RX23W Group

Apple Notification Center Service Sample Program

Introduction

This document is application Note for sample application of Apple Notification Center Service (ANCS) client role.

Target Device

- Target Board for RX23W

Related Documents

- Target Board for RX23W User's Manual (R20UT4634)
- e² studio 2020-04 and e² studio v7.8 Getting Started Guide (R20UT4819)
- Renesas Flash Programmer V3.08 Flash memory programming software User's Manual (R20UT4813)
- Bluetooth Low Energy Protocol Stack Basic Package: User's Manual (R01UW0205)
- RX23W Group Bluetooth Low Energy Profile Developer's Guide (R01AN4553)
- Apple Notification Center Service (ANCS) Specification
  (https://developer.apple.com/library/archive/documentation/CoreBluetooth/Reference/AppleNotificationCenterServiceSpecification/Introduction/Introduction.html#//apple_ref/doc/uid/TP40013460-CH2-SW1)
- Bundle IDs for native iOS and iPadOS apps in mobile device management
  (https://support.apple.com/guide/mdm/bundle-ids-for-native-ios-and-ipados-apps-mdm90f60c1ce/web)

Contents

1. Overview

ANCS sample application operates on Target Board for RX23W (hereinafter called “target board”) and connects to iOS devices with Bluetooth® Low Energy (hereinafter called “Bluetooth LE”) wireless communication.

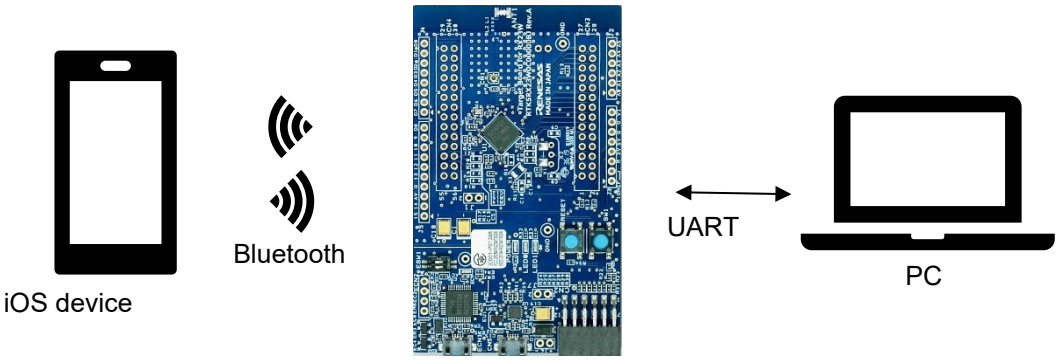

Figure 1.1 Operation environment

ANCS sample application starts Advertising after enabling the Resolvable Private Address (RPA) feature. This connects with iOS device by iOS device’s action.

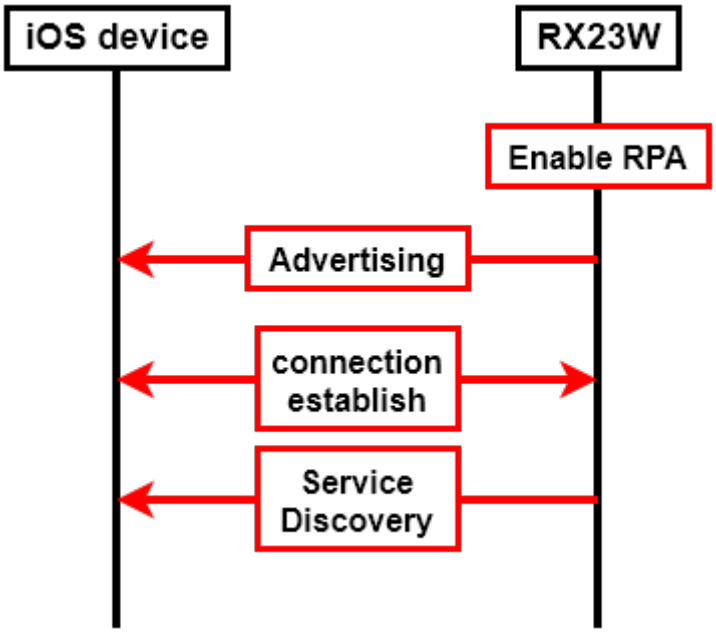

Figure 1.2 Operation flow for connection

After Paring with iOS device for use of ANCS, target device receives various notification generated from iOS (hereinafter called “iOS notification”). You can also request details of iOS notification to connected iOS device.

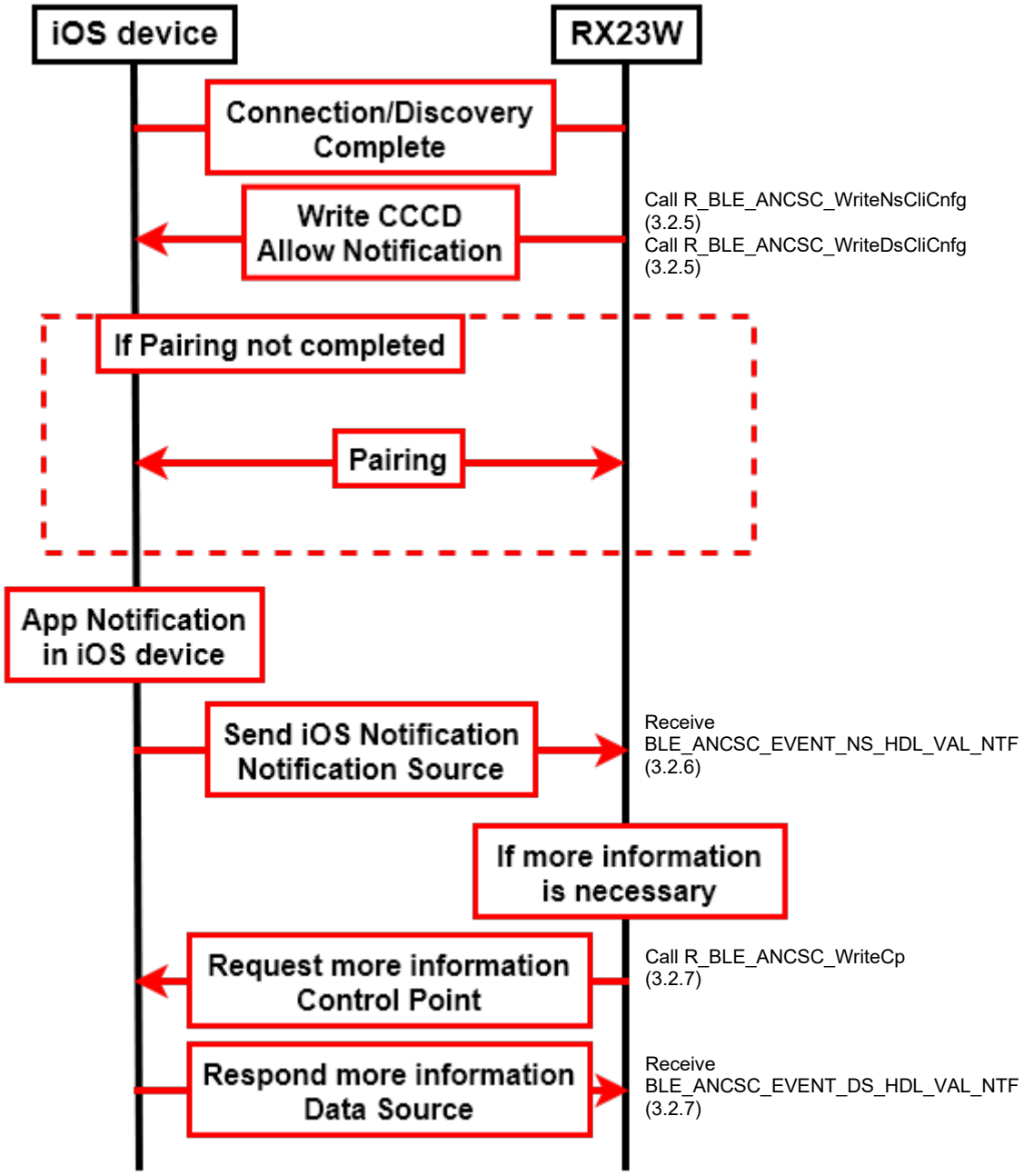

Figure 1.3 Operation flow for ANCS

1.1 Operation Requirements

ANCS sample application provided in this application note is confirmed to work with the following hardware and software environments.

Table 1.1 Hardware environments

Hardware	Version, etc.
Target Board for RX23W	RTK5RX23W0C00000BJ
USB cable	USB A - Micro B cable
Windows 10 PC	---
iOS device	iOS14.2

Table 1.2 Software environments

Software	Version, etc.
e^{2}studio	V2021-04
CC-RX compiler	V2.08.01
QE for BLE	V1.10
BLE FIT module	V2.11
Renesas Flash Programmer	V3.08
Tera Term (VT100 compatible terminal emulator)	UART Setting Baud rate：115200bps Data：8bit Parity：None Stop bits：None Flow Control：None Character encoding：UTF-8

1.2 Directory/File Structure

This application note provides a sample project for the ANCS client role (hereinafter called “sample project”). The structure of this project is as follows.

Table 1.3 ANCS Sample project file structure

Directory/File structure				Description
Release\	ancs_client.json ble_sample_tbrx23w_ancs_client.mot			QE for BLE file pre-built mot file
src\	smc_gen\	Config_BLE_PROFILE\	app_main.c	BLE application main code
			gatt_db.c gatt_db.h	GATT database code
			r_ble_ANCSc.c r_ble_ANCSc.h	profile code
		r_ble_rx23w\		BLE protocol stack program
		general\ r_ble_qe_utility\ r_bsp\ r_byteq\ r_cmt_rx\ r_config\ r_flash_rx\ r_gpio_rx\ r_irq_rx\ r_lpc_rx\ r_pincfg\ r_sci_rx\		Other files generated from smart configurator
	ble_sample_tbrx23w_ancs_client.c			project main code
.cproject .project ble_sample_tbrx23w_ancs_client HardwareDebug.launch				CC-RX project file
ble_sample_tbrx23w_ancs_client.scfg				Smart configurator file

1.3 Import sample project

This project is provided in the form of a zip file and can be imported into e²studio. For import procedures, please refer to the [*e² studio 2020-04 and e² studio v7.8 Getting Started Guide (R20UT4819)*]. The zip file that can be imported is as follows:

- ble_sample_tbrx23w_ancs_client.zip

1.4 Programming Firmware

“Release” folder contains pre-built mot files. You can confirm operation shown in **Figure 1.1** by programming mot file to target board. Please refer to [*Target Board for RX23W User's Manual (R20UT4634)*] and [*Renesas Flash Programmer V3.08 Flash memory programming software User's Manual (R20UT4813)*] for the procedure to writing mot file to target board. The available mot files are as follows:

- ble_sample_tbrx23w_ancs_client.mot

2. Operation of ANCS Sample Project

This chapter describes the operating procedure of this project.

2.1 Application Option

You can change some of the behavior by configuring the following options in the program.

Table 2.1 ANCS Sample project file structure

Option	Description
ANCS_STRING_LEN	The maximum length of string data used in such as ANCS command. default: 32 range: 1 to 244
ANCS_RPA_ENABLE	Enable or Disable RPA 0: Disable (default) 1: Enable Note: Remote RPA will be resolved even if this option is disabled.

2.2 Operation Flow of ANCS sample application

This section describes the procedure for connection with iOS device and ANCS data communication after running sample project. For description of source code, refer [3.2 Application Program].

2.2.1 Connection with iOS Device

This section describes procedure of connection and pairing between target board and iOS device. In this section, iPhone8 (iOS14.2) is used as iOS device.

1. Connect the Micro B side of the USB cable to CN5 of the target board and the Type A side to any USB port on the PC.

2. Start the VT100 compatible terminal emulator (hereinafter called “terminal emulator”) on your PC. UART setting of terminal emulator is shown in **Table 1.2**.

3. Press the RESET button on the target board. After pressing, the target board will start Advertising operation. When Advertising operation starts, the following text is displayed in the terminal emulator:

```
receive BLE_GAP_EVENT_ADV_ON result : 0x0000, adv_hdl : 0x0000
```

Figure 2.1 Display in Beginning of Advertising Operation

4. Select “Bluetooth” on the Setting screen of iOS device. Tap “REL_ANCS” in DEVICE field.

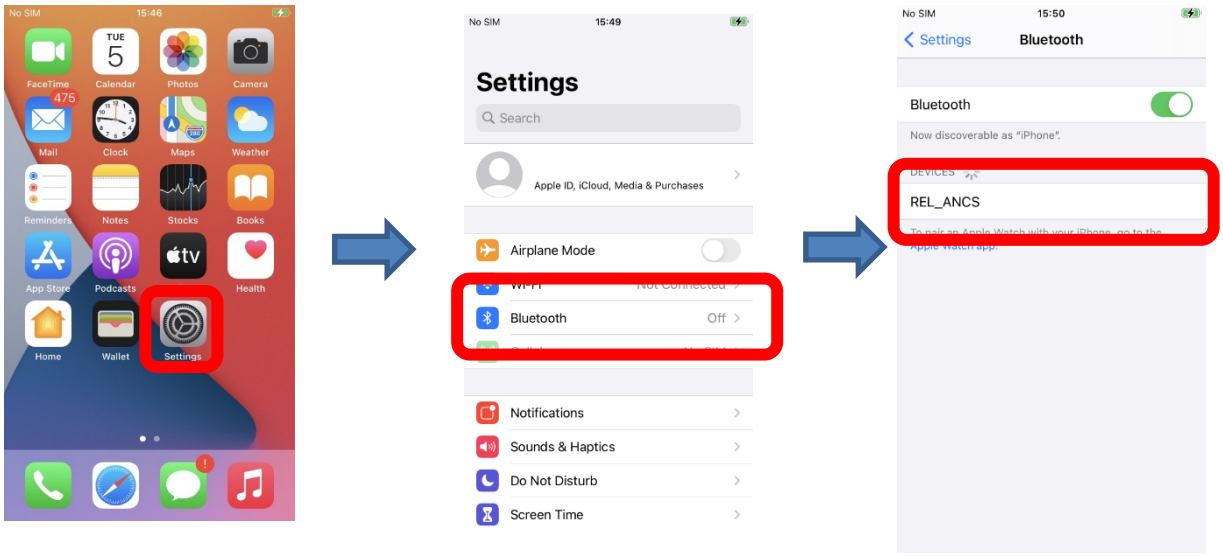

Figure 2.2 Bluetooth Connection between iOS device and target board

5. When target board and iOS device is connected for the first time, a dialog asking for pairing will appear. Please allow pairing with “REL_ANCS”.

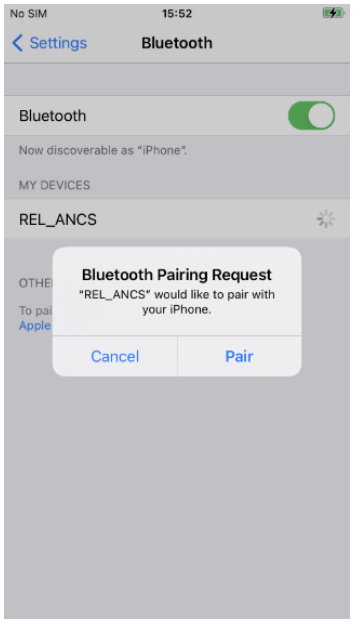

Figure 2.3 Pairing between iOS device and target board

6. Dialog asking for allowing notification will appear. Please allow notifications.

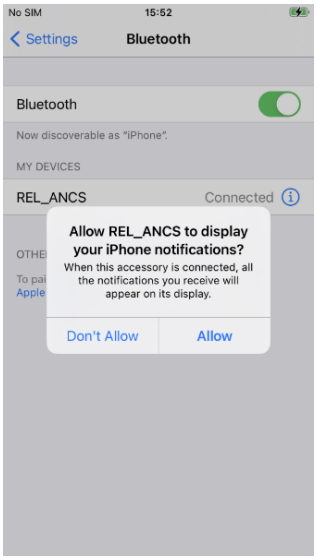

Figure 2.4 Allow iOS notification to target device

When all procedure above is completed and ANCS data communication is possible, following text will be displayed in the terminal emulator.

```
ANCS: Notification Enabled.
```

Figure 2.5 Ready for ANCS communication

2.2.2 Receive Notification from iOS Device

You can receive iOS notifications after connecting to your iOS device in the [2.2.1 Connection with iOS Device]. You can also get more information about the iOS notifications you received and interact with them. In this section, operation of sample project using iOS notification from calendar app as an example. For more information on Event IDs, Event Flags, etc., refer [*Apple Notification Center Service (ANCS) Specification*].

1. Launch calendar app in iOS device. Set an appointment at any time and wait until the scheduled time.

2. In appointment time, iOS notification will be sent from iOS device to target board. When target board receives the notification, following text will appear in terminal emulator in PC connected with target board. Sample project can use command defined in section 2.3 to get more information about iOS notification or make action to iOS notification. Example usage of this command will be shown after next section.

```
ANCS : Notification Source received.
Event ID          = 0x00
Event Flags       = 0x10
Category ID       = 0x05
Category Count    = 0x01
Notification UID  = 0x00000001
```

```
Category ID: 0x05
  = Schedule
```

Figure 2.6 Receive notification from iOS device

3. Example of getting Notification Attribute using “get_att” command is shown below.

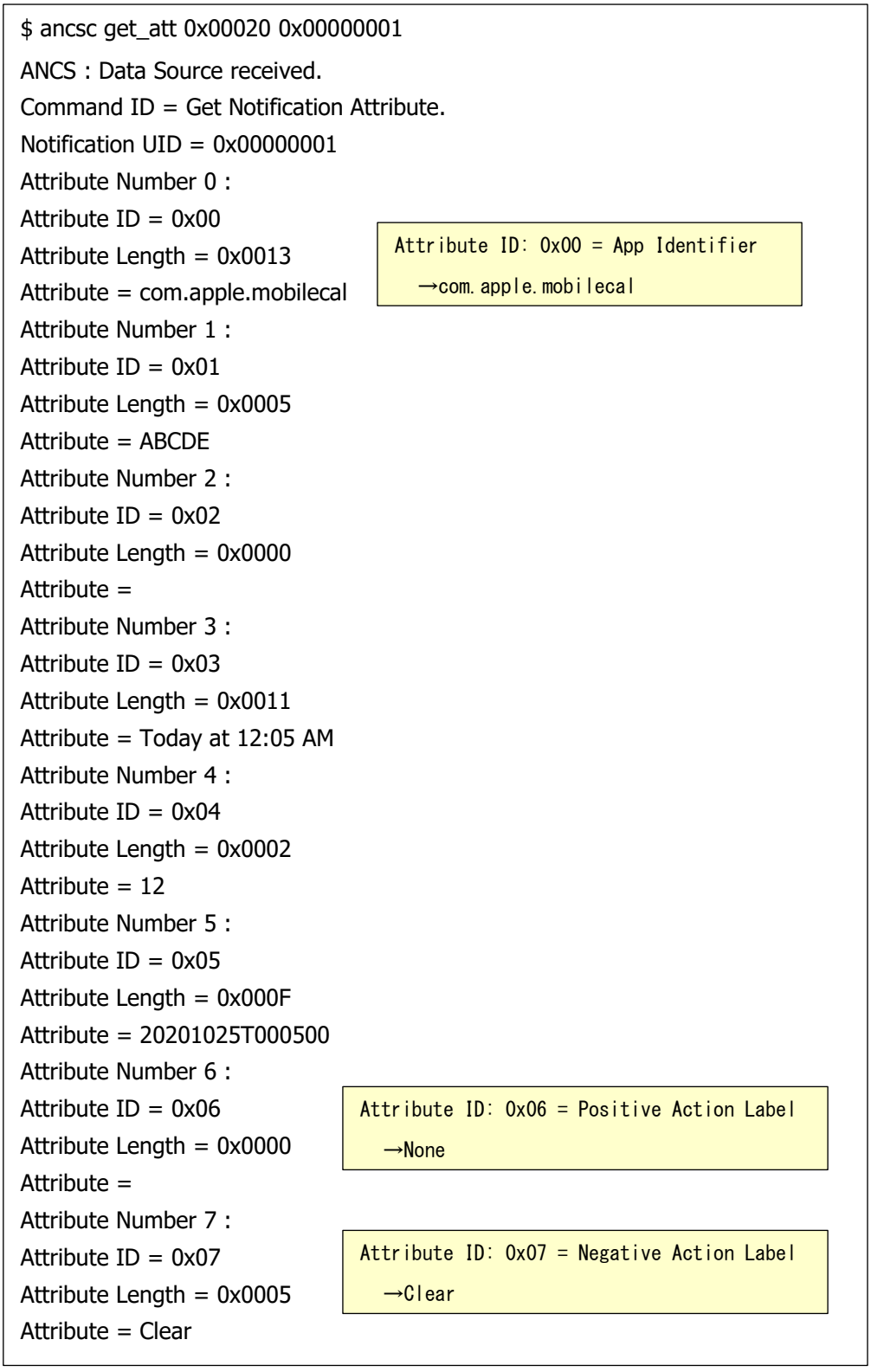

Figure 2.7 Get Notification Attributes operation

4. Example of getting Application Attribute using “get_app” command and App Identifier from procedure 3 is shown below.

```
$ ancsc get_app 0x0020 com.apple.mobilecal
ANCS : Data Source received.
Command ID = Get App Attribute.
App Identifier = com.apple.mobilecal
Attribute Number 0 :
Attribute ID = 0x00
Attribute Length = 0x0008
Attribute = Calendar
```

Figure 2.8 Get App Attributes operation

5. If the Notification Attribute obtained in procedure 3 defines the behavior of iOS notification in Positive Action Label or Negative Action Label, the behavior can be performed using “ntf_act” command. Example of erasing iOS notification using “ntf_act” command is shown below.

```
$ ancsc ntf_act 0x0020 0x00000001 neg
ANCS : Notification Source received.
Event ID          = 0x02
Event Flags       = 0x10
Category ID       = 0x05
Category Count    = 0x01
Notification UID  = 0x00000001
```

```
Event ID: 0x02
  = Notification Removed
```

Figure 2.9 Perform Notification Action operation

2.3 ANCS command

This section describes the command defined in sample project. These commands can be used by typing it into the terminal emulator after connecting with iOS device.

get_att command		
Format:	ancsc get_att [conn_hdl] [Notification UID]	
Description:	Get Notification Attributes operation will be performed. After executing this command, Write Request of Control Point characteristic with command of Get Notification Attributes will be sent. Following data will be sent. Command ID: 0x00 Notification UID: 0xXXXXXXXX Attribute ID & Length: 0x00, 0x01, 0xXXXX*1, 0x02, 0xXXXX*1, 0x03, 0xXXXX*1, 0x04, 0x05, 0x06, 0x07 *1: ANCS_STRING_LEN will be used as data length.	
Parameter:	[conn_hdl]	Specifies the Connection Handle to send Write Request.
	[Notification UID]	Specifies the Notification UID. Notification UID needs to be specified from received Notification Source.
Example:	$ ancsc get_att 0x0020 0x00000001 Perform Get Notification Attributes operation to iOS notification with Notification UID = 0x00000001 on device connected by Connection Handle = 0x0020.	

get_app command		
Format:	ancsc get_app [conn_hdl] [App Identifier]	
Description:	Get App Attributes operation will be performed. After executing this command, Write Request of Control Point characteristic with command of Get App Attributes will be sent. Following data will be sent. Command ID: 0x01 App Identifier: “com.apple.XXX” Attribute ID: 0x00	
Parameter:	[conn_hdl]	Specifies the Connection Handle to send Write Request.
	[App Identifier]	Type App Identifier as string. App Identifier needs to be specified from received Data Source by Get Notification Attributes operation. The maximum length of string is ANCS_STRING_LEN. Refer [*Bundle IDs for native iOS and iPadOS apps in mobile device management*] for valid values.
Example:	$ ancsc get_app 0x0020 com.apple.mobilecal Perform Get App Attributes operation to app specified with App Identifier = “com.apple.mobilecal” on device connected by Connection Handle = 0x0020.	

ntf_act command		
Format:	ancsc ntf_act [conn_hd] [Notification UID] [pos/neg]	
Description:	Perform Notification Action operation will be performed. After executing this command, Write Request of Control Point characteristic with command of Perform Notification Action will be sent. Following data will be sent. Command ID: 0x02 Notification UID: 0xXXXXXXXX Action ID: 0xXX	
Parameter:	[conn_hdl]	Specifies the Connection Handle to send Write Request.
	[Notification UID]	Specifies the Notification UID. Notification UID needs to be specified from received Notification Source.
	[pos/neg]	Select behavior performed by Perform Notification Action Operation. The behavior performed by [pos/neg] varies depending on the iOS notification. Check behavior with Get Notification Attributes operation. pos: Perform “Positive Action Label” behavior neg: Perform “Negative Action Label” behavior
Format:	$ ancsc ntf_act 0x0020 0x00000001 pos Perform Perform Notification Action operation which specifies Positive Action to iOS notification with Notification UID = 0x00000001 on device connected by Connection Handle = 0x0020.	

3. Program of ANCS sample program

This chapter describes the program of ANCS sample project.

3.1 ANCS Program

This section describes API defined in r_ble_ANCSc.h.

3.1.1 Overview of ANCS

Table 3.1 shows UUIDs and Attribute Properties defined in ANCS.

Table 3.1 Service information of ANCS

Service	UUID	Properties
Apple Notification Center Service	7905F431-B5CE-4E99-A40F-4B1E122D00D0	---
Characteristic	**UUID**	**Properties**
Notification Source	9FBF120D-6301-42D9-8C58-25E699A21DBD	Notification
Control Point	69D1D8F3-45E1-49A8-9821-9BBDFDAAD9D9	Write
Data Source	22EAC6E9-24D6-4BB5-BE44-B36ACE7C7BFB	Notification

When iOS notification is issued, the Bluetooth device is notified as the Notification of Notification Source. For more information about iOS notification, Bluetooth device use Write of Control Point to send what information you need from iOS device and receive Notification of Data Source as response. Pairing is required in communication using ANCS. Since Notification Source and Data Source operates Notification, you must access the Client Characteristic Configuration Descriptor (hereinafter call “CCCD”) to allow Notification when data communication is ready.

3.1.2 API of service

This section describes API used as a whole service in ANCS.

Table 3.2 ANCS service function

Init function		
Definition:	R_BLE_ANCSC_Init(ble_servc_app_cb_t cb)	
Description:	Initialize ANCS and register the callback function. ANCS characteristic events will be notified to registered callback function.	
Argument:	cb	Callback function to be registered

ServDiscCb function		
Definition:	R_BLE_ANCSC_ServDiscCb(uint16_t conn_hdl, uint8_t serv_idx, uint16_t type, void *p_param)	
Description:	ANCS callback function for service discovery procedure. This function is used as one of parameter in R_BLE_DISC_Start function.	
Argument:	conn_hdl	Connection handle
	serv_idx	Service index
	type	Event type
	p_param	Pointer to event parameter

Getattlhdl function		
Definition:	R_BLE_ANCSC_GetServAttrHdl(const st_ble_dev_addr_t *p_addr, st_ble_gatt_hdl_range_t *p_hdl)	
Description:	Gets attribute handle of ANCS.	
Argument:	p_addr	Pointer to BD address of target device to get attribute handle
	p_hdl	Pointer to store attribute handle

When an error occurs in GATT communication, GATTC event “BLE_GATTC_EVENT_ERROR_RSP” is notified to application which has error code as data. If error occurs in the ANCS data communication, the error code specified by ANCS may notified with this event. **Table 3.3** shows the ANCS error codes.

Table 3.3 ANCS error code

Error code	Description
BLE_ANCSC_UNKNOWN_COMMAND_ERROR Value: 0xA0	Value specified with Command ID is wrong.
BLE_ANCSC_INVALID_COMMAND_ERROR Value: 0xA1	Data format of “Control Point” is wrong.
BLE_ANCSC_INVALID_PARAMETER_ERROR Value: 0xA2	Parameter such as Notification UID do not match with iOS notification.
BLE_ANCSC_ACTION_FAILED_ERROR Value: 0xA3	Action specified by Command ID was not performed.

3.1.3 API of Notification Source

This section describes API of Notification Source characteristic. Refer section 3.2.5 and 3.2.6 for example.

Table 3.4 Notification Source function

Getattlhdl function		
Definition:	R_BLE_ANCSC_GetNsAttrHdl(const st_ble_dev_addr_t *p_addr, st_ble_ANCSc_ns_attr_hdl_t *p_hdl)	
Description:	Gets attribute handle of Notification Source.	
Argument:	p_addr	Pointer to BD address of target device to get attribute handle
	p_hdl	Pointer to store attribute handle

Read function for Client Characteristic Configuration Descriptor		
Definition:	R_BLE_ANCSC_ReadNsCliCnfg(uint16_t conn_hdl)	
Description:	Send Read Request to the CCCD of Notification Source. “BLE_ANCSC_EVENT_NS_CLI_CNFG_READ_RSP” event occurs when Read Response is received.	
Argument:	conn_hdl	Connection handle

Write function for Client Characteristic Configuration Descriptor		
Definition:	R_BLE_ANCSC_WriteNsCliCnfg(uint16_t conn_hdl, const uint16_t *p_value)	
Description:	Send Write Request to the CCCD of Notification Source. To receive Notification of Notification Source, you must access CCCD to allow notification using this API. “BLE_ANCSC_EVENT_NS_CLI_CNFG_WRITE_RSP” event occurs when Write Response is received.	
Argument:	conn_hdl	Connection handle
	p_value	Pointer to data sending with Write Request

Table 3.5 Notification Source event

Event	Description
BLE_ANCSC_EVENT_NS_HDL_VAL_NTF	Receive Notification from Notification Source. Data : ble_ancsc_ns_t Refer **Table 3.6** for data format.
BLE_ANCSC_EVENT_NS_CLI_CNFG_READ_RSP	Receive Read Response of CCCD included in Notification Source. Data : uint16_t
BLE_ANCSC_EVENT_NS_CLI_CNFG_WRITE_RSP	Receive Write Response of CCCD included in Notification Source. Data : None

Table 3.6 Notification Source data structure

Structure :	st_ble_ancsc_ns_t	
Member variable :	uint8_t	eventid
	uint8_t	eventflag
	uint8_t	categoryid
	uint8_t	categorycount
	uint32_t	notificationuid

Data structure of Notification Source is always same and decode function is implemented. Therefore, the application can use the data in the format above.

3.1.4 API of Control Point

This section describes API of Control Point characteristic. Refer section 3.2.7 for example.

Table 3.7 Control Point function

Getattlhdl function		
Definition:	R_BLE_ANCSC_GetCpAttrHdl(const st_ble_dev_addr_t *p_addr, st_ble_ANCSc_cp_attr_hdl_t *p_hdl);	
Description:	Gets attribute handle of Control Point	
Argument:	p_addr	Pointer to BD address of target device to get attribute handle
	p_hdl	Pointer to store attribute handle

Write function		
Definition:	R_BLE_ANCSC_WriteCp(uint16_t conn_hdl, const st_ble_seq_data_t *p_value);	
Description:	Send Write Request to the CCCD of Control Point. “BLE_ANCSC_EVENT_CP_WRITE_RSP” event occurs when Write Response is received.	
Argument:	conn_hdl	conn_hdl
	p_value	Pointer to sending data. Refer “**Control Point data structure**” for format.

Table 3.8 Control Point Event

Event	Description
BLE_ANCSC_EVENT_CP_WRITE_RSP	Receive Write Response from Control Point. Data: None

Control Point data structure

Control Point has different data format depending on use case. Therefore, R_BLE_ANCSC_WriteCp, the Write API of Control Point characteristic, has “st_ble_seq_data_t” which is data format for data with variable length is used as argument.

Application must store data in available format for each use cases. Refer [*Apple Notification Center Service (ANCS) Specification*] for more information about data format.

3.1.5 API of Data Source

This section describes API of Data Source characteristic. Refer section 3.2.5 and 3.2.7 for example.

Table 3.9 Data Source function

Getattlhdl function		
Definition:	R_BLE_ANCSC_GetDsAttrHdl(const st_ble_dev_addr_t *p_addr, st_ble_ANCSc_ds_attr_hdl_t *p_hdl)	
Description:	Gets attribute handle of Data Source.	
Argument:	p_addr	Pointer to BD address of target device to get attribute handle
	p_hdl	Pointer to store attribute handle

Read function for Client Characteristic Configuration Descriptor		
Definition:	R_BLE_ANCSC_ReadDsCliCnfg(uint16_t conn_hdl)	
Description:	Send Read Request to the CCCD of Data Source. “BLE_ANCSC_EVENT_DS_CLI_CNFG_READ_RSP” event occurs when Read Response is received.	
Argument:	conn_hdl	Connection handle

Write function for Client Characteristic Configuration Descriptor		
Definition:	R_BLE_ANCSC_WriteDsCliCnfg(uint16_t conn_hdl, const uint16_t *p_value)	
Description:	Send Write Request to the CCCD of Data Source. To receive Notification of Data Source, you must access CCCD to allow notification using this API. “BLE_ANCSC_EVENT_DS_CLI_CNFG_WRITE_RSP” event occurs when Write Response is received.	
Argument:	conn_hdl	Connection handle
	p_value	Pointer to data sending with Write Request

Table 3.10 Data Source event

Event	Description
BLE_ANCSC_EVENT_DS_HDL_VAL_NTF	Receive Notification from Data Source. Data: st_ble_seq_data_t Refer “**Data Source data structure**” for format.
BLE_ANCSC_EVENT_DS_CLI_CNFG_READ_RSP	Receive Read Response of CCCD included in Data Source. Data: uint16_t
BLE_ANCSC_EVENT_DS_CLI_CNFG_WRITE_RSP	Receive Write Response of CCCD included in Data Source. Data: None

Data Source data structure

Data Source has different data format depending on use case. Therefore, BLE_ANCSC_EVENT_DS_HDL_VAL_NTF, the Notification receive event of Data Source characteristic, has “st_ble_seq_data_t” which is data format for data with variable length is used as event data.

Application must store data in available format for each use cases. Refer [*Apple Notification Center Service (ANCS) Specification*] for more information about data format.

3.2 Application Program

Sample project provides program to connect with iOS device using RX23W and perform data communication of ANCS client role over Bluetooth LE communication. This section describes behavior of program and how to change them.

Sample project consists of app_main.c which is base of user application, GATT database, API and event code for accessing ANCS. From next section, ANCS sample application mainly implemented in app_main.c is described.API and event code for accessing ANCS is described in [3.1 ANCS Program].

3.2.1 Initialization

Initialization of ANCS is executed by R_BLE_ANCSC_Init() API. This API is called in ble_init() defined in app_main.c. Therefore, no additional processing is required by the user.

```
static ble_status_t ble_init(void)
{
    ble_status_t status;
……
    /* Initialize Apple Notification Center Service client API */
    status = R_BLE_ANCSC_Init(ANCSc_cb);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }
    return status;
}
```

Code 3.1 Initialization

3.2.2 Enable Resolvable Private Address function

After initialization is completed, sample project enables Resolvable Private Address (RPA) function.

app_main.c/gap_cb performs the following:

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
.....
    switch(type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            /* Set device Privacy mode */
            st_ble_dev_addr_t lc_id_addr;
            uint8_t lc_irk[BLE_GAP_IRK_SIZE];
            uint8_t lc_csrk[BLE_GAP_CSRK_SIZE];
            uint8_t irk_check[BLE_GAP_IRK_SIZE];

            ble_sta = R_BLE_SECD_ReadLocInfo(&lc_id_addr, lc_irk, lc_csrk);
            memset(irk_check, 0x00, BLE_GAP_IRK_SIZE);

            if((ble_sta == BLE_SUCCESS) && (0 != memcmp(irk_check, lc_irk, BLE_GAP_IRK_SIZE)))
            {
                R_BLE_ABS_SetLocPrivacy(lc_irk, BLE_ABS_PRIV_NET_STATIC_IDADDR);
            }
            else
            {
                R_BLE_ABS_SetLocPrivacy(NULL, BLE_ABS_PRIV_NET_STATIC_IDADDR);
            }
        } break;
.....
}
```

Read local IRK of local device from data flash, and set privacy mode to Network Privacy mode using abstraction API.

Code 3.2 Enable RPA function

3.2.3 Start Advertising

After RPA is enabled, start advertising to establish connection with iOS device. To resolve RPA of remote device, add bonding information stored in data flash to Resolving List.

app_main.c/gap_cb performs the following:

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
.....
    switch(type)
    {
.....
        case BLE_GAP_EVENT_RPA_EN_COMP:
        {
            st_ble_dev_addr_t idaddr_list[BLE_CFG_NUM_BOND + 1] = {0};
            st_ble_gap_rslv_list_key_set_t key_set_list[BLE_CFG_NUM_BOND + 1] = {0};
            uint8_t d_cnt;

            R_BLE_SECD_GetIdInfo(idaddr_list, key_set_list, &d_cnt);

            /* Start Advertising if data not stored in data flash */
            if(0 == d_cnt)
            {
                R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
            }
            else
            {
#if ANCS_RPA_ENABLE
                uint8_t i;
                uint8_t is_included_dummy;

                is_included_dummy = 0;
                /* check registration dummy remote address & irk */
                for(i=0; i<d_cnt; i++)
                {
                    if(0 == memcmp(&idaddr_list[i], &gs_dummy_addr, sizeof(st_ble_dev_addr_t)))
                    {
                        is_included_dummy = 1;
                        break;
                    }
                }
                if(0 == is_included_dummy)
                {
                    /* add dummy remote */
                    idaddr_list[d_cnt] = gs_dummy_addr;
                    memset(key_set_list[d_cnt].remote_irk, 0x00, BLE_GAP_IRK_SIZE);
                    key_set_list[d_cnt].local_irk_type = BLE_GAP_RL_LOC_KEY_REGISTERED;
                    d_cnt++;
                }
#endif
                /* register remote address & irk */
                gs_app_rsv_list = true;
                R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV, idaddr_list, key_set_list, d_cnt);
            }
        } break;

        case BLE_GAP_EVENT_RSLV_LIST_CONF_COMP:
        {
            /* Start Advertising after setting resolving list */
            if(true == gs_app_rsv_list)
            {
                R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
                gs_app_rsv_list = false;
            }
        } break;
```

Read Identity Address and Remote IRK of bonded remote device from data flash.

Start Advertising if there is no bonding information.

Check for dummy address being included to bonding information list. If not, add it to list.

After registration of Resolving List, Start Advertising.

Code 3.3 Start Advertising

3.2.4 Service Discovery

When the connection with iOS device is established by the operation of [2.2.1 Connection with iOS Device], sample project starts service discovery procedure. The code for starting service discovery procedure is implemented in app_main.c/gattc_cb BLE_GATTC_EVENT_CONN_IND event process. The R_BLE_DISC_Start API which is API for starting service discovery procedure is included in BLE FIT module.

```
static void gattc_cb(uint16_t type, ble_status_t result, st_ble_gattc_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
……
      switch(type)
      {
            case BLE_GATTC_EVENT_CONN_IND:
            {
                  /* Start discovery operation after connection established */
                  R_BLE_DISC_Start(p_data->conn_hdl, gs_disc_entries,
                                    ARRAY_SIZE(gs_disc_entries), disc_comp_cb);
.........
            } break;
.........
}
```

Start service discovery after connection with iOS device has established.

Code 3.4 Start Service Discovery

3.2.5 Pairing and enable Notification

In app_main.c/disc_comp_cb function, which is callback function called when service discovery has completed, sample project sends Write Request to CCCD of Notification Source to iOS device. If pairing with iOS device has completed, 0x01 is written to CCCD of Notification Source.

```
static void disc_comp_cb(uint16_t conn_hdl)
{
/* Hint: Input process such as GATT operation */
……
      /* Enabling CCCD */
      /* If pairing is necessary, BLE_GATTC_EVENT_ERROR_RSP will happen and pairing will
         perform at the event */
      uint16_t cccd_value = BLE_GATTS_CLI_CNFG_NOTIFICATION;
      R_BLE_ANCSC_WriteNsCliCnfg(conn_hdl, &cccd_value);

      /* End user code. Do not edit comment generated here */
      return;
}
```

Send Write Request to CCCD of Notification Source.

Code 3.5 Service Discovery Completed

In case of connection established with iOS device was for the first time, pairing with the iOS device has not been completed when service discovery procedure has finished. In such case, BLE_GATTC_EVENT_ERROR_RSP event is respond for the Write Request of CCCD. This event means that the security requirements required to access characteristic of iOS device are not sufficient, so pairing is needed. Sample project starts pairing with iOS device using R_BLE_ABS_StartAuth API from abstraction API in the error response.

```
static void gattc_cb(uint16_t type, ble_status_t result, st_ble_gattc_evt_data_t *p_data)
{
    switch(type)
    {
……
        case BLE_GATTC_EVENT_ERROR_RSP:
        {
            st_ble_gattc_err_rsp_evt_t *err_rsp_data =
              (st_ble_gattc_err_rsp_evt_t *)p_data->p_param;

            /* Start Authentication if CCCD write is error */
            st_ble_ANCSc_ns_attr_hdl_t ns_hdl;
            R_BLE_ANCSC_GetNsAttrHdl(&g_conn_bd_addr, &ns_hdl);
            if(err_rsp_data->attr_hdl == ns_hdl.cli_cnfg_desc_hdl)
            {
                R_BLE_ABS_StartAuth(p_data->conn_hdl);
            }
}
```

Start Pairing in error response.

Code 3.6 Start Pairing

BLE_GAP_EVENT_PEER_KEY_INFO event occurs when pairing progresses and the key exchange has completed. In this event, you can get LTK, IRK, and Identity Address of iOS device. To resolve RPA of iOS device, register IRK and Identity Address to Resolving List using R_BLE_GAP_Conf_RslvList API.

BLE_GAP_EVENT_ENC_CHG event occurs when pairing has completed. In this event, Write Request to CCCD of Notification Source is sent.

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_gattc_evt_data_t *p_data)
{
    switch(type)
    {
……
        case BLE_GAP_EVENT_PEER_KEY_INFO:
        {
            /* Add remote device information to resolving list */
            st_ble_gap_peer_key_info_evt_t *p_peer_key_info_param =
                        (st_ble_gap_peer_key_info_evt_t *)p_data->p_param;

            memcpy(key_set.remote_irk,
                    p_peer_key_info_param->key_ex_param.p_keys_info->id_info,
                    BLE_GAP_IRK_SIZE);
            key_set.local_irk_type = BLE_GAP_RL_LOC_KEY_REGISTERED;
            memcpy(remote_id_addr.addr,
                      &p_peer_key_info_param->key_ex_param.p_keys_info->id_addr_info[1],
                      BLE_BD_ADDR_LEN);
            remote_id_addr.type = p_peer_key_info_param->key_ex_param.p_keys_info->id_addr_info[0];

            R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV, &remote_id_addr, &key_set, 1);
        } break;

……
}
```

Register Key information to Resolving List.

Code 3.7 Register key information

After Writing to CCCD of Notification Source has completed, send Write Request of CCCD included in Data Source for enabling notification from iOS device.

```
static void ANCSc_cb(uint16_t type, ble_status_t result, st_ble_servc_evt_data_t *p_data)
{
    switch(type)
    {
……
        case BLE_ANCSC_EVENT_NS_CLI_CNFG_WRITE_RSP:
        {
            if (BLE_SUCCESS == result)
            {
                /* Enable Notification of Data Source */
                uint16_t cccd_value = BLE_GATTS_CLI_CNFG_NOTIFICATION;
                R_BLE_ANCSC_WriteDsCliCnfg(p_data->conn_hdl, &cccd_value);
            }
        } break;

        case BLE_ANCSC_EVENT_DS_CLI_CNFG_WRITE_RSP:
        {
            if (BLE_SUCCESS == result)
            {
                /* ANCS Notification is Enabled */
                R_BLE_CLI_Printf("ANCS : Notification Enabled \n");
            }
        } break;
……
    }
}
```

Start write to CCCD of Data Source when write to CCCD of Notification Source has completed.

Notify ANCS is ready when write to CCCD of Data Source has completed.

Code 3.8 Allow Notification

3.2.6 Receive iOS notification with Notification Source

When RX23W receives Notification from Notification Source in iOS device, BLE_ANCSC_EVENT_NS_HDL_VAL_NTF event occurs. The event process needs to be implemented in app_main.c/ANCSc_cb. If generated from QE for BLE, the user must implement the event process by yourself. In sample project, each member variable of received Notification Source is displayed in the terminal emulator.

```
static void ANCSc_cb(uint16_t type, ble_status_t result, st_ble_servc_evt_data_t *p_data)
{
……
    switch(type)
    {
……
        case BLE_ANCSC_EVENT_NS_HDL_VAL_NTF:
        {
            if(BLE_SUCCESS == result)
            {
                /* Display Notification Source */
                st_ble_ancsc_ns_t *ns_ntf_data = (st_ble_ancsc_ns_t *)p_data->p_param;

                  R_BLE_CLI_Printf("ANCS : Notification Source received.\n");
                R_BLE_CLI_Printf("Event ID                = 0x%02X\n",ns_ntf_data->eventid);
                R_BLE_CLI_Printf("Event Flags             = 0x%02X\n",ns_ntf_data->eventflags);
                R_BLE_CLI_Printf("Category ID             = 0x%02X\n",ns_ntf_data->categoryid);
                R_BLE_CLI_Printf("Category Count          = 0x%02X\n",ns_ntf_data->categorycount);
                R_BLE_CLI_Printf("Notification UID    = 0x%08X\n",ns_ntf_data->notificationuid);
            }
        } break;
……
```

Receive event data with [st_ble_ancsc_ns_t] format.

Display each data.

Code 3.9 Receive Notification Source

3.2.7 Data communication using Control Point and Data Source

You can use the information of iOS notification received in [3.2.6 Receive iOS notification with Notification Source] to get more information about iOS notification or perform certain action to iOS notification. If getting more information about the iOS notification is written, further information is notified from Data Source characteristic. This section describes how to get Notification Attribute as example.

To get Notification Attributes, send Write Request including get Notification Attributes command to Control Point characteristic using R_BLE_ANCSC_WriteCP API. Sample project implements this process in cmd_ancsc_get_att function which is called when get_att command is executed.

```
static void cmd_ancsc_get_att(int argc, char *argv[])
{
    uint16_t conn_hdl;
    uint8_t ntf_data[19];
    st_ble_seq_data_t seq_cp_ntf_att_data=
    {
        .len = 19,
        .data = ntf_data
    };
    st_ble_ancsc_cp_ntf_att_t    ntf_att_data;

    conn_hdl = (uint16_t)strtol(argv[1], &str_check, 0);
    if(*str_check != '\0')
    {
        R_BLE_CLI_Printf("ancsc %s: wrong parameter\n", argv[0]);
        return;
    }

    ntf_att_data.notification_uid = (uint32_t)strtol(argv[2], &str_check, 0);
    if(*str_check != '\0')
    {
        R_BLE_CLI_Printf("ancsc %s: wrong parameter\n", argv[0]);
        return;
    }

    /* Set value to control point for get notification attribute */
    ntf_att_data.command_id = BLE_ANCSC_CPDS_COMMANDID_GETNOTIFICATIONATTRIBUTE;
    BT_PACK_LE_1_BYTE(&ntf_data[0], &ntf_att_data.command_id);

    BT_PACK_LE_4_BYTE(&ntf_data[1], &ntf_att_data.notification_uid);

    ntf_att_data.id_app_id = BLE_ANCSC_CPDS_NOTIFICATIONATTRIBUTEID_APPIDENTIFIER;
    BT_PACK_LE_1_BYTE(&ntf_data[5], &ntf_att_data.id_app_id);

    ntf_att_data.id_title = BLE_ANCSC_CPDS_NOTIFICATIONATTRIBUTEID_TITLE;
    ntf_att_data.id_title_len = ANCS_STRING_LEN;
    BT_PACK_LE_1_BYTE(&ntf_data[6], &ntf_att_data.id_title);
    BT_PACK_LE_2_BYTE(&ntf_data[7], &ntf_att_data.id_title_len);
.....
    /* Write Command */
    result = R_BLE_ANCSC_WriteCp(conn_hdl, &seq_cp_ntf_att_data);
.....
}
```

Change command argument to number.

Store Control Point data in format of [st_ble_seq_data_t].

Send Write Request of Control Point.

Code 3.10 Write Control Point

After get Notification Attribute command is written to Control Point characteristic, iOS device sends Notification from Data Source characteristic. When Notification is received, BLE_ANCSC_EVENT_DS_HDL_VAL_NTF event is notified to application. Sample project implements this process in app_main.c/ANCSc_cb. If generated from QE for BLE, the user must implement the event process by yourself. In sample project, received data is displayed in the terminal emulator.

```
static void ANCSc_cb(uint16_t type, ble_status_t result, st_ble_servc_evt_data_t *p_data)
{
    switch(type)
    {
……
    case BLE_ANCSC_EVENT_DS_HDL_VAL_NTF:
    {
        if(BLE_SUCCESS == result)
        {
            st_ble_seq_data_t *ds_ntf_data = (st_ble_seq_data_t *)p_data->p_param;

            /* Display Data Source for Get Notification Attribute Command */
            if(BLE_ANCSC_CPDS_COMMANDID_GETNOTIFICATIONATTRIBUTE == ds_ntf_data->data[0])
            {
                uint16_t pos = 1;
                uint32_t ntf_uid;
                uint8_t attr_count = 0;

                R_BLE_CLI_Printf("ANCS : Data Source received.\n");
                R_BLE_CLI_Printf("Command ID = Get Notification Attribute.\n");

                BT_UNPACK_LE_4_BYTE(&ntf_uid, &ds_ntf_data->data[pos]);
                R_BLE_CLI_Printf("Notification UID = 0x%08X\n", ntf_uid);
                pos = pos + 4;

                /* Display Each Attribute list */
                while(pos < ds_ntf_data->len)
                {
                    st_ble_ancsc_attribute_data_t attr_data;

                    R_BLE_CLI_Printf("Attribute Number %d :\n", attr_count);

                    BT_UNPACK_LE_1_BYTE(&attr_data.id, &ds_ntf_data->data[pos]);
                    R_BLE_CLI_Printf("Attribute ID = 0x%02X\n", attr_data.id);
                    pos++;

                    BT_UNPACK_LE_2_BYTE(&attr_data.length, &ds_ntf_data->data[pos]);
                    R_BLE_CLI_Printf("Attribute Length = 0x%04X\n", attr_data.length);
                    pos = pos + 2;

                    attr_data.data = &ds_ntf_data->data[pos];
                    R_BLE_CLI_Printf("Attribute = ");
                    for(uint16_t i = 0; i < attr_data.length; i++)
                    {
                        R_BLE_CLI_Printf("%c", attr_data.data[i]);
                    }
                    R_BLE_CLI_Printf("\n");
                    pos = pos + attr_data.length;

                    attr_count++;
                }
            }……
    }
}
```

Receive event data in format of [st_ble_seq_data_t]

Display [st_ble_seq_data_t] format data while decoding.

Code 3.11 Receive Control Point

Get App Attributes operation and Perform Notification Action operation are also implemented in sample project in similar codes. For Get App Attributes operation, sending Write Request to Control Point process is implemented in cmd_ancsc_get_app function, and receiving Notification from Data Source process is implemented in BLE_ANCSC_EVENT_DS_HDL_VAL_NTF event in app_main.c/ANCSc_cb. For Perform Notification Action operation, sending Write Request to Control Point process is implemented in cmd_ancsc_ntf_act function.

3.3 Development using QE for BLE

Sample project is developed based on programs generated using QE for BLE's custom service generation feature. When adding other services to project, QE for BLE makes development easier. For instructions about usage of QE for BLE, refer [*RX23W Group Bluetooth Low Energy Profile Developer's Guide (R01AN4553)*].
To create ANCS client role, import Release/ancs_client.json using import function of QE for BLE.

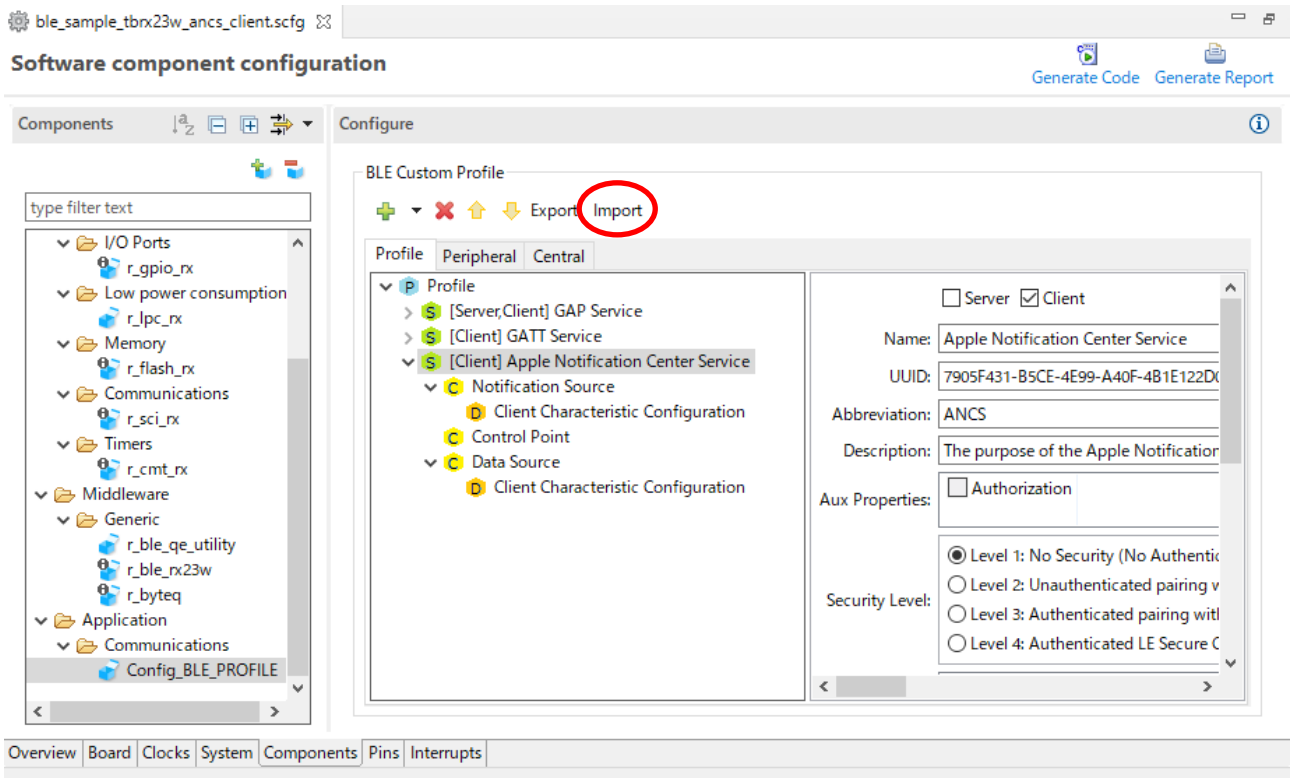

Figure 3.1 Screen of QE for BLE

After code generation, please add following code to r_ble_ANCSc.h. For more information about data format or value used in ANCS, refer [*Apple Notification Center Service (ANCS) Specification*].

```
/**********************************************************************************//**
  * @brief Control Point/Data Source Command ID enumeration.
**********************************************************************************/
typedef enum {
      BLE_ANCSC_CPDS_COMMANDID_GETNOTIFICATIONATTRIBUTE = 0, /**< Get notification attribute Command */
      BLE_ANCSC_CPDS_COMMANDID_GETAPPATTRIBUTE = 1, /**< Get app attribute Command */
      BLE_ANCSC_CPDS_COMMANDID_PERFORMNOTIFICATIONACTION = 2, /**< Perform notification action Command
*/
} e_ble_ancsc_cpds_commandid_t;

/**********************************************************************************//**
  * @brief Control Point/Data Source Notification Attribute ID enumeration.
**********************************************************************************/
typedef enum {
      BLE_ANCSC_CPDS_NOTIFICATIONATTRIBUTEID_APPIDENTIFIER = 0, /**< Notifiaction Attribute ID: App Identifier */
      BLE_ANCSC_CPDS_NOTIFICATIONATTRIBUTEID_TITLE = 1, /**< Notifiaction Attribute ID: Title */
      BLE_ANCSC_CPDS_NOTIFICATIONATTRIBUTEID_SUBTITLE = 2, /**< Notifiaction Attribute ID: Sub title */
      BLE_ANCSC_CPDS_NOTIFICATIONATTRIBUTEID_MESSAGE = 3, /**< Notifiaction Attribute ID: Message */
      BLE_ANCSC_CPDS_NOTIFICATIONATTRIBUTEID_MESSAGESIZE = 4, /**< Notifiaction Attribute ID: Message size */
      BLE_ANCSC_CPDS_NOTIFICATIONATTRIBUTEID_DATE = 5, /**< Notifiaction Attribute ID: Date */
      BLE_ANCSC_CPDS_NOTIFICATIONATTRIBUTEID_POSITIVEACTIONLABEL = 6, /**< Notifiaction Attribute ID:
Positive action label */
      BLE_ANCSC_CPDS_NOTIFICATIONATTRIBUTEID_NEGATIVEACTIONLABEL = 7, /**< Notifiaction Attribute ID:
Negative action label */
} e_ble_ancsc_cpds_notificationattributeid_t;

/**********************************************************************************//**
  * @brief Control Point/Data Source App Attribute ID enumeration.
**********************************************************************************/
typedef enum {
      BLE_ANCSC_CPDS_APPATTRIBUTEID_DISPLAYNAME = 0, /**< App Attribute ID: Display name */
} e_ble_ancsc_cpds_appattributeid_t;

/**********************************************************************************//**
  * @brief Control Point/Data Source Action ID enumeration.
**********************************************************************************/
typedef enum {
      BLE_ANCSC_CPDS_ACTIONID_POSITIVE = 0, /**< Positive action */
      BLE_ANCSC_CPDS_ACTIONID_NEGATIVE = 1, /**< Negative action */
} e_ble_ancsc_cpds_actionid_t;
```

Code 3.12 Add value macros

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jun.7.21	-	First edition issued.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.