PHY62XX

ANCS (Apple Notification Center Service)

Application Note

Version 0.1

Author: Eagle Lao

Security: Public

Date: 2021.3

PhyPlus

Revision History

Revision	Author	Participant	Date	Description
V0.1	Eagle Lao		07/10/2018	Draft file

Table of Contents

1 Introduction

This file introduces the implementation of ANCS (Apple Message Center Service) in the service part of the PHY62XX SDK and the introduction of related applications.
Purpose of Apple Notification Center Service (ANCS) is to provide Bluetooth peripherals with a simple and convenient way to obtain notification information from iOS devices.
There is no dependency on the use of ANCS. It is a subset of GATT. Any device that implements GATT client can easily obtain notification information from ios devices.
The PHY62XX SDK code provides the realization of ANCS service and an application example, respectively in the following directories:

ANCS Profile	Trunk\components\profiles\ancs
Application samples	Trunk\example\ble_peripheral\ancs

2 ANCS Profile

ANCS Profile mainly realizes the data interaction between Service discovery based on GATT Client and ANCS application layer.
For applications, ANCS provides a set of APIs and a message interface for pushing messages to applications.

2.1 bStatus_t ble_ancs_init(ancs_evt_hdl_t evt_hdl, uint8_t task_ID)

ANCS profile initialization. Through this function, the application can initialize ANCS and register the message corresponding function.

- Parameter

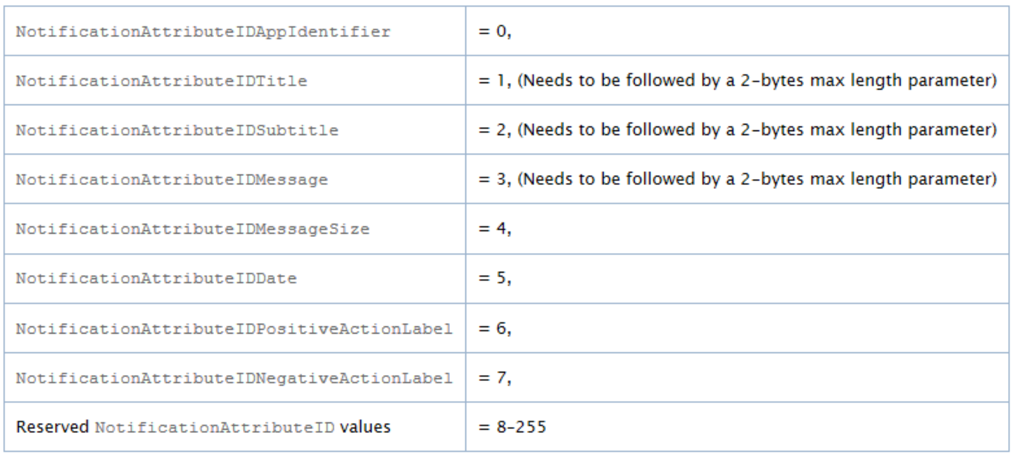
Type	Parameter name	Description
ancs_evt_hdl_t	evt_hdl	ANCS profile callback function, used to push module messages after registration.
uint8_t	task_ID	OSAL task ID.

- Return value

SUCCESS	Initialization succeeded.
Other values	Reference<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 configuration adds notification attributes. This needs to be configured before ANCS Discovery, usually after ble_ancs_init() is completed, 0~7 are standard notification attributes, 8~255 are reserved values, please refer to the following table for details:



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

Notification attribute will only take effect after configuration. When the application calls ble_ancs_get_notif_attrs() to get the notification attribute, the notification event (via the registered callback function) will return the parameter value of the configured notification attribute.

- Parameter

Type	Parameter name	Description
const ancs_notif_attr	Id	GPIO pin.
uint8_t *	p_data	Allocated memory is used to globally store the parameter value of the corresponding attribute (usually a UTF-8 string).
const uint16_t	Len	P_data byte size.

- Return value

SUCCESS	Initialization succeeded.
Other values	Reference< comdef.h>

2.3 bStatus_t ble_ancs_get_notif_attrs(const uint8_t * pNotificationUID)

After receiving the ANCS Notify message, the application can request notification attributes through the Notification UID. After the function is executed, the valid request data will be returned in the form of a message through the callback function.

- Parameter

Type	Parameter name	Description
const uint8_t *	pNotificationUID	Notification UID, the ID is a string in UTF-8 format, ending with 0

- Return value

SUCCESS	Initialization succeeded.
Other values	Reference<comdef.h>

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

After receiving the callback event of the notification attribute, if the attr_id of the event is BLE_ANCS_NOTIF_ATTR_ID_APP_IDENTIFIER, then the content of the app attribute can be requested through this function.

- Parameter

Type	Parameter name	Description
const uint8_t *	p_app_id	App ID, this field is given in the callback event of the notification property.
uint8_t	app_id_len	App ID length.

- Return value

SUCCESS	Initialization succeeded.
Other values	Reference<comdef.h>

2.5 bStatus_t ble_ancs_start_descovery(uint16_t conn_handle)

Start the ANCS service on the Discovery host. This function needs to be executed after the SMP is established. After Discovery is completed, the BLE_ANCS_EVT_DISCOVERY_COMPLETE event will be pushed through the callback function. On the contrary, if it fails, the BLE_ANCS_EVT_DISCOVERY_FAILED event will be pushed through the callback function.

- Parameter

Type	Parameter name	Description
uint16	conn_handle	Connection handle.

- Return value

SUCCESS	Initialization succeeded.
Other values	Reference<comdef.h>

2.6 bStatus_t ble_ancs_handle_gatt_event(gattMsgEvent_t* pMsg)

In response to GATT events, the ANCS profile needs to respond to GATT events of the host ANCS service. The function will ignore irrelevant events and will not make any changes to the input data.

- Parameter

Type	Parameter name	Description
gattMsgEvent_t*	pMsg	GATT event.

- Return value

SUCCESS	Initialization succeeded.
Other values	Reference< comdef.h>

3 ANCS application example

3.1 OSAL task initialization

Refer to the code in bold in the text box below, the initialization process requires:

- Call ancs initialization function
- Call the function ble_ancs_attr_add() to add attributes required for configuration notification.

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

//OSAL Task initialization configuration

   // 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_M
AX);
ble_ancs_attr_add(BLE_ANCS_NOTIF_ATTR_ID_SUBTITLE,m_attr_subtitle,ANCS_ATTR_DATA_MA
X);
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,ANC
S_ATTR_DATA_MAX);
ble_ancs_attr_add(BLE_ANCS_NOTIF_ATTR_ID_NEGATIVE_ACTION_LABEL,m_attr_negaction,ANC
S_ATTR_DATA_MAX);
osal_set_event( AncsApp_TaskID, START_DEVICE_EVT );
}
```

3.2 Start Discovery

After the device is connected to the iOS device, if the SMP process has been completed, you can call the function ble_ancs_start_descovery() to start Discovery, refer to the bold part of the following text box to call.

```
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 Responding to GATT events

Please refer to the call in the bold part of the text box below.

```
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
//The following code is used by the application itself to respond to GATT events, skip it here
    return (TRUE);
}
```

3.4 ANCS application layer events

In this example, the ANCS application layer event is handled by the function on_ancs_evt(ancs_evt_t * p_evt).