PHY62XX

ANCS（苹果消息中心服务）

应用指南

Version 0.1

Phyplus Microelectronics Limited

2018/07/10

版本控制信息

版本/状态	作者	参与者	起止日期	备注
V0.1	Eagle		07/10/2018	文档初稿

目录

1 简介

本文档介绍 ANCS（苹果消息中心服务）在 PHY62XX SDK 中服务部分的实现和相关应用介绍。

苹果通知中心（Apple Notification Center Service, ANCS）的目的是提供给蓝牙外设一种简单、方便的获取 iOS 设备通知信息的方式。

ANCS 的使用没有依赖，它是 GATT 的一个子集，任何一个实现了 GATT client 的设备可以方便的从 ios 设备获取通知信息。

PHY62XX SDK 代码提供 ANCS 服务的实现和一个应用的例子，分别在下列目录：

ANCS Profile	Trunk\components\profiles\ancs
应用样例	Trunk\example\ble_peripheral\ancs

2 ANCS Profile

ANCS Profile 主要实现基于 GATT Client 的 Service discovery 和 ANCS 应用层的数据交互。

对于应用，ANCS 提供了一组 API 以及一个消息接口用于向应用推送消息。

2.1 bStatus_t ble_ancs_init(ancs_evt_hdl_t evt_hdl, uint8_t task_ID)

ANCS profile 初始化，通过该函数，应用可以初始化 ANCS，并且注册消息相应函数。

- 参数

类型	参数名	说明
ancs_evt_hdl_t	evt_hdl	ANCS profile 回调函数，注册之后用于推送模块消息。
uint8_t	task_ID	OSAL task ID，。

- 返回值

SUCCESS	初始化成功。
其他数值	参考<comdef.h>

2.2 bStatus_t ble_ancs_attr_add(const ancs_notif_attr_id id, uint8_t * p_data, const uint16_t len)

ANCS 配置添加通知属性。这个需要在 ANCS Discovery 之前配置，通常是在 ble_ancs_init() 完成之后进行，0~7 为标准的通知属性，8~255 为保留值，具体参考下表：

NotificationAttributeIDAppIdentifier	= 0,
NotificationAttributeIDTitle	= 1, (Needs to be followed by a 2-bytes max length parameter)
NotificationAttributeIDSubtitle	= 2, (Needs to be followed by a 2-bytes max length parameter)
NotificationAttributeIDMessage	= 3, (Needs to be followed by a 2-bytes max length parameter)
NotificationAttributeIDMessageSize	= 4,
NotificationAttributeIDDate	= 5,
NotificationAttributeIDPositiveActionLabel	= 6,
NotificationAttributeIDNegativeActionLabel	= 7,
Reserved NotificationAttributeID values	= 8-255

通知属性只有配置之后才会生效，当应用调用 ble_ancs_get_notif_attrs()获取通知属性时候，通知事件（通过注册的回调函数）会返回已配置的通知属性的参数值。

- 参数

类型	参数名	说明
const ancs_notif_attr	Id	GPIO pin。
uint8_t *	p_data	已分配的内存，用于全局存储对应属性的参数值（通常是 UTF-8 字符串）。
const uint16_t	Len	p_data 字节大小。

- 返回值

SUCCESS	初始化成功。
其他数值	参考<comdef.h>

2.3 bStatus_t ble_ancs_get_notif_attrs(const uint8_t * pNotificationUID)

在收到 ANCS Notify 消息之后，应用可以通过 Notification UID 去请求获取通知属性，该函数执行之后，有效的请求数据会以消息形式通过回调函数返回。

- 参数

类型	参数名	说明
const uint8_t *	pNotificationUID	Notification UID，该 ID 为 UTF-8 格式的字符串，以 0 结尾

- 返回值

SUCCESS	初始化成功。
其他数值	参考<comdef.h>

2.4 bStatus_t ble_ancs_get_app_attrs(const uint8_t * p_app_id, uint8_t app_id_len)

在收到通知属性的回调事件之后，如果事件的 attr_id 为 BLE_ANCS_NOTIF_ATTR_ID_APP_IDENTIFIER，那么可以通过本函数请求 app 属性的内容。

- 参数

类型	参数名	说明
const uint8_t *	p_app_id	App ID，该字段在通知属性的回调事件中给出。
uint8_t	app_id_len	App ID 长度。

- 返回值

SUCCESS	初始化成功。
其他数值	参考<comdef.h>

2.5 bStatus_t ble_ancs_start_descovery(uint16_t conn_handle)

开始 Discovery 主机一侧的 ANCS 服务，该函数需要在 SMP 建立之后执行，Discovery 完成之后，BLE_ANCS_EVT_DISCOVERY_COMPLETE 事件会通过回调函数推送，反之，如果失败，BLE_ANCS_EVT_DISCOVERY_FAILED 事件会通过回调函数推送。

- 参数

类型	参数名	说明
uint16	conn_handle	连接句柄。

- 返回值

SUCCESS	初始化成功。
其他数值	参考<comdef.h>

2.6 bStatus_t ble_ancs_handle_gatt_event(gattMsgEvent_t* pMsg)

响应 GATT 事件，ANCS profile 需要响应 host ANCS 服务的 GATT 事件，函数会忽略无关的事件，并且不对输入数据做任何改变。

- 参数

类型	参数名	说明
gattMsgEvent_t*	pMsg	GATT 事件。

- 返回值

SUCCESS	初始化成功。
其他数值	参考<comdef.h>

3 ANCS 应用例程

3.1 OSAL task 初始化

参考下面文本框中黑体部分代码，初始化过程需要：

- 调用 ancs 初始化函数
- 调用函数 ble_ancs_attr_add()添加配置通知需要的属性。

```
void AncsApp_init( uint8 task_id )
{
  AncsApp_TaskID = task_id;


  //OSAL Task 初始化配置


  // Initialize GATT attributes
  GGS_AddService(GATT_ALL_SERVICES);                  // GAP GATT Service
  GATTServApp_AddService(GATT_ALL_SERVICES);      // GATT Service
  DevInfo_AddService();                                          // Device Information Service

// For ANCS, the device must register an a GATT client, whereas the
// iPhone acts as a GATT server.
ble_ancs_init(on_ancs_evt, AncsApp_TaskID);
ble_ancs_attr_add(BLE_ANCS_NOTIF_ATTR_ID_APP_IDENTIFIER,m_attr_appid,ANCS_ATTR_DATA_MAX);
//ble_ancs_attr_add(BLE_ANCS_APP_ATTR_ID_DISPLAY_NAME,m_attr_disp_name,sizeof(m_attr_disp_name));
ble_ancs_attr_add(BLE_ANCS_NOTIF_ATTR_ID_TITLE,m_attr_title,ANCS_ATTR_DATA_MAX);
ble_ancs_attr_add(BLE_ANCS_NOTIF_ATTR_ID_MESSAGE,m_attr_message,ANCS_ATTR_DATA_MAX);
ble_ancs_attr_add(BLE_ANCS_NOTIF_ATTR_ID_SUBTITLE,m_attr_subtitle,ANCS_ATTR_DATA_MAX);
ble_ancs_attr_add(BLE_ANCS_NOTIF_ATTR_ID_MESSAGE_SIZE,m_attr_message_size,ANCS_ATTR_DATA_MAX);
ble_ancs_attr_add(BLE_ANCS_NOTIF_ATTR_ID_DATE,m_attr_date,ANCS_ATTR_DATA_MAX);
ble_ancs_attr_add(BLE_ANCS_NOTIF_ATTR_ID_POSITIVE_ACTION_LABEL,m_attr_posaction,ANCS_ATTR_DATA_MAX);
ble_ancs_attr_add(BLE_ANCS_NOTIF_ATTR_ID_NEGATIVE_ACTION_LABEL,m_attr_negaction,ANCS_ATTR_DATA_MAX);
osal_set_event( AncsApp_TaskID, START_DEVICE_EVT );
}
```

3.2 启动 Discovery

设备连接 iOS 设备之后，如果 SMP 过程已经完成，可以调用函数 ble_ancs_start_descovery()启动 Discovery，参考下列文本框中黑体部分调用。

```
static void AncsApp_processPairState(uint8_t state, uint8_t status)
{
   if (state == GAPBOND_PAIRING_STATE_STARTED){
      LOG("Pairing started\n");
   }
   else if (state == GAPBOND_PAIRING_STATE_COMPLETE){
      if (status == SUCCESS){
         LOG("Pairing Successful\n");
         // Now that the device has successfully paired to the iPhone,
         // the subscription will not fail due to insufficient authentication.
        ble_ancs_start_descovery(gapConnHandle);
      }
      else{
         LOG("Pairing fail: %d\n", status);
      }
   }
   else if (state == GAPBOND_PAIRING_STATE_BONDED){
      if (status == SUCCESS){
         LOG("Bonding Successful\n");
         ble_ancs_start_descovery(gapConnHandle);
      }
   }
}
```

3.3 响应 GATT 事件

请参考下列文本框黑体部分的调用。

```
static uint8_t AncsApp_processGATTMsg(gattMsgEvent_t *pMsg)
{
   ble_ancs_handle_gatt_event(pMsg);

   //ANCS requires authentication, if the NP attempts to read/write chars on the
   //NP without proper authentication, the NP will respond with insufficent_athen
   //error to which we must respond with a slave security request

   //以下代码用于应用本身响应GATT事件，此处略过

   return (TRUE);
}
```

3.4 ANCS 应用层事件

本例中，ANCS 应用层事件由函数 on_ancs_evt(ancs_evt_t * p_evt)进行处理。