0.195s]
writing 'recovery'...
OKAY [ 1.857s]
finished. total time: 2.056s

E:\>fastboot flash recoveryfs E:\7600\LE11B01U01SIM76000L\recoveryfs.img
target reported max download size of 134217728 bytes
sending 'recoveryfs' (8576 KB)...
OKAY [ 0.280s]
writing 'recoveryfs'...
OKAY [ 2.643s]
finished. total time: 2.928s
```

## 2.5 应用程序编译和运行

## 2.5.1 demo 应用

Demo 的目录结构:



说明：本文所有提到的 demo 都在二次环境压缩包内，目录为 simcom-demo

### 1) Demo 程序在 windows 下编译

将 simcom-demo 目录下 Makefile\_Windows 重命名为 Makefile, 用 make 命令直接编译, 在 simcom-demo/bin/ 目录下生成 demo\_app 应用程序。通过 adb push 的方式把编译出来的可执行文件导入到 SIM7600 的 Linux 系统的 data 目录下, 增加执行权限, 然后可以通过 adb shell 进入到 SIM7600 的 Linux 控制台里, 手动调试

执行程序。调试阶段客户通过 `adb shell` 的控制终端启动和调试应用程序，代码中可以用 `printf` 的方式加调试 `log` 和信息。

## 2) Demo 程序在 Linux 下编译

编译方法见 2.2.1.6 节

### 2.5.2 helloworld 应用

#### 1)helloworld 编译

执行 `make helloworld` 即可,生成的应用在 `sim_open_sdk/output` 目录下

#### 2) helloworld 自启动设置

将编译生成的应用程序 `helloworld` 拷到 `sim_open_sdk/sim_usrfs` 目录下，之后将 `sim_open_sdk/helloworld/start_helloworld` 文件拷到 `sim_open_sdk/sim_rootfs/etc/init.d/` 目录下并在 `sim_open_sdk` 目录下执行如下命令

```
sudo ln -sf ../init.d/start_helloworld sim_rootfs/etc/rc5.d/S99start_helloworld
```

之后按照 2.2.1.4 节重新生成 `rootfs` 镜像即可

## 2.6 系统分区和文件保护

系统分区中，`system,data,cache` 三个为一个物理分区，三个逻辑分区的大小可以动态调整，调整文件为 `sim_open_sdk/tools/ubinize_system_userdata.cfg`

```
[sysfs_volume]
```

```
mode=ubi
```

```
image=./mdm9607-perf-sysfs.ubifs
```

```
vol_id=0
```

```
vol_type=dynamic
```

```
vol_name=rootfs
```

```
vol_size=55MiB //此处调整 system 分区大小
```

```
[usrfs_volume]
```

```
mode=ubi
```

```
image=./usrfs/mdm9607-perf-usrfs.ubifs
```

```
vol_id=1
```

```
vol_type=dynamic
```

```
vol_name=usrfs
```

```
vol_flags = autoresize//data 分区大小自动设置
```

```
[cache_volume]
```

```
mode=ubi
```

```
vol_id=2
```

```
vol_type=dynamic
```

```
vol_name=cachefs
```

```
vol_size=45MiB //此处调整 cache 分区大小
```

系统中所有必要文件必须都放到 **system** 目录下，临时生成的文件放到 **data** 目录下，注意 **system** 为只读目录，**data** 文件作为临时文件存放处，在一定的极限条件下可能会丢失。

在系统运行时，为了保证进程的安全，采用/data 和/cache 备份的方案。

(参考开发环境中的 helloworld/helloworld.c 以及 helloworld/helloworld\_auto\_restore.sh)

模块/data 下应用程序：/data/helloworld

模块/data 下应用程序的版本标识：/data/helloworld\_ver.txt

模块/cache 下应用程序：/cache/helloworld\_bak

模块/cache 下应用程序的版本标识：/cache/helloworld\_bak\_ver.txt

(1)模块开机执行检测脚本/etc/init.d/helloworld\_auto\_restore.sh

(2)第一次开机会将/data/helloworld 复制到/cache/helloworld\_bak，再执行/data/helloworld

(3)后续/data/helloworld 不存在，则将/cache/helloworld\_bak 复制到/data/helloworld；/cache/helloworld\_bak 不存在，则将/data/helloworld 复制到/cache/helloworld\_bak。

(4)两者都存在，则判断两者对应的版本标识。若两者不一致，则将高版本的程序覆盖到低版本的程序。

(5)应用程序升级策略

- 1) /data/helloworld 顺序删除/cache/helloworld\_bak\_ver.txt、/cache/helloworld\_bak
- 2) /data/helloworld 下载目标程序到/cache/helloworld\_bak
- 3) /data/helloworld 启动/cache/helloworld\_bak update &
- 4) /cache/helloworld\_bak 写下自己对应的版本标识
- 5) 重启

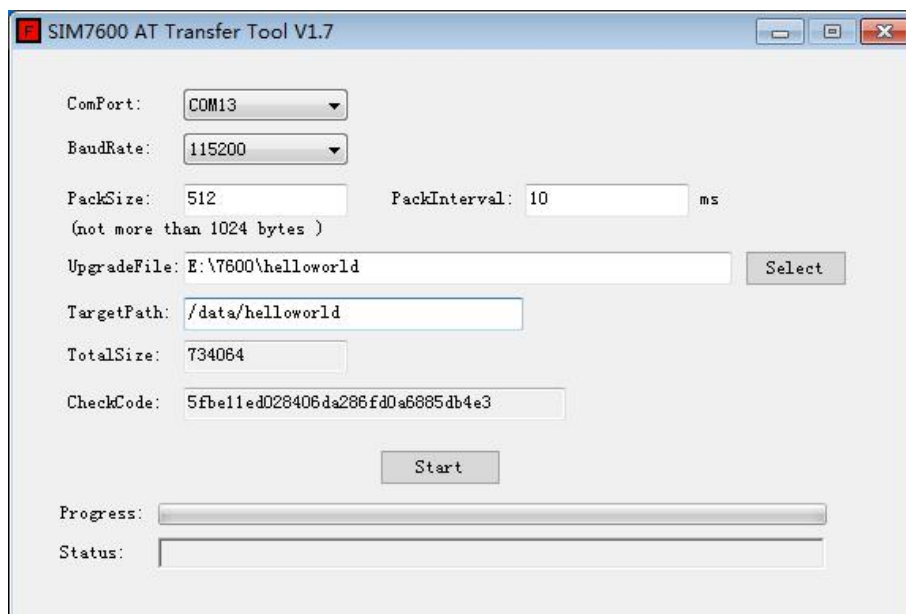
注意：

由于 Linux 系统的缓存原理，对文件操作不会立即写入 flash，异常断电会导致文件损坏或丢失。所以对重要的文件操作，需要执行 sync 命令，来保证操作完成。

## 2.7 产线生产模式

### 2.7.1 单独下载 APP

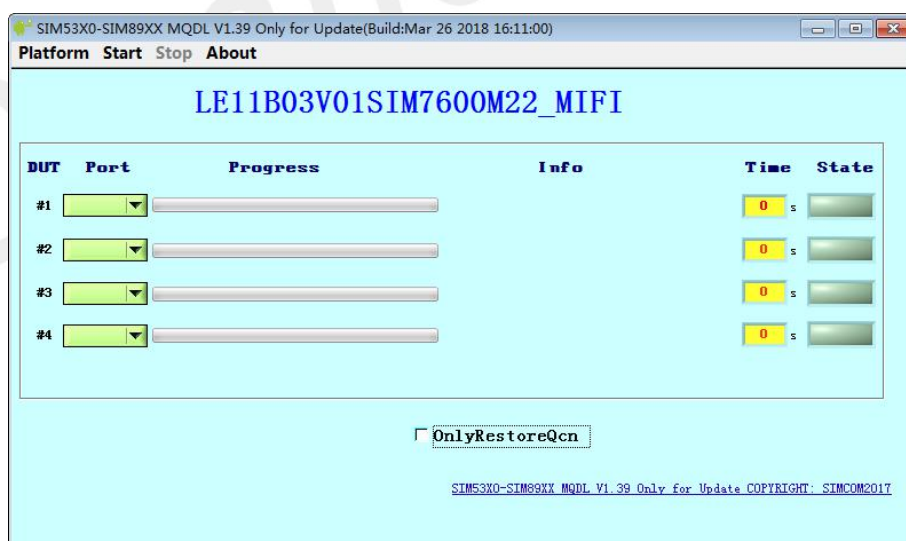
SIMCom 提供一个二次开发的工具用来产线下载客户开发好的应用程序。如下图所示：



这个工具的基本原理是客户选择自己的编译好的应用程序文件，然后在 **TargetPath** 里填入目标路径，点击 **start**，工具会把这个程序放到模块对应的路径内，然后自动修改成可执行权限。如上图，把 PC 上 E 盘里的 **7600\helloworld** 文件下载到模块的 **/data/** 目录中，并命名为 **helloworld**。模块下次开机就会自动检查 **data** 下是不是有 **helloworld**，如果有的话就会自动执行，完成客户的程序启动。

## 2.7.2 下载编译出的 Images

提供产线工具，供客户直接下载使用，可以一拖四来下载，工具安装好后会有使用说明。



## 3. 编程指导

### 3.1 系统基本 API

SIM7600 Linux 系统的基本 API 和所有的 Linux 系统一样。Timer 和 RTC 定时器有所不同，RTC 定时器在系统休眠之后还是准确的。TCP(socket) demo 同 linux 的使用，具体可以参考代码中的 DEMO。

### 3.2 嵌入式 AT 收发

Linux 二次开发应用程序可以使用所有 AT 指令的功能。通过使用 Linux 中 tty 设备节点/dev/smd8，用于支持 AT 命令通讯。

#### 3.2.1 发送 AT 命令接口

接口	int sendATCmd(char* atCmd, char* finalRsp, char *buff, uint32_t buffSize, long timeout_ms)
输入	atCmd :AT 命令 finalRsp: 期望的返回值中所带字符串 buff : AT 返回值所有内容指针 buffSize: AT 返回值所有内容指针大小，至少 100 字节 timeout_ms : AT 超时时间
输出	无
返回值	0: 成功    -1: 失败    其他: AT 返回内容长度
NOTE	

### 3.3 UART

**SIM7600CE-T:** 双路高速串口 UART1 和 UART2，设备节点分别是 UART1: /dev/ttyHS0（可配置成蓝牙功能）和 UART2: /dev/ttyHS1。所有的串口均不支持 AT 指令的使用。

**SIM7600CE-AT:** 最多可以配置成三路高速串口，所有的串口均不支持 AT 指令的使用。串口配置的说明文档，请参考《SIM7600 Open Linux UART&SPI 说明文档》。设备节点分别是：



UART1: Device ID 是/dev/ttyHS1;

UART2: 默认是蓝牙功能。如果配置为高速串口, Device ID 是/dev/ttyHS0;

UART3: 默认是 debug with console 功能, Device ID 是/dev/ttyHSL0。如果配置为高速串口, Device ID 是/dev/ttyHS2。

具体参数如下表:

Parameter s	Value ranges	Defaults	Setting method
起始位	1bit	1bit	ioctl
数据位	7bit,8bit	8bit	ioctl
停止位	1bit,2bit	1bit,	ioctl
校验位	Odd,Even,Space, None	None	ioctl
波特率	300,600,1200,2400,4800,9600,19200,38400,57600,115200,230400,460800,921600,1000000,1152000,1500000,3000000,3200000,3686400	115200	ioctl
单 帧 发 送 大 小	512B	512B	
单 帧 响 应 延 时	(1-10) ms (每单帧同步发送完成后, Host 需要等待的时间)	建议 1ms	
最大 buffer	10KB	10KB	

SIM7600CE-T 的 PIN 脚定义如下表:

PIN No	PINNAME	Default Function	Pull	Wakeup Interrupt
6	SPI_CLK	UART1_RTS	B-PD	
7	SPI_MISO	UART1_RX	B-PD	
8	SPI_MOSI	UART1_TX	B-PD	
9	SPI_CS	UART1_CTS	B-PD	
66	RTS	UART2_RTS	B-PD	
67	CTS	UART2_CTS	B-PD	
68	RXD	UART2_RX	B-PD	
71	TXD	UART2_TX	B-PD	

SIM7600CE-AT 的 PIN 脚定义如下表:

PIN No	PINNAME	Default Function	Pull	Wakeup Interrupt
6	SPI_CLK	UART2_RTS	B-PD	
7	SPI_MISO	UART2_RX	B-PD	
8	SPI_MOSI	UART2_TX	B-PD	
9	SPI_CS	UART2_CTS	B-PD	
66	RTS	UART1_RTS	B-PD	
67	CTS	UART1_CTS	B-PD	
68	RXD	UART1_RX	B-PD	



71	TXD	UART1_TX	B-PD	
42	DGB_TXD	UART3_TX	B-PD	
55	SCL	UART3_RTS	B-PD	
56	SDA	UART3_CTS	B-PU	
106	DGB_RXD	UART3_RX	B-PD	

### 3.4 GPIO

如下是 SIM7600CE-T GPIO 可控脚的说明

PI N No	PINNAME	SYS GPIO No.	Default Function	Function n 0	Function 1	Pull	Wakeup Interrupt
33	GPIO3	GPIO_1020	MIFI_POWER_EN	GPIO	MIFI_POWER_EN	B-P U	
34	GPIO6	GPIO_1023	MIFI_SLEEP_CLK	GPIO	MIFI_SLEEP_CLK	B-P D	
48	SD_DET	GPIO_26	GPIO	GPIO	SD_DET	B-P D	*
49	STATUS	GPIO_52	Status	GPIO	Status	B-P D	*
50	GPIO43	GPIO_36	MIFI_COEX	GPIO	MIFI_COEX	B-P D	
51	NETLIGHT	GPIO_18	NETLIGHT	GPIO	NETLIGHT	B-P D	
52	GPIO41	GPIO_79	BT_PCM	GPIO	BT_PCM	B-P D	*
54	FLIGHTMODE	GPIO_76	BT_PCM	GPIO	BT_PCM	B-P D	*
69	UART_RI	GPIO_50	GPIO(RI)	GPIO	GPIO	B-P D	
70	UART_DCD	GPIO_51	GPIO	GPIO	GPIO	B-P D	
72	UART_DTR	GPIO_74	GPIO(DTR)	GPIO	GPIO	B-P D	*
86	COEX3	GPIO_78	BT_PCM	GPIO	BT_PCM	B-P D	
87	GPIO77	GPIO_77	BT_PCM	GPIO	BT_PCM	B-P D	

注：

- 如果不需要 BT\_PCM 功能，可以 export 对应的 SYS GPIO No.作为 GPIO 使用。
- 如果 UART 配置为低速串口，CTS，RTS 可以作为 GPIO 使用

如下是 SIM7600CE-AT GPIO 可控脚的说明：

PIN No	PINNAME	SYS GPIO No.	Default Function	Function 0	Function 1	Pull	Wakeup Interrupt
33	GPIO3	GPIO_1020	MIFI_POWER_EN	GPIO	MIFI_POWER_EN	B-P U	
34	GPIO6	GPIO_1023	MIFI_SLEEP_CLK	GPIO	MIFI_SLEEP_CLK	B-P D	
48	SD_DET	GPIO_26	GPIO	GPIO	SD_DET	B-P D	*
49	STATUS	GPIO_52	Status	GPIO	Status	B-P D	*
50	GPIO43	GPIO_36	MIFI_COEX	GPIO	MIFI_COEX	B-P D	
69	UART_RI	GPIO_50	GPIO(RI)	GPIO	GPIO	B-P D	
70	UART_DC D	GPIO_51	GPIO	GPIO	GPIO	B-P D	
72	UART_DTR	GPIO_74	GPIO(DTR)	GPIO	GPIO	B-P D	*

注：

- PIN50,PIN52,PIN87 开机阶段作为 BOOT\_CFG 使用,所以如果开发中要使用这几个 GPIO，只作为输出。
- PIN69 固定作为唤醒 MCU 的 PIN 脚，PIN72 固定作为 MCU 唤醒模块的 PIN 脚。

实例：

#### 1) GPIO 配置

关于 GPIO 的操作方法：

通过/sys/class/gpio/\*\*\*文件节点,这种方法需要注意 PIN No.和 SYS GPIO No.的对应关系.

1)).动态建立 GPIO 文件节点,向/sys/class/gpio/export 文件中写入 GPIO 号,比如使用 PIN50:

```
echo 36 > /sys/class/gpio/export
```

2)).拉高拉低 GPIO,先设置 GPIO 为输出,再设置输出高低  
比如拉高 PIN50:

```
echo 36 > /sys/class/gpio/export
```

```
echo out> /sys/class/gpio/gpio36/direction
```

```
echo 1> /sys/class/gpio/gpio36/value
```

3)).读取 GPIO,先设置 GPIO 为输入,再读取状态

```
echo in> /sys/class/gpio/gpio36/direction
```

```
cat /sys/class/gpio/gpio36/value
```

## 2) GPIO 中断配置

方法:

```
echo "in" > /sys/class/gpio/gpio36/direction
```

```
echo "both" > /sys/class/gpio/gpio36/edge
```

需 poll value 节点的状态，实时触发响应。

## 3.5 ADC

模块提供了二个 ADC 模数转换接口，对应模块的 PIN46.PIN47. 分别对应 ADC2 和 ADC1  
开发者可以读取相对应的设备文件获取电压值。

接口:	int read_adc(int ch)
输入:	ch :通道
输出:	无
返回值:	0: 成功     -1: 失败
NOTE:	

## 3.6 I2C

**SIM7600CE-T:** SDA,SCL 分别对应模块的 PIN55 和 PIN56。开发者先 open /dev/i2c-5 设备获取句柄，然后调用读写函数对设备进行操作。

**SIM7600CE-AT:** SDA,SCL 分别对应模块的 PIN51 和 PIN119。开发者先 open /dev/i2c-4 设备获取句柄，然后调用读写函数对设备进行操作。

### 3.6.1 写 I2C 接口

接口:	int sim_i2c_write(uint8_t slave_address, uint16_t reg, uint8_t *buf, int len)
输入:	slave_address    设备从地址 reg                设备寄存器 buf                要写入的数据 buffer len                要写入的数据长度
输出:	无
返回值:	0: 成功     -1: 失败其他: 写入的数据长度
NOTE:	

### 3.6.2 读 I2C 接口

接口:	int sim_i2c_read(uint8_t slave_address, uint16_t reg, uint8_t *buf, int len)	
输入:	slave_address	设备从地址
	reg	设备寄存器
	len	要写入的数据长度
输出:	buf	读到的数据 buffer
返回值:	0: 成功    -1: 失败	
NOTE:		

## 3.7 SD Card/EMMC flash

通过 SDIO PIN (21-26), 外接 SD Card 或 EMMC, 默认 mount 在/media/card 上, 快捷方式为/sdcard。分区和格式化需要在产线上做一次, 使用 AT 命令 AT+CFDISK, 之后系统默认只 mount 第一个分区为/sdcard, 其它分区需要在 APP 中用命令再做 mount.

### 3.7.1 分区

使用 AT 指令 AT+CFDISK=<num>[,<size>]进行分区。

### 3.7.2 格式化

首次分区后, 需使用 AT 指令 AT+CFDISK 进行格式化。

### 3.7.3 挂载

模块开机自动挂载/dev/mmcblk0p1 到/media/card, 如有其它分区则需要用户挂载。

例如挂载/dev/mmcblk0p2 到/mnt:

```
mount -t auto /dev/mmcblk0p2 /mnt
```

### 3.7.4 CFDISK 命令

AT+CFDISK	
Test Command <b>AT+ CFDISK=?</b>	Response a)If successfully: <b>+CFDISK: (1-4)[...]</b> <b>OK</b> b)If failed: <b>ERROR</b>
Read Command <b>AT+ CFDISK?</b>	Response a)If successfully: <b>+CFDISK: &lt;num&gt;,&lt;size&gt;</b> <b>OK</b> b)If failed: <b>ERROR</b>
Write Command <b>AT+ CFDISK=&lt;num&gt;[,&lt;size&gt;,...]</b>	Response a)If successfully: <b>OK</b> b)If failed: <b>ERROR</b>
Write Command(Formatting all partitions) <b>AT+ CFDISK</b>	Response a)If successfully: <b>OK</b> b)If failed: <b>ERROR</b>

#### Defined values

<b>&lt;num&gt;</b>	Partition number
<b>&lt;size&gt;</b>	Partition size.The unit is KB. <b>NOTE:</b> The last partition size does not need to be set. The size of the last partition is the size of the disk remaining.

#### Examples

```

AT+CFDISK=?
OK+CFDISK: (1-4)[...]

OK
AT+CFDISK=4,50000,50000,50000
OK
AT+CFDISK
OK
AT+CFDISK?
+CFDISK: 1,50040
+CFDISK: 2,50048

```

```
+CFDISK: 3,50048
+CFDISK: 4,3708288
```

```
OK
```

## 3.8 UIM

客户可以使用 UIM 的相关函数获取一些关于 SIM 卡的信息，比如 ICCID 和 IMSI 等。

### 3.8.1 查询 SIM 卡状态

接口:	int getSimCardStatus(SimCard_Status_type *simStatus);
输入:	无
输出:	SimStatus card_status: 0: 未检测到 SIM 卡 1: 检测到 SIM 卡 2: 未知错误 app_type: SIM 卡类型 1: SIM 2: USIM 3: RUIM 4: CSIM 5: ISIM app_state: 1: detected 2: pin block 3: puk block 4: person check 5: Permanently blocked 6: illegal pin_state: 1: unknown 2: Enabled and not verified 3: Enabled and verified 4: Disabled 5: Blocked 6: Permanently blocked) pin_retries: 剩余解 pin 次数 puk_retries: 剩余解 puk 次数
返回值:	0: 成功 -1: 失败
NOTE:	

### 3.8.2 查询 SIM 卡 ICCID

接口:	int get_iccid(char *piccid);
输入:	无
输出:	piccid: ICCID
返回值:	0: 成功 -1: 失败
备注:	piccid 至少 21 个字节长度的指针或数组，并且内容已经初始化为 0

### 3.8.3 查询 SIM 卡 IMSI

接口:	int get_imsi(char *imsi);
输入:	无
输出:	imsi: IMSI
返回值:	0: 成功    -1: 失败
备注:	imsi 至少 16 个字节长度的指针或数组, 并且内容已经初始化为 0

## 3.9 SMS

实现短信的收发

### 3.9.1 短信初始化

接口:	int sms_init(sms_ind_cb_fcn sms_ind_cb);
输入:	sms_ind_cb: 用于处理短信接收
输出:	无
返回值:	0: 成功    -1: 失败
NOTE:	初始化后, 再进行短信其他操作

### 3.9.2 接收短信格式设置

接口:	void sms_set_format(sms_format format);
输入:	format: 1:ASCII    not use 2:UTF8     default 3:UCS2
输出:	无
返回值:	无
NOTE:	接收英文短信输出格式为 ASCII, 中文短信默认输出格式为 UTF8, 可通过此函数设置为 UCS2

### 3.9.3 短信发送

接口:	<code>int send_message(int smsMode, char *phoneNumber, unsigned char *content, int content_len);</code>
输入:	smsMode: 1:ASCII 2:UTF8 3:UCS2 phoneNumber: 目的手机号码 content: 短信内容 content_len: 短信长度
输出:	无
返回值:	0: 成功 -1: 失败
NOTE:	

### 3.9.4 回调函数处理短信接收

接口:	<code>void process_simcom_ind_message(simcom_event_e event,void *cb_usr_data);</code>
输入:	无
输出:	event=SIMCOM_EVENT_SMS_PP_IND cb_usr_data: 对应结构体 typedef struct { boolean is_concat;     // 长短信标志, 1 表示长短信 uint8 msg_ref;         // 长短信消息号, 相同的消息号表示一条短信 uint8 seq_num;         // 长短信当前条数 uint8 total_sm;        // 长短信总条数 sms_format format;     // 1:ASCII 2:UTF8 3: UCS2 int message_len;       // 短信长度 char message_content[SMS_MAX_CONTENT_LENGTH];     // 短信内容 char source_address[SMS_MAX_ADDRESS_LENGTH];     // 手机号码 }sms_info_type; is_concat: 长短信标志, 1 表示长短信 msg_ref: 长短信消息号, 相同的消息号表示一条短信 seq_num: 长短信当前条数 total_num: 长短信总条数 format: 1:ASCII 2:UTF8 3: UCS2 message_len: 短信长度 message_content: 短信内容 source_address: 手机号码
返回值:	无
NOTE:	该函数在 sms_init 作为参数传递进去



## 3.10 Voice Call

用来拨打和接听语音电话，同时可以监控语音电话的状态。

### 3.10.1 电话初始化

接口:	int Init();
输入:	无
输出:	无
返回值:	0: 成功    -1: 失败
NOTE:	初始化后，再进行电话其他操作

### 3.10.2 拨打电话

接口:	int voice_dial(char *phoneNum);
输入:	phoneNum: 需要拨打的号码
输出:	无
返回值:	0: 成功    -1: 失败
NOTE:	

### 3.10.3 处理当前通话

接口:	int voice_mtcall_process(voice_call_option option, unsigned char call_id);
输入:	Option: 1: 挂断电话，针对当前只有一个通话 2: 接听，针对当前只有一个通话 3: 通话保持，保持当前的通话，接听新的来电 4: 挂断所有通话 5: 挂断保持的通话 6: 挂断当前通话 Call_id:option 等于 1 或者 2 时有效
输出:	无
返回值:	0: 成功    -1: 失败
NOTE:	

### 3.10.4 得到指定通话的状态

接口:	int get_call_info(uint8_t callid, call_info2_type *pcall_info);
输入:	callid 非 0 值
输出:	pcall_info 结构体包含参数: 1 call_id, - 跟输入的一致 2 call_state: 1: 发起通话 2: 来电 3: 通话建立 5: 响铃 6: 通话保持 7: 通话等待 8: 正在挂断 9: 挂断 3 方向: 1: 呼出 2: 呼入 4 号码
返回值:	0: 成功 -1: 失败
NOTE:	

### 3.10.5 得到所有的通话状态

接口	int get_all_call_info(call_info_type *pcall_info_list);
输入	
输出	pcall_info_list 结构体包含参数: 1. call_num - 当前共有几路通话 2. call_info: 2.1 call_id, - 跟输入的一致 2.2 call_state: 1: 发起通话 2: 来电 3: 通话建立 5: 响铃 6: 通话保持 7: 通话等待 8: 正在挂断 9: 挂断 2.3 方向: 1: 呼出 2: 呼入 2.4 号码

返回值	0: 成功    -1: 失败
-----	-----------------

NOTE
------

### 3.10.6 回调函数

接口	void      process_simcom_ind_message(simcom_event_e      event,void *cb_usr_data)
----	--

输入 输出
----------

Event = SIMCOM\_EVENT\_VOICE\_CALL\_IND

cb\_usr\_data 结构体包含参数:

1. call\_num    - 当前共有几路通话
2. call\_info:
  - 2.1 call\_id,    - 跟输入的一致
  - 2.2 call\_state:
    - 1: 发起通话
    - 2: 来电
    - 3: 通话建立
    - 5: 响铃
    - 6: 通话保持
    - 7: 通话等待
    - 8: 正在挂断
    - 9: 挂断
  - 2.3 方向:
    - 1: 呼出
    - 2: 呼入
  - 2.4 号码

返回值	0: 成功    -1: 失败
-----	-----------------

NOTE
------

## 3.11 NAS

NAS 接口主要用于获取一些关于网络状态的信息，比如注册网络类型和信号等。

### 3.11.1 查询注册网络状态

接口	int get_NetworkType(nas_serving_system_type_v01 *serving_system);
----	---

输入	无
----	---

## 输出

serving\_system

registration\_state: 注册状态

- 0x00 –未注册

- 0x01 – 已注册

- 0x02 –搜索网络中

- 0x03 –注册被拒接

- 0x04 –未知状态

cs\_attach\_state: CS 域状态，是否附着成功决定是否能通话

- 0x00 –未知

- 0x01 –附着成功

- 0x02 –未附着成功

ps\_attach\_state: PS 域状态，是否附着成功决定是否能拨号上网

- 0x00 -- 未知

- 0x01 -- 附着成功（注：附着成功不表示能上网，还需要拨号）

- 0x02 -- 未附着成功

selected\_network; 网络类型

- 0x00 –未知

- 0x01 –3GPP2

- 0x02 –3GPP

radio\_if\_len: 注册的网络个数（一般有效个数是一个，但是电信卡有可能有两个，同时注册上 LTE 和 CDMA）

radio\_if[]; 注册网络数组

-0x00 –无服务

-0x01 –CDMA\_1X

-0x02 –CDMA\_EVDO

-0x04 -- GSM

-0x05 -- UMTS

-0x08 -- LTE

## 返回值

0: 成功 -1: 失败

## NOTE

### 3.11.2 查询信号

## 接口

int get\_SignalStrength(int \* signalStrength);

## 输入

无

## 输出

signalStrength: 信号值

## 返回值

0: 成功 -1: 失败

## 备注

## 3.12 WDS

WDS 接口主要用于查询和设置 APN。

### 3.12.1 查询 APN

接口	int get_apnInfo(int profile_index, int* pdp_type, char* apn_str, char *username, char *password);
输入	无
输出	profile_index: APN index pdp_type: pdp 类型 - 0 -- IPv4 - 2 -- IPv6 - 3 -- IPv4 and IPv6 apn_str: APN 字符串 username: 用户名 password: 密码
返回值	0: 成功 -1: 失败
备注	

### 3.12.2 设置 APN

接口	int set_apnInfo(int profile_index, int pdp_type, char* apn_str, char *username, char *password);
输入	profile_index: (1~16) : 需要查询的 APN index pdp_type: pdp 类型, 一般选择 0 - 0 -- IPv4 - 2 -- IPv6 - 3 -- IPv4 and IPv6 apn_str: APN 字符串 username: 用户名, 如果没有, 设置为 NULL password: 密码, 如果没有, 设置为 NULL
输出	无
返回值	0: 成功 -1: 失败
备注	

### 3.12.3 Data Call

用来完成设置/查询 APN，CDMA 网络下拨号用户名和密码，连接网络，断开网络连接等功能。最多同时支持 8 个链接，APP 使用的数据链接建议从第 7 个 profile 开始。

### 3.12.4 初始化网络

接口	int Init();
输入	无
输出	无
返回值	无
NOTE	

### 3.12.5 建立数据链接

接口	int start_dataCall(app_tech_e tech, int ip_family, int profile);
输入	tech: 1 umts 2 cdma 3 1x 4 do 5 lte 6 auto ip_family: 4ipv4 6 ipv6 10 ipv4 and ipv6 profile: profile 序列号
输出	无
返回值	0: 成功    -1: 失败
NOTE	

### 3.12.6 获取数据链接参数

接口	int get_datacall_info(int profile, datacall_info_type *pcallinfo);
输入	profile: profile 序列号

输出	<pre> datacall_info_type *pcallinfo typedef struct {     dsi_hndl_t handle; //链接句柄     enum app_tech_e tech; //使用制式     int family; //IP 类型     int profile; //profile 序列     datacall_status_type status; //当前链接状态     char if_name[32]; //Interface 名字     char ip_str[16]; //IP 地址     char pri_dns_str[16]; //主 DNS     char sec_dns_str[16]; //次 DNS     char gw_str[16]; //网关     unsigned int mask; //掩码 }datacall_info_type; </pre>
返回值	0: 成功    -1: 失败
NOTE	

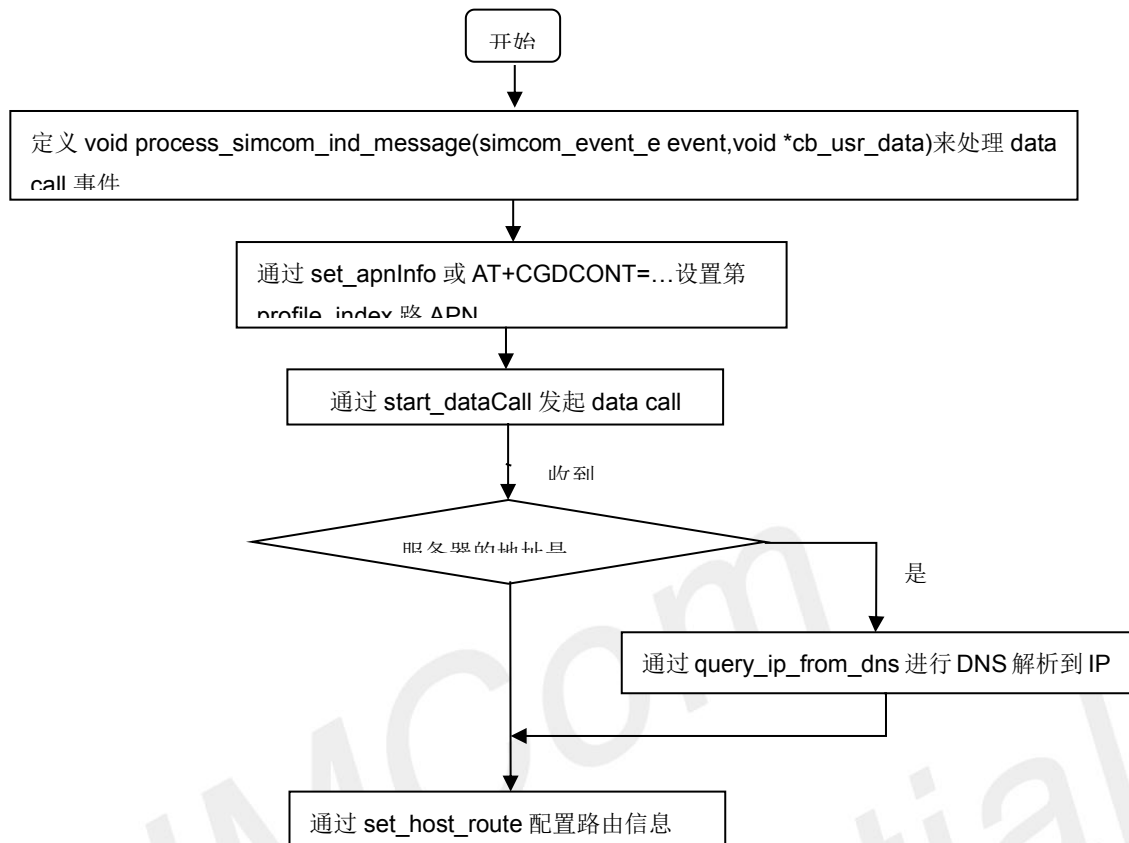
### 3.12.7 释放数据链接

接口	int stop_dataCall(int profile);
输入	profile: profile 序列号
输出	无
返回值	0: 成功    -1: 失败
NOTE	

### 3.12.8 释放网络资源

接口	int Delnit();
输入	无
输出	无
返回值	无
NOTE	

### 3.12.9 Data Call 流程说



### 3.13 GNSS

实现 GNSS 的开启关闭,XTRA 的使用, 以及 NMEA 和经纬度的输出。

#### 3.13.1 初始化 GNSS

接口	static boolean gps_init();
输入	无
输出	无
返回值	TRUE:成功 FALE:失败
NOTE	该函数必须在所有其他操作之前执行

#### 3.13.2 使能 XTRA

接口	static void gps_xtra_enable();
输入	无



输出	无
返回值	无
NOTE	如需使用 XTRA，必须在开启 GNSS 之前 ENABLE,且需要模块联网

### 3.13.3 禁止 XTRA

接口	static void gps_xtra_disable();
输入	无
输出	无
返回值	无
NOTE	

### 3.13.4 GNSS 冷启动

接口	static boolean gps_coldstart();
输入	无
输出	无
返回值	TRUE:成功 FALE:失败
NOTE	信号良好的情况下，冷启动约 35 秒

### 3.13.5 GNSS 热启动

接口	static boolean gps_hotstart();
输入	无
输出	无
返回值	TRUE:成功 FALE:失败
NOTE	

### 3.13.6 停止定位

接口	static boolean gps_stop();
输入	无
输出	无
返回值	TRUE:成功 FALE:失败

## NOTE

### 3.13.7 回调函数输出简要位置信息

接口	void process_simcom_ind_message(simcom_event_e event,void *cb_usr_data)
输入	无
输出	无
返回值	无
NOTE	<p>开启 GNSS 且定位后，持续且每秒输出经纬度，时间，高度，速度等</p> <p>Event = SIMCOM_EVENT_LOC_IND</p> <p>cb_usr_data 结构类型：</p> <pre>typedef struct {     double        latitude; //经度     double        longitude; //纬度     double        altitude; //高度     float         speed;     //速度     float         bearing;   //角度     float         accuracy; //精度     //time[0] = year-2000     //time[1] = month     //time[2] = day     //time[3] = time     //time[4] = minute     //time[5] = second     uint8_t time[6]; } GpsInfo;</pre>

### 3.13.8 回调函数输出 NMEA 语句

接口	void process_simcom_ind_message(simcom_event_e event,void *cb_usr_data)
输入	无
输出	无
返回值	无
NOTE	<p>开启 GNSS 后，持续且每秒输出 NMEA 语句</p> <p>Event = SIMCOM_EVENT_NMEA_IND</p> <p>cb_usr_data 类型 char</p>

### 3.14 WIFI

功能描述:

客户可以选择 W58 或者 W58L 模块。

模块具体的 Spec:

SDIO 3.0 接口, 相关 PIN 脚 (27-32)

20MHz/40MHz (2.4GHz)

作为 AP 模式, 连接节点 30 个

1X1 单天线设计, 低成本方案

软件功能:

支持 WIFI AP/station 混合模式

支持连接 WIFI 热点的设备和连接 Module 设备之间的通讯

支持隐藏 WIFI 热点的功能

提供 AT 指令, 对 WIFI 和路由协议功能的配置

特殊说明:

get\_wifi\_mode 和 set\_wifi\_mode 适用于 W58, W58L 不能使用。

sta\_init 和 get\_sta\_status 使用于 W58L, W58 不能使用。

#### 3.14.1 获取当前 WIFI 模式设置(for W58)

接口	wifi_mode_type get_wifi_mode()
输入	无
输出	无
返回值	wifi_mode_type 类型: 0: 单 AP 模式 1: 双 AP 模式 2: AP+STA 模式
NOTE	

#### 3.14.2 设置 WIFI 模式(for W58)

WIFI 的模式有三种:

0: 单 AP 模式 提供一个热点供手机上网

1: 双 AP 模式 提供两个热点供手机上网

2: AP+STA 模式提供一个热点工手机上网, 同时提供一个 WIFI 客户端让模块连接其他路由器上网

一般来说，如果设备设计的 WIFI 模式是固定的，最好将 WIFI 的模式预设好。

预置 WIFI 模式方法：

修改 sim\_open\_sdk/sim\_usrfs/mobileap\_cfg.xml 和 sim\_open\_sdk/sim\_rootfs/etc/default\_mobileap\_cfg.xml 中的节点：

<WlanMode>AP-AP</WlanMode>值：

AP 模式：            AP

双 AP 模式：         AP-AP

AP+STA 模式：        AP-STA

b.修改 sim\_open\_sdk/sim\_usrfs/mobileap\_enable\_cfg 和  
sim\_open\_sdk/sim\_rootfs/etc/default\_mobileap\_enable\_cfg 值

AP 模式：            0

AT+AP 模式：         1

AP+STA 模式：        2

注意： a b 模式保持一致

如果设备的 WIFI 模式需要根据场景动态切换。则可以调用下面接口进行调整：

接口	wifi_mode_type set_wifi_mode(wifi_mode_type mode)
输入	mode: wifi_mode_type 类型 0: 单 AP 模式 1: 双 AP 模式 2: AP+STA 模式
输出	无
返回值	boolean 类型： TRUE or FALSE
NOTE	

### 3.14.3 WIFI 开关

接口	boolean wifi_power(int act)
输入	act : int 类型 1: 打开 0: 关闭
输出	无
返回值	boolean 类型： TRUE or FALSE
NOTE	

### 3.14.4 获取 WIFI 状态

接口	boolean get_wifi_status(uint8 *flag)
----	--------------------------------------

输入	无
输出	flag : uint8 *类型: 1: 打开 0: 关闭
返回值	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.5 设置 WIFI 热点名称

接口	boolean set_ssid(char *SSID, ap_index_type ap_index)
输入	1. SSID: Char*类型 热点名称字符串 2. ap_index: ap_index_type 类型 AP 模式下, 为 1 AP-AP 模式下, 为 0 或 1 STA-AP 模式下, 为 2
输出	无
返回值	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.6 获取 WIFI 热点名称

接口	boolean get_ssid(char *str_SSID, ap_index_type ap_index)
输入	ap_index: ap_index_type 类型 AP 模式下, 为 1 AP-AP 模式下, 为 0 或 1 STA-AP 模式下, 为 2
输出	SSID: Char*类型 热点名称字符串
返回值	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.7 设置 AP auth 类型, 加密模式, 密码

接口	boolean set_auth(char *str_pwd, int auth_type, int encrypt_mode, ap_index_type ap_index)
输入	1. str_pwd: char*类型 密码 2. auth_type: int 类型 auth 类型 3. encrypt_mode: int 类型 加密模式 4. ap_index: ap_index_type 类型 AP 模式下, 为 1 AP-AP 模式下, 为 0 或 1 STA-AP 模式下, 为 2
输出	无
返回值	boolean 类型: TRUE or FALSE
NOTE	1.当 auth type 输入为 0 或 1 时, encrypt mode 的输入值为 0 或 1; 2.当 auth type 输入为 2 时, encrypt mode 的输入值只能为 1; 3.当 auth type 的输入值大于 3 时, encrypt mode 的输入值必须大于等于 2; 4.当 encrypt mode 输入为 0 时, 不需要输入 password; 5.当 encrypt mode 输入为 1 时, password 必须输入。 Password 输入格式必须满足: 长度为 5 或者 13 的 ASCII 编码字符串, 或者长度为 10 或者 26 的十六进制编码字符串; 6.当 encrypt mode 的输入值大于等于 2 时, password 必须输入。 Password 输入的格式必须满足: 长度为 8 到 63 的 ASCII 编码字符串, 或者长度为 64 的十六进制编码字符串。  Default value : int authType = 5; int encryptMode = 4; ap_index_type: 0-> ap 0-> ap & ap 2-> ap & sta

### 3.14.8 获取 AP auth 类型, 加密模式, 密码

接口	boolean get_auth(int *auth_type_ptr, int *encrypt_mode_ptr, char *pwd_str, ap_index_type ap_index)
输入	ap_index: ap_index_type 类型 AP 模式下, 为 1 AP-AP 模式下, 为 0 或 1 STA-AP 模式下, 为 2
输出	1. authType: int*类型

返回值	2.encrypt_mode_ptr: int*类型
	3. pwd_str: char*类型
NOTE	boolean 类型:
	TRUE or FALSE

### 3.14.9 设置 WIFI 广播开关

接口	boolean set_bcast(int broadcast, ap_index_type ap_index)
输入	1. Broadcast: int 类型 2. ap_index: ap_index_type 类型 AP 模式下, 为 1 AP-AP 模式下, 为 0 或 1 STA-AP 模式下, 为 2
输出	无
返回值	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.10 获取 WIFI 广播设定

接口	boolean get_bcast(int *broadcast, ap_index_type ap_index)
输入	ap_index: ap_index_type 类型 AP 模式下, 为 1 AP-AP 模式下, 为 0 或 1 STA-AP 模式下, 为 2
输出	Broadcast: int 类型
返回值	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.11 获取 DHCP 设定

接口	boolean get_dhcp(char *host_ip_str, char *start_ip_str, char *end_ip_str, char *time_str)
输入	无
输出	1. host_ip_str: 主机地址

返回值	2. start_ip_str: 开始地址 3. end_ip_str: 结束地址 4. time_str: 授权时长
	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.12 获取连接的客户端数

接口	int get_client_count(ap_index_type ap_index)
输入	ap_index: ap_index_type 类型 AP 模式下, 为 1 AP-AP 模式下, 为 0 或 1 STA-AP 模式下, 为 2
输出	无
返回值	Int 类型, 连接 AP 的客户端数目
NOTE	

### 3.14.13 获取 STA 模式下的 IP 地址

接口	Boolean get_sta_ip(char *ip_str, int len)
输入	无
输出	Ip_str: IP 地址
返回值	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.14 获取 WIFI 物理地址

接口	boolean get_mac_addr(char *mac_addr, ap_index_type ap_index)
输入	ap_index: ap_index_type 类型 AP 模式下, 为 1 AP-AP 模式下, 为 0 或 1 STA-AP 模式下, 为 2
输出	mac_addr: char*类型, 物理地址。第一条为 HOST 的 MAC
返回值	boolean 类型: TRUE or FALSE



## NOTE

### 3.14.15 设置 STA 连接外部热点后获取到的 IP

接口	boolean get_sta_ip(char *ip_str, int len)
输入	无
输出	Ip_str: ip 地址
返回值	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.16 设置 STA 连接外部 AP 的 SSID 和密码

接口	boolean set_sta_cfg(char *ssid_str, char *psk_value)
输入	1. ssid_str: 热点名称 2. psk_value: 密码
输出	无
返回值	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.17 获取 STA 设置的 SSID 和 密码

接口	boolean get_sta_cfg(char *ssid_str, char *psk_value)
输入	无
输出	1. ssid_str: 热点名称 2. psk_value: 密码
返回值	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.18 wifi sta 扫描可用热点

接口	boolean sta_scan(char *list_str)
----	----------------------------------

输入	无
输出	list_str: char 类型的指针，用来存放扫描到的可用热点列表。 格式如下： /r/n<mac_addr>,<name>/r/n <mac_addr>,<name>/r/n .....  说明： <mac_addr>: MAC 地址 <name>: 热点名称
返回值	boolean 类型： TRUE or FALSE
NOTE	

### 3.14.19 wifi sta 扫描可用热点（显示信号强度）

接口	boolean sta_scan(char *list_str)
输入	无
输出	list_str: char 类型的指针，用来存放扫描到的可用热点列表。 格式如下： /r/n<mac_addr>,<name>/r/n <mac_addr>,<name>/r/n .....  说明： <mac_addr>: MAC 地址 <name>: 热点名称
返回值	boolean 类型： TRUE or FALSE
NOTE	

### 3.14.20 设置 cdma 模式拨号的用户名和密码

接口	boolean set_user_name_pwd(char *sz_username, char *sz_usrpwd)
输入	1. sz_username: 用户名 2. sz_usrpwd: 密码
输出	无
返回值	boolean 类型： TRUE or FALSE
NOTE	

### 3.14.21 获取 cdma 模式拨号的用户名和密码

接口	boolean get_user_name_pwd(char *sz_username, int len_name, char *sz_usrpwd, int len_pwd)
输入	无
输出	1. sz_username: 用户名 2. sz_usrpwd: 密码
返回值	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.22 获取网络状态

接口	boolean get_net_status(char *net_enable_str)
输入	无
输出	net_enable_str: 网络状态
返回值	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.23 恢复 wifi 设定

接口	void restore_wifi()
输入	无
输出	无
返回值	无
NOTE	

### 3.14.24 设置 AP 热点名称和 auth 类型, 加密模式, 密码

接口	boolean set_ssid_and_auth(char *SSID, char *str_pwd, int auth_type, int encrypt_mode, ap_index_type ap_index)
输入	1. SSID: Char*类型

	热点名称字符串 2.str_pwd: char*类型 密码 3.auth_type: int 类型 auth 类型 4.encrypt_mode: int 类型 加密模式 5.ap_index: ap_index_type 类型 AP 模式下, 为 1 AP-AP 模式下, 为 0 或 1 STA-AP 模式下, 为 2
输出	无
返回值	boolean 类型: TRUE or FALSE
NOTE	1.当 auth type 输入为 0 或 1 时, encrypt mode 的输入值为 0 或 1; 2.当 auth type 输入为 2 时, encrypt mode 的输入值只能为 1; 3.当 auth type 的输入值大于 3 时, encrypt mode 的输入值必须大于等于 2; 4.当 encrypt mode 输入为 0 时, 不需要输入 password; 5.当 encrypt mode 输入为 1 时, password 必须输入。 Password 输入格式必须满足: 长度为 5 或者 13 的 ASCII 编码字符串, 或者长度为 10 或者 26 的十六进制编码字符串; 6.当 encrypt mode 的输入值大于等于 2 时, password 必须输入。 Password 输入的格式必须满足: 长度为 8 到 63 的 ASCII 编码字符串, 或者长度为 64 的十六进制编码字符串。  Default value : int authType = 5; int encryptMode = 4; ap_index_type: 0-> ap 0-> ap & ap 2-> ap & sta

### 3.14.25 开启/关闭 STA 模式(for W58L)

接口	boolean sta_init(int sta_enable)
输入	sta_enable: int 类型 0    -关闭 STA 模式 1    -开启 STA 模式
输出	无
返回值	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.26 获取 STA 模式开启状态(for W58L)

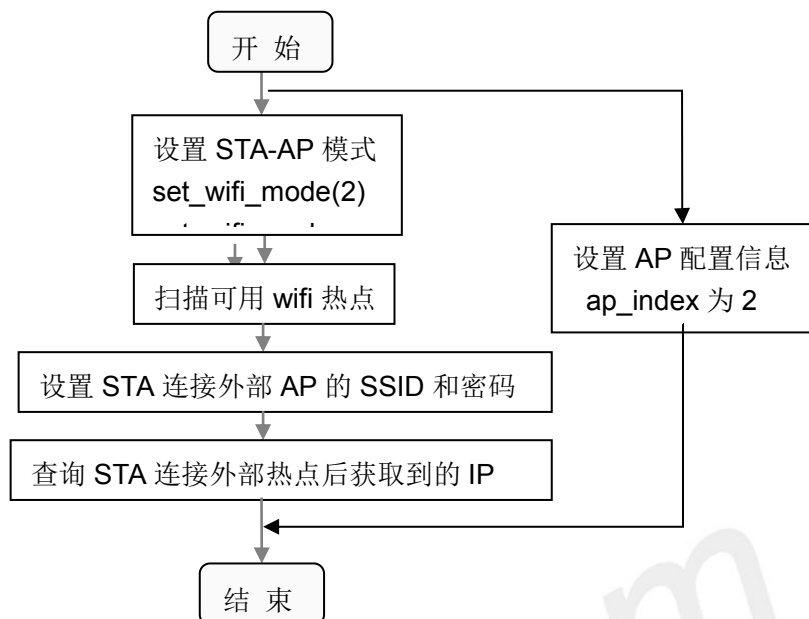
接口	boolean get_sta_status(uint8 *flag)
输入	无
输出	Flag: uint8 *类型 1 – 开启状态 0 – 关闭状态
返回值	boolean 类型: TRUE or FALSE
NOTE	

### 3.14.27 获取操作的结果码

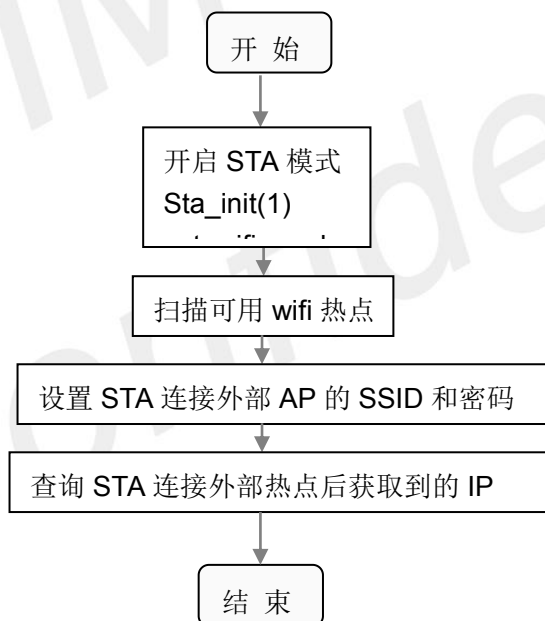
接口	Uin8wifi_get_err_code()
输入	无
输出	无
返回值	Uin8 类型: 0– 成功 1– 无效参数 2–申请内存出错 3–发送/接收消息出错 4–打开文件出错 5–读写文件出错 6–无效的返回值 7–AP 的 ID 错误 8–获取信息出错 9–不是 W58L 10–其他错误
NOTE	结果码保留的是最后一次操作的。 如果调用某个操作的 API 出错，可以调用该接口获取本次操作的结果码。

### 3.14.28 STA-AP 功能使用说明

W58:

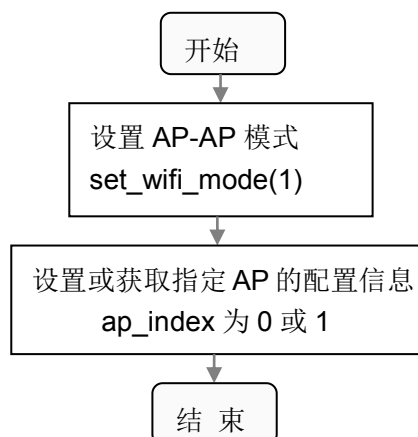


W58L:



### 3.14.29 AP-AP 功能使用说明

AP-AP 功能使用可参考下面的流程图:



### 3.15 SPI

**SIM7600CE-T:** SPI 接口在标准版本上无法使用，需要更新支持 SPI 的版本，可以将 UART1 或者 UART2 配置成 SPI，具体请参考《SIM7600 Open Linux UART&SPI 说明文档》。模块的 SPI 只支持 Master 模式，不支持 Slave 模式，SPI 的版本启动后会生成/dev/spidev2.0 设备节点，默认 SPI 接口只能接单个外设。

**SIM7600CE-AT:** 有专门的 SPI 接口（PIN 脚从 73-76）。模块的 SPI 只支持 Master 模式，不支持 Slave 模式，SPI 的版本启动后会生成/dev/spidev1.0 设备节点，默认 SPI 接口只能接单个外设。

### 3.16 3.17 USB OTG

需要外围电路配合设计，目前仅支持 U 盘，二次开发环境默认支持。

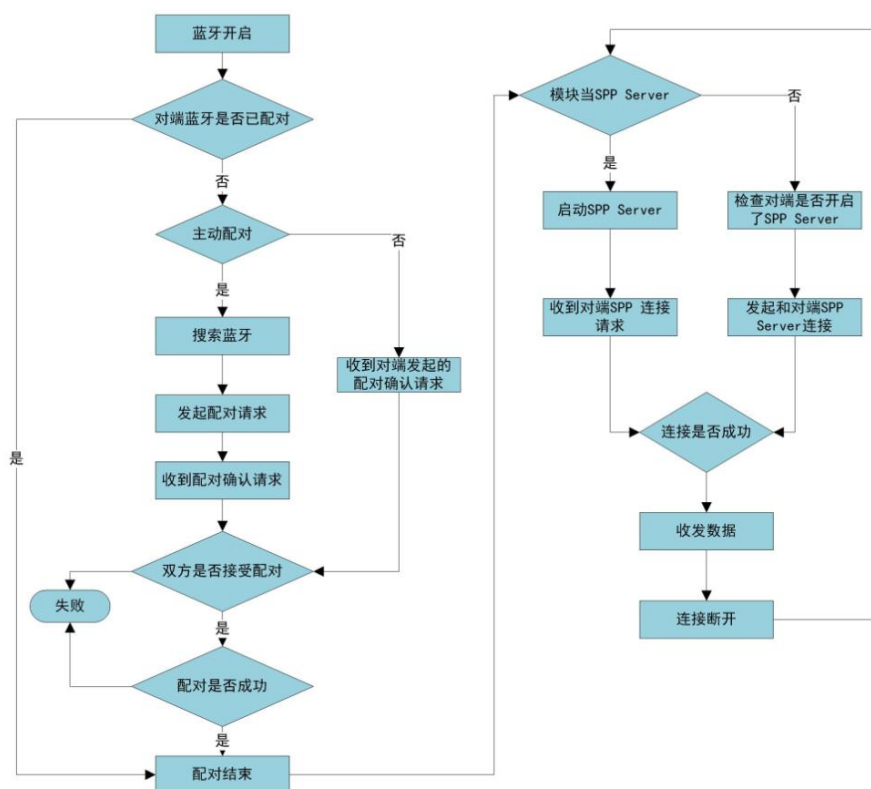
### 3.17 蓝牙

推荐客户选择 W58 模块，封装了高通 QCA9377 的蓝牙芯片。

本章介绍如何通过调用蓝牙的 API，实现 SPP 数据收发，GATT 数据传送等。

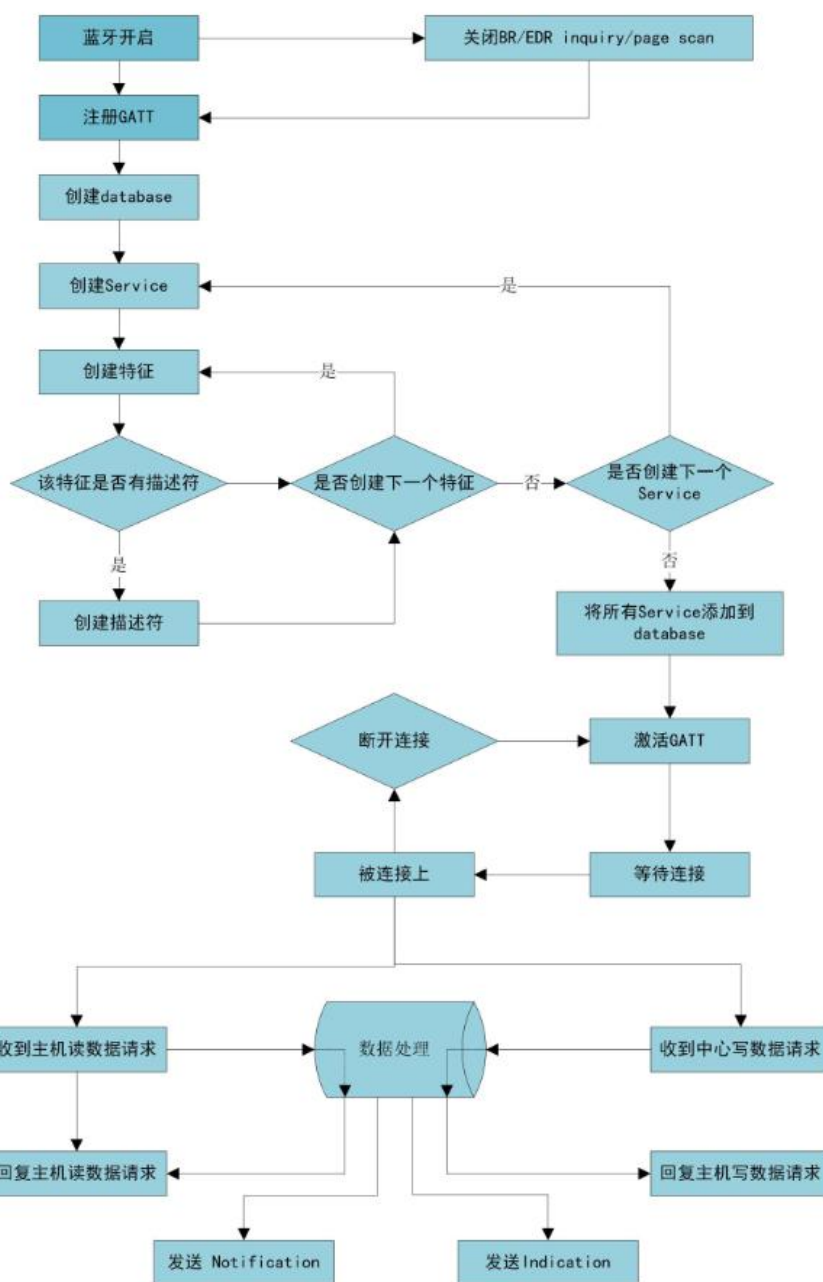
SPP 数据收发数据，首先开启蓝牙，如果和对端蓝牙没有配对，需要先配对。配对结束后，启动 SPP 服务器供对端蓝牙连接，或者直接连接对端蓝牙 SPP server

连接结束后可以进行数据传送



蓝牙 SPP 配对及 SPP 数据收发流程图。





GATT 数据收发流程图

下面是 SPP 功能中相关的 API 接口：

### 3.17.1 蓝牙接口初始化

接口	int Init(bt_ind_cb_fcn handle);
输入	typedef void (*bt_ind_cb_fcn)(void *pData); 回调函数
输出	无

返回值	0: 成功    -1: 失败
NOTE	

### 3.17.2 蓝牙开关

接口	int simcom_bt_power_on(int bPower);
输入	bPower        0: 关闭        1: 打开
输出	无
返回值	0: 成功    -1: 失败
NOTE	

### 3.17.3 获取已配对列表

接口	int get_bonded_device(void);
输入	无
输出	在回调函数以消息 BT_BONDED_COMMAND 返回已配对列表 消息具体参数参考回调函数说明
返回值	0: 成功    -1: 失败
NOTE	

### 3.17.4 搜索蓝牙

接口	int search_devices(int subcmd, int mode, int timeout);
输入	无
输出	搜索列表在回调函数中以消息 BT_SEARCH_COMMAND 返回 消息具体参数参考回调函数说明
返回值	0: 成功    -1: 失败
NOTE	

### 3.17.5 蓝牙配对

接口	int search_devices(int subcmd, int mode, int timeout);
输入	index:    搜索蓝牙列表中的索引号
输出	在回调函数以消息 BT_ACCEPT_COMMAND 返回配对确认通知

	或者以消息 BT_BOND_COMMAND 返回配对结果 消息具体参数参考回调函数说明
返回值	0: 成功     -1: 失败
NOTE	

### 3.17.6 配对确认请求

接口	Init 注册的回调函数
输入	无
输出	上报 BT_ACCEPT_COMMAND 消息，消息具体参数参考回调函数说明
返回值	
NOTE	

### 3.17.7 接受配对

接口	int accept_bond(int isAccept, accept_mode mode);
输入	isAccept: 是否接收配对 mode: 配对模式 ACCEPT_MODE_NONE = 0 不接受的时候模式可以忽略. ACCEPT_MODE_COMPARE = 1 对比配对密码 ACCEPT_MODE_PASSKEY = 2 需要输入密码 ACCEPT_MODE_REBOND = 3 重新配对 ACCEPT_MODE_NOTIFICATION = 4 仅通知，不需要回复 ACCEPT_MODE_JUSTWORK = 5 直接配对 ACCEPT_MODE_PINCODE = 6 PIN 码配对模式
输出	在回调函数以消息 BT_BOND_COMMAND 返回配对配对结果 消息具体参数参考回调函数说明
返回值	0: 成功     -1: 失败
NOTE	此接口在收到回调函数 BT_ACCEPT_COMMAND 消息的时候去执行

### 3.17.8 配对结果

接口	Init 注册的回调函数
输入	无
输出	上报 BT_BOND_COMMAND 消息，消息具体参数参考回调函数说明
返回值	
NOTE	

### 3.17.9 开启 SPP Server

接口	int spp_server(int bActive);
输入	bActive: 1: 开启 0: 关闭
输出	返回值
返回值	0: 成功 -1: 失败
NOTE	通过返回值判断是否开启或关闭成功

#### 3.17.10 查询对端蓝牙是否开启 SPP Server

接口	int bt_get_profile(int index, int *isSupportSPP);
输入	index: 已配对列表索引号
输出	isSupportSPP: 1: 已开启 0: 未开启
返回值	0: 成功 -1: 失败
NOTE	

#### 3.17.11 发起 SPP 连接

接口	int spp_conncet_device(int index);
输入	index: 已配对列表索引
输出	返回值
返回值	0: 成功 -1: 失败
NOTE	通过返回值判断是否连接成功

#### 3.17.12 断开 SPP 连接

接口	int spp_disconncet_device(void);
输入	无
输出	返回值
返回值	0: 成功 -1: 失败
NOTE	通过返回值判断是否断开成功

### 3.17.13 非主动连接 SPP

接口	Init 注册的回调函数
输入	无
输出	上报 BT_CONNECT_COMMAND 消息，消息具体参数参考回调函数说明
返回值	
NOTE	

### 3.17.14 非主动断开 SPP 连接

接口	Init 注册的回调函数
输入	无
输出	上报 BT_DISCONNECT_COMMAND 消息，消息具体参数参考回调函数说明
返回值	
NOTE	

### 3.17.15 查询本地 SPP Server 状态

接口	int bt_get_server_status(int *isActive);
输入	无
输出	isActive      0: 关闭      1: 开启
返回值	0: 成功      -1: 失败
NOTE	

### 3.17.16 查询 SPP 连接状态

接口	int bt_get_connect_status(int *isConnected);
输入	无
输出	isConnected:      0: 未连接      1: 连接
返回值	0: 成功      -1: 失败
NOTE	

### 3.17.17 SPP 发送数据

接口	int spp_send_data(uint8_t *data, uint16_t data_len);	
输入	Data	数据
	data_len	数据长度
输出	返回值	
返回值	0: 成功	-1: 失败
NOTE		

### 3.17.18 SPP 接收数据

接口	Init 注册的回调函数	
输入	无	
输出	上报 BT_SPP_RECV_COMMAND 消息，消息具体参数参考回调函数说明	
返回值		
NOTE		

### 3.17.19 查询本地蓝牙名称和 MAC 地址

接口	int host_device(char *device_name, SIMCOM_BT_ADDR_T *paddr);	
输入	无	
输出	device_name	蓝牙名称
	paddr	MAC 地址
返回值	0: 成功	-1: 失败
NOTE	此接口是重载函数，参数如本节所示的时候是查询。如果只有一个 char * 参数的时候是设置蓝牙地址	

### 3.17.20 设置本地蓝牙名称

接口	int host_device(char *device_name);	
输入	device_name	蓝牙名称
输出	无	
返回值	0: 成功	-1: 失败
NOTE	此接口是重载函数，参数如本节所示的时候是设置。	

### 3.17.21 设置 PIN 码

接口	int pin_set(char *pincode);
输入	pincode: pin 码 4~16 位阿拉伯数字字符串
输出	无
返回值	0: 成功 -1: 失败
NOTE	pin 码是相对不安全的模式，容易可破解，不建议使用。默认关闭了这种模式。如果客户有特别的需求，请参考 demo 启动部分源码，修改蓝牙启动参数。

### 3.17.22 读取 PIN 码

接口	int pin_read(char *pincode);
输入	无
输出	pincode
返回值	0: 成功 -1: 失败
NOTE	

### 3.17.23 设置 BR/EDR inquiry scan 和 page scan

接口	int bt_set_scan_enable(int inquiry_scan_status, int page_scan_status);
输入	inquiry_scan_status 0: 关闭 1: 打开 page_scan_status 0: 关闭 1: 打开
输出	无
返回值	0: 成功 -1: 失败
NOTE	inquiry_scan: 如果关闭，别的蓝牙设备将无法发现本蓝牙 page_scan: 如果关闭，别的蓝牙将无法连接本蓝牙

### 3.17.24 查询 BR/EDR inquiry scan 和 page scan 设定

接口	int bt_set_scan_enable(int inquiry_scan_status, int page_scan_status);
输入	无
输出	inquiry_scan_status 0: 关闭 1: 打开 page_scan_status 0: 关闭 1: 打开
返回值	0: 成功 -1: 失败
NOTE	inquiry_scan: 如果关闭，别的蓝牙设备将无法发现本蓝牙 page_scan: 如果关闭，别的蓝牙将无法连接本蓝牙

### 3.17.25 GATT 注册

接口	int bt_gatt_register(int bRegister);
输入	bRegister: 1: 注册 0: 取消注册
输出	无
返回值	0: 成功 -1: 失败
NOTE	

### 3.17.26 创建 database

接口	int bt_gatt_createdatabase(int bCreateDb);
输入	bCreateDb 1: 创建 0: 删除
输出	无
返回值	0: 成功 -1: 失败
NOTE	

### 3.17.27 创建 16 位 UUID 服务

接口	int bt_gatt_create_service(int uuid16);
输入	uuid16 16bit UUID
输出	无
返回值	0: 成功 -1: 失败
NOTE	此接口是重载接口

### 3.17.28 创建 128 位 UUID 服务

接口	int bt_gatt_create_service(unsigned char *uuid128);
输入	uuid128 128bit UUID
输出	无
返回值	0: 成功 -1: 失败
NOTE	此接口是重载接口



### 3.17.29 创建 16 位特征

接口	int bt_gatt_create_characteristic(int uuid16, int pro, int permission, uint32_t *attrHandle);	
输入	uuid16	16 位 UUID
	pro	property
	permission	权限
输出	无	
返回值	0: 成功    -1: 失败	
NOTE	此接口是重载接口	

### 3.17.30 创建 128 位特征

接口	int bt_gatt_create_characteristic(unsigned char *uuid128, int pro, int permission, uint32_t *attrHandle);	
输入	uuid128	128 位 UUID
	pro	property
	permission	权限
输出	无	
返回值	0: 成功    -1: 失败	
NOTE	此接口是重载接口	

### 3.17.31 创建描述符

接口	int bt_gatt_create_descriptor();	
输入	无	
输出	无	
返回值	0: 成功    -1: 失败	
NOTE	此接口目前还为完善配置，暂时用默认配置	

### 3.17.32 将创建的服务添加到数据库

接口	int bt_gatt_add_service_2_db();	
输入	无	
输出	无	
返回值	0: 成功    -1: 失败	

## NOTE

### 3.17.33 发送 notification

接口	int bt_gatt_notification(uint16_t attrhandle, char *data, uint32_t data_len);
输入	attrhandle 属性句柄 data 数据 data_len 数据长度
输出	无
返回值	0: 成功    -1: 失败
NOTE	

### 3.17.34 发送 indication

接口	int bt_gatt_indication (uint16_t attrhandle, char *data, uint32_t data_len);
输入	attrhandle 属性句柄 data 数据 data_len 数据长度
输出	无
返回值	0: 成功    -1: 失败
NOTE	

### 3.17.35 返回主机从本地读数据请求

接口	int bt_gatt_read_cfm(uint16_t attrHandle, int rspCode, uint8_t *data, uint32_t data_len);
输入	attrhandle 属性句柄 rspCode 错误码 data 数据 data_len 数据长度
输出	无
返回值	0: 成功    -1: 失败
NOTE	

### 3.17.36 返回主机从本地写数据请求

接口	int bt_gatt_write_cfm(uint16_t attrHandle, int rspCode);
输入	attrhandle 属性句柄 rspCode 错误码
输出	无
返回值	0: 成功    -1: 失败

### 3.17.37 回调函数

```
static void simcom_process_response(bt_msg *recv_msg)
{
    switch(recv_msg->command)
    {
        case BT_SEARCH_COMMAND:
            扫描结果输出
            param1: search status: search end
            param2: search result index
            param3: rssi;
            address: device mac address
            data_len: data name length
            data: device name
            break;
        case BT_BOND_COMMAND:
            配对返回结果
            param1: bond result 0: FAILED 1:SUCCESS
            address: device mac address
            data_len device name length
            data: if data_len > 0 deviceName
            break;
        case BT_ACCEPT_COMMAND:
            提示用户确认配对
            param1: accept mode
            param2: if ((param1 == ACCEPT_MODE_NOTIFICATION) ||
                    (param1 == ACCEPT_MODE_COMPARE) ||
                    (param1 == ACCEPT_MODE_PASSKEY))

            param2 is passkey value
            else
            ignore
            address: device address
            data_len: data length
            data: device name (maybe null)
```

```
break;
case BT_GET_BONDED_COMMAND:
    获取已配对设备列表
    param1 = total bonded number
    param2 = current index;
    param3 : if total bonded unnumber more than 0, 1: last one
    address = devices;          break;

case BT_CONNECT_COMMAND:
    其他蓝牙设备通过 SPP 连接模块，上报此消息。
break;

case BT_DISCONNECT_COMMAND:
    非模块主动断开 SPP 连接，上报此消息。
break;

case BT_SPP_RECV_COMMAND:
    SPP 收数据
    data_len: receive data len
    data:      receive data
    break;

case BT_GATT_CONNECT_COMMAND:
    GATT 连接上，上报此消息。
break;
case BT_GATT_READ_IND_COMMAND:
    其他设备试图通过蓝牙读模块数据，上报此消息。
break;
case BT_GATT_WRITE_IND_COMMAND:
    其他设备试图通过蓝牙写模块数据，上报此消息。
break;
    }
}
```

## 3.18 ETH

### 3.18.1 网卡模式设置

以太网分三种模式:

0: ETH\_LAN 外设通过模块上网，外设 IP 通过模块 HDCP 分配。

- 1: ETH\_WAN 模块通过以太网上网，
- 2: ETH\_LAN\_STATIC 外设通过模块上网，模块 ETH 和外设 IP 设置为静态 IP

一般来说，以太网的功能是设计的时候预定义好了。所以可以在编译二次开发环境前先设定好模式。修改二次开发环境源码 `data/lan_mode` 中的值，即可修改预设模式。模式所对应的值就是上述列表数值。

客户可以参考 2.6 章节的文件保护，在 `/cache` 下进行备份，若 `/data` 下不存在 `lan_mode`，则默认是 ETH\_LAN 模式。

### 3.18.2 网卡型号选择

接口	Init(ethernet_type_info type)
输入	type
	1 BCM898XX
	2 AT803X
输出	无
返回值	0: 成功 -1: 失败

### 3.18.3 驱动加载

接口	int Install_driver()
输入	无
输出	无
返回值	0: 成功 -1: 失败

### 3.18.4 驱动卸载

接口	int Uninstall_driver()
输入	无
输出	无
返回值	0: 成功 -1: 失败
NOTE	如果休眠的时候需要卸载以太网驱动

### 3.18.5 从 NV 中读取预设 MAC 地址

接口	int get_mac_address_from_nv(dms_device_mac_enum_v01 device_type, uint8_t *mac_addr);
输入	DMS_DEVICE_MAC_WLAN_V01 = 0 ---- WIFI MAC 地址 DMS_DEVICE_MAC_BT_V01 = 1 ---- 蓝牙 MAC 地址 DMS_DEVICE_MAC_LAN_V01 = 2 ---- ETH MAC 地址
输出	无
返回值	0: 成功 -1: 失败
NOTE	

### 3.18.6 设置 MAC 地址

接口	int set_eth_mac_address(char *mac_address)
输入	mac_address: MAC 地址字符串 如: 64:00:6A:04:65:A1
输出	无
返回值	0: 成功 -1: 失败
NOTE	如果 NV 中未读取到 MAC 地址, 可以设置一个随机的

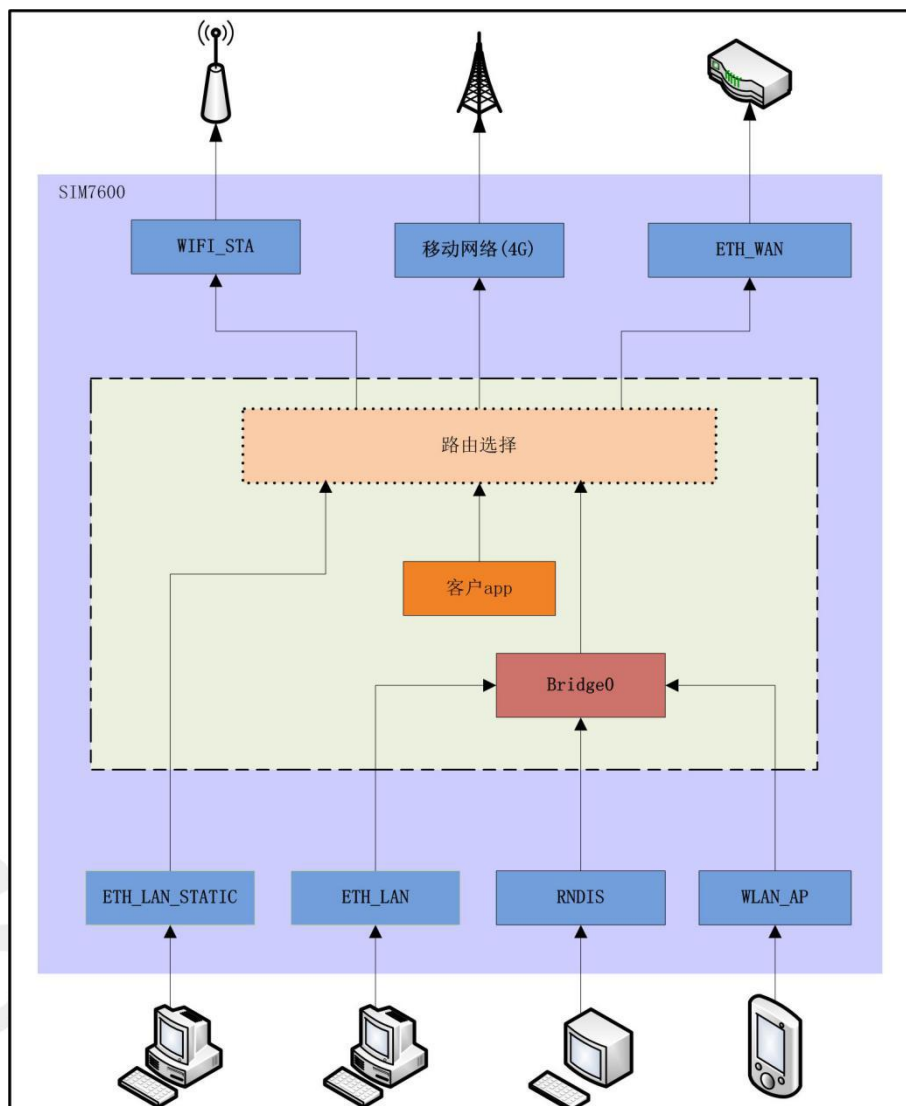
### 3.18.7 设置 IP

接口	int set_eth_ip_address(char *ip_address)
输入	ip_address: IP 地址: 192.168.*.*
输出	无
返回值	无
NOTE	模块睡眠需要卸载 ETH 驱动

## 3.19 网络设置

本节主要介绍模块支持的网络访问方式和网络设置

## 3.20 网络访问方式



模块访问外部网络方式主要有 3 种：

1. 移动网络
2. WIFI\_STA                      WIFI 开启 STA
- ETH\_WAN 以太网当 WAN 口

外部设备连接模块方式有 4 种

1. WIFI\_AP
2. RNDIS
3. ETH\_LAN 以太网连接到模块内部 bridge0 上，模块给设备自动分配 IP
4. ETH\_LAN\_STATIC 模块以太网和主控设备上的以太网口都设置静态 IP

第 1,2,3 中模式：主控设备和模块连接是以局域网的形式连接。 会从模块 HDCP 得到一个 192.168.225.\* 的 IP。IP 获取成功后，主控可以通过模块连接外网，也可以和模块内部进程（APP）通过局域网 IP 进行互访。

第 4 种模式：需要由客户 APP 和主控自行设置 IP,网关，NDS.

IP:                      192.168.\*.\* (和 APP 给 ETH\_LAN\_STATIC 设置的 IP 同样网段)

网关:                      192.168.\*.\* (和 APP 给 ETH\_LAN\_STATIC 设置的 IP 相同)

DNS 服务器： 192.168.225.1 （固定）

### 3.20.1 默认路由优先级预置

修改如下两个文件中的节点：sim\_open\_sdk/sim\_usrfs/mobileap\_cfg.xml 和 sim\_open\_sdk/sim\_rootfs/etc/default\_mobileap\_cfg.xml

```
<FirstPreferredBackhaul>usb_cradle</FirstPreferredBackhaul>
<SecondPreferredBackhaul>wlan</SecondPreferredBackhaul>
<ThirdPreferredBackhaul>wwan</ThirdPreferredBackhaul>
usb_cradle:    以太网(ETH_WAN)
wlan: WLAN_STA
wwan:          移动网络
```

模块根据当前模块状态和配置文件设置，选择一个做默认路由。

比如：

在当前设置下，模块是有 WLAN\_STA 和 ETH\_WAN 都连接的时候，默认路由将设置为 ETH\_WAN. 就是说访问外网会通过 ETH 访问。

设置不变，如果以太网线拔掉，默认路由将变为 WLAN. 插入以太网线，待以太网络连接成功后，默认路由将切换为 ETH\_WAN.

客户如果希望 WLAN\_STA 和 ETH\_WAN 都连接的时候优先使用 ETH\_WAN, 可以调整他们两之间的顺序。同理如果客户希望 4G 网络拨号成功的时候都通过 4G 网络，那么把 4G 的优先级跳到最高即可。

## 3.21 ALSA

### 3.21.1 设置内部扬声器的音量输出

接口	int set_clvl_value(int clvl_value)
输入	clvl_value: 音量值 (0-5)
输出	无
返回值	0: 成功 -1: 失败
NOTE	



### 3.21.2 获取内部扬声器的音量

接口	<b>int get_clvl_value(void)</b>
输入	无
输出	无
返回值	失败返回-1. 成功时返回音量值
NOTE	

### 3.21.3 设置 mic 增益

接口	<b>int set_micgain_value(int micgain_value)</b>
输入	<b>micgain_value:</b> 需要设置的增益值(0-8)
输出	无
返回值	0: 成功 -1: 失败
NOTE	

### 3.21.4 获取 mic 增益

接口	<b>int get_micgain_value(void)</b>
输入	无
输出	无
返回值	失败返回-1. 成功时返回 mic 增益值
NOTE	

### 3.21.5 切换语音通道

接口	<b>int set_csdvc_value(int csdvc_value)</b>
输入	<b>csdvc_value</b> 1: 听筒 3: 扬声器
输出	无
返回值	0: 成功 -1: 失败
NOTE	

### 3.21.6 查询当前语音通道

接口	<code>int get_csdvc_value(void)</code>
输入	无
输出	无
返回值	失败时返回-1，成功时返回代表当前语音通道的整形值
NOTE	

### 3.21.7 启动录音

接口	<code>Int simcom_audio_start_record(simcom_audio_record_type_e record_type, char *record_path);</code>
输入	record_type: 1: local 本地录音 2: remote 远端录音（通话时才支持） 3: local and remote 本地和远端同时录音 record_path:录音文件存储路径
输出	无
返回值	0: 成功 -1: 失败
NOTE	record_path 需要指定读写分区，如/data /cache

### 3.21.8 停止录音

接口	<code>int simcom_audio_stop_record();</code>
输入	无
输出	无
返回值	0: 成功 -1: 失败
NOTE	

### 3.21.9 查询录音状态

接口	<code>int simcom_audio_get_record_state();</code>
----	---

输入	无
输出	无
返回值	失败时返回-1，成功时返回 0：当前无录音， 1：正在录音
NOTE	

## 3.22 设备控制

### 3.22.1 进入 recovery 模式

接口	<b>int exec_cdeltcmd(const char *path)</b>
输入	path 指定升级包的路径， 当 path 为 NULL 时使用默认路径/cache/update_ota.zip
输出	无
返回值	0：成功 -1：失败
NOTE	

### 3.22.2 adb 设置

接口	<b>int exec_cusbadb_cmd(bool value)</b>
输入	value 1：打开 adb 口 0：关闭 adb 口
输出	无
返回值	0：成功 -1：失败
NOTE	

## 3.23 DMS

### 3.23.1 初始化

接口	<b>int dms_init()</b>
输入	无
输出	无
返回值	0: 成功 -1: 失败
接口	<b>int dms_init()</b>

### 3.23.2 获取 imei

接口	<b>int get_imei(char *plmei)</b>
输入	<b>plmei</b> : 存储 imei 的 buf 的指针
输出	无
返回值	0: 成功 -1: 失败
NOTE	

### 3.23.3 获取 meid

接口	<b>int get_meid(char *pMeid);</b>
输入	<b>pMeid</b> : 存储 meid 的 buf 的指针
输出	无
返回值	0: 成功 -1: 失败
NOTE	

### 3.23.4 获取固件版本识别码

接口	<b>int get_rev_id(char *pRev_id)</b>
输入	<b>pRev_id</b> : 存储 Rev_id 的 buf 的指针
输出	无
返回值	0: 成功 -1: 失败
NOTE	

### 3.23.5 设置 UE 工作模式

接口	<b>int dms_set_operating_mode(unsigned char mode)</b>
输入	mode: 0 Online 1 Low power 2 Factory Test mode 3 Offline 4 Resetting 5 Shutting down 6 Persistent low power 7 Mode-only low power
输出	无
返回值	0: 成功 -1: 失败
NOTE	

### 3.23.6 释放

接口	<b>void dms_deinit()</b>
输入	无
输出	无
返回值	无
NOTE	

## 4.客户版本维护

客户的版本信息保存在/etc/simcom\_ap\_ver.ini 中，版本信息的最大长度为 35 个字节。可用通过 cat 查看文件内容，也可通过 AT+CSUB=1 来查询版本信息。

SIMCom  
Confidential