A76xx Series Open SDK_ BLE_应用指导

LTE 模组

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
技术支持邮箱：support@simcom.com
官网：www.simcom.com

名称：	A76xx Series Open SDK_BLE_应用指导
版本：	V1.01
类别：	应用文档
状态：	已发布

版权声明

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
邮箱：simcom@simcom.com
官网：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

Version History

Version	Date	Owner	What is new
V1.00	2022-11-17		第一版
V1.01	2023-09-06		新增接口 sAPI_BleSetGapName sAPI_BleSetAppearance sAPI_BleSetPairEnable sAPI_BleGetPairInfo sAPI_BleReadRssi sAPI_BleGetDeviceName sAPI_BleSetDeviceName

About this Document

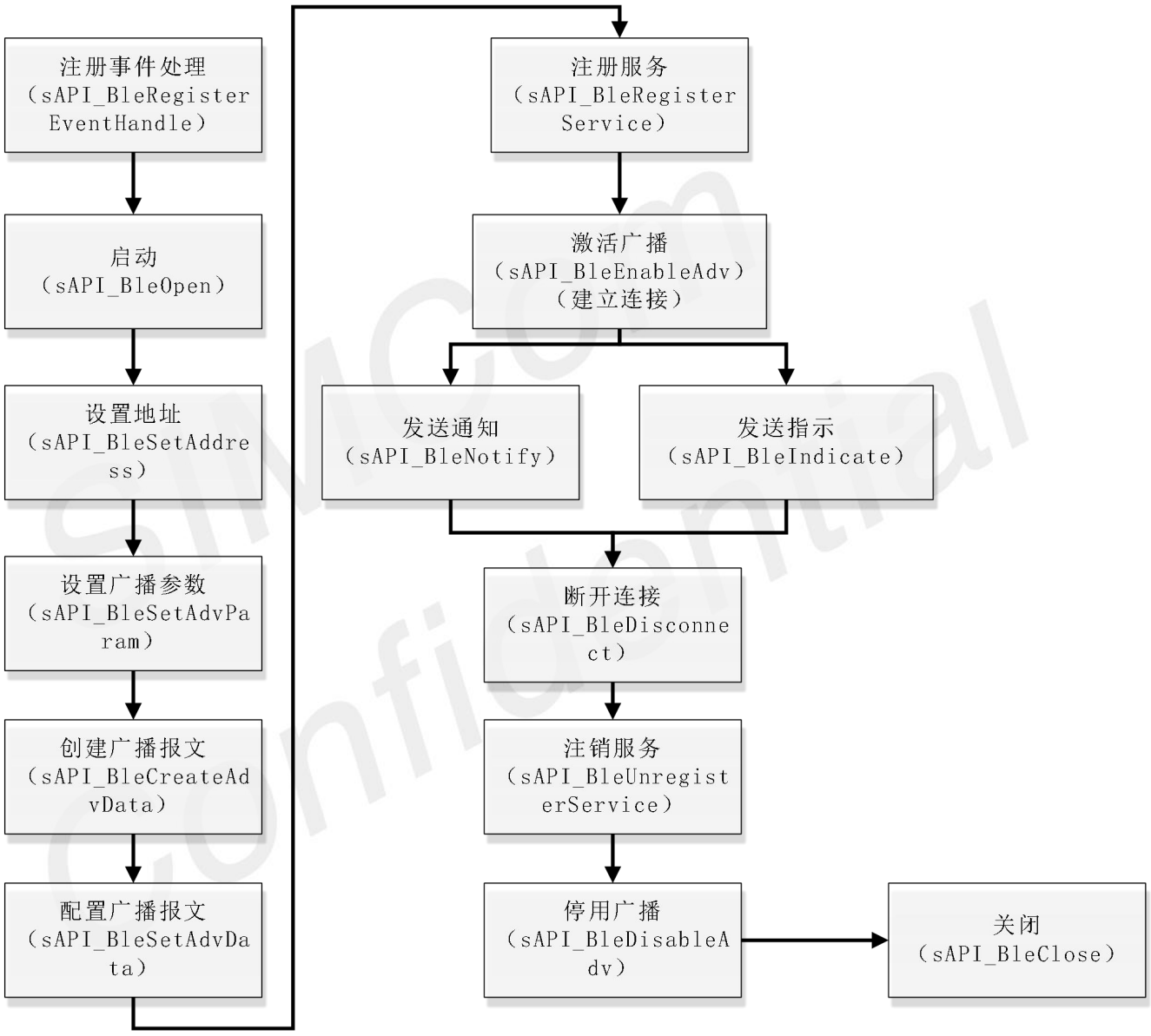
本文档适用于 A1603 open 系列、A1606 open 系列。

目录

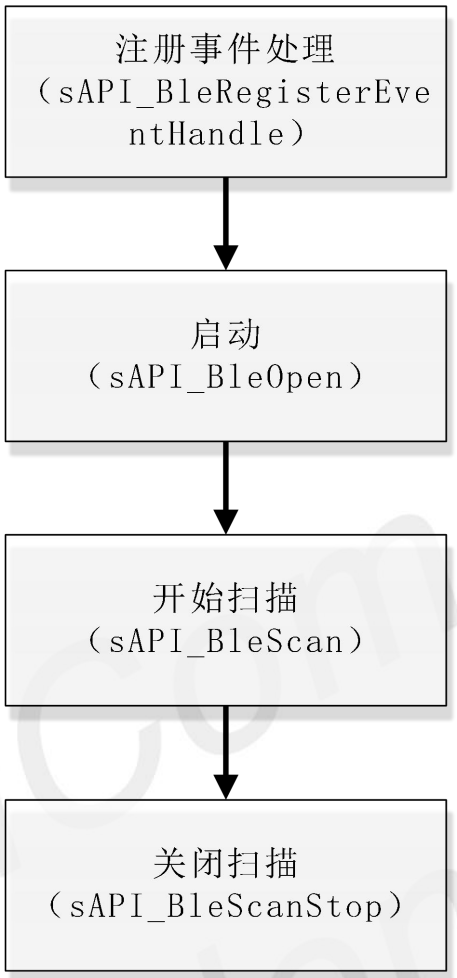
缩略语

BLE	Bluetooth low energy
PUD	protocol data unit

1BLE 使用流程



外围角色执行程序流程图



外围角色扫描过程流程图

在 BLE 中存在两个角色，一个是中心角色，一个是外围角色，蓝牙设备都可以独立作为中心角色或外围角色。外设角色的作用是为中心角色提供各种数据，中心角色可以扫描并接收多个外设角色数据（ 外围角色中的设备进行广播，中心角色的设备扫描寻找广播）。

在 BLE demo 中主要实现了蓝牙事件处理程序的注册/注销，BLE 功能的启动/停止，广播地址配置，广播参数设置，广播报文创建，广播报文配置，服务注册/注销，广播激活/停用以及通知或指示的发送。当程序流程进行到激活广播时，标志着连接建立。

通知和指示在作用上很相似，都是在客户端请求后，服务器发相应的数据给客户端。不同的地方在协议层，通知是没有安全性质的发送，但是指示是有应答机制的。简言之，通知无应答，速度快；指示有应答，更安全。

2API 介绍

BLE 头文件名：simcom_ble.h

2.1 sAPI_BleOpen

该接口用于打开 ble 设备。

接口:	int **sAPI_BleOpen**(sMsgQRef msgQ, int flag);
参数:	[in] [msgQ]：接收函数结果的消息队列。 [in] [flag] ：异步信号。如果 flag=1，函数结果将通过消息队列传递。如果 flag=0，则函数结果通过返回值传递。
返回值:	0 表示成功，其他值都为错误。
NOTE:	无

2.2 sAPI_BleClose

该接口用于关闭 ble 设备。

接口:	void **sAPI_BleClose**(void);
参数:	无
返回值:	无
NOTE:	无

2.3 sAPI_BleRegisterEventHandle

该接口用于注册蓝牙 LE 事件处理程序，仅处理 LE 相关事件。

接口:	int **sAPI_BleRegisterEventHandle**(SC_BLE_EVENT_HANDLE_T handle);
参数:	[in] [handle] BLE 事件处理程序
返回值:	0 表示成功，其他值都为错误。
NOTE:	无

2.4 sAPI_BleCreateRandomAddress

该接口用于为 ble 设备创建随机地址。

接口:	void **sAPI_BleCreateRandomAddress**(SC_BLE_ADDR_T *address);
参数:	[out] [address] BLE 地址
返回值:	无
NOTE:	无

2.5 sAPI_BleSetAddress

配置 ble 设备的广播地址。

接口:	int **sAPI_BleSetAddress**(SC_BLE_ADDR_T *address);
参数:	[in] [address] BLE 设备地址
返回值:	0 表示成功，其他值都为错误。
NOTE:	

2.6 sAPI_BleCreateAdvData

该接口用于为 ble 设备创建广播报文。

接口:	int **sAPI_BleCreateAdvData**(int dataType, void *advData, int advDataSize, const void *data, int length);
参数:	[in] [dataType] BLE 广播数据包数据类型 [out] [advData] BLE 广播包 [in] [advDataSize] BLE 广播包大小 [in] [data] BLE 广播包数据 [in] [length] BLE 广播包数据长度
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.7 sAPI_BleSetAdvData

该接口用于为 ble 设备配置广播报文。

接口:	int **sAPI_BleSetAdvData**(const void *advData, int length);
参数:	[in] [advData] BLE 广播包 [in] [length] BLE 广播包数据长度
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.8 sAPI_BleSetAdvParam

该接口用于广播参数设置。

接口:	int **sAPI_BleSetAdvParam**(SC_BLE_ADV_PARAM_T *param);
参数:	[in] [param] BLE 广播参数
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.9 sAPI_BleRegisterService

该接口用于注册 gatt 服务。

接口:	int **sAPI_BleRegisterService**(SC_BLE_SERVICE_T *service, int length);
参数:	[in] [service] BLE 广播服务 [in] [length] BLE 广播数据长度
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.10 sAPI_BleUnregisterService

该接口用于销毁 ble 设备的注册服务。

接口:	int **sAPI_BleUnregisterService**(void);
参数:	无
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.11 sAPI_BleEnableAdv

该接口用于启动 ble 广播。

接口:	int **sAPI_BleEnableAdv**(void);
参数:	无
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.12 sAPI_BleDisableAdv

该接口用于停止广播。

接口:	int **sAPI_BleDisableAdv**(void);
参数:	无
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.13 sAPI_BleIndicate

该接口用于发送一个指示。

接口:	int **sAPI_BleIndicate**(unsigned short att_handle, const void *data, int size);
参数:	[in] [att_handle]特征处理 [in] [data] 等待发送的数据 [in] [size] 数据的长度
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.14 sAPI_BleNotify

该接口用于发送一个通知。

接口:	int **sAPI_BleNotify**(unsigned short att_handle, const void *data, int size);
参数:	[in] [att_handle]特征处理 [in] [data] 等待发送的数据 [in] [size] 数据的长度
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.15 sAPI_BleScan

该接口用于扫描周围 BLE 设备。

接口:	int **sAPI_BleScan**(unsigned char type, unsigned short interval, unsigned short window, unsigned char own_address_type);
参数:	[in] [type] scan type。 例如: LE_ACTIVE_SCAN, LE_PASSIVE_SCAN [in] [interval] 扫描间隔. 范围: 0x0004-0x4000 [in] [window] 扫描窗口. 范围: 0x0004-0x4000 [in] [own_address_type] 扫描地址类型. 有公共地址(LE_ADDRESS_TYPE_PUBLIC)和随机地址

	(LE_ADDRESS_TYPE_RANDOM)两种
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.16 sAPI_BleConnect

该接口用于连接远端 ble 设备。

接口:	int **sAPI_BleConnect**(SC_BLE_ADDR_T *addr, int type);
参数:	[in] [addr] 远端 BLE 设备地址 [in] [type] address type.
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.17 sAPI_BleDisconnect

该接口用于断开所有连接。

接口:	int **sAPI_BleDisconnect**(void);
参数:	无
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.18 sAPI_BleMtuRequest

该接口用于发送 mtu 请求。

接口:	int **sAPI_BleMtuRequest**(unsigned short mtu_size);
参数:	[in] [mtu_size] mtu 大小，现在只支持 185。
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.19 sAPI_BleReadByGroupTypeRequest

该接口用于发送按组读取类型请求。

接口:	int **sAPI_BleReadByGroupTypeRequest**(unsigned short start, unsigned short end, SC_UUID_T *attribute_group_type);
参数:	[in] [start]启动句柄 [in] [end]结束句柄 [in] [attribute_group_type]属性组类型

返回值:	0 表示成功，其他值都为错误
NOTE:	

2.20 sAPI_BleReadByTypeRequest

该接口用于发送按类型读取请求。。

接口:	int **sAPI_BleReadByTypeRequest**(unsigned short start, unsigned short end, SC_UUID_T *attribute_type);
参数:	[in] [start]启动句柄 [in] [end]结束句柄 [in] [attribute_type]属性类型
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.21 sAPI_BleFindInformationRequest

该接口用于发送查找信息请求。

接口:	int **sAPI_FindInformationRequest**(unsigned short start, unsigned short end);
参数:	[in] [start]启动句柄 [in] [end]结束句柄
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.22 sAPI_BleReadRequest

该接口用于发送读请求。

接口:	int **sAPI_BleReadRequest**(unsigned short handle);
参数:	[in] [handle]值句柄
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.23 sAPI_BleWriteRequest

该接口用于发送写请求。

接口:	int **sAPI_BleWriteRequest**(unsigned short handle, const void *data, unsigned short size);

参数:	[in] [handle]值句柄 [in] [data]待写入的数据 [in] [size]数据长度
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.24 sAPI_BleWriteCommand

该接口用于发送写命令。

接口:	Int **sAPI_BleWriteCommand**(unsigned short handle, const void *data, unsigned short size);
参数:	[in] [handle]值句柄。 [in] [data]待写入的数据。 [in] [size]数据长度
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.25 sAPI_BleScanStop

该接口用于停止扫描。

接口:	int sAPI_BleScanStop(void);
参数:	无
返回值:	0 表示成功，其他值都为错误
NOTE:	

2.26 sAPI_BleRegisterEventHandleEx

该接口用于注册蓝牙事件处理程序，可以处理所有蓝牙事件。

接口:	int **sAPI_BleRegisterEventHandleEx**(void (*function_cb)(void *msg));
参数:	[in] [function] 蓝牙事件处理程序
返回值:	0 表示成功，其他值都为错误。
NOTE:	无

2.27 sAPI_BleSetGapName

该接口用于设置 LE GAP 名称，如果 client 是 IOS 设备，可以使用该 api 设置设备名称与广播数据中的 name 相同，即可使 IOS client 多扫描连接为同一名称设备。

接口:	int **sAPI_BleSetGapName**(char *name);
参数:	[in] [handle] 蓝牙事件处理程序
返回值:	0 表示成功，其他值都为错误。
NOTE:	无

2.28 sAPI_BleSetGapAppearance

该接口用于设置蓝牙图标处理程序，适用于带 UI 的产品。

接口:	int **sAPI_BleSetGapAppearance**(UINT16 appearance);
参数:	[in] [appearance] 蓝牙协议定义中枚举
返回值:	0 表示成功，其他值都为错误。
NOTE:	无

2.29 sAPI_BleSetPairEnable

该接口用于设置默认接受或者拒绝配对请求，只能在发起配对之前进行设置。

接口:	int **sAPI_BleSetPairEnable**(unsigned char enable);
参数:	[in] [enable] 0:默认拒绝配对请求，1:默认接受配对请求
返回值:	0 表示成功，其他值都为错误。
NOTE:	无

2.30 sAPI_ BleGetPairInfo

该接口用于获取 LE 配对信息。

接口:	int **sAPI_BleGetPairInfo**(int index,SC_BLE_DEVICE_INFO_RECORD *record);
参数:	[in] [index] 配对列表 index [out][record] 配对设备信息
返回值:	0 表示成功，其他值都为错误。
NOTE:	无

2.31 sAPI_BleDeleteSinglePairInfo

该接口用于删除 LE 配对列表中指定 index 配对信息。

接口:	int **sAPI_BleDeleteSinglePairInfo**(int index);

参数:	[in] [index] 配对列表 index
返回值:	0 表示成功，其他值都为错误。
NOTE:	无

2.32 sAPI_BleClearPairInfo

该接口用于删除 LE 所有配对信息。

接口:	int **sAPI_BleClearPairInfo**(void);
参数:	无
返回值:	0 表示成功，其他值都为错误。
NOTE:	无

2.33 sAPI_BleReadRssi

该接口用于查询连接设备的 RSSI，通过 COMMON_RSSI 事件返回。

接口:	int **sAPI_BleReadRssi**(void);
参数:	无
返回值:	0 表示成功，其他值都为错误。
NOTE:	无

2.34 sAPI_BleGetDeviceName

该接口用于获取设备名称。

接口:	char ***sAPI_BleGetDeviceName**(void);
参数:	无
返回值:	返回设备名称。
NOTE:	无

2.35 sAPI_BleSetDeviceName

该接口用于设置设备名称。

接口:	int **sAPI_BleSetDeviceName**(char *name);
参数:	[in] [name] 设备名称
返回值:	0 表示成功，其他值都为错误。
NOTE:	无

3 错误码信息

<err>	Meaning
0	成功.
1	未知错误
2	状态警报
3	参数错误
4	打开设备错误

4 变量定义

4.1 SC_BLE_ATT_ERROR_RESPONSE_T

```
typedef enum
{
    SC_BLE_ERR_OK = 0,    //OK
    SC_BLE_ERR_INVALID_HANDLE,    //处理无效
    SC_BLE_ERR_READ_NOT_PERMITTED,    //读操作不被允许
    SC_BLE_ERR_WRITE_NOT_PERMITTED,    //写操作不被允许
    SC_BLE_ERR_INVALID_PDU,    //不可用的 PDU
    SC_BLE_ERR_INSUFFICIENT_AUTHEN,    //身份验证不充分
    SC_BLE_ERR_REQUEST_NOT_SUPPORT,    //请求不被允许
    SC_BLE_ERR_INVALID_OFFSET,    //不可用的 OFFSET
    SC_BLE_ERR_INSUFFICIENT_AUTHOR,    //作者不足
    SC_BLE_ERR_PREPARE_QUEUE_FULL,    //准备队列已满
    SC_BLE_ERR_ATTRIBUTE_NOT_FOUND,    //标志未找到
    SC_BLE_ERR_ATTRIBUTE_NOT_LONG,    //标志长度不够
    SC_BLE_ERR_INSUFFICIENT_ENCRY_KEY,    //加密秘钥不足
    SC_BLE_ERR_INVALID_ATTRIBUTE_VALUE,    //不可用的属性值
    SC_BLE_ERR_UNLIKELY_ERROR,    //非典型错误
    SC_BLE_ERR_INSUFFICIENT_ENCRY,    //内存空间不足
    SC_BLE_ERR_UNSUPPORTED_GTOUP_TYPE,    //不支持 GTOUP 类型
    SC_BLE_ERR_INSUFFICIENT_RESOURCES,    //资源不足
} SC_BLE_ATT_ERROR_RESPONSE_T;
```

4.2 SC_BLE_RETURNCODE_T

```
typedef enum
{
    SC_BLE_RETURNCODE_OK,    //OK
    SC_BLE_RETURNCODE_NOT_KNOW_ERROR,    //未知错误
    SC_BLE_RETURNCODE_ALERT,    //警告
    SC_BLE_RETURNCODE_PARAM_ERROR,    //参数错误
    SC_BLE_RETURNCODE_OPEN_FAIL,    //打开失败
    SC_BLE_RETURNCODE_MSGQ_ERROR,    //消息队列错误
} SC_BLE_RETURNCODE_T;
```

4.3 SC_COMMON_EVENT_TYPE_T

```
typedef enum
{     COMMON_INQUIRY_RESULT,    //查询结果
      COMMON_INQUIRY_COMPLETE,    //查询结束
      COMMON_PAIRING_REQUEST,    //请求配对
      COMMON_PAIRED,    //已经配对
      COMMON_PIN_REQUEST,    //PIN 请求
      COMMON_POWERUP_COMPLETE,    //开机完成
      COMMON_POWERUP_FAILED,    //开机失败
      COMMON_SHUTDOWN_COMPLETE,    //关机完成
      COMMON_BTFIRMWARE_ASSERT,    //BT 固件声明
      COMMON_HCI_COMPLETE_EVENT,    //HCI 完成
      COMMON_NULL,    //空
      COMMON_NAME,    //名字
      COMMON_RSSI,    //接收信号强度指示器
      COMMON_SLAVE_LE_BOND_COMPLETE,    //从机建立完成
} SC_COMMON_EVENT_TYPE_T;
```

4.4 SC_BLE_EVENT_TYPE_T

```
typedef enum
{
      BLE_SCAN_EVENT,    //扫描
      BLE_CONNECT_EVENT,    //连接
      BLE_DIS_CONNECT_EVENT,    //断开连接
      BLE_INDICATE_EVENT,    //指示
      BLE_ERROR_RESPONSE_EVENT,    //响应错误
      BLE_MTU_EXCHANGED_EVENT,    //交换 MTU
      BLE_CLIENT_MTU_EXCHANGED_EVENT,    //客户端交换 MTU
      BLE_CLIENT_READ_BY_GROUP_TYPE_RSP_EVENT,    //客户端按组读取响应
      BLE_CLIENT_READ_BY_TYPE_RSP_EVENT,    //客户端按类型读取响应
      BLE_CLIENT_FIND_INFOMATION_RSP_EVENT,    //客户端查找返回信息
      BLE_CLIENT_READ_RSP_EVENT,    //客户端读取 RSP
      BLE_CLIENT_READ_BLOB_RSP,    //客户端读取 BLOB 响应
      BLE_CLIENT_HANDLE_NOTIFY_EVENT,    //客户端处理通知
      BLE_CLIENT_HANDLE_INDIATION_EVENT,    //客户端处理指示
      BLE_WHITE_LIST_SIZE,    //白名单大小
      BLE_SMP_PASSKEY,    //SMP 秘钥
      BLE_ADV_PHY_TXPOWER,    //广播物理层发射功率
      BLE_CLIENT_WRITE_RSP,    //客户端写应答
} SC_BLE_EVENT_TYPE_T;
```

4.5 SC_BLE_EVENT_HANDLE_T

```
typedef struct
{
    unsigned short int event_type;    //事件类型
    unsigned short int event_id;    //事件 ID
    int payload_length;    //有效负载长度
    void *payload;    //有效负载
} SC_BLE_EVENT_MSG_T;

typedef int (*SC_BLE_EVENT_HANDLE_T)(SC_BLE_EVENT_MSG_T *msg);
```

4.6 SC_BLE_ADDR_T

```
typedef struct
{
    unsigned char bytes[6];    //地址字节
} SC_BLE_ADDR_T;
```

4.7 SC_BLE_ADV_PARAM_T

```
typedef struct
{
    unsigned short int interval_min;    //最小时间间隔
    unsigned short int interval_max;    //最大时间间隔
    unsigned char advertising_type;      //广告类型
    unsigned char own_address_type;    //自身地址类型
    unsigned char peer_address_type;    //对端地址类型
    SC_BLE_ADDR_T peer_address;    //对端地址
    unsigned char filter; /* 0x00: process scan and connection request from all devices
                             0x01: process connection request from all devices
                                   and scan request only from White List
                             0x02: process scan request from all devices
                                   and conneciton request only from White List
                             0x03: process scan and connection reques only from in the White List
                          */
} SC_BLE_ADV_PARAM_T;
```

4.8 SC_BLE_ADV_DATATYPE_T

```
typedef enum
{
    BLE_ADV_DATA_TYPE_FLAG = 0x01,
    BLE_ADV_DATA_TYPE_COMPLETE_SERVICE_LIST = 0x03,
    BLE_ADV_DATA_TYPE_SHORT_NAME = 0x08,
    BLE_ADV_DATA_TYPE_COMPLETE_NAME = 0x09,
    BLE_ADV_DATA_TYPE_APPEARANCE = 0x19,
    BLE_ADV_DATA_TYPE_SPECIFIC_DATA = 0xff,
} SC_BLE_ADV_DATATYPE_T;
```

4.9 SC_BLE_SERVICE_T

```
typedef struct
{
    SC_UUID_T *uuid;    //通用唯一标识符
    void *value;    //值
    int size;    //大小
    unsigned short handle;    //句柄
    unsigned char permission;    //许可
    int (*write_cb)(void *arg);
    int (*read_cb)(void *arg);
} SC_BLE_SERVICE_T;
```

4.10 SC_UUID_T

```
typedef struct
{
    unsigned char type;
} SC_UUID_T;
```

4.11 SC_UUID_16_T

```
typedef struct
{
    SC_UUID_T type;
    unsigned short value;
} SC_UUID_16_T;
```

4.12 SC_UUID_128_T

```
typedef struct
{
    SC_UUID_T type;
    unsigned char value[16];
} SC_UUID_128_T;
```

4.13 SC_UUID_COMMON_T

```
typedef struct
{
    union
    {
        SC_UUID_16_T uuid_16;
        SC_UUID_128_T uuid_128;
    };
} SC_UUID_COMMON_T;
```

4.14 SC_BLE_SERVICE_T

```
typedef struct
{
    SC_UUID_T *uuid;
    void *value;
    int size;
    unsigned short handle;
    unsigned char permission;
    int (*write_cb)(void *arg);
    int (*read_cb)(void *arg);
} SC_BLE_SERVICE_T;
```

4.15 SC_BLE_EVENT_MSG_T

```
typedef struct
{
    unsigned short int event_type;
    unsigned short int event_id;
    int payload_length;
    void *payload;
```

```
} SC_BLE_EVENT_MSG_T;
```

4.16 SC_BLE_CHARACTERISTIC_16_T

```
typedef struct
{
    unsigned char properties;
    unsigned short handle;
    unsigned short uuid;
} SC_BLE_CHARACTERISTIC_16_T;
```

4.17 SC_BLE_CHARACTERISTIC_128_T

```
typedef struct
{
    unsigned char properties;
    unsigned short handle;
    unsigned char uuid[16];
} SC_BLE_CHARACTERISTIC_128_T;
```

4.18 SC_BLE_RW_T

```
typedef struct
{
    unsigned short handle;
    unsigned short length;
    unsigned short offset;
    unsigned char *data;
} SC_BLE_RW_T;
```

4.19 SC_BLE_SCAN_EVENT_T

```
typedef struct
{
    unsigned char adv_type;
    unsigned char address_type;
    SC_BLE_ADDR_T address;
    unsigned char length;
    unsigned char data[31];
```

```
    char rssi;
} SC_BLE_SCAN_EVENT_T;
```

4.20 SC_BLE_CONNECT_EVENT_T

```
typedef struct
{
    SC_BLE_ADDR_T address;
    unsigned char address_type;
    int acl_handle;
    int role;
    unsigned char peer_irk[16];
} SC_BLE_CONNECT_EVENT_T;
```

4.21 SC_BLE_MTU_EXCHANGE_EVENT_T

```
typedef struct
{
    int mut;
    int acl_handle;
} SC_BLE_MTU_EXCHANGE_EVENT_T;
```

4.22 SC_BLE_READ_BY_GROUP_TYPE_EVENT_T

```
typedef struct
{
    unsigned char size;
    unsigned char value[255];
} SC_BLE_READ_BY_GROUP_TYPE_EVENT_T;
```

4.23 SC_BLE_READ_BY_TYPE_EVENT_T

```
typedef struct
{
    unsigned char size;
    unsigned char value[255];
} SC_BLE_READ_BY_TYPE_EVENT_T;
```

4.24 SC_BLE_FIND_INFORMATION_EVENT_T

```
typedef struct
{
    unsigned char size;
    unsigned char value[255];
} SC_BLE_FIND_INFORMATION_EVENT_T;
```

4.25 SC_BLE_READ_EVENT_T

```
typedef struct
{
    unsigned char size;
    unsigned char value[255];
} SC_BLE_READ_EVENT_T;
```

4.26 SC_BLE_HANDLE_VALUE_IND_EVENT_T

```
typedef struct
{
    unsigned short acl_handle;
    unsigned short handle;
    unsigned char value[255];
    int size;
} SC_BLE_HANDLE_VALUE_IND_EVENT_T;
```

4.27 SC_BLE_HANDLE_VALUE_NTF_EVENT_T

```
typedef struct
{
    unsigned short acl_handle;
    unsigned short handle;
    unsigned char value[255];
    int size;
} SC_BLE_HANDLE_VALUE_NTF_EVENT_T;
```

4.28 SC_BLE_ERROR_RSP_EVENT_T

```
typedef struct
{
    unsigned char request;
    unsigned short att_handle;
    unsigned char code;
} SC_BLE_ERROR_RSP_EVENT_T;
```

4.29 SC_ARR_CHARA_CCB_EX_T

```
typedef struct
{
    unsigned short configurationBits;
    SC_BLE_ADDR_T addr;
}SC_ATT_CHARA_CCB_EX_T;
```

4.30 SC_BLE_DEVICE_INFO_RECORD

```
typedef struct
{
    unsigned char valid;
    unsigned char address[6];
    unsigned char addr_type;
    unsigned short deviceState;
    unsigned char ltk[16];
    unsigned char keyType;
    unsigned short ediv;
    unsigned char rand[8];
    unsigned char enc_size;
    unsigned char csrk[16];
    unsigned char peer_csrk[16];
    unsigned char signCounter;
    unsigned char irk[16];
    unsigned char peer_irk[16];
}__attribute__((packed)) SC_BLE_DEVICE_INFO_RECORD;
```

4.31 SC_BT_ EVENT_ACL_CONNECT_T

```
typedef struct
```

```
{
    unsigned char addr[6];
    unsigned short int handle;
}SC_BT_EVENT_ACL_CONNECT_T;
```

4.32 SC_ATT_BOND_COMPLETE

```
typedef struct
{
    unsigned char address_type;
    SC_BLE_ADDR_T address;
}SC_ATT_BOND_COMPLETE;
```

5Example

5.1 sAPI_BleOpen

Examples

```
#include “simcom_ble.h”

Int result = sAPI_BleOpen(NULL, 0);
If (result == 0)
{
    sAPI_Debug(“open ble device success.”);
}
else
{
    sAPI_Debug(“open ble device fail. error code = %d”, result);
}
```

5.2 sAPI_BleClose

Examples

```
#include “simcom_ble.h”

sAPI_BleClose();
```

5.3 sAPI_BleCreateRandomAddress

Examples

```
#include “simcom_ble.h”

SC_BLE_ADDR_T addr;
sAPI_BleCreateRandomAddress(&addr);
```

5.4 sAPI_BleSetAddress

Examples

```
#include “simcom_ble.h”

SC_BLE_ADDR_T address;

address.bytes[0] = 0xdf;
address.bytes[1] = 0x45;
address.bytes[2] = 0xe6;
address.bytes[3] = 0x29;
address.bytes[4] = 0x65;
address.bytes[5] = 0xc0;

Int result = sAPI_BleSetAddress(&address);
If (result == 0)
{
    sAPI_Debug(“set address success.”);
}
else
{
    sAPI_Debug(“set address fail. error code = %d”, result);
}
```

5.5 sAPI_BleCreateAdvData

Examples

```
#include “simcom_ble.h”

char advData[31];
int size = sAPI_BleCreateAdvData(BLE_ADV_DATA_TYPE_COMPLETE_NAME, advData,
sizeof(advData), "SIMCOM BLE", strlen("SIMCOM BLE"));
```

5.6 sAPI_BleSetAdvData

Examples

```
#include “simcom_ble.h”
```

```
char advData[31];
int size = sAPI_BleCreateAdvData(BLE_ADV_DATA_TYPE_COMPLETE_NAME, advData,
sizeof(advData), "SIMCOM BLE", strlen("SIMCOM BLE"));
int result = sAPI_BleSetAdvData(advData, size);
if (result == 0)
{
    sAPI_Debug(“set broadcast name success.”);
}
else
{
    sAPI_Debug(“set broadcast name fail. error code = %d”, result);
}
```

5.7 sAPI_BleSetAdvParam

Examples

```
#include “simcom_ble.h”

SC_BLE_ADV_PARAM_T param;

param.interval_min = 0x800; // 1.28s
param.interval_max = 0x800;
param.advertising_type = LE_ADV_TYPE_IND;
param.own_address_type = LE_ADDRESS_TYPE_RANDOM;

int result = sAPI_BleSetAdvParam(&param);;
if (result == 0)
{
    sAPI_Debug(“set broadcast parameter success.”);
}
else
{
    sAPI_Debug(“set broadcast parameter fail. error code = %d”, result);
}
```

5.8 sAPI_BleUnregisterService

Examples

```
#include “simcom_ble.h”
```

```
int result = sAPI_BleUnregisterService();
if (result == 0)
{
    sAPI_Debug("destroy the registered service success.");
}
else
{
    sAPI_Debug("destroy the registered service fail. error code = %d", result);
}
```

5.9 sAPI_BleEnableAdv

Examples

```
#include "simcom_ble.h"

int result = sAPI_BleEnableAdv();
if (result == 0)
{
    sAPI_Debug("start broadcast success.");
}
else
{
    sAPI_Debug("start broadcast fail. error code = %d", result);
}
```

5.10 sAPI_BleDisableAdv

Examples

```
#include "simcom_ble.h"

int result = sAPI_BleDisableAdv();
if (result == 0)
{
    sAPI_Debug("stop broadcast success.");
}
else
{
    sAPI_Debug("stop broadcast fail. error code = %d", result);
}
```

5.11 sAPI_BleIndicate

Examples

```
#include “simcom_ble.h”

int result = sAPI_BleIndicate(handle, “first data”, strlen(“first data”));
if (result == 0)
{
    sAPI_Debug(“send a indicate success.”);
}
else
{
    sAPI_Debug(“send a indicate fail. error code = %d”, result);
}
```

5.12 sAPI_BleNotify

Examples

```
#include “simcom_ble.h”

int result = sAPI_BleNotify(handle, “first data”, strlen(“first data”));
if (result == 0)
{
    sAPI_Debug(“send a notify success.”);
}
else
{
    sAPI_Debug(“send a notify fail. error code = %d”, result);
}
```

5.13 sAPI_BleScan

Examples

```
#include “simcom_ble.h”

int result = sAPI_BleScan(LE_ACTIVE_SCAN,0x0800,0x0400,LE_ADDRESS_TYPE_RANDOM);
if (result == 0)
{
    sAPI_Debug(“scan success.”);
```

```
}
else
{
    sAPI_Debug(“scan fail. error code = %d”, result);
}
```

5.14 sAPI_BleConnect

Examples

```
#include “simcom_ble.h”

SC_BLE_ADDR_T addr;

addr.bytes[0] = 0xdf;
addr.bytes[1] = 0x45;
addr.bytes[2] = 0xe6;
addr.bytes[3] = 0x29;
addr.bytes[4] = 0x65;
addr.bytes[5] = 0xc0;

int result = sAPI_BleConnect(&addr, LE_ADDRESS_TYPE_PUBLIC);
if (result == 0)
{
    sAPI_Debug(“Connect to the remote device success.”);
}
```

5.15 sAPI_BleDisconnect

Examples

```
#include “simcom_ble.h”

int result = sAPI_BleDisconnect();
if (result == 0)
{
    sAPI_Debug(“close all connections success.”);
}
```

5.16 sAPI_BleMtuRequest

Examples

```
#include “simcom_ble.h”

int result = sAPI_BleMtuRequest(185);
if (result == 0)
{
    sAPI_Debug(“Send the mtu request success.”);
}
```

5.17 sAPI_BleReadByGroupTypeRequest

Examples

```
#include “simcom_ble.h”

SC_UUID_16_T uuid;

uuid.type.type = LE_UUID_TYPE_16;
uuid.value = LE_ATT_UUID_PRIMARY;

int result = sAPI_BleReadByGroupTypeRequest(0x0, 0xffff, (SC_UUID_T *)&uuid);
if (result == 0)
{
    sAPI_Debug(“Send the read by group type request success.”);
}
```

5.18 sAPI_BleReadByTypeRequest

Examples

```
#include “simcom_ble.h”

SC_UUID_16_T uuid;

uuid.type.type = LE_UUID_TYPE_16;
uuid.value = LE_ATT_UUID_CHARC;

int result = sAPI_BleReadByTypeRequest(0x0, 0xffff, (SC_UUID_T *)&uuid);
```

```
if (result == 0)
{
    sAPI_Debug("Send the read by type request success.");
}
```

5.19 sAPI_BleFindInformationRequest

Examples

```
#include "simcom_ble.h"

int result = sAPI_BleFindInformationRequest(0x0, 0xffff);
if (result == 0)
{
    sAPI_Debug("Send the find information request success.");
}
```

5.20 sAPI_BleReadRequest

Examples

```
#include "simcom_ble.h"

int result = sAPI_BleReadRequest(0x12);
if (result == 0)
{
    sAPI_Debug("Send the read request success.");
}
```

5.21 sAPI_BleWriteRequest

Examples

```
#include "simcom_ble.h"

const char *data = "hello world.";

int result = sAPI_BleWriteRquest(0x12, data, strlen(data));
if (result == 0)
{
    sAPI_Debug("Send the write request success.");
}
```

5.22 sAPI_BleWriteCommand

Examples

```
#include “simcom_ble.h”

const char *data = “hello world.”;

int result = sAPI_BleWriteRquest(0x12, data, strlen(data));
if (result == 0)
{
      sAPI_Debug(“Send the write command success.”);
}
```

5.23 sAPI_BleScanStop

Examples

```
#include “simcom_ble.h”

int result = sAPI_BleScanStop();
   if(result == 0)
    {
       sAPI_Debug("scan stop success.");
       PrintfResp("\r\nscan stop success.\r\n");
     }
     else
     {
        sAPI_Debug("scan stop fail. error code = %d", result);
        sprintf(urc, "\r\nscan stop fail. error code = %d\r\n", result);
     }
```

5.24 sAPI_BleSetGapName

Examples

```
#include “simcom_ble.h”

int result = sAPI_BleSetGapName("SIMCOM BLE DEMO");
   if(result == 0)
    {
```

```
        sAPI_Debug("gap name set success.");
        PrintfResp("\r\n gap name set success.\r\n");
      }
      else
      {
         sAPI_Debug("gap name set fail. error code = %d", result);
         sprintf(urc, "\r\ns gap name set fail. error code = %d\r\n", result);
      }
```

5.25 sAPI_BleSetAppearance

Examples

```
#include “simcom_ble.h”

int result = sAPI_BleSetAppearance(960);//Generac HID
   if(result == 0)
    {
       sAPI_Debug("set appearance success.");
       PrintfResp("\r\n set appearance success.\r\n");
      }
      else
      {
         sAPI_Debug("set appearance fail. error code = %d", result);
         sprintf(urc, "\r\n set appearance fail. error code = %d\r\n", result);
      }
```

5.26 sAPI_BleSetPairEnable

Examples

```
#include “simcom_ble.h”

int result = sAPI_BleSetpairEnable(1);
   if(result == 0)
    {
       sAPI_Debug("set enbale success.");
       PrintfResp("\r\n set enbale success.\r\n");
      }
      else
      {
         sAPI_Debug("set enbale fail. error code = %d", result);
```

```
        sprintf(urc, "\r\n set enbale fail. error code = %d\r\n", result);
    }
```

5.27 sAPI_BleGetPairInfo

Examples

```
#include “simcom_ble.h”

 int flag = 0;
 SC_BLE_DEVICE_INFO_RECORD info = {0};
 for(int i = 0;i < 10;i ++)
 {
      if(sAPI_BleGetPairInfo(i,&info) == 0)
      {
          sAPI_Debug("pair info[%d] %2X:%2X:%2X:%2X:%2X:%2X",i,
              info.address[5],info.address[4],info.address[3],
              info.address[2],info.address[1],info.address[0]);
          sprintf(urc, "\r\npair info[%d] %2X:%2X:%2X:%2X:%2X:%2X\r\n",i,
              info.address[5],info.address[4],info.address[3],
              info.address[2],info.address[1],info.address[0]);
          flag ++;
      }
 }
 if(flag == 0)
 {
      sAPI_Debug("NOT pair info");
      sprintf(urc, "\r\nNOT pair info\r\n");
}
```

5.28 sAPI_BleReadRssi

Examples

```
#include “simcom_ble.h”

int result = sAPI_BleReadRssi();
   if(result == 0)
    {
      sAPI_Debug("read rssi success.");
      PrintfResp("\r\n read rssi success.\r\n");
     }
```

```
    else
    {
       sAPI_Debug("read rssi fail. error code = %d", result);
       sprintf(urc, "\r\n read rssi fail. error code = %d\r\n", result);
    }
```

5.29 sAPI_BleGetDeviceName

Examples

```
#include “simcom_ble.h”

  Char *name = sAPI_ BleGetDeviceName();
  if(name != NULL)
   {
     sAPI_Debug("device name is %s.",name);
    }
    else
    {
       sAPI_Debug("get device name fail.");
       sprintf(urc, "\r\ns get device name fail.\r\n");
    }
```

5.30 sAPI_BleSetDeviceName

Examples

```
#include “simcom_ble.h”

int result = sAPI_BleSetDeviceName("SC_TEST_NAME");
  if(result == 0)
   {
     sAPI_Debug("set device name success.");
     PrintfResp("\r\n set device name success.\r\n");
    }
    else
    {
       sAPI_Debug("set device name fail. error code = %d", result);
       sprintf(urc, "\r\n set device name fail. error code = %d\r\n", result);
    }
```

6Demo

```
#ifdef BT_SUPPORT

#include "stdlib.h"
#include "string.h"
#include "stdio.h"
#include "simcom_os.h"
#include "simcom_ble.h"
#include "simcom_common.h"
#include "simcom_debug.h"
#include "simcom_uart.h"

typedef enum{
    SC_BLE_DEMO_OPEN                    = 1,
    SC_BLE_DEMO_CLOSE                   = 2,
    SC_BLE_DEMO_SET_ADDRESS             = 3,
    SC_BLE_DEMO_CREATE_ADV_DATA         = 4,
    SC_BLE_DEMO_SET_ADV_DATA            = 5,
    SC_BLE_DEMO_SET_ADV_PARAM           = 6,
    SC_BLE_DEMO_REGISTER_SERVICE        = 7,
    SC_BLE_DEMO_UNREGISTER_SERVICE      = 8,
    SC_BLE_DEMO_REGISTER_EVENT_HANDLE   = 9,
    SC_BLE_DEMO_ENABLE_ADV              = 10,
    SC_BLE_DEMO_DISABLE_ADV             = 11,
    SC_BLE_DEMO_DISCONNECT              = 12,
    SC_BLE_DEMO_INDICATE                = 13,
    SC_BLE_DEMO_NOTIFY                  = 14,
    SC_BLE_DEMO_SCAN_START              = 15,
    SC_BLE_DEMO_SCAN_STOP               = 16,
}SC_BLE_DEMO_TYPE;

extern sMsgQRef simcomUI_msgq;
extern void PrintfOptionMenu(char *options_list[], int array_size);
extern void PrintfResp(const char *format, ...);

#define CHARACTERISTIC_VALUE_LENGTH         512

static char custom_characteristic_value[CHARACTERISTIC_VALUE_LENGTH]    = {"hello world"};
static unsigned short   custom_client_characteristic_config_value       = 0;
```

```
SC_UUID_16_T sc_ble_uuid_primary_service =
{
    .type.type = LE_UUID_TYPE_16,
    .value = LE_ATT_UUID_PRIMARY
};

SC_UUID_16_T sc_ble_uuid_client_charc_config =
{
    .type.type = LE_UUID_TYPE_16,
    .value = LE_ATT_UUID_CLIENT_CHARC_CONFIG
};

SC_UUID_16_T sc_ble_uuid_characteristic =
{
    .type.type = LE_UUID_TYPE_16,
    .value = LE_ATT_UUID_CHARC
};

static SC_UUID_16_T custom_service_declare =
{
    .type.type = LE_UUID_TYPE_16,
    .value = 0xFFF0
};

static SC_BLE_CHARACTERISTIC_16_T custom_characteristic_declare =
{
    .uuid = 0xFFF1,
    .properties = LE_ATT_CHARC_PROP_READ | LE_ATT_CHARC_PROP_WWP |
LE_ATT_CHARC_PROP_NOTIFY | LE_ATT_CHARC_PROP_INDICATE
};

static SC_UUID_16_T custom_characteristic_attribute_value =
{
    .type.type = LE_UUID_TYPE_16,
    .value = 0xFFF1
};


static int custom_characteristic_write_cb(void *param)
{
    SC_BLE_RW_T *rw = (SC_BLE_RW_T *)param;

    sAPI_Debug("%s: handle %04x, length %d, offset %d\r\n", __func__, rw->handle, rw->length,
rw->offset);
```

```
    if((rw->offset >= CHARACTERISTIC_VALUE_LENGTH) || (rw->length >
CHARACTERISTIC_VALUE_LENGTH))
    {
        sAPI_Debug("Offset specified was past the end of the attribute.\r\n");
        PrintfResp("\r\nOffset specified was past the end of the attribute.\r\n");
        return SC_BLE_ERR_INVALID_OFFSET;
    }

    memset(custom_characteristic_value, 0, CHARACTERISTIC_VALUE_LENGTH);
    memcpy(custom_characteristic_value, rw->data, rw->length);
    sAPI_Debug("%s: receive data:", __func__);
    for(int i=0; i<rw->length; i++)
    {
        sAPI_Debug("%02X", custom_characteristic_value[i]);
    }

    return SC_BLE_ERR_OK;
}

static int custom_characteristic_read_cb(void *param)
{
    SC_BLE_RW_T *rw = (SC_BLE_RW_T *)param;

    sAPI_Debug("%s: handle %d, length %d, offset %d\r\n", __func__, rw->handle, rw->length,
rw->offset);

    if((rw->offset >= CHARACTERISTIC_VALUE_LENGTH) || (rw->length >
CHARACTERISTIC_VALUE_LENGTH))
    {
        sAPI_Debug("Offset specified was past the end of the attribute.\r\n");
        PrintfResp("\r\nOffset specified was past the end of the attribute.\r\n");
        return SC_BLE_ERR_INVALID_OFFSET;
    }

    unsigned char *p = (unsigned char *)sAPI_Malloc(CHARACTERISTIC_VALUE_LENGTH);
    if(p == NULL)
    {
        sAPI_Debug("%s: malloc fail\r\n", __func__);
        return 0x80;
    }

    memset(p, 0, CHARACTERISTIC_VALUE_LENGTH);
    memcpy(p, custom_characteristic_value, strlen(custom_characteristic_value));
    rw->data = p;

    return strlen(custom_characteristic_value);
```

```
}


static int custom_client_char_config_write_cb(void *param)
{
    SC_BLE_RW_T *rw = (SC_BLE_RW_T *)param;

    sAPI_Debug("%s: handle %d, length %d, offset %d\r\n", __func__, rw->handle, rw->length,
rw->offset);

    if(rw->data[0] > 0x03)//bit1:bit0 = indication:notification
    {
        sAPI_Debug("%s: written data illegal.\r\n", __func__);
        PrintfResp("\r\n%s: written data illegal.\r\n", __func__);
        return 0x81;
    }

    sAPI_Debug("%s: receive data:0x%02X", __func__, rw->data[0]);
    custom_client_characteristic_config_value = rw->data[0];

    return SC_BLE_ERR_OK;
}

static SC_BLE_SERVICE_T custom_attrs[] =
{
    //custom service declare
    SC_BLE_DECLARE_PRIMARY_SERVICE(sc_ble_uuid_primary_service,
custom_service_declare, LE_UUID_TYPE_16),

    //custom characteristic declare
    SC_BLE_DECLARE_CHARACTERISTIC(sc_ble_uuid_characteristic,
custom_characteristic_declare, custom_characteristic_attribute_value,
LE_ATT_PM_READABLE|LE_ATT_PM_WRITEABLE,
                                  custom_characteristic_write_cb, custom_characteristic_read_cb,
                                  custom_characteristic_value, sizeof(custom_characteristic_value)),

    //custom client characteristic config declare
    SC_BLE_DECLARE_CLINET_CHRAC_CONFIG(sc_ble_uuid_client_charc_config,
custom_client_characteristic_config_value, custom_client_char_config_write_cb),
};

static int custom_ble_handle_event(SC_BLE_EVENT_MSG_T *msg)
{
    char urc[50] = {0};
    SC_BLE_SCAN_EVENT_T *scan = NULL;
```

```
    sAPI_Debug("%s: event_type:%d event_id:%d", __func__, msg->event_type, msg->event_id);
    switch(msg->event_type)
    {
        case EVENT_TYPE_COMMON:
            switch(msg->event_id)
            {
                case COMMON_POWERUP_FAILED:
                    sAPI_Debug("COMMON_POWERUP_FAILED.");
                    PrintfResp("\r\nCOMMON_POWERUP_FAILED.\r\n");
                    ble_handle_open(-1);
                    break;
                case COMMON_POWERUP_COMPLETE:
                    sAPI_Debug("COMMON_POWERUP_COMPLETE.");
                    PrintfResp("\r\nCOMMON_POWERUP_COMPLETE.\r\n");
                    ble_handle_open(0);
                    break;
                default:break;
            }
            break;
        case EVENT_TYPE_LE:
            switch(msg->event_id)
            {
                case BLE_SCAN_EVENT:
                    scan = (SC_BLE_SCAN_EVENT_T *)(msg->payload);
                    sAPI_Debug("remote device addr:%02x:%02x:%02x:%02x:%02x:%02x rssi:%d",

scan->address.bytes[5],scan->address.bytes[4],scan->address.bytes[3],

scan->address.bytes[2],scan->address.bytes[1],scan->address.bytes[0],
                                        scan->rssi);
                    sprintf(urc, "remote device addr:%02X:%02X:%02X:%02X:%02X:%02X
rssi:%d\r\n",
                    scan->address.bytes[5],scan->address.bytes[4],scan->address.bytes[3],
                    scan->address.bytes[2],scan->address.bytes[1],scan->address.bytes[0],
                    scan->rssi);
                    PrintfResp(urc);
                    break;

                case BLE_CONNECT_EVENT:
                    sAPI_Debug("BLE_CONNECT_EVENT.");
                    PrintfResp("\r\nBLE_CONNECT_EVENT.\r\n");
                    break;

                case BLE_DIS_CONNECT_EVENT:
                    sAPI_Debug("BLE_DIS_CONNECT_EVENT.");
                    PrintfResp("\r\nBLE_DIS_CONNECT_EVENT.\r\n");
```

```
                    break;

                case BLE_MTU_EXCHANGED_EVENT:
                    sAPI_Debug("BLE_MTU_EXCHANGED_EVENT.");
                   PrintfResp("\r\nBLE_MTU_EXCHANGED_EVENT.\r\n");
                    break;

                case BLE_INDICATE_EVENT:
                    sAPI_Debug("BLE_INDICATE_EVENT.");
                    PrintfResp("\r\nBLE_INDICATE_EVENT.\r\n");
                    break;

                case BLE_CLIENT_MTU_EXCHANGED_EVENT:
                    sAPI_Debug("BLE_CLIENT_MTU_EXCHANGED_EVENT.");
                    PrintfResp("\r\nBLE_CLIENT_MTU_EXCHANGED_EVENT.\r\n");
                    break;

                case BLE_CLIENT_READ_BY_GROUP_TYPE_RSP_EVENT:
                    sAPI_Debug("BLE_CLIENT_READ_BY_GROUP_TYPE_RSP_EVENT.");
                    PrintfResp("\r\nBLE_CLIENT_READ_BY_GROUP_TYPE_RSP_EVENT.\r\n");
                    break;

                case BLE_CLIENT_WRITE_RSP:
                    sAPI_Debug("BLE_CLIENT_WRITE_RSP.");
                    PrintfResp("\r\nBLE_CLIENT_WRITE_RSP.\r\n");
                    break;

                default:
                    sAPI_Debug("%s: default.", __func__);
                    PrintfResp("\r\n%s: default.\r\n", __func__);
                    break;
            }
            break;
        default:break;
    }

    return 0;
}

void BLEDemo(void)
{
    int result = 0;
    SC_BLE_ADDR_T address = {0};
    int advDataSize = 0;
    char advData[31] = {0};
    unsigned char advData_type_Flags = 0;
```

```
    SC_BLE_ADV_PARAM_T param = {0};
    char urc[50] = {0};

    SIM_MSG_T optionMsg ={0,0,0,NULL};
    UINT32 opt = 0;
    char *note = "\r\nPlease select an option to test from the items listed below.\r\n";
    char *options_list[] =
    {
        "1. open",
        "2. close",
        "3. set address",
        "4. create adv data",
        "5. set adv data",
        "6. set adv param",
        "7. register service",
        "8. unregister service",
        "9. register event handle",
        "10. enable adv",
        "11. disable adv",
        "12. disconnect",
        "13. indicate",
        "14. notify",
        "15. scan start",
        "16. scan stop",
    };
    //gap peripheral role execute procedure: 9->1->3->6->4->5->7->10->(establish ble
link)->(13/14)->12->8->11->2
    //gap observer role scan procedure: 9->1->15->16

    while(1)
    {
        PrintfResp(note);
        PrintfOptionMenu(options_list,sizeof(options_list)/sizeof(options_list[0]));
        sAPI_MsgQRecv(simcomUI_msgq,&optionMsg,SC_SUSPEND);
        if(SRV_UART != optionMsg.msg_id)
        {
            sAPI_Debug("%s,msg_id is error!!",__func__);
            break;
        }

        sAPI_Debug("arg3 = [%s]",optionMsg.arg3);
        opt = atoi(optionMsg.arg3);
        free(optionMsg.arg3);
        result = 0;
        memset(urc, 0, 50);
```

```
switch(opt)
{
    case SC_BLE_DEMO_OPEN:
        result = sAPI_BleOpen(NULL, 0);
        if(result == 0)
        {
            sAPI_Debug("open ble device success.");
            PrintfResp("\r\nopen ble device success.\r\n");
        }
        else
        {
            sAPI_Debug("open ble device fail. error code = %d", result);
            sprintf(urc, "\r\nopen ble device fail. error code = %d\r\n", result);
            PrintfResp(urc);
        }
        break;
    case SC_BLE_DEMO_CLOSE:
        sAPI_BleClose();
        sAPI_Debug("close ble device finish.");
        PrintfResp("\r\nclose ble device finish.\r\n");
        break;
    case SC_BLE_DEMO_SET_ADDRESS:
        address.bytes[0] = 0x01;
        address.bytes[1] = 0x00;
        address.bytes[2] = 0x00;
        address.bytes[3] = 0x00;
        address.bytes[4] = 0x00;
        address.bytes[5] = 0xc0;
        result = sAPI_BleSetAddress(&address);
        if(result == 0)
        {
            sAPI_Debug("set address success.");
            PrintfResp("\r\nset address success.\r\n");
        }
        else
        {
            sAPI_Debug("set address fail. error code = %d", result);
            sprintf(urc, "\r\nset address fail. error code = %d\r\n", result);
            PrintfResp(urc);
        }
        break;
    case SC_BLE_DEMO_CREATE_ADV_DATA:
        advData_type_Flags = 0x18;
        advDataSize = 0;
        memset(advData, 0, sizeof(advData));
        result = sAPI_BleCreateAdvData(BLE_ADV_DATA_TYPE_FLAG, advData,
```

```
sizeof(advData), &advData_type_Flags, 1);
                    if(result == -1)
                    {
                        advDataSize = -1;
                        sAPI_Debug("create adv data fail. error code = %d", result);
                        sprintf(urc, "\r\ncreate adv data fail. error code = %d\r\n", result);
                        PrintfResp(urc);
                        break;
                    }

                    advDataSize += result;
                    result = sAPI_BleCreateAdvData(BLE_ADV_DATA_TYPE_COMPLETE_NAME,
&advData[advDataSize], sizeof(advData)-advDataSize, "SIMCOM BLE DEMO", strlen("SIMCOM BLE
DEMO"));
                    if(result == -1)
                    {
                        advDataSize = -1;
                        sAPI_Debug("create adv data fail. error code = %d", result);
                        sprintf(urc, "\r\ncreate adv data fail. error code = %d\r\n", result);
                        PrintfResp(urc);
                        break;
                    }

                    advDataSize += result;
                    sAPI_Debug("create adv data success.");
                    PrintfResp("\r\ncreate adv data success.\r\n");
                    break;
                case SC_BLE_DEMO_SET_ADV_DATA:
                    if((advDataSize == 0) || (advDataSize == -1))
                    {
                        sAPI_Debug("set adv data fail, please create adv data.");
                        PrintfResp("\r\nset adv data fail, please create adv data.\r\n");
                    }
                    else
                    {
                        result = sAPI_BleSetAdvData(advData, advDataSize);
                        if(result == 0)
                        {
                            sAPI_Debug("set adv data success.");
                            PrintfResp("\r\nset adv data success.\r\n");
                        }
                        else
                        {
                            sAPI_Debug("set adv data fail. error code = %d", result);
                            sprintf(urc, "\r\nset adv data fail. error code = %d\r\n", result);
                            PrintfResp(urc);
```

```
                    }
                }
                break;
            case SC_BLE_DEMO_SET_ADV_PARAM:
                memset(&param, 0, sizeof(SC_BLE_ADV_PARAM_T));
                param.interval_min = 400;
                param.interval_max = 400;
                param.advertising_type = LE_ADV_TYPE_IND;
                param.own_address_type = LE_ADDRESS_TYPE_RANDOM;
                result = sAPI_BleSetAdvParam(&param);
                if(result == 0)
                {
                    sAPI_Debug("set broadcast parameter success.");
                    PrintfResp("\r\nset broadcast parameter success.\r\n");
                }
                else
                {
                    sAPI_Debug("set broadcast parameter fail. error code = %d", result);
                    sprintf(urc, "\r\nset broadcast parameter fail. error code = %d\r\n", result);
                    PrintfResp(urc);
                }
                break;
            case SC_BLE_DEMO_REGISTER_SERVICE:
                result = sAPI_BleRegisterService(custom_attrs, sizeof(custom_attrs) /
sizeof(SC_BLE_SERVICE_T));
                if(result == 0)
                {
                    sAPI_Debug("service register success.");
                    PrintfResp("\r\nservice register success.\r\n");
                }
                else
                {
                    sAPI_Debug("service register fail. error code = %d", result);
                    sprintf(urc, "\r\nservice register fail. error code = %d\r\n", result);
                    PrintfResp(urc);
                }
                break;
            case SC_BLE_DEMO_UNREGISTER_SERVICE:
                result = sAPI_BleUnregisterService();
                if(result == 0)
                {
                    sAPI_Debug("service unregister success.");
                    PrintfResp("\r\nservice unregister success.\r\n");
                }
                else
                {
```

```
            sAPI_Debug("service unregister fail. error code = %d", result);
            sprintf(urc, "\r\nservice unregister fail. error code = %d\r\n", result);
            PrintfResp(urc);
        }
        break;
    case SC_BLE_DEMO_REGISTER_EVENT_HANDLE:
        sAPI_BleRegisterEventHandle(custom_ble_handle_event);
        sAPI_Debug("register ble event handle finish.");
        PrintfResp("\r\nregister ble event handle finish.\r\n");
        break;
    case SC_BLE_DEMO_ENABLE_ADV:
        result = sAPI_BleEnableAdv();
        if(result == 0)
        {
            sAPI_Debug("enable adv success.");
            PrintfResp("\r\nenable adv success.\r\n");
        }
        else
        {
            sAPI_Debug("enable adv fail. error code = %d", result);
            sprintf(urc, "\r\nenable adv fail. error code = %d\r\n", result);
            PrintfResp(urc);
        }
        break;
    case SC_BLE_DEMO_DISABLE_ADV:
        result = sAPI_BleDisableAdv();
        if(result == 0)
        {
            sAPI_Debug("disable adv success.");
            PrintfResp("\r\ndisable adv success.\r\n");
        }
        else
        {
            sAPI_Debug("disable adv fail. error code = %d", result);
            sprintf(urc, "\r\ndisable adv fail. error code = %d\r\n", result);
            PrintfResp(urc);
        }
        break;
    case SC_BLE_DEMO_DISCONNECT:
        result = sAPI_BleDisconnect();
        if(result == 0)
        {
            sAPI_Debug("disconnect ble link success.");
            PrintfResp("\r\ndisconnect ble link success.\r\n");
        }
        else
```

```
                {
                    sAPI_Debug("disconnect ble link fail. error code = %d", result);
                    sprintf(urc, "\r\ndisconnect ble link fail. error code = %d\r\n", result);
                    PrintfResp(urc);
                }
                break;
            case SC_BLE_DEMO_INDICATE:
                sAPI_Debug("start sending indication. gatt_handle:%d",
custom_characteristic_declare.handle);
                if(custom_characteristic_declare.handle == 0)
                {
                    sAPI_Debug("send indication fail, please register service.");
                    PrintfResp("\r\nsend indication fail, please register service.\r\n");
                }
                else
                {
                    result = sAPI_BleIndicate(custom_characteristic_declare.handle, "simcom is
testing indicate", strlen("simcom is testing indicate"));
                    if(result == 0)
                    {
                        sAPI_Debug("indicate send success.");
                        PrintfResp("\r\nindicate send success.\r\n");
                    }
                    else
                    {
                        sAPI_Debug("indicate send fail. error code = %d", result);
                        sprintf(urc, "\r\nindicate send fail. error code = %d\r\n", result);
                        PrintfResp(urc);
                    }
                }
                break;
            case SC_BLE_DEMO_NOTIFY:
                sAPI_Debug("start sending notification. gatt_handle:%d",
custom_characteristic_declare.handle);
                if(custom_characteristic_declare.handle == 0)
                {
                    sAPI_Debug("send notification fail, please register service.");
                    PrintfResp("\r\nsend notification fail, please register service.\r\n");
                }
                else
                {
                    result = sAPI_BleNotify(custom_characteristic_declare.handle, "simcom is testing
notify", strlen("simcom is testing notify"));
                    if(result == 0)
                    {
                        sAPI_Debug("notify send success.");
```

```
                        PrintfResp("\r\nnotify send success.\r\n");
                    }
                    else
                    {
                        sAPI_Debug("notify send fail. error code = %d", result);
                        sprintf(urc, "\r\nnotify send fail. error code = %d\r\n", result);
                        PrintfResp(urc);
                    }
                }
                break;
            case SC_BLE_DEMO_SCAN_START:
                result = sAPI_BleScan(LE_ACTIVE_SCAN, 0x1F40, 0x1F40,
LE_ADDRESS_TYPE_RANDOM);
                if(result == 0)
                {
                    sAPI_Debug("scan start success.");
                    PrintfResp("\r\nscan start success.\r\n");
                }
                else
                {
                    sAPI_Debug("scan start fail. error code = %d", result);
                    sprintf(urc, "\r\nscan start fail. error code = %d\r\n", result);
                    PrintfResp(urc);
                }
                break;
            case SC_BLE_DEMO_SCAN_STOP:
                result = sAPI_BleScanStop();
                if(result == 0)
                {
                    sAPI_Debug("scan stop success.");
                    PrintfResp("\r\nscan stop success.\r\n");
                }
                else
                {
                    sAPI_Debug("scan stop fail. error code = %d", result);
                    sprintf(urc, "\r\nscan stop fail. error code = %d\r\n", result);
                    PrintfResp(urc);
                }
                break;
            default:break;
        }
    }
}

#endif
```