RX23W グループ

Apple Notification Center Service サンプルプログラム

要旨

本書は Apple Notification Center Service(以下、ANCS と略します)クライアントロールのサンプルアプリケーションに関するアプリケーションノートです。

動作確認デバイス

Target Board for RX23W

関連ドキュメント

- Target Board for RX23W ユーザーズマニュアル(R20UT4634)
- 統合開発環境 e²studio ユーザーズマニュアル入門ガイド(R20UT4819)
- フラッシュ書き込みソフトウェアユーザーズマニュアル(R20UT4813)
- Bluetooth Low Energy プロトコルスタック基本パッケージ(R01UW0205)
- RX23W グループ Bluetooth Low Energy プロファイル開発者ガイド(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)
- iOS 及び iPadOS APP のバンドル ID (https://support.apple.com/ja-jp/guide/mdm/mdm90f60c1ce/web)

目次

1. 概要

ANCS サンプルアプリケーションは、Target Board for RX23W(以下、ターゲットボードと記します)上で動作し、Bluetooth® Low Energy(以下、Bluetooth LE と略します)無線通信で iOS デバイスと接続します。

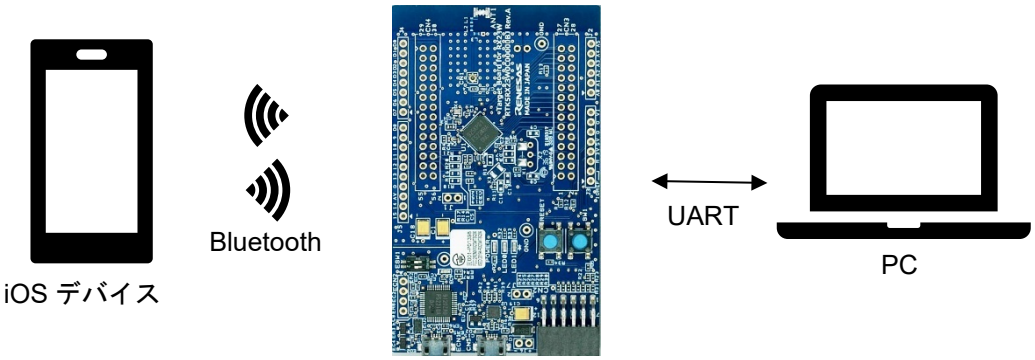

図 1.1　動作環境

ANCS サンプルアプリケーションは Resolvable Private Address(RPA)機能を有効化した後、Advertising を開始します。その後 iOS デバイスと接続を確立します。

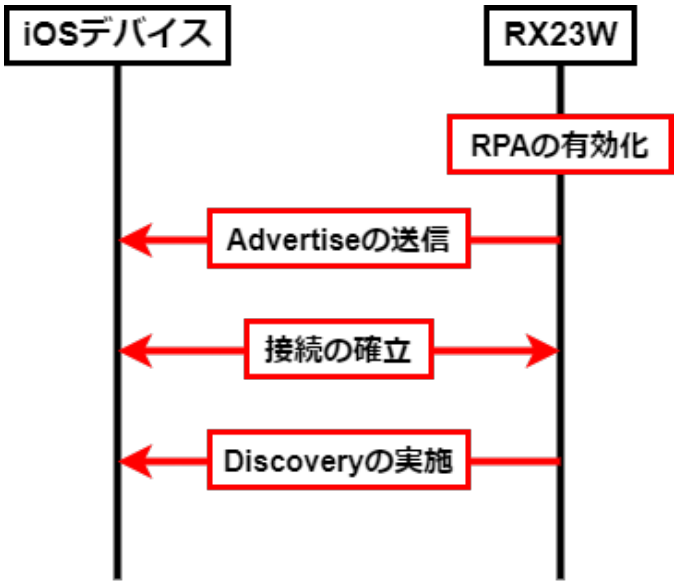

図 1.2　接続時の動作フロー

ANCS を使用するためにはペアリングが必要です。ペアリングを終えると、RX23W は iOS デバイスが出力する様々な通知(以下、iOS 通知と称します)を受信します。また iOS 通知の詳細を iOS デバイスに要求することができます。

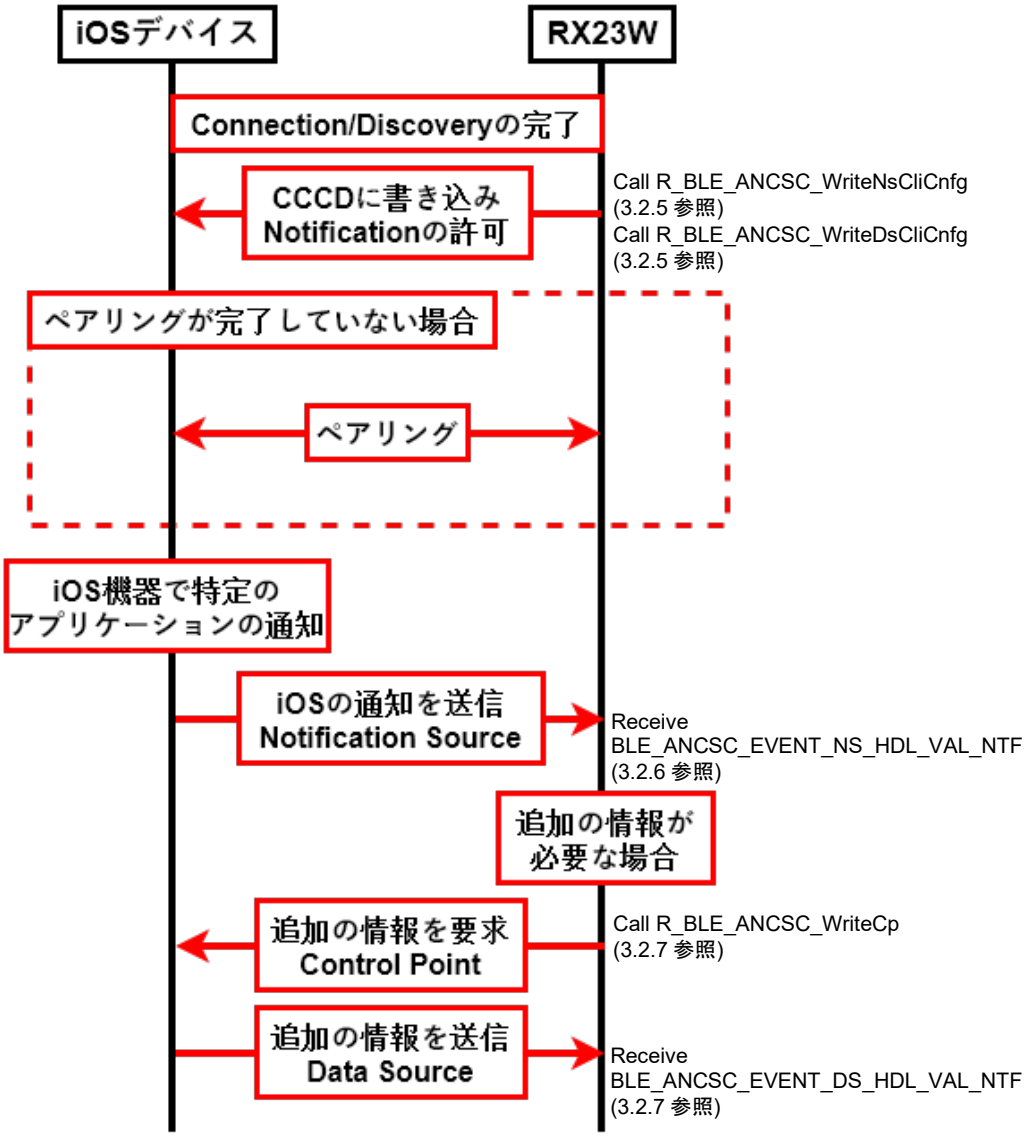

図 1.3　ANCS の動作フロー

1.1　動作環境

本アプリケーションノートで提供する ANCS サンプルアプリケーションは以下のハードウェア並びにソフトウェア環境にて動作確認しています。

表 1.1　動作環境　(ハードウェア)

ハードウェア	型名等
Target Board for RX23W	RTK5RX23W0C00000BJ
USB ケーブル	USB A - Micro B ケーブル
Windows 10 PC	---
iOS デバイス	iOS14.2

表 1.2　動作環境　(ソフトウェア)

ソフトウェア	バージョン等
e^{2}studio	V2021-04
CC-RX コンパイラ	V2.08.01
QE for BLE	V1.10
BLE FIT モジュール	V2.11
Renesas Flash Programmer	V3.08
Tera Term (VT100 互換ターミナルエミュレータ)	UART 設定 Baud rate：115200bps Data：8bit Parity：なし Stop bits：なし Flow Control：なし 文字コード：UTF-8

1.2 ディレクトリ・ファイル構成

本アプリケーションノートは、ANCS クライアントロール用のサンプルプロジェクト(以下、本プロジェクトと記します)を提供します。本プロジェクトの構成は以下の通りです。

表 1.3 ANCS サンプルプロジェクトファイル構成

ディレクトリ・ファイル構成				概要
Release¥	ancs_client.json ble_sample_tbrx23w_ancs_client.mot			QE for BLE 用ファイル ビルド済みの mot ファイル
src¥	smc_gen¥	Config_BLE_PROFILE¥	app_main.c	BLE アプリケーションメインコード
			gatt_db.c gatt_db.h	GATT データベースコード
			r_ble_ANCSc.c r_ble_ANCSc.h	プロファイルコード
		r_ble_rx23w¥		BLE protocol stack のプログラム
		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¥		その他スマート・コンフィグレータから生成されるプログラム
	ble_sample_tbrx23w_ancs_client.c			プロジェクトメインコード
.cproject .project ble_sample_tbrx23w_ancs_client HardwareDebug.launch				CC-RX 用プロジェクトファイル
ble_sample_tbrx23w_ancs_client.scfg				スマート・コンフィグレータ用ファイル

1.3 サンプルプロジェクトのインポート

本プロジェクトは zip ファイルの形式で提供されており、e²studio にインポートすることができます。インポート手順は「*統合開発環境 e²studio ユーザーズマニュアル入門ガイド(R20UT4819)*」を参照下さい。インポートする zip ファイルは本アプリケーションノートに同梱されており、フィル名は以下の通りです。

- ble_sample_tbrx23w_ancs_client.zip

1.4 ファームウェアの書き込み

Release フォルダにはビルド済みの mot ファイルを収めてあります。同 mot ファイルをターゲットボードに書き込み、次節からの手順を実行することで図 1.1 の動作を確認できます。mot ファイルのターゲットボードへの書き込み手順は「*Target Board for RX23W ユーザーズマニュアル(R20UT4634)*」及び「*フラッシュ書き込みソフトウェアユーザーズマニュアル(R20UT4813)*」を参照下さい。利用可能な mot ファイルは以下の通りです。

- ble_sample_tbrx23w_ancs_client.mot

2. ANCS サンプルプロジェクトの動作

本章では本プロジェクトの動作手順について解説します。

2.1 アプリケーションオプション

本プロジェクトを e²studio でビルド・実行する際、app_main.c に定義した以下のマクロを変更することにより、一部動作を変更することができます。

表 2.1 アプリケーションオプション

オプション	説明
ANCS_STRING_LEN	ANCS コマンドなどで使用する文字列データの最大長を設定します。 default: 32 range: 1~244
ANCS_RPA_ENABLE	ローカルデバイスの RPA 生成の有効／無効を設定します。無効の場合、static アドレスを使用します。 0: Disable (default) 1: Enable 【注】本オプションが無効でもリモート RPA の解決は行います。

2.2 ANCS サンプルアプリケーションの動作フロー

本節では ANCS プロジェクトを実行したあとに iOS デバイスと接続し、ANCS のデータ通信を行う手順について解説します。本プロジェクトのソースコードに関する解説は「3.2 アプリケーションのプログラム」を参照してください。

2.2.1 iOS デバイスとの接続

本節では本プロジェクトを書き込んだターゲットボードと iOS デバイスとの接続及びペアリング手順を説明します。本節では iOS デバイス(iOS14.2)を使用して解説します。

1. USB ケーブルの Micro B 側をターゲットボードの CN5 に接続、Type A 側を PC の任意の USB ポートに接続して下さい。

2. PC で VT100 互換ターミナルエミュレータ(以下、ターミナルエミュレータと記します)を起動し、表 1.2 に示す UART 設定を実施して下さい。ターミナルエミュレータとして Tera Term が使用できます。

3. ターゲットボードの RESET ボタンを押下して下さい。押下後、ターゲットボードは Advertising を開始します。Advertising が開始されると、以下の文字列がターミナルエミュレータに表示されます。

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

図 2.1 Advertising の開始表示

4. iOS デバイスの Setting 画面にて Bluetooth を選択してください。DEVICE 欄に現れる”REL_ANCS”をタップして下さい。

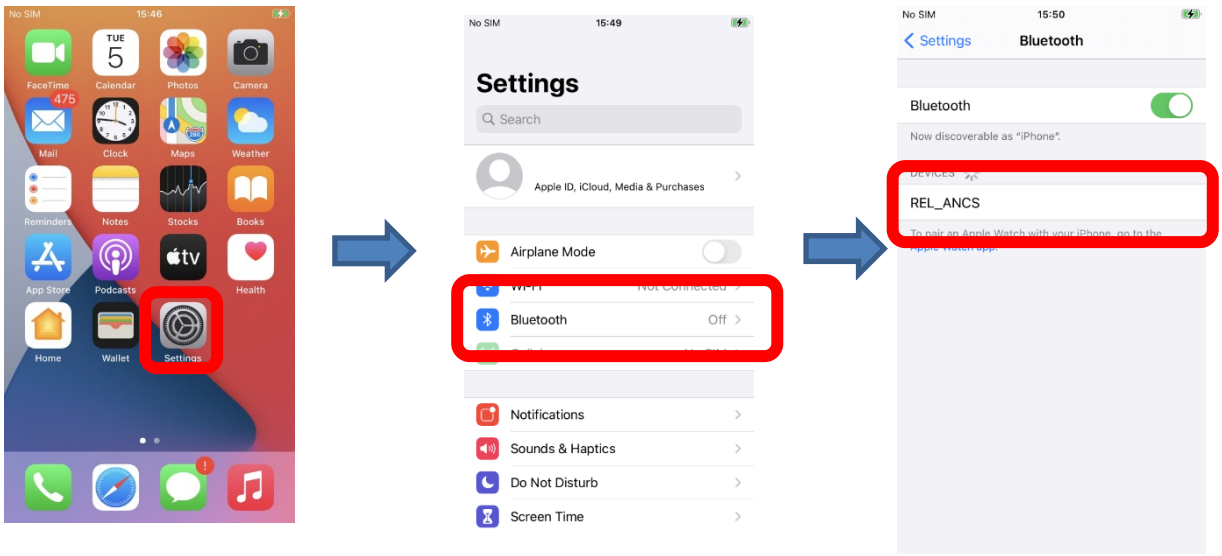

図 2.2　iOS デバイスとターゲットボードの Bluetooth 接続

5. ターゲットボードと iOS デバイスを初めて接続する場合、“REL_ANCS”とのペアリング可否を問うダイアログが現れますので、ペアリングを許可して下さい。

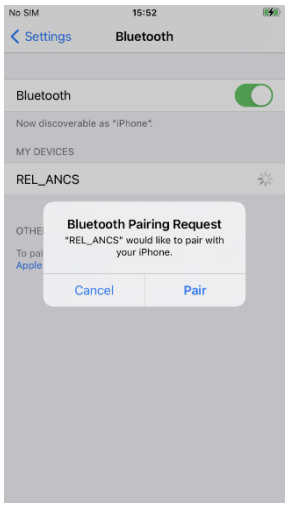

図 2.3　iOS デバイスとターゲットボードのペアリング許可

6. “REL_ANCS”との通知の共有可否を問うダイアログが現れますので、通知の共有を許可して下さい。

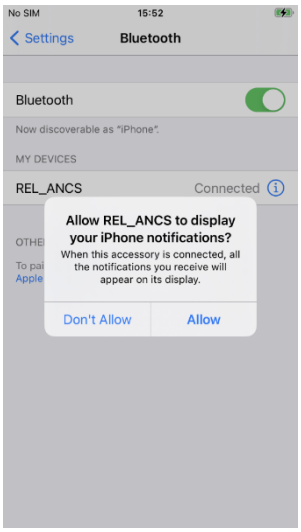

図 2.4　iOS デバイス通知の共有許可

ANCS からの通知が受信可能になると、下記の文字列がターミナルエミュレータに表示されます。

```
ANCS: Notification Enabled.
```

図 2.5　ANCS の準備完了

2.2.2　iOS デバイスからの通知の受信

「2.2.1 iOS デバイスとの接続」の手順で iOS デバイスと接続後、iOS デバイスからスケジュール等の通知(以下、iOS 通知)を受信することができます。また受信した iOS 通知に関する情報を取得したり、iOS 通知に対して操作したりすることも可能です。ここではカレンダーアプリからの通知を例にして本プロジェクトの動作を解説します。Event ID や Event Flag 等の用語の詳細は「*Apple Notification Center Service (ANCS) Specification*」を参照下さい。

1. iOS デバイスにて、カレンダーアプリを起動して下さい。任意の時刻に予定をセットし、予定時刻に至るまで待機して下さい。

2. 1 項で予定をセットした時刻に至ると、iOS デバイスからターゲットボードに iOS 通知が送信されます。ターゲットボードがこの通知を受信するとターゲットボードと接続した PC のターミナルエミュレータに以下の文字列が表示されます。本サンプルアプリケーションは、2.3 節に定義したコマンドを使用して、iOS 通知の詳細や通知の消去等の動作を iOS デバイスに指示できます。次項以降にこのコマンドの使用例を記載します。

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

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

図 2.6　iOS デバイスからの通知受信

3. get_att コマンド使用して Data Source から Notification Attribute を取得する例を以下に示します。

```
$ ancsc get_att 0x0020 0x00000001
ANCS : Data Source received.
Command ID = Get Notification Attribute.
Notification UID = 0x00000001
Attribute Number 0 :
Attribute ID = 0x00
Attribute Length = 0x0013
Attribute = com.apple.mobilecal
Attribute Number 1 :
Attribute ID = 0x01
Attribute Length = 0x0005
Attribute = ABCDE
Attribute Number 2 :
Attribute ID = 0x02
Attribute Length = 0x0000
Attribute =
Attribute Number 3 :
Attribute ID = 0x03
Attribute Length = 0x000C
Attribute = 今日 17:45
Attribute Number 4 :
Attribute ID = 0x04
Attribute Length = 0x0002
Attribute = 12
Attribute Number 5 :
Attribute ID = 0x05
Attribute Length = 0x000F
Attribute = 20201015T174500
Attribute Number 6 :
Attribute ID = 0x06
Attribute Length = 0x0000
Attribute =
Attribute Number 7 :
Attribute ID = 0x07
Attribute Length = 0x0006
Attribute = 消去
```

```
Attribute ID: 0x00 = App Identifier
  →com.apple.mobilecal
```

```
Attribute ID: 0x06 = Positive Action Label
  →なし
```

```
Attribute ID: 0x07 = Negative Action Label
  →消去
```

図 2.7　Get Notification Attributes 動作の実行

4. 3 項で取得した App Identifier と get_app コマンドを使用して Data Source から Application Attribute を取得する例を以下に示します。

```
$ 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 = 0x000F
Attribute =  カレンダー
```

図 2.8　Get App Attributes 動作の実行

5. 3 項で取得した Notification Attribute に Positive Action Label もしくは Negative Action Label で iOS 通知に対する動作が定義されている場合は、ntf_act コマンドを使用して通知の消去等を iOS デバイスに指示できます。ntf_act コマンドの使用例を以下に示します。

```
$ 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
```

図 2.9　Perform Notification Action 動作の実行

2.3 ANCS コマンド

本節では本サンプルプログラムで定義したコマンド仕様を説明します。これらのコマンドは iOS デバイスと接続した後、ターミナルエミュレータに入力することで使用することができます。

get_att コマンド		
書式：	ancsc get_att [conn_hdl] [Notification UID]	
説明：	Get Notification Attributes 動作を実行します。 コマンドを実行後 Control Point キャラクタリスティックへ Notification Attribute 通知を指示する Write Request が送信されます。 以下の情報が送信されます。 Command ID: 0x00 Notification UID: 0xXXXXXXXX Notification Attribute ID & length: 0x00, 0x01, 0xXXXX※, 0x02, 0xXXXX※, 0x03, 0xXXXX※, 0x04, 0x05, 0x06, 0x07 ※：データ長は ANCS_STRING_LEN が使用されます。	
パラメータ：	[conn_hdl]	Write Request を送信するコネクションハンドルを指定します。
	[Notification UID]	Notification UID を指定します。 Notification Source で受信した Notification UID から指定してください。
使用例：	$ ancsc get_att 0x0020 0x00000001 Connection Handle = 0x0020 で接続しているデバイスの Notification UID = 0x00000001 の通知に対して Get Notification Attributes 動作を実行する。	

get_app コマンド		
書式：	ancsc get_app [conn_hdl] [App Identifier]	
説明：	Get App Attributes 動作を実行します。 コマンドを実行後 Control Point キャラクタリスティックへ Application Attributes 通知を指示する Write Request が送信されます。 以下の情報が送信されます。 Command ID: 0x01 App Identifier: com.apple.XXX App Attribute ID: 0x00	
パラメータ：	[conn_hdl]	Write Request を送信するコネクションハンドルを指定します。
	[App Identifier]	App Identifier を文字列で入力します。 Get Notification Attributes 動作によって Data Source で受信した App Identifier を指定してください。 入力できる文字列の最大長は[ANCS_STRING_LEN]です。 有効な値は「*iOS 及び iPadOS APP のバンドル ID*」を参照してください。
使用例：	$ ancsc get_app 0x0020 com.apple.mobilecal Connection Handle = 0x0020 で接続しているデバイスの App Identifier = com.apple.mobilecal で指定されるアプリに対して Get App Attributes 動作を実行する。	

ntf_act コマンド		
書式：	ancsc ntf_act [conn_hd] [Notification UID] [pos/neg]	
説明：	Perform Notification Action 動作を実行します。 コマンドを実行後 Control Point キャラクタリスティックへ、通知された iOS 通知に対する動作を指示する Write Request が送信されます。 以下の情報が送信されます。 Command ID: 0x02 Notification UID: 0xXXXXXXXX Action ID: 0xXX	
パラメータ：	[conn_hdl]	Write Request を送信するコネクションハンドルを指定します。
	[Notification UID]	Notification UID を指定します。 Notification Source で受信した Notification UID から指定してください。
	[pos/neg]	Perform Notification Action 動作によって実行する動作を選択します。pos/neg によって実行される動作は iOS 通知によって異なるので Get Notification Attributes 動作で確認してください。 pos: Positive Action Label の動作を実行する neg: Negative Action Label の動作を実行する
使用例：	$ ancsc ntf_act 0x0020 0x00000001 pos Connection Handle = 0x0020 で接続しているデバイスの Notification UID = 0x00000001 の通知に対して Positive Action を行うための Perform Notification Action 動作を実行する。	

3. ANCS サンプルプロジェクトのプログラム

本章では ANCS サンプルプロジェクトのプログラムを解説します。

3.1 ANCS サービスのプログラム

本節では r_ble_ANCSc.h に記述されている API についての説明を行います。

3.1.1 ANCS の概要

表 3.1 に ANCS に定義されている UUID と Attribute Properties を示します。

表 3.1 ANCS のサービス情報

サービス名	UUID	Properties
Apple Notification Center Service	7905F431-B5CE-4E99-A40F-4B1E122D00D0	---
キャラクタリスティック名	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

iOS 通知が発行されると Notification Source の Notification として Bluetooth 機器に通知されます。通知された iOS 通知に関する詳細を確認するには Control Point の Write を使用してどのような情報が必要か、を iOS デバイスに送信し、その応答として Data Source の Notification を受信します。また、ANCS を使用した通信を行う場合、ペアリングを行う必要があります。Notification Source と Data Source は Notification による通信を行うため、データ通信の準備が整った後、Client Characteristic Configuration Descriptor(以下、CCCD と略します)にアクセスし Notification を許可する必要があります。

3.1.2　サービスの API

本項では ANCS のサービス全体として使用される API について解説します。

表 3.2　ANCS のサービスの関数

Init 関数		
関数定義：	R_BLE_ANCSC_Init(ble_servc_app_cb_t cb)	
説明：	ANCS の初期化を行い、コールバック関数を登録します。 登録されたコールバック関数には ANCS のキャラクタリスティックのイベントが通知されます。	
引数：	cb	登録するコールバック関数

ServDiscCb 関数		
関数定義：	R_BLE_ANCSC_ServDiscCb(uint16_t conn_hdl, uint8_t serv_idx, uint16_t type, void *p_param)	
説明：	サービス検索時のコールバック関数です。 R_BLE_DISC_Start 関数のパラメータの一つとして使用します。	
引数：	conn_hdl	コネクションハンドル
	serv_idx	サービスインデックス
	type	イベントのタイプ
	p_param	イベントのパラメータのポインタ

Getattlhdl 関数		
関数定義：	R_BLE_ANCSC_GetServAttrHdl(const st_ble_dev_addr_t *p_addr, st_ble_gatt_hdl_range_t *p_hdl)	
説明：	ANCS のアトリビュートハンドルを取得します。	
引数：	p_addr	アトリビュートハンドルを取得する対象の BD アドレスのポインタ
	p_hdl	取得したアトリビュートハンドルを格納するためのポインタ

GATT での通信でエラーが発生したときの GATTC のイベント「BLE_GATTC_EVENT_ERROR_RSP」ではエラーコードが通知されます。ANCS のデータ通信でエラーが発生した場合は、ANCS サービス独自のエラーコードが通知される場合があります。表 3.3 に ANCS のエラーコードを示します。

表 3.3　ANCS のエラー一覧

エラーコード	説明
BLE_ANCSC_UNKNOWN_COMMAND_ERROR Value: 0xA0	Command ID で指定した数値が間違っている。
BLE_ANCSC_INVALID_COMMAND_ERROR Value: 0xA1	Control Point で書き込まれたデータの形式が間違っている。
BLE_ANCSC_INVALID_PARAMETER_ERROR Value: 0xA2	Notification UID などで指定されたパラメータが iOS の通知と合っていない。
BLE_ANCSC_ACTION_FAILED_ERROR Value: 0xA3	Command ID で指定された動作が実行されなかった。

3.1.3　Notification Source の API

本項では Notification Source キャラクタリスティック関連の API 仕様を記載します。API の使用例については 3.2.5 項および 3.2.6 項を参照してください。

表 3.4　Notification Source の関数

Getattlhdl 関数		
関数定義：	R_BLE_ANCSC_GetNsAttrHdl(const st_ble_dev_addr_t *p_addr, st_ble_ANCSc_ns_attr_hdl_t *p_hdl)	
説明：	Notification Source のアトリビュートハンドルを取得します。	
引数：	p_addr	アトリビュートハンドルを取得する対象の BD アドレスのポインタ
	p_hdl	取得したアトリビュートハンドルを格納するためのポインタ

Client Characteristic Configuration Descriptor の Read 関数		
関数定義：	R_BLE_ANCSC_ReadNsCliCnfg(uint16_t conn_hdl)	
説明：	Notification Source の CCCD に Read Request を送信します。Read Response を受信すると BLE_ANCSC_EVENT_NS_CLI_CNFG_READ_RSP イベントが返ります。	
引数：	conn_hdl	コネクションハンドル

Client Characteristic Configuration Descriptor の Write 関数		
関数定義：	R_BLE_ANCSC_WriteNsCliCnfg(uint16_t conn_hdl, const uint16_t *p_value)	
説明：	Notification Source の CCCD に Write Request を送信します。Notification Source からの Notification を可能にする為、本 API を使用して CCCD にアクセスして Notification を有効にする必要があります。Write Response を受信すると BLE_ANCSC_EVENT_NS_CLI_CNFG_WRITE_RSP イベントが返ります。	
引数：	conn_hdl	コネクションハンドル
	p_value	Write Request で送信するデータのポインタ

表 3.5　Notification Source のイベント

イベント	説明
BLE_ANCSC_EVENT_NS_HDL_VAL_NTF	Notification Source の notification を受信した データ：st_ble_ancsc_ns_t 表 3.6 を参照してください
BLE_ANCSC_EVENT_NS_CLI_CNFG_READ_RSP	Notification Source の CCCD の Read Response を受信した データ：uint16_t
BLE_ANCSC_EVENT_NS_CLI_CNFG_WRITE_RSP	Notification Source の CCCD の Write Reaponse を受信した データ：なし

表 3.6　Notification Source のデータ構造

構造体名：	st_ble_ancsc_ns_t	
メンバ変数：	uint8_t	eventid
	uint8_t	eventflag
	uint8_t	categoryid
	uint8_t	categorycount
	uint32_t	notificationuid

Notification Source のデータ構造は常に同じなのでデコード関数が実装されています。そのため、アプリケーションでは上記の形式でデータを利用することができます。

3.1.4　Control Point の API

本項では Control Point キャラクタリスティック関連の API 仕様を記載します。API の使用例については 3.2.7 項を参照してください。

表 3.7　Control Point の関数

Getattlhdl 関数		
関数定義：	R_BLE_ANCSC_GetCpAttrHdl(const st_ble_dev_addr_t *p_addr, st_ble_ANCSc_cp_attr_hdl_t *p_hdl);	
説明：	Control Point のアトリビュートハンドルを取得します。	
引数：	p_addr	アトリビュートハンドルを取得する対象の BD アドレスのポインタ
	p_hdl	取得したアトリビュートハンドルを格納するためのポインタ

Write 関数		
関数定義：	R_BLE_ANCSC_WriteCp(uint16_t conn_hdl, const st_ble_seq_data_t *p_value);	
説明：	Control Point に Write Request を送信します。Write Response を受信すると BLE_ANCSC_EVENT_CP_WRITE_RSP イベントが返ります。	
引数：	conn_hdl	コネクションハンドル
	p_value	送信するデータのポインタ **Control Point のデータ構造**を参照してください

表 3.8　Control Point のイベント

イベント	説明
BLE_ANCSC_EVENT_CP_WRITE_RSP	Control Point の Write Response を受信した データ：なし

Control Point のデータ構造

Control Point は利用ケースごとにデータ形式が異なります。よって Control Point キャラクタリスティックへの Write API　R_BLE_ANCSC_WriteCp は、可変長データを扱うためのデータ形式である[st_ble_seq_data_t]を引数に持ちます。

アプリケーションは、利用ケースごとにデータを利用できる形式で格納する必要があります。形式の詳細は「*Apple Notification Center Service (ANCS) Specification*」を参照してください。

3.1.5 Data Source の API

本項では Data Source キャラクタリスティック関連の API 仕様を記載します。API の使用例については 3.2.5 項および 3.2.7 項を参照してください。

表 3.9 Data Source の関数

Getattlhdl 関数		
関数定義：	R_BLE_ANCSC_GetDsAttrHdl(const st_ble_dev_addr_t *p_addr, st_ble_ANCSc_ds_attr_hdl_t *p_hdl)	
説明：	Data Source のアトリビュートハンドルを取得します。	
引数：	p_addr	アトリビュートハンドルを取得する対象の BD アドレスのポインタ
	p_hdl	取得したアトリビュートハンドルを格納するためのポインタ

Client Characteristic Configuration Descriptor の Read 関数		
関数定義：	R_BLE_ANCSC_ReadDsCliCnfg(uint16_t conn_hdl)	
説明：	Data Source の CCCD に Read Request を送信します。Read Response を受信すると BLE_ANCSC_EVENT_DS_CLI_CNFG_READ_RSP イベントが返ります。	
引数：	conn_hdl	コネクションハンドル

Client Characteristic Configuration Descriptor の Write 関数		
関数定義：	R_BLE_ANCSC_WriteDsCliCnfg(uint16_t conn_hdl, const uint16_t *p_value)	
説明：	Data Source の CCCD に Write Request を送信します。Data Source からの Notification を可能にする為、本 API を使用して CCCD にアクセスして Notification を可能にする必要があります。Write Response を受信すると BLE_ANCSC_EVENT_DS_CLI_CNFG_WRITE_RSP イベントが返ります。	
引数：	conn_hdl	コネクションハンドル
	p_value	Write Request で送信するデータのポインタ

表 3.10 Data Source のイベント

イベント	説明
BLE_ANCSC_EVENT_DS_HDL_VAL_NTF	Data Source の notification を受信した データ：st_ble_seq_data_t **Data Source のデータ構造**を参照
BLE_ANCSC_EVENT_DS_CLI_CNFG_READ_RSP	Data Source の CCCD の Read Response を受信した データ：uint16_t
BLE_ANCSC_EVENT_DS_CLI_CNFG_WRITE_RSP	Data Source の CCCD の Write Reaponse を受信した データ：なし

Data Source のデータ構造

Data Source は利用ケースごとにデータ形式が異なります。よって、Data Source から Notification を受信したことを示す BLE_ANCSC_EVENT_DS_HDL_VAL_NTF イベントのデータ形式は可変長データを扱うためのデータ形式である[st_ble_seq_data_t]です。アプリケーションは、利用ケースごとにデータを利用できる形式で展開する必要があります。形式の詳細は「*Apple Notification Center Service (ANCS) Specification*」を参照してください。

3.2　アプリケーションのプログラム

本プロジェクトは RX23W を使用して iOS と接続し、Bluetooth LE 通信で ANCS のデータを送受信する ANCS クライアントロールのプログラムを提供しています。本節ではプログラムのコード上の動作フローや利用方法、プログラムの変更方法について説明します。

本プロジェクトは、ユーザアプリケーションのひな形となる app_main.c、GATT データベース、ANCS に定義されたキャラクタリスティックへアクセスする API とイベントコードから構成されます。次節より、主に app_main.c の ANCS サンプルアプリケーションの動作を解説します。ANCS に定義されたキャラクタリスティックへアクセスする API とイベントコードの仕様は「3.1 ANCS サービスのプログラム」に記載します。

3.2.1　初期化

ANCS の初期化は R_BLE_ANCSC_Init() API によって実行されます。同 API は app_main.c の ble_init()でコールしています。よって、ユーザによる追加の処理は不要です。

```
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 初期化

3.2.2　Resolvable Private Address(RPA)機能の有効化

初期化完了後、本サンプルアプリケーションは iOS デバイスのアドレスを解決するため、Resolvable Private Address(RPA)機能を有効にします。app_main.c/gap_cb で以下の処理が実行されます。

```
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;
.........
}
```

データフラッシュからローカルデバイスのローカル IRK を読み取り、抽象 API を使用して Privacy モードを Network Privacy mode に設定する。

Code 3.2 RPA 機能の有効化

3.2.3　Advertising の開始

RPA が有効化された後、iOS との接続を開始するために Advertising を開始します。リモートデバイスの RPA を解決するためにデータフラッシュに格納されたボンディング情報を Resolving List に追加します。app_main.c/gap_cb で以下の処理が実行されます。

```
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;
```

データフラッシュからボンディングしたリモートデバイスの Identity アドレス、リモート IRK を読み取る。

ボンディング情報がなければ Advertising 開始する。

ボンディング情報のリストにダミーアドレスがあるか確認、なければ追加して Resolving List に登録する。

Resolving list の登録後、Advertising を開始する。

Code 3.3　Advertising の開始

3.2.4　サービス検索

「2.2.1 iOS デバイスとの接続」の操作により iOS デバイスとの接続が確立すると、サービス検索を開始します。本サンプルアプリケーションでは、サービス検索の開始に関するコードは app_main.c/gattc_cb BLE_GATTC_EVENT_CONN_IND イベント処理に以下のように実装しています。なお、サービス検索開始 API は R_BLE_DISC_Start API として BLE FIT モジュールに含まれています。

```
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;
.........
}
```

iOS デバイスとの接続確立後、サービス検索を開始する。

Code 3.4　サービス検索開始

3.2.5　ペアリングと Notification の有効化

サービス検索完了時に実行されるコールバック関数 app_main.c/disc_comp_cb で、本サンプルアプリケーションは iOS デバイスの Notification Source の CCCD に書き込み要求を発行します。iOS デバイスとのペアリングが完了している場合はサービス検索完了時にて、Notification Source の CCCD に 1 が書き込まれます。

```
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;
}
```

iOS デバイスの Notification Source の CCCD に書き込み要求を発行する。

Code 3.5　サービス検索完了

iOS デバイスとの初回接続時、サービス検索完了時点では iOS デバイスとのペアリングが完了していません。よって、Notification Source の CCCD への書き込み要求に対して BLE_GATTC_EVENT_ERROR_RSP イベントが返ります。同イベントは iOS デバイスのキャラクタリスティックへのアクセスに必要なセキュリティ要件が整っていない、即ちペアリングが必要であることを意味します。本サンプルアプリケーションは、同エラーレスポンスをトリガに抽象 API の R_BLE_ABS_StartAuth API を使用して、iOS デバイスとのペアリングを開始します。

```
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);
            }
}
```

エラーレスポンスをトリガに
ペアリングを開始する。

Code 3.6　ペアリングの開始

ペアリングが進行し、iOS デバイスとの鍵交換が完了すると BLE_GAP_EVENT_PEER_KEY_INFO イベントが発生します。同イベントにて iOS デバイスと交換した LTK、IRK、Identity アドレスを取得できます。IRK 並びに Identity アドレスは、iOS デバイスの RPA 解決に必要なため、R_BLE_GAP_ConfRslvList API を用いて Resolving List に登録します。また、ペアリング完了時に発生する BLE_GAP_EVENT_ENC_CHG イベントにて再度 Notification Source の CCCD に書き込み要求を発行します。

```
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;

……
}
```

交換した鍵情報を
Resolving List に登
録する。

Code 3.7 鍵情報の登録

Notification Source の CCCD 書き込み完了後、Data Source の CCCD に 1 を書き込み、ANCS の Notification 送信を許可します。

```
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;
……
    }
}
```

Notification Source の CCCD 書き込み完了をトリガに Data Source の CCCD へ書き込みを開始する

Data Source の CCCD への書き込み完了で ANCS が利用可能であることを通知する

Code 3.8　Notification の許可

3.2.6　iOS 通知を Notification Source で受信

iOS デバイスの Notification Source から Notification を受信すると、BLE_ANCSC_EVENT_NS_HDL_VAL_NTF イベントが発生します。同イベント処理は app_main.c/ANCSc_cb に記述します。QE for BLE から生成した場合、同イベント処理のひな形が作成されないため、ユーザによるイベント処理の実装が必要です。本プロジェクトでは、受信した Notification Source の各メンバ変数をターミナルエミュレータに表示します。

```
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;
……
```

イベントのデータを [st_ble_ancsc_ns_t]型で受け取る

各データを表示する

Code 3.9　Notification Source の受信

3.2.7　Control Point と Data Source を使用したデータ通信

「3.2.6 iOS 通知を Notification Source で受信」で得た iOS 通知の情報を使用し、Control Point キャラクタリスティックへ iOS 通知の詳細や iOS 通知の処理を指示することができます。iOS 通知の詳細取得を指示した場合、詳細情報は Data Source キャラクタリスティックから通知されます。本節では Notification Attribute の取得を例に説明します。

Notification Attribute を取得するためには、Control Point キャラクタリスティックに R_BLE_ANCSC_WriteCp API を使用して、Data Source からの Notification Attribute 送信指示を書き込みます。本プロジェクトでは get_att コマンドを使用時に実行される cmd_ancsc_get_att 関数内に実装しています。

```
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);
……
}
```

コマンドの引数を数値に変換

Control Point のデータを [st_ble_seq_data_t]型にエンコードしながら格納する

Control Point の Write Request 発行する

Code 3.10　Control Point の Write 実行

get_att コマンドによる Control Point キャラクタリスティックへ Data Source からの Notification Attribute 送信指示書き込みが完了した後、iOS デバイスの Data Source キャラクタリスティックから Notification 送信されます。Notification 受信時、BLE_ANCSC_EVENT_DS_HDL_VAL_NTF イベントがアプリケーションに通知されます。本サンプルアプリケーションでは、このイベント処理を app_main.c/ANCSc_cb に実装しています。QE for BLE から生成した場合、このイベント処理のひな形が作成されないため、ユーザによるイベント処理の実装が必要です。本プロジェクトでは受信したデータの内容をターミナルエミュレータに表示します。

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

イベントのデータを
[st_ble_seq_data_t]型で受け取る

[st_ble_seq_data_t]型のデータ
をデコードしながら表示する

Code 3.11　Notification の許可

Get App Attributes 動作と Perform Notification Action 動作も同様の形式で本プロジェクトに実装されています。Get App Attributes 動作の Control Point への Write Request 送信処理は cmd_ancsc_get_app 関数内に、Data Source での Notification 受信処理は app_main.c/ANCSc_cb の BLE_ANCSC_EVENT_DS_HDL_VAL_NTF イベントに実装されています。また Perform Notification Action 動作の Control Point への Write Request 送信処理は cmd_ancsc_ntf_act 関数内に実装されています。

3.3　QE for BLE による開発

本プロジェクトは QE for BLE のカスタムサービス生成機能を使用して生成されたプログラムをベースに開発されています。プロジェクトに他のサービスを追加する場合、QE for BLE を使用すると開発が容易になります。 QE for BLE の操作方法は「*RX23W グループ Bluetooth Low Energy プロファイル開発者ガイド (R01AN4553)*」を参照ください。

ANCS クライアントロールを作成するには、QE for BLE のインポート機能を使用し、本プロジェクトの Release フォルダの ancs_client.json を選択してください。

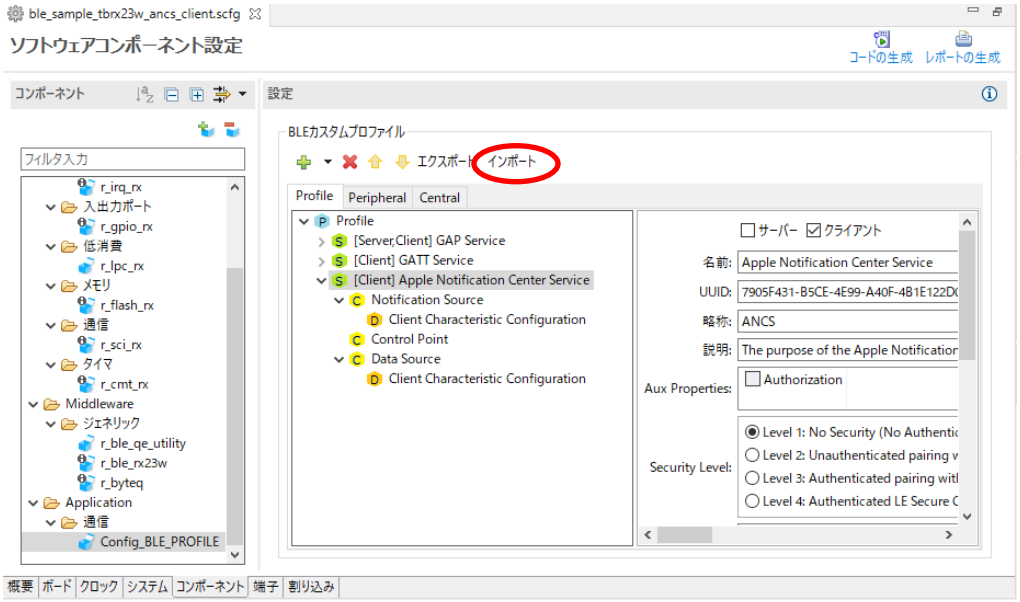

図 3.1　QE for BLE の画面

コード生成後、生成された r_ble_ANCSc.h に以下のコードを追加して下さい。ANCS で使用されるデータ形式や値などの詳細は「*Apple Notification Center Service (ANCS) Specification*」を参照してください。

```
/**********************************************************************************//**
 * @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 マクロの追加

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2021/7/7	-	新規作成

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。