RX23W グループ

Bluetooth Low Energy プロファイル開発者ガイド

要旨

本ガイドは、下記対象デバイスの Bluetooth Low Energy(LE)プロファイル開発者向けに Bluetooth LE ソフトウェアを使用してプロファイルを生成し、それを利用する方法をガイドします。

対象デバイス

- RX23W グループ

関連ドキュメント

- Bluetooth Core Specification 【https://www.bluetooth.com】
- Core Specification Supplement 【https://www.bluetooth.com】
- RX23W グループ ユーザーズマニュアル ハードウェア編(R01UH0823)
- Firmware Integration Technology ユーザーズマニュアル(R01AN1833)
- RX ファミリ ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)
- RX ファミリ e² studio に組み込む方法 Firmware Integration Technology (R01AN1723)
- RX スマート・コンフィグレータ ユーザーガイド: e² studio 編(R20AN0451)
- RX23W グループ BLE モジュール Firmware Integration Technology(R01AN4860)
- Bluetooth Low Energy プロトコルスタック基本パッケージ ユーザーズマニュアル (R01UW0205)
- RX23W グループ Bluetooth Low Energy アプリケーション開発者ガイド (R01AN5504)
- QE for BLE[RA,RE,RX] V1.4.0 リリースノート(R20UT5109)

目次

1.　概要

1.1　プロファイルの構成

Bluetooth LE では、主に汎用アトリビュートプロファイル(GATT)を使用してデータのやり取りを行います。GATT はクライアントロールとサーバロールを定義し、通信はクライアント・サーバ間で実行されます。プロファイルは 1 つ以上のサービスから構成される様々な用途ごとに策定されたプロトコルであり、同一のプロファイルに対応しているデバイスの間で通信することが可能です。サーバのデバイスは GATT データベースの中にプロファイルデータを持ち、クライアントのデバイスは Bluetooth LE の通信を行うことでプロファイルデータにアクセスします。またサーバのデバイスはプロファイルデータをクライアント側へ通知することもできます。それらを使用することで Bluetooth LE 通信における送受信を実現します。

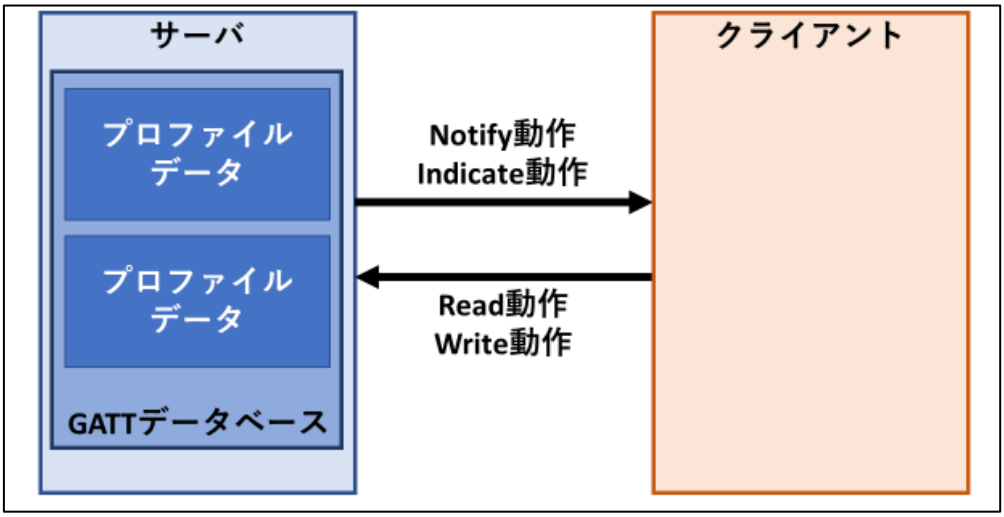

図 1.1 プロファイル通信の概要

Bluetooth LE ソフトウェアにおけるプロファイルの構成を図 1.2 に示します。本 Bluetooth LE ソフトウェアではユーザアプリケーションとプロファイルは BLE Protocol Stack の上で動作します。プロファイルは次の 3 種類のプログラムで構成されます。

- ユーザアプリケーションから Bluetooth LE 機能とプロファイルを利用するためのフレームワーク
- プロファイルで利用するサービスのデータ構造を定義した GATT データベース
- プロファイルのデータにアクセスするための API プログラム

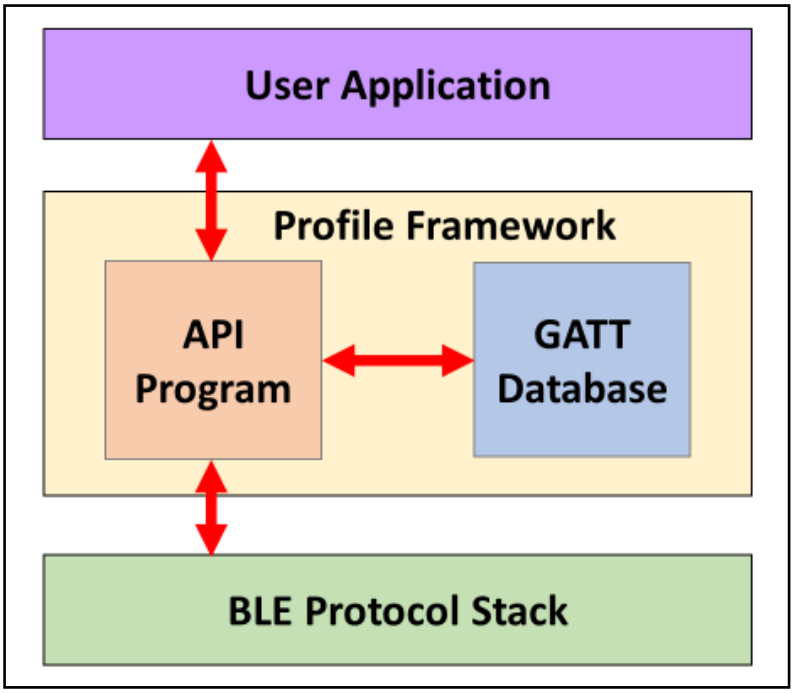

図 1.2 プロファイルの構成

Bluetooth LE ソフトウェアではプロファイルのデータ構造を GATT データベースで定義します。データへアクセスするためのインタフェースはプロファイルを構成する各サービス単位の API プログラムで提供します。ユーザアプリケーションはサービスの API プログラムを使用してプロファイルのデータにアクセスし、プロファイル通信を実行します。

Bluetooth 仕様では、Bluetooth SIG Inc.によっていくつかのサービスが定義されており、本書では、これらを SIG 標準サービスと呼びます。一方、SIG 標準サービスでサポートされない機能を実現する場合には、ユーザが独自のサービスを定義する必要があります。本書では、独自に定義されたサービスをカスタムサービスと呼びます。

1.2　プロファイルの開発手順

プロファイル開発の手順を図 1.3 に示します。

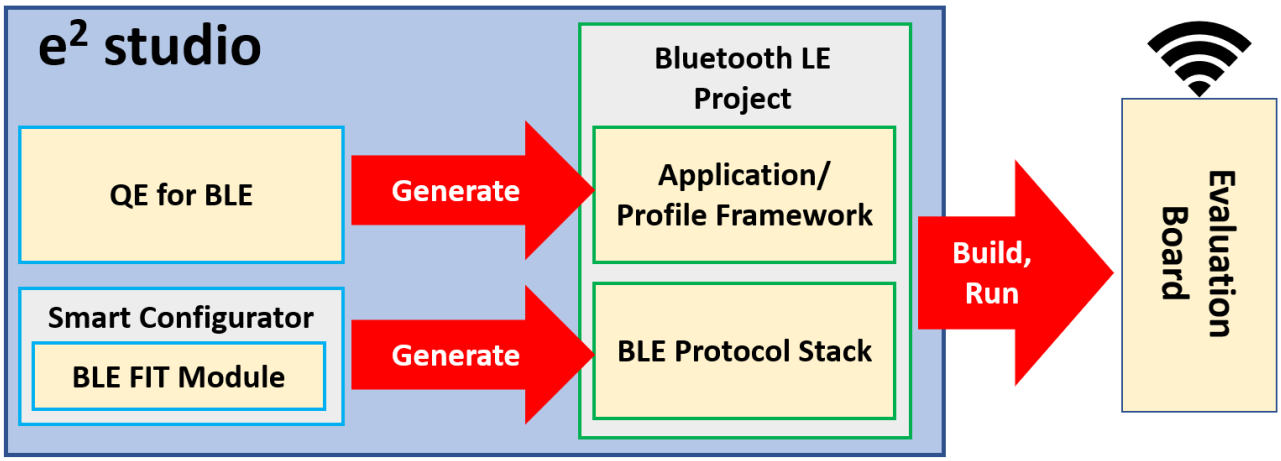

図 1.3 プロファイル開発の手順

プロファイル開発の手順は以下の通りです。

1) Bluetooth LE 通信プロジェクトの作成

RX23W 用 Bluetooth LE 通信プロジェクトを e²studio に追加します。詳細は「2. 開発環境」をご覧ください。

2) QE for BLE でプロファイルを設計

QE for BLE の GUI を操作してプロファイルを設計します。詳細は「3. プロファイルの設計」をご参照ください。QE for BLE は、設計したプロファイルの API プログラムと GATT データベース、フレームワークを e² studio 上のユーザプロジェクトに生成します。

3)アプリケーションの実装

QE for BLE から生成されたプログラムを利用してユーザプロジェクト上にアプリケーションを実装します。生成されたプログラムを用いたユーザアプリケーションの実装については、「4. プログラムの実装」をご参照ください。

4)実装したアプリケーションの実行

実装したアプリケーションは評価用ボードを使用して実行することができます。詳細は「5. 作成したプロファイルのビルド及び実行」をご参照ください。

2. 開発環境

2.1 ソフトウェア条件

本ドキュメントは以下のソフトウェアを使用する環境をもとに説明します。

- e² studio 2022-01 (64bit 版)
- CC-RX コンパイラ v2.08.00
- BLE FIT モジュール v2.30
- QE for BLE [RA, RE, RX] v1.4.0
- QE for BLE [RA, RE, RX] Utility v1.40

QE for BLE [RA, RE, RX]については「2.2 QE for BLE」、開発環境の構築方法については「2.3 開発環境の構築」を参照してください。

2.2　QE for BLE

QE for BLE は GUI 上でプロファイルを設計し、それをプログラムとして生成します。QE for BLE が生成するプログラムの一覧を表 2.1 に示します。

表 2.1 QE for BLE が生成するプログラム

ファイル名	説明
app_main.c	ユーザアプリケーションとプロファイルのフレームワーク アプリケーションとプロファイル開発のベースとなるスケルトンプログラムです。
gatt_db.c gatt_db.h	GATT データベースのプログラム QE for BLE で[サーバー]にチェックをつけたサービスのデータ構造が定義されています。
r_ble_[略称][s or c].c r_ble_[略称][s or c].h	プロファイルの API プログラム プロファイルのデータを送受信するための API プログラムです。プロファイルを構成するサービスごとにファイルが生成されます。それぞれのファイル名は QE for BLE で設定される[略称]と[サーバー]、[クライアント]をもとに決められます。[略称][s]がサーバ側、[略称][c]がクライアント側です。 Example) [略称]=[sig]、サーバ：r_ble_sigs.c, r_ble_sigs.h [略称]=[cus]、クライアント：r_ble_cusc.c, r_ble_cusc.h

各プログラムは [プロジェクト名]/qe_gen/ble 内に生成されます。

スケルトンプログラムはプロファイル及びアプリケーション開発のベースとなります。スケルトンプログラムの実装方法は、「4. プログラムの実装」でガイドします。

QE for BLE では Renesas で作成した SIG 標準サービスのサービス API プログラムを提供します。これらの多くは認証取得済みのため、そのまま使用することができます。 表 2.2 に QE for BLE がサポートする SIG 標準サービスの一覧を示します。また表 2.3 に QE for BLE サポートするプロファイルの一覧を示します。各サービス、各プロファイルの詳細な仕様については Bluetooth SIG のホームページ (https://www.bluetooth.com)をご参照ください。

表 2.2 BLE QE Utility モジュールがサポートするサービス一覧

サービス名	略称	バージョン	サービス名	略称	バージョン
Alert Notification Service	ANS	1.0	Automation IO Service	AIOS	1.0
Battery Service	BAS	1.0	Blood Pressure Service	BLS	1.0
Body Composition Service	BCS	1.0	Bond Management Service	BMS	1.0
Continuous Glucose Monitoring Service	CGMS	1.0.1	Current Time Service	CTS	1.1
Cycling Power Service	CPS	1.1	Cycling Speed and Cadence Service	CSCS	1.0
Device Information Service	DIS	1.1	Environmental Sensing Service	ESS	1.0
Fitness Machine Service	FTMS	1.0	Glucose Service	GLS	1.0
Health Thermometer Service	HTS	1.0	Heart Rate Service	HRS	1.0
Human Interface Device Service	HIDS	1.0	Immediate Alert Service	IAS	1.0
Insulin Delivery Service	IDS	1.0	Link Loss Service	LLS	1.0.1
Location and Navigation Service	LNS	1.0	Next DST Change Service	NDCs	1.0
Object Transfer Service	OTS	1.0	Phone Alert Status Service	PASS	1.0
Pulse Oximeter Service	PLXS	1.0	Reconnection Configuration Service	RCS	1.0
Reference Time Update Service	RTUS	1.0	Running Speed and Cadence Service	RSCS	1.0
Scan Parameters Service	ScPS	1.0	Tx Power Service	TPS	1.0
User Data Service	UDS	1.0	Weight Scale Service	WSS	1.0
GATT Service	GATS	-	GAP Service	GAPS	-

【注】Object Transfer Service は認証を取得しておりません。このサービスを使用した製品開発をご検討の際は弊社までご相談ください。

表 2.3 BLE QE Utility がサポートするプロファイル一覧

プロファイル	プロファイルを構成するサービス			
Blood Pressure Profile [BLP]	BLS	DIS		
Health Thermometer Profile [HTP]	HTS	DIS		
Heart Rate Profile [HRP]	HRS	DIS		
Glucose Profile [GLP]	GLS	DIS		
Pulse Oximeter Profile [PLXP]	PLXS	DIS	(BAS)	(CTS)
	(BMS)			
Continuous Glucose Monitoring Profile [CGMP]	CGMS	DIS	(BMS)	
Reconnection Configuration Profile [RCP]	RCS	(BMS)		
Insulin Delivery Profile [IDP]	IDS	DIS	(BAS)	(CTS)
	(BMS)	(IAS)		
Cycling Power Profile [CPP]	CPS	(DIS)	(BAS)	
Cycling Speed and Cadence Profile [CSCP]	CSCS	(DIS)		
Running Speed and Cadence Profile [RSCP]	RSCS	(DIS)		
Location and Navigation Profile [LNP]	LNS	(DIS)	(BAS)	
Weight Scale Profile [WSP]	WSS	DIS	(BCS)	(BAS)
	(CTS)	(UDS)		
Fitness Machine Profile [FTMP]	FTMS	(DIS)	(UDS)	
Environmental Sensing Profile [ESP]	ESS	(DIS)	BAS)	
Find Me Profile [FMP]	IAS			
Proximity Profile [PXP]	IAS	(LLS)	(TPS)	
Alert Notification Profile [ANP]	ANS			
Phone Alert Status Profile [PASP]	PASS			
Time Profile [TIP]	CTS	(NDCS)	(RTUS)	
HID over GATT Profile [HOGP]	HIDS	DIS	BAS	(ScPS)
Scan Parameters Profile [ScPP]	ScPS			
Automation IO Profile [AIOP]	AIOS			

【注】()のないサービスは Mandatory のサービスであり、()のついたサービスは Optional のサービスです。QE for BLE でプロファイルを追加した場合、プロファイルツリーには Mandatory のサービスのみ追加されます。

2.3　開発環境の構築

本節では、QE for BLE を使用した開発環境の構築方法を説明します。

2.3.1　Bluetooth LE 通信プロジェクトの追加

以下のドキュメントの 4 章「BLE FIT モジュールのプロジェクト」を参考にして、e²studio のワークスペースに Bluetooth LE 機能を使用可能なプロジェクトを追加してください。」

・RX23W グループ BLE モジュール Firmware Integration Technology(R01AN4860)

2.3.2　QE for BLE のインストール

QE for BLE は以下のページからダウンロードすることができます。

https://www.renesas.com/qe-ble

インストールの方法は以下の通りです：

1. e² studio を起動する。
2. [Renesas Views]→[Renesas Software Installer]メニューを選択し、[Renesas Software Installer]ダイアログを開く。
3. [Renesas QE]を選択し、[次へ(N)>]ボタンを押下する。
4. [QE for BLE[RA,RE,RX](v1.4.0)]チェックボックスをチェックし、[終了(F)]ボタンを押下する。
5. [インストール]ダイアログで[Renesas QE for BLE[RA,RE,RX]]チェックボックスと[Renesas QE for BLE[RA,RE,RX] Utility]チェックボックスがチェックされていることを確認し、[次へ(N)>]ボタンを押下する。
6. インストール対象が[Renesas QE for BLE[RA,RE,RX]]と[Renesas QE for BLE[RA,RE,RX] Utility]となっていることを確認し、[次へ(N)>]ボタンを押下する。
7. ライセンスを確認した後、ライセンスに同意できる場合は[使用条件の条項に同意します(A)]ラジオボタンを選択し、[終了(F)]ボタンを押下する。
8. 信頼する証明書の選択ダイアログが表示された場合、表示された証明書をチェックした後、[OK]ボタンを押下してインストールを継続する。
9. e² studio の再起動を促されるので再起動を行う。
10. e² studio の[Renesas Views]ー[Renesas QE]メニューより本製品を起動します。本製品の使い方は e² studio の[ヘルプ]メニューから、QE の項目を参照してください。

3.　プロファイルの設計

本章では、QE for BLE を使用してプロファイルを設計するための方法をガイドします。

3.1　プロファイル設計の概要

本節では、QE for BLE によるプロファイル開発の手順を説明します。

3.1.1　QE for BLE の起動

e² studio で[Renesas Views]→[Renesas QE]→[R_BLE カスタムプロファイル RA,RE,RX(QE)]を選択して QE for BLE を起動します。

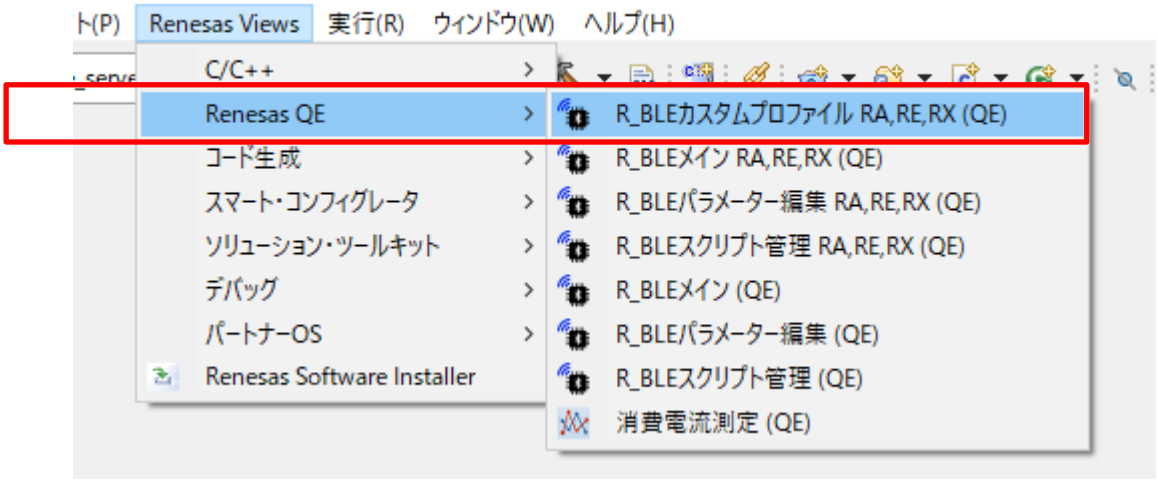

図 3.1 QE for BLE の開き方

注：旧バージョンの QE for BLE がプロジェクトに含まれている場合には、最新の QE for BLE への移行が促されます。表示される手順に従ってスマートコンフィギュレータのコンポーネントを削除します。

- QE for BLE [RA,RE,RX] V1.40 リリースノート

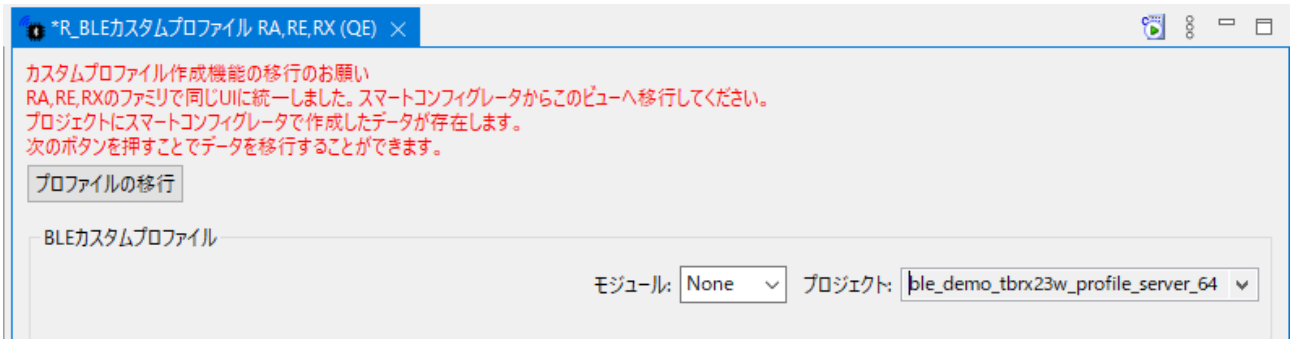

図 3.2 旧バージョンのプロジェクト使用時のプロファイルの移行

右上のプロジェクト選択欄から、コードを追加するプロジェクトを選択します。

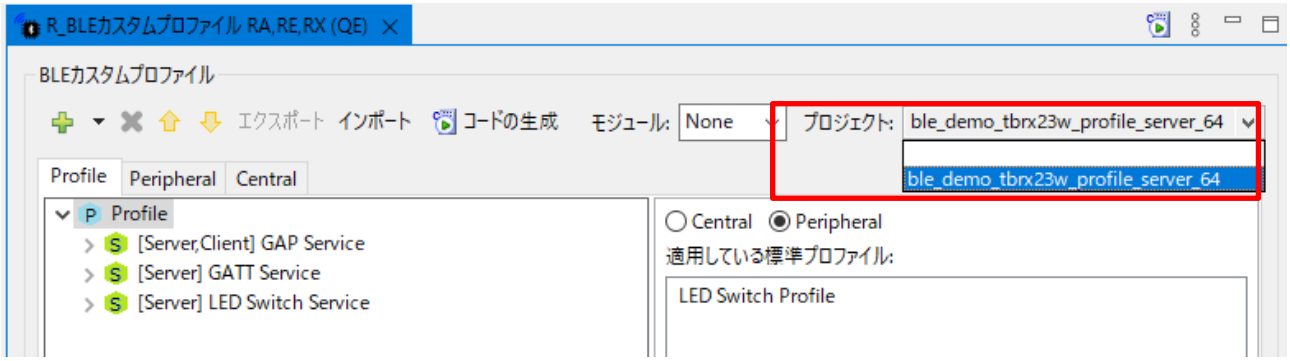

図 3.3 プロジェクトの選択

3.1.2　QE for BLE によるプロファイルの設計

設計画面のそれぞれのタブからプロファイル・アプリケーションを設計します。

- Profile タブ：サービス、キャラクタリスティック、ディスクリプタの設定を含むプロファイルを設定できます。ユーザはこのタブで GAP ロールを選択できます。詳しくは「3.2 章　プロファイルの設計」を参照してください。
- Peripheral タブ：ペリフェラルのデバイスとして動作するため、アドバタイジングに関する設定ができます。詳しくは「3.3 章　ペリフェラルの設計」を参照してください
- Central タブ：セントラルのデバイスとして動作するため、スキャン及びコネクションに関する設定ができます。詳しくは「3.4 章　セントラルの設計」を参照してください。

3.1.3　コード生成

プロファイルの設計後に、「コードの生成」ボタンを押すと選択したプロジェクトにコードが生成されます。コードは、プロジェクトフォルダ内の qe_ben/ble フォルダに生成されます。

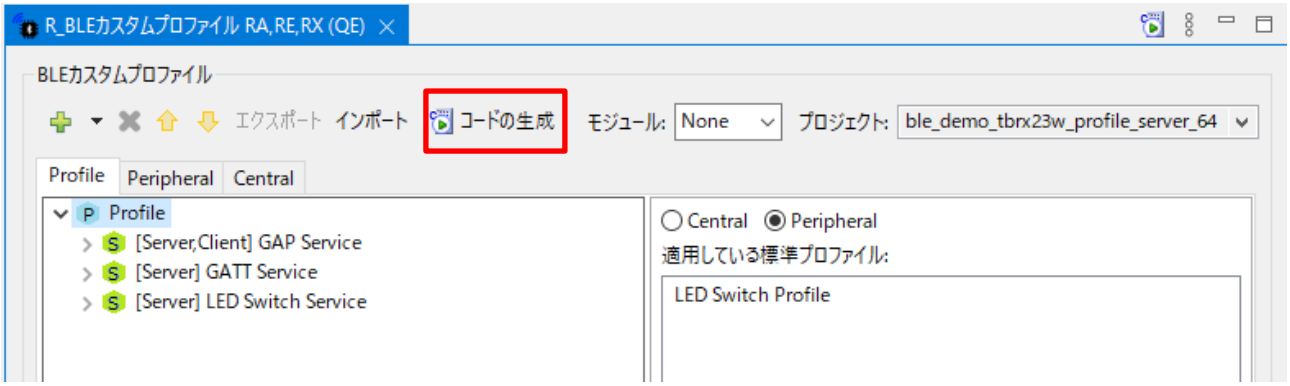

図 3.4 コード生成ボタン

3.1.4　プログラムの実装

これらの生成されたプログラムを使用してユーザアプリケーションを作成します。カスタムサービスの API プログラムの変更方法は「4.2 カスタムサービスの実装」、アプリケーションのフレームワークの変更方法は「4.3 アプリケーションの実装」を参照してください。

3.2　プロファイルの設計

QE for BLE の「プロファイル」タブからユーザアプリケーションで使用するプロファイルを設計することができます。図 3.5 にプロファイル設計画面を示します。

図 3.5 「プロファイル」タブの設定画面

プロファイルは 1 個以上のサービス、サービスは 1 個以上のキャラクタリスティック、キャラクタリスティックは 0 個以上のディスクリプタで構成されるツリー状の構造になっています。プロファイルツリーを確認することで現在設計しているプロファイルの全体の構成を確認することができます。

プロファイルツリーの各要素を選択すると、詳細設定エリアに選択した要素の種類ごとに設定項目が表示されます。詳細設定エリアに表示された項目を編集することでプロファイルツリーに追加した要素の機能を設計することができます。設計したプロファイルはプロジェクトフォルダ内に保存されます。プロファイルを任意のフォルダに保存したい場合は「エクスポート」を使用するとサービス単位で保存することができます。

ツールバーを使用することでプロファイルツリーの要素の追加や削除を行うことができます。ツールバーにあるアイコンとそれぞれに対応する動作は以下の通りです。

- 「  」：プロファイルツリーに要素を追加します。追加される要素はプロファイルツリーで選択されている要素によって異なります。
- 「」：プロファイルツリーで選択している要素を削除します。
- 「  」「  」：プロファイルツリーで選択している要素を移動します。プロファイルツリーで要素を並べ変えるときにご利用ください。
- 「エクスポート」：設計したサービスを JSON ファイルとして出力します。
- 「インポート」：サービスが定義されている JSON ファイルを読み込んでプロファイルに追加します。

3.2.1　要素の追加

QE for BLE のツールバーにある「」をクリックすることでプロファイルツリーに要素を追加することができます。追加される要素はプロファイルツリーで選択されている要素によって異なります。ここでは選択されている要素によってどの要素が追加されるかを説明します。

プロファイルを選択した状態で要素を追加するとプロファイルにサービスを追加することができます(図 3.6)。カスタムサービスを追加する場合は「新規サービスを追加」、SIG 標準サービスを追加する場合は「サービスを追加」を選択してください。追加することができる SIG 標準サービスは表 2.2 に示します。「プロファイルを追加」を選択した場合、追加するサービスをプロファイル単位で選択できます。プロファイルに含まれるサービスの一覧を表 2.3 に示します。プロファイル内に含まれるサービスでも任意で追加するサービスについてはプロファイルツリーに追加されません。それらのサービスをプロファイルツリーに追加したい場合は個別に追加してください。

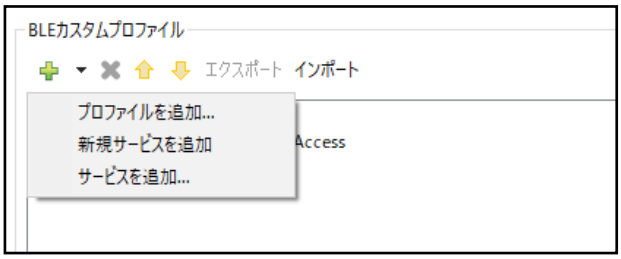

図 3.6 サービスの追加

サービスを選択した状態でツールバーから「」ボタン をクリックすることでキャラクタリスティックを追加することができます(図 3.7)。カスタムキャラクタリスティックを追加する場合は「新規キャラクタリスティックを追加」、Bluetooth SIG で仕様が定義されているキャラクタリスティックを追加する場合は「キャラクタリスティックを追加」を選択してください。

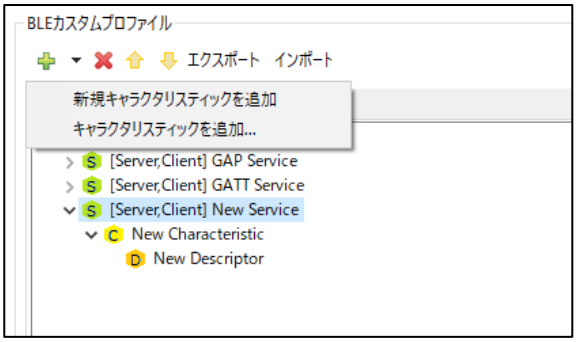

図 3.7 キャラクタリスティックの追加

キャラクタリスティックを選択した状態でツールバーから「  」ボタン をクリックすることでディスクリプタを追加することができます(図 3.8)。カスタムディスクリプタを追加する場合は「新規ディスクリプタを追加」、Bluetooth SIG で仕様が定義されているディスクリプタを追加する場合は「ディスクリプタを追加」を選択してください。

図 3.8 ディスクリプタの追加

3.2.2　プロファイルの設定

プロファイルツリーでプロファイル「  」を選択することで詳細設定エリアにプロファイルの設定項目(図 3.9)を表示することができます。

プロファイルの設定項目では GAP ロールを選択することができます。ラジオボタンで「セントラル」と「ペリフェラル」のどちらに設定するかを選択してください。ここで選択した項目に沿ってスケルトンプログラムが生成されます。「ペリフェラル」を選んだ場合、アドバタイズを行うプログラムが生成されます。「セントラル」を選んだ場合、スキャンを行い、接続要求を発行するプログラムが生成されます。

「適用している標準プロファイル」にはプロファイルツリーに登録されているサービスからどのプロファイルに対応しているかを表示します。表示するプロファイルは表 2.3 に示します。

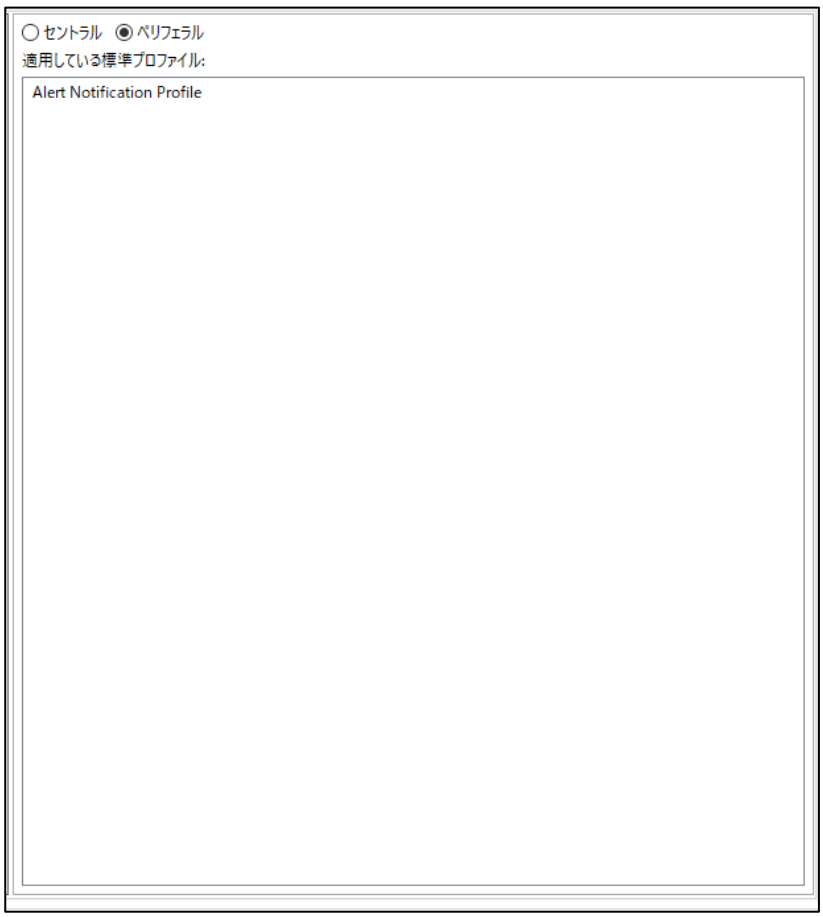

図 3.9 プロファイルの設定項目

3.2.3　サービスの設定

プロファイルツリーのサービス「 」 を選択することで詳細設定エリアにサービスの設定項目(図 3.10)を表示することができます。各設定項目の説明を表 3.1 に示します。

【注】GAP Service と GATT Service は必須のサービスです。これらのサービスは削除しないようにしてください。

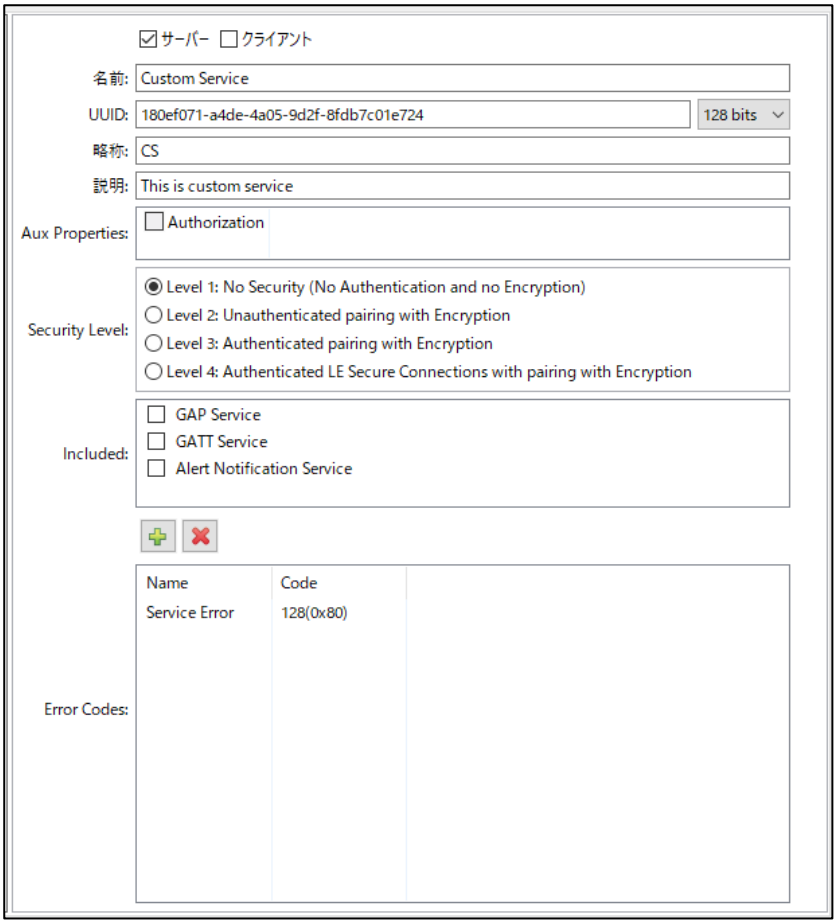

図 3.10 サービスの設定項目

表 3.1 サービスの設定項目の説明

項目名	説明	
サーバー [任意]	チェックを入れるとサービスのサーバとしてのプログラムを生成するように設定します。 また GATT データベースにキャラクタリスティックとディスクリプタを追加します。	
クライアント [任意]	チェックを入れるとサービスのクライアントとしてのプログラムを生成するように設定します。	
名前 [必須]	サービスの名前を設定します。 Example) Custom service	
UUID [必須]	サービスの UUID を設定します。 カスタムサービスの場合、128bit を選択してください。 初期値にはランダムな値が設定されていますが、任意の値に変更することも可能です。 Example) 16bit: 0xe237 128bit: 96FE7990-2C76-89AB-DC49-AB7F123DEF9C 【注】”0x”や”-”は入力されなくてもコード生成に問題ありません。	
略称 [必須]	サービスの名前の略称を設定します。 サービスのファイル名や関数名、変数名などに使用されるため、他のサービスと競合しないようにしてください。 Example) cs	
説明 [任意]	サービスに関する説明を設定します。 生成されるコードのコメントに使用されます。用途や動作を任意で説明してください。 Example) This service used for sending sensor data.	
Aux properties [任意]	サービスへのアクセスに必要となる条件を設定します。 以下の項目について設定することができます。	
	Authorization	ユーザの承認が必要となります。 関数 R_BLE_GAP_AuthorizeDev( )を使用して承認を行います。
Security Level [必須]	クライアントがサービスへアクセスする際に必要なセキュリティのレベルを選択します。 以下の項目から選択してください。	
	Level 1: No Security	クライアントはペアリングせずにアクセスでき、通信は暗号化されません。
	Level 2: Unauthenticated pairing with Encryption	クライアントは Just Works 方式による認証なしのペアリング後にアクセスでき、通信は暗号化されます。
	Level 3: Authenticated pairing with Encryption	クライアントは Passkey Entry もしくは OOB 方式による認証を伴うペアリング後にアクセスでき、通信は暗号化されます。
	Level 4: Authenticated LE Secure Connections with pairing with Encryption	クライアントは LE Secure Connections によるペアリング後にアクセスでき、通信は暗号化されます。
Included [任意]	Included service を設定します。 Included service として設定したいサービスにチェックを入れてください。	
Error Codes [任意]	サービスのエラーコードを追加します。 追加したエラーコードは関数 R_BLE_GATTS_SendErrRsp( )で使用することができます。	
	Name	エラーコードの名称を設定します。 Example) Value not Supported
	Code	エラーコードの値を設定します。 任意の数値を選択してください。

SIG 標準サービスを選択したときのサービスの設定項目を、図 3.11 に示します。この状態では「サーバー」、「クライアント」、「Aux Properties」、「Security Level」、「Included」の項目が設定可能です。「カスタマイズ」ボタンをクリックすることですべての項目を編集できるようになります。SIG 標準サービスをもとにカスタムサービスを作成する場合にご利用ください。

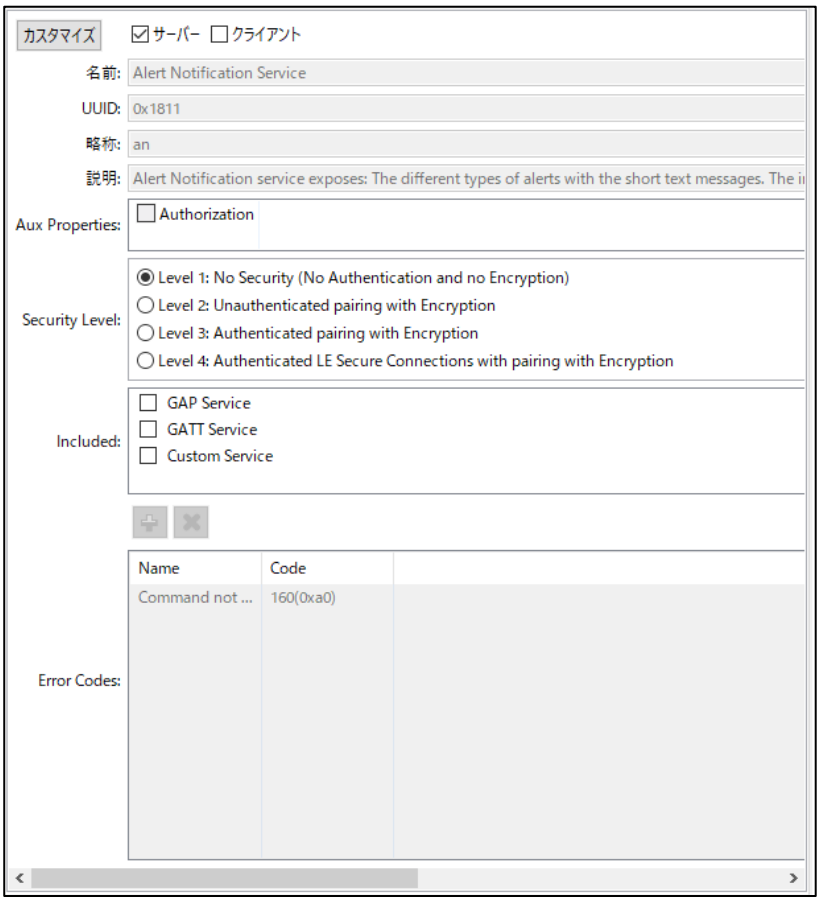

図 3.11 SIG 標準サービスの設定項目

3.2.4　キャラクタリスティックの設定

QE for BLE のプロファイルツリーのキャラクタリスティック「C」 を選択することで詳細設定エリアにキャラクタリスティックの設定項目（図 3.12）を表示することができます。各設定項目の説明を表 3.2、表 3.3 に示します。

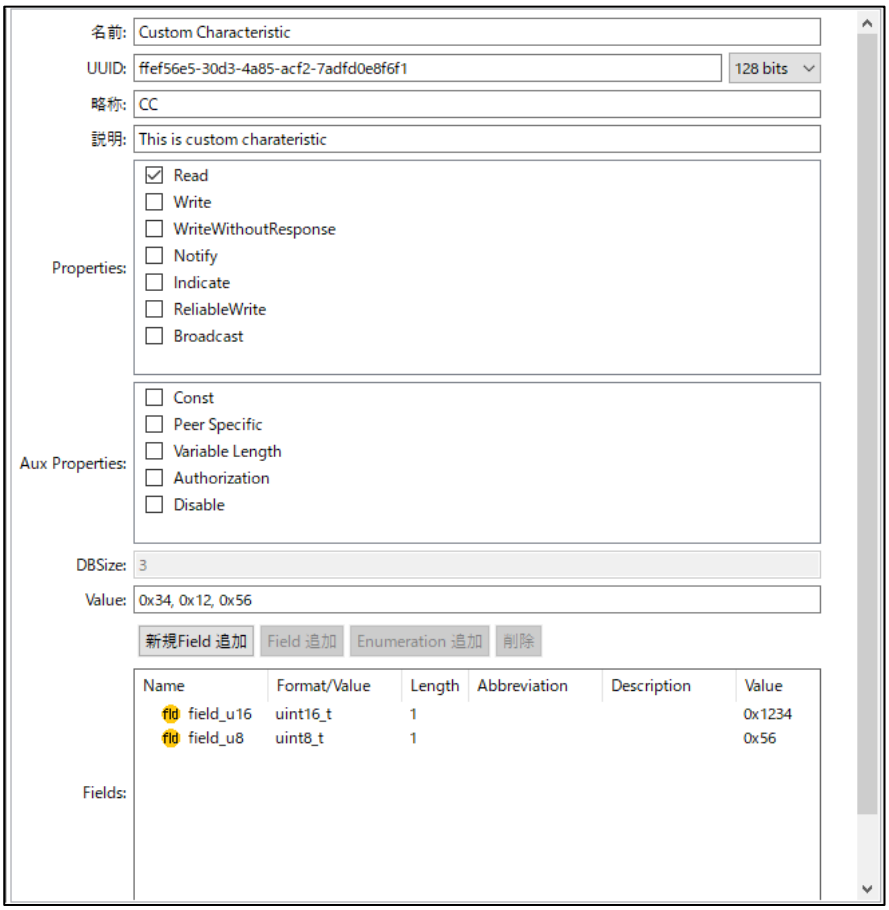

図 3.12 キャラクタリスティックの設定項目

表 3.2 キャラクタリスティックの設定項目の説明

項目名	説明	
名前 [必須]	キャラクタリスティックの名前を設定します。 Example) Custom Characteristic	
UUID [必須]	キャラクタリスティックの UUID を設定します。 カスタムキャラクタリスティックの場合、128bit を選択してください。 初期値にはランダムな値が入力されていますが、任意の値に変更することも可能です。 Example) 16bit: 0xe237 128bit: 96FE7990-2C76-89AB-DC49-AB7F123DEF9C 【注】”0x”や”-”は入力されなくてもコード生成に問題ありません。	
略称 [必須]	キャラクタリスティックの名前の略称を設定します。 キャラクタリスティックの関数やデータの名称に使われるため、他のキャラクタリスティックと競合しないようにしてください。 Example) cc	
説明 [任意]	キャラクタリスティックに関する説明を設定します。 生成されるコードのコメントで使用されます。用途などを任意で説明してください。 Example) This Characteristic is used for sending sensor data	
Properties [必須]	キャラクタリスティックの動作を定義します。 設定した値に対応する API・イベントが生成されます。 「Broadcast」と「ReliableWrite」については、その動作上 API・イベントが生成されないためご注意ください。また Notify もしくは Indicate を選択すると Client Characteristic Configuration Descriptor が追加されます。設定できる値は以下の通りです。	
	Read	Read 動作が可能になります。
	Write	Write 動作が可能になります。
	WriteWithoutResponse	Write Without Response 動作が可能になります。
	Notify	Notify 動作が可能になります。
	Indicate	Indicate 動作が可能になります。
	ReliableWrite	Reliable Write 動作が可能になります。
	Broadcast	Broadcast 動作が可能になります。
Aux Properties [任意]	キャラクタリスティックの補助属性を設定します。 設定できる値は以下の通りです。	
	Const	値の変更ができなくなります。
	Peer Specific	接続相手毎に個別に値を保持するようになります。 通常は、すべての接続相手が同じ値を参照します。
	Variable Length	値を可変長にします。
	Authorization	ユーザの承認が必要となります。 関数 R_BLE_GAP_AuthorizeDev( )を使用して承認を行います。
	Disable	アトリビュートを無効にします。
DBSize [必須]	キャラクタリスティックのサイズです。単位は BYTE です。 Field に対応したサイズが自動的に計算されます。 「st_ble_seq_data_t」の Field を追加した場合、データの最大長を入力してください。	
Value [任意]	キャラクタリスティックの初期値です。 数値で入力する場合、8bit ごとに区切って入力してください。 文字列の場合””で囲むことで簡単に入力することができます。 Example) 数値の場合：0x12,0x34,56,78 文字列の場合：”example”	
Field [必須]	キャラクタリスティックのデータ構造を設定します。 設定できる値については表 3.3 を参照してください。	

表 3.3 Field の設定項目

新規 Field 追加	新しい Field を追加します。 Field には以下の項目を設定することができます。		
	Name [必須]	Field の名前を設定します。 Example) field_name	
	Format/Value [必須]	Field のフォーマットを設定します。 選択できる値は以下の通りです。	
		bool	boolean 型を設定します。
		char	char 型を設定します。
		uint8_t	符号なし 8bit データ型を設定します。
		uint16_t	符号なし 16bit データ型を設定します。
		uint32_t	符号なし 32bit データ型を設定します。
		int8_t	符号あり 8bit データ型を設定します。
		int16_t	符号あり 16bit データ型を設定します。
		int32_t	符号あり 32bit データ型を設定します。
		st_ble_ieee_11073_float_t	IEEE-11073 32bit FLOAT 型を設定します。
		st_ble_ieee_11073_sfloat_t	IEEE-11073 16bit SFLOAT 型を設定します。
		st_ble_date_time_t	日付及び時間情報を保持するデータ構造を設定します。
		st_ble_dev_addr_t	Bluetooth LE のアドレスのデータ構造を設定します。
		st_ble_seq_data_t	可変長データのためのデータ配列です。 Field が 1 つで length が 2 以上のとき設定します。
		struct	構造体を設定します。 「Field 追加」を使用した場合に設定されます。
	Length [必須]	Field のデータ長を設定します。 Format/Value で設定したフォーマットをここで設定したデータ長分確保します。	
	Abbreviation [任意]	Field の略称を設定します。 変数名などに利用されます。設定されない場合、Name に設定した値が利用されます。	
	Description [任意]	Field の説明を設定します。	
	Value [任意]	Field 単位で初期値を設定します。 入力した値はキャラクタリスティック・ディスクリプタの「Value」に反映されます。	
Field 追加	選択されている Field 内部に新しく Field を追加します。 階層構造や入れ子構造のデータを持つ場合にご利用ください。 選択されている Field の Format/Value は[struct]に設定されます。 追加された Field には新規 Field 追加と同じ項目が設定できます。		
Enumeration 追加	選択されている Field の値として利用可能な Enumeration を定義します。 Enumeration では以下の項目を設定することができます。		
	Name [必須]	Enumeration の名前を設定します。	
	Format/Value [必須]	Enumeration の値を設定します。	
	Description [任意]	Enumeration の説明を設定します。	
削除	選択されている Field を削除します。		

3.2.5　ディスクリプタの設定

プロファイルツリーのディスクリプタ「D」を選択することで詳細設定エリアにディスクリプタの設定項目（図 3.13）を表示することができます。各設定項目の説明を表 3.4 に示します。

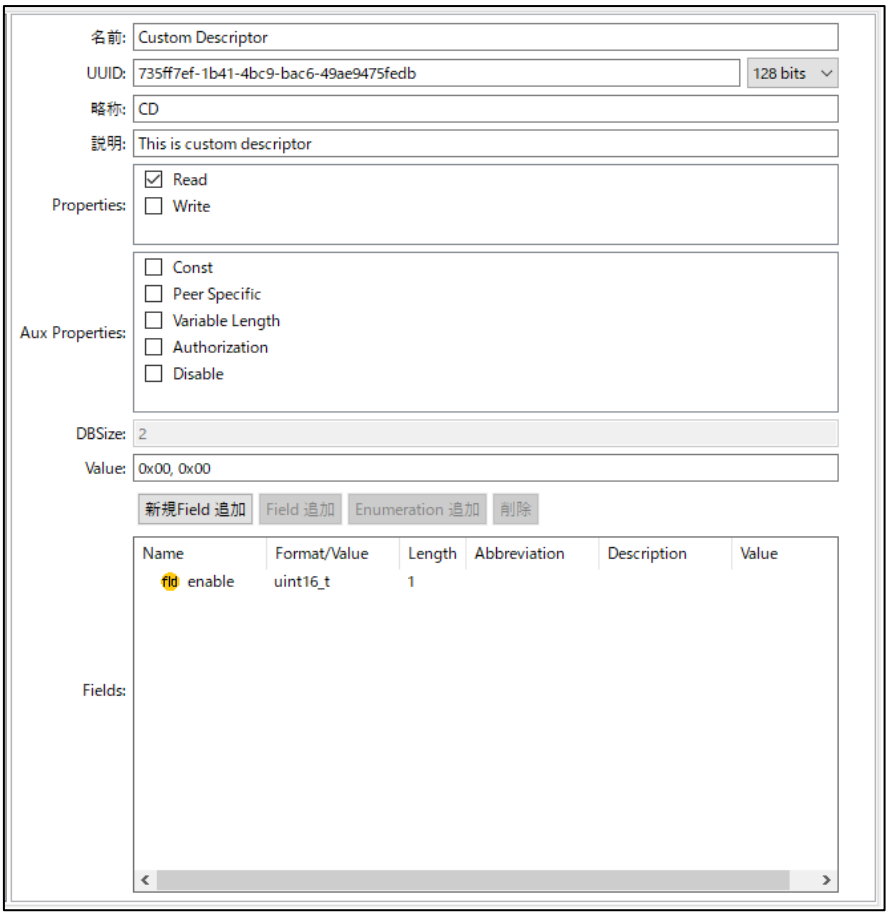

図 3.13 ディスクリプタの設定項目

表 3.4 ディスクリプタの設定項目の説明

項目名	説明	
名前 [必須]	ディスクリプタの名前を設定します。 Example) 　Custom Descriptor	
UUID [必須]	ディスクリプタの UUID を設定します。 カスタムディスクリプタの場合、128bit を選択してください。 初期値にはランダムな値が設定されていますが、任意の値に変更することも可能です。 Example) 16bit: 0xe237 128bit: 96FE7990-2C76-89AB-DC49-AB7F123DEF9C 【注】”0x”や”-”は入力されなくてもコード生成に問題ありません。	
略称 [必須]	ディスクリプタの名前の略称を設定します。 ディスクリプタの関数やデータの名称に使われるため、他のディスクリプタと競合しないようにしてください。 Example) 　cd	
説明 [任意]	ディスクリプタに関する説明を設定します。 生成されるコードのコメントで使用されます。用途などを任意で説明してください。 Example) 　This Descriptor is used to enable data sending	
Properties [必須]	ディスクリプタの動作を定義します。 設定した値に対応する API・イベントが生成されます。 設定できる値は以下の通りです。	
	Read	Read 動作が可能になります。
	Write	Write 動作が可能になります。
Aux Properties [任意]	ディスクリプタの補助属性を設定します。 設定できる値は以下の通りです。	
	Const	値の変更ができなくなります。
	Peer Specific	接続相手毎に個別に値を保持するようになります。 通常は、すべての接続相手が同じ値を参照します。
	Variable Length	値を可変長にします。
	Authorization	ユーザの承認が必要となります。 関数 R_BLE_GAP_AuthorizeDev( )を使用して承認を行います。
	Disable	アトリビュートを無効にします。
DBSize [必須]	キャラクタリスティックのサイズです。単位は BYTE です。 Field に対応したサイズが自動的に計算されます。 「st_ble_seq_data_t」の Field を追加した場合、データの最大長を入力してください。	
Value [任意]	ディスクリプタの初期値です。 数値で入力する場合、8bit ごとに区切って入力してください。 文字列の場合””で囲むことで簡単に入力することができます。 Example) 　数値の場合：0x12,0x34,56,78 　文字列の場合：”example”	
Field [必須]	ディスクリプタのデータ構造を設定します。 設定できる値については表 3.3 を参照してください。	

3.3　ペリフェラルの設計

「Peripheral」タブでは GAP ペリフェラルロールで動作する際に必要なパラメータを設定する事ができます。このタブで設定した値は「Profile」タブで「Peripheral」を選択したときのアプリケーションフレームワークを作成する際に使用されます。このタブでは以下の項目を設定することができます。

表 3.5 ペリフェラルタブで設定できる項目

項目名	説明
Advertising Data	Advertising 動作時に送信されるアドバタイジングデータを設定することができます。
Scan Response Data	Advertising 動作時に送信されるスキャンレスポンスデータを設定することができます。
Advertising Parameter	Advertising 動作のパラメータを設定することができます。

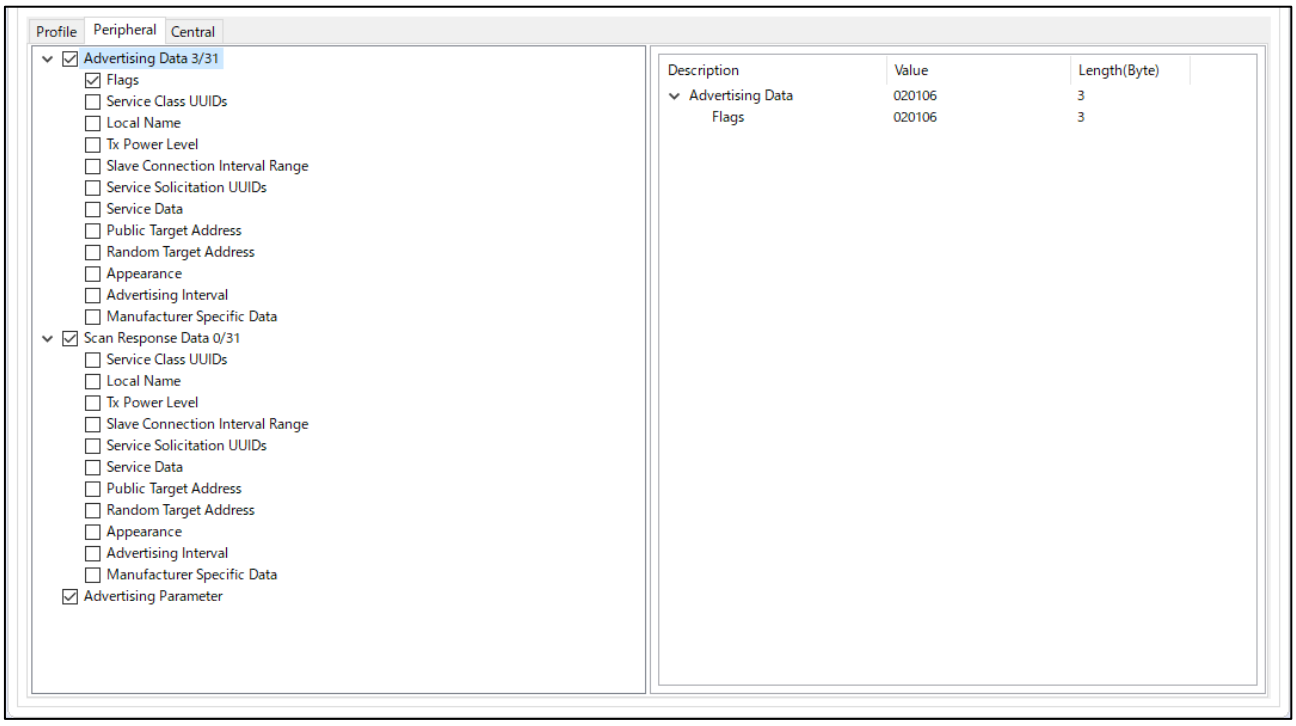

図 3.14 Peripheral タブの設定画面

3.3.1　Advertising Data の設定

Advertising Data ではアドバタイズ動作で送信されるアドバタイジングデータを設定することができます。チェックをつけた全てのデータ型がアドバタイジングデータとして追加されます。それぞれのデータ型について、ユーザは詳細な値を設定することができます。ユーザが追加することが可能なデータ型は表 3.6 に示します。設定できるアドバタイズデータの最大長は 31 バイトであるため、その値を超過しないように設定してください。もし 31 バイトを超えるデータを設定したい場合、「3.3.2 Scan Response Data の設定」で設定できるスキャンレスポンスデータもご利用ください。各アドバタイジングデータの詳細については「Core Specification Supplement【https://www.bluetooth.com】」を参照してください。

表 3.6 設定できるデータ型の一覧

データ型名	データ	
Flags	このデータ型はアドバタイジングデータのフラグを示します。 接続可能なデバイスのアドバタイジング動作時には必要なデータ型です。 このデータ型はスキャンレスポンスデータとして設定することはできません。 Discoverable Mode を選択した後、追加する情報をチェックしてください。	
	LE Limited Discoverable Mode	一定期間デバイスが検索可能であることを示します。
	LE General Discoverable Mode	常時デバイスが検索可能であることを示します。
	Non-Discoverable Mode	デバイスが検索不可能であることを示します。
	BR/EDR Not Supported	Bluetooth LE のみサポートすることを示します。
	Simultaneous LE and BR/EDR to same Device Capable (Controller)	Bluetooth LE と BR/EDR のコントローラ側として同時に動作することができることを示します。
	Simultaneous LE and BR/EDR to same Device Capable (Host)	Bluetooth LE と BR/EDR のホスト側として同時に動作することができることを示します。
Service Class UUIDs	このデータ型はアドバタイズを行うデバイスが対応するサービスのリストを示します。 「Profile」タブで追加したサービスから選択することができ、チェックしたサービスをリストに追加することができます。	
Local Name	このデータ型はデバイスの名前を示します。 名前の種類を選択した後、名前を入力してください。 名前の種類は以下から選ぶ事ができます。	
	Short local name	デバイスの省略した名前を示します。 デバイスの名前が長く、データの最大長を超過してしまう場合にご利用ください。
	Complete local name	デバイスの完全な名称を示します。
TX Power Level	このデータ型はアドバタイズを行うデバイスの送信電力を示します。	
Slave Connection Interval Range	このデータ型はアドバタイズを行うデバイスが推奨するコネクションインターバルを示します。 最大値・最小値を入力してください。	
Service Solicitation UUIDs	このデータ型はアドバタイズを行うデバイスが要求するサービスのリストを示します。 「Profile」タブで追加したサービスから選択することができ、チェックしたサービスをリストに追加することができます。	
Service Data	このデータ型はサービスで使用するデータを示します。 このデータの値はサービスの UUID とサービスのデータから構成されます。 Example) Service UUID [0x1234]　Service Data [0x56, 0x78, 0x9a, 0xbc] →設定する値 [123456789abc]	
Public Target Address	このデータ型はアドバタイジングデータの送信相手をパブリックデバイスアドレスで示します。 Example) Public BD Address [0x12:0x34:0x56:0x78:0x9a:0xbc] →設定する値 [123456789abc]	
Random Target Address	このデータ型はアドバタイジングデータの送信相手をランダムデバイスアドレスで示します。 Example) Random BD Address [0x12:0x34:0x56:0x78:0x9a:0xbc] →設定する値 [123456789abc]	
Appearance	このデータ型はデバイスの見た目を示します。 各見た目に対応する値は Bluetooth SIG の仕様ページの Assigned Numbers をご確認ください。 https://www.bluetooth.com	
Advertising Interval	このデータ型はデバイスのアドバタイジングインターバルを示します。 ここで設定した値はアドバタイジングデータとしてのみ使用されます。 アドバタイズ動作時のパラメータとしては使用されません。	
Manufacturer Specific Data	このデータは製造者独自のデータを設定することができます。 このデータの値はカンパニーID とデータから構成されます。 Example) Company ID [0x1234]　Specific Data [0x56, 0x78, 0x9a, 0xbc] →設定する値 [341256789abc]	

3.3.2　Scan Response Data の設定

Scan Response Data では Advertising 動作時に送信されるスキャンレスポンスデータを設定することができます。

チェックをつけた全てのデータ型がアドバタイジングデータとして追加されます。それぞれのデータ型について、ユーザは詳細な値を設定することができます。ユーザが追加することが可能なデータ型は表 3.6 に示します。

3.3.3　Advertising Parameter の設定

Advertising Parameter では Advertising 動作時に使用されるパラメータを設定する事ができます。

設定できるパラメータは表 3.7 に示します。

【注】デフォルトの設定で接続しづらい場合、「Slow Advertising Interval」のパラメータを変更してください。

表 3.7 Advertising 動作のパラメータ

パラメータ名	説明	
Fast	Advertising 動作時のタイミングに関するパラメータを設定できます。 このパラメータは「Enable Fast Advertising」がチェックされているときに設定できます。 チェックされていないときはこのパラメータは反映されません。 以下の項目が設定できます。	
	Advertising Interval	アドバタイジングインターバルを設定します。
	Advertising period	「Fast」のパラメータを使用する期間を設定します。
Slow	Advertising 動作時のタイミングに関するパラメータを設定できます。 「Enable Fast Advertising」がチェックされているとき、「Fast Advertising Period」に設定した期間の後に使用されます。チェックされていないときはこのパラメータが Advertising 動作時に最初から使用されます。	
	Advertising Interval	アドバタイジングインターバルを設定します。
	Advertising period	「Slow」のパラメータを使用する期間を設定します。 設定した期間の後、Advertising 動作は停止します。 このパラメータは「Set Advertising Period」をチェックしたときに設定できます。
Advertising channel	アドバタイジングチャンネルを設定します。 チェックした全てのチャンネルに送信されます。	
Address type	Advertising 動作時に使用するアドレスの種類を選択できます。 以下からアドレスの種類を選択してください。	
	Public address	Advertising 動作時にパブリックアドレスが使用されます。
	Random Address	Advertising 動作時にランダムアドレスが使用されます。 MCU 固有のスタティックアドレスが値として使用されます。

3.4　セントラルの設計

「Central」タブでは GAP セントラルロールで動作する際に必要なパラメータを設定する事ができます。このタブで設定した値は「Profile」タブで「Central」を選択したときのアプリケーションフレームワークを作成する際に使用されます。このタブでは以下の項目を設定することができます。

表 3.8 セントラルタブで設定できる項目

項目名	説明
Scan Parameter	Scan 動作のパラメータを設定することができます。
Scan filter data	Scan 動作時に使用されるフィルタのデータを設定することができます。
Connection Parameter	デバイスのコネクション時に使用されるパラメータを設定します。 ここで設定されたパラメータは Connection Request 送信時に使用されます。

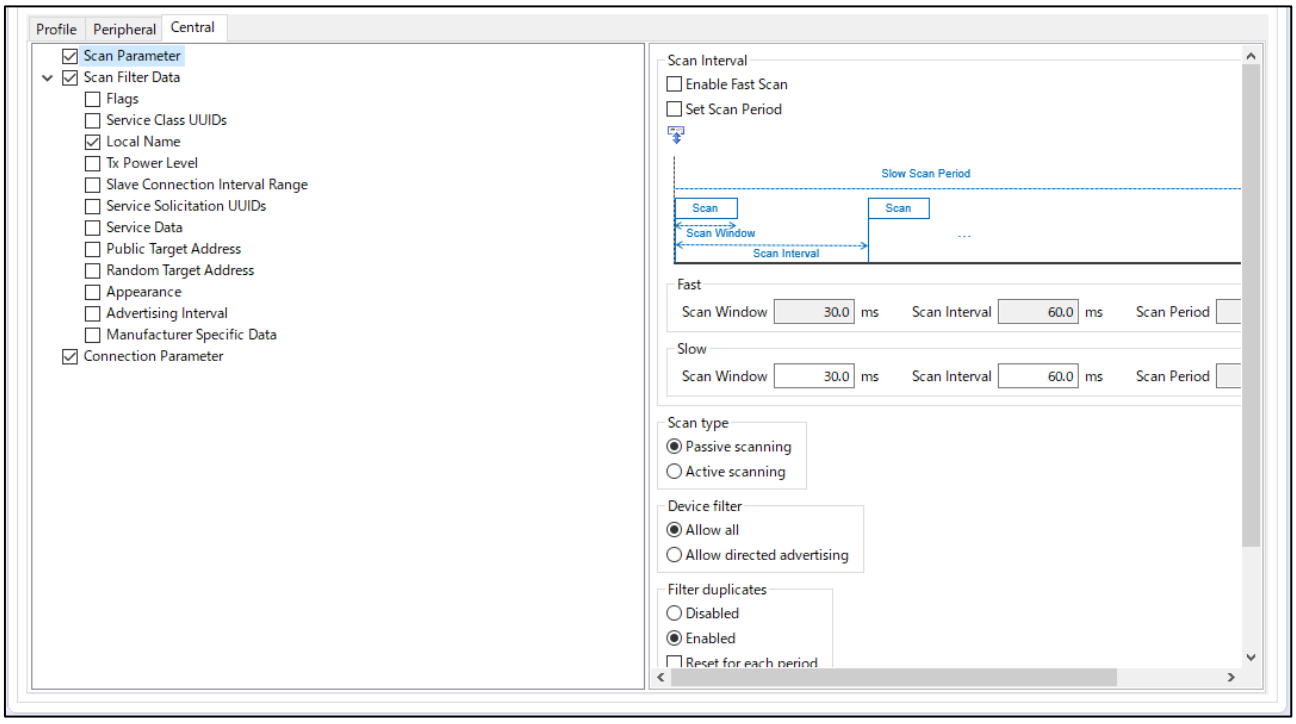

図 3.15 セントラルタブの設定画面

3.4.1　Scan Parameter の設定

Scan Parameter では Scan 動作時に使用されるパラメータを設定することができます。

設定できるパラメータは表 3.9 に示します。

【注】デフォルトの設定で接続しづらい場合、「Slow Scan Interval」と「Slow Scan Window」のパラメータを変更してください。

表 3.9 Scan 動作のパラメータ

パラメータ名	説明	
Fast	Scan 動作時のタイミングに関するパラメータを設定できます。 このパラメータは「Enable Fast Scan」がチェックされているときに設定できます。チェックされていないときはこのパラメータは反映されません。 以下の項目が設定できます。	
	Scan Window	スキャンウィンドウを設定します。
	Scan Interval	スキャンインターバルを設定します。
	Scan Period	「Fast」のパラメータを使用する期間を設定します。
Slow	Scan 動作時のタイミングに関するパラメータを設定できます。 「Enable Fast Scan」がチェックされているとき、「Fast Scan Period」に設定した期間の後に使用されます。チェックされていないときはこのパラメータが Scan 動作時に最初から使用されます。	
	Scan Window	スキャンウィンドウを設定します。
	Scan Interval	スキャンインターバルを設定します。
	Scan Period	「Slow」のパラメータを使用する期間を設定します。 設定した期間の後、Scan 動作は停止します。 このパラメータは「Set Scan Period」をチェックしたときに設定できます。
Scan type	Scan Type を選択することができます。 以下から Scan Type を選択してください。	
	Passive Scanning	パッシブスキャンが Scan 動作で行われます。
	Active Scanning	アクティブスキャンが Scan 動作で行われます。
Device filter	Scan 動作のフィルタポリシを設定できます。 以下からフィルタポリシを選択してください。	
	Allow all	デバイスのアドレスを指定していないダイレクトアドバタイジング PDU を除く、全てのアドバタイズ及びスキャンレスポンス PDU を受信します。
	Allow directed advertising	ターゲットアドレスがアイデンティティアドレスでもデバイスのアドレスと異なっているダイレクトアドバタイジング PDU を除く、全てのアドバタイズ及びスキャンレスポンス PDU を受信します。 デバイスのレゾルバブルプライベートアドレスを指定するダイレクトアドバタイジング PDU は受信することができます。
Filter duplicates	Scan 動作で同じデバイスから重複するアドバタイズ受信のフィルタを設定できます。 以下から選択してください	
	Disable	重複するアドバタイズ受信のフォルタを無効にし、重複するアドバタイズを全て受信します。
	Enabled	重複するアドバタイズ受信のフィルタを有効にし、重複するアドバタイズを受信しなくなります。 「Reset for each period」をチェックすると「Scan Period」ごとにフィルタをリセットします。

3.4.2　Scan Filter Data の設定

Scan Filter Data では Scan 動作時に使用されるフィルタのデータを設定することができます。

フィルタに設定したデータを含むアドバタイジングデータのみアプリケーションに通知されるようになります。チェックをつけたデータ型がフィルタデータとして設定されます。また設定したデータにユーザは詳細な値を設定することができます。ユーザが追加することが可能なデータ型は表 3.6 に示します。

【注】フィルタデータとして一つのデータ型のみ設定できます。

3.4.3　Connection Parameter の設定

Connection Parameter ではコネクション動作時に使用されるパラメータを設定することができます。

このパラメータは Connection Request 送信時に使用されます。

表 3.10 Connection 動作のパラメータ

パラメータ名	説明	
Parameter	コネクション動作のパラメータを設定する事ができます。 ここで設定したパラメータは Connection Request で送信され、コネクション完了後に使用されます。 以下の項目が設定できます。	
	Connection Interval	コネクションインターバルを設定します。
	Connection Latency	スレイブレイテンシを設定します。
	Connection Supervision Timeout	スーパービジョンタイムアウトを設定します。
Connection cancel	コネクション動作をキャンセルするためのパラメータを設定することができます。 以下の項目が設定できます。	
	Connection Timeout	コネクション動作のタイムアウトを設定します。 ペリフェラルデバイスがこのタイムアウトを超えて Connection Request に返答がない場合、コネクション動作はキャンセルされます。

3.5　その他の注意

3.5.1　QE for BLE から生成したコードを使用して 2 つの MCU を接続する場合

Bluetooth LE 通信が可能な評価ボードを 2 つ以上使用することで、Bluetooth LE 通信での相互接続が可能です。Bluetooth LE 通信での相互接続を行う場合、以下の点に注意して QE for BLE を設定してください。

●片側のデバイスをペリフェラル、もう片側のデバイスをセントラルに設定する

QE for BLE から生成されるアプリケーションフレームワークはペリフェラルのプログラムとセントラルのプログラム間で接続できるように作られています。

●Advertising Data と Scan Filter Data が合うように設定する

セントラルのプログラムで Scan を実行した場合、Scan Filter Data で設定した値を含む Advertising Event のみが検知されます。また、検知した Advertising Event を送信するデバイスに対して接続要求を送信します。そのため、ペリフェラルで設定した固有の Advertising Data を元に Scan Filter data に設定することでペリフェラルのデバイスと接続することができます。固有のデータとして「local name」を使用することを推奨します。

●「Enable Fast Advertising/Scan」を設定する

Advertising や Scan の動作について、QE for BLE のデフォルトの設定では「Slow」のパラメータが使用されるように設定されています。この設定では低い動作率にすることで消費電力を抑えることが可能ですが、互いのデータを容易に検知することが難しくなります。「Fast」を使用することで消費電力は増加してしまいますが、互いのデータを検知しやすくなり素早い接続が可能になります。

4.　プログラムの実装

この章では、QE for BLE から生成されたプログラムにユーザアプリケーションを追加する方法についてガイドします。QE for BLE から生成されたプログラムの一例を図 4.1 に示します。

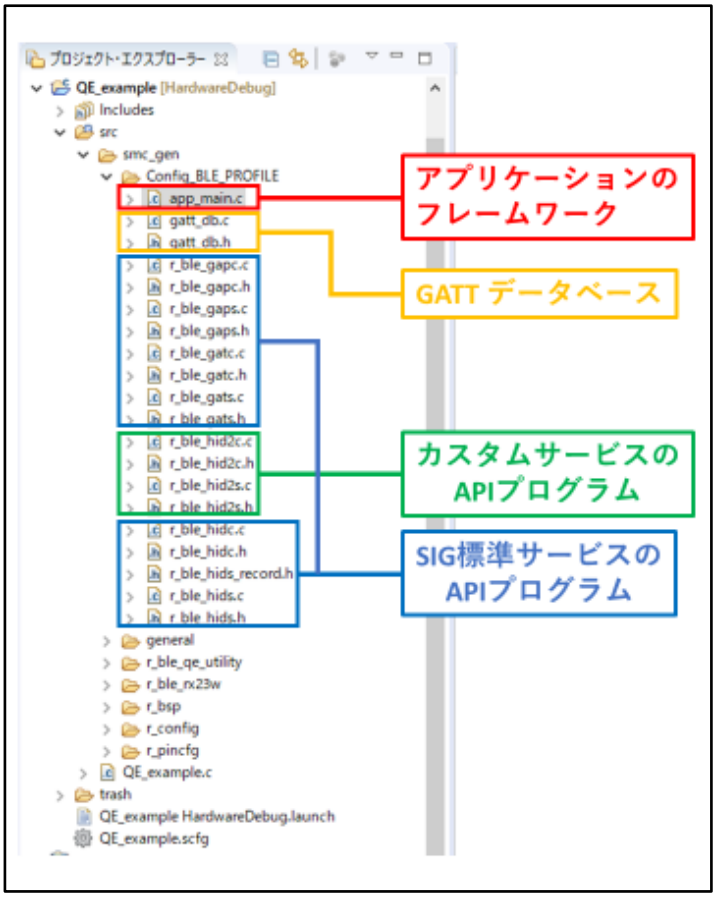

図 4.1 QE for BLE から生成されたプログラム

QE for BLE から生成されるプログラムは QE for BLE を使用するたびに新しく生成されます。BLE QE Utility モジュールのバージョン 1.10 以降を使用している場合、アプリケーションフレームワークにはユーザが実装したコードを保護するためにコードブロック機能を持つコメント行を実装しています(図 4.2)。ユーザはこのコメント行の間にコードを実装することで QE for BLE を使用しても新しいアプリケーションフレームワークに実装したコードを残すことができます。

```
/* Start user code for XXXX. Do not edit comment generated here */

    Implement user code here

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

図 4.2 ユーザコードの実装場所

コードを新しく生成した際に再生成前のプログラムはプロジェクト内の trash フォルダにコピーされます(図 4.3)。そのため、コメント行の間にユーザコードを追加できない場合は、必要なコードを適宜コピーし直してください。

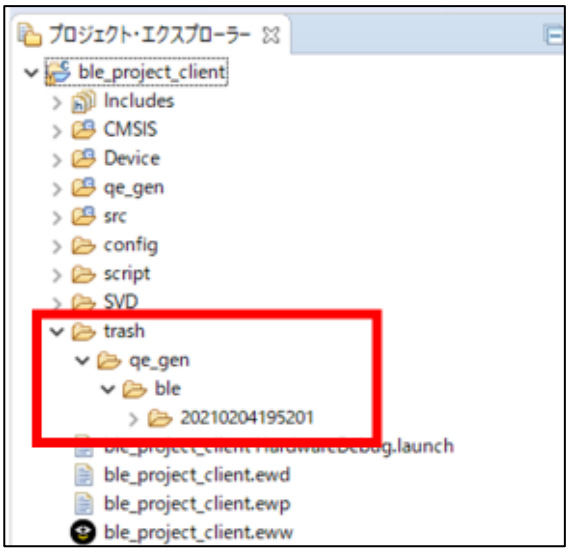

図 4.3 trash フォルダ

4.1　サービスのプログラム

SIG 標準サービスの API プログラムとカスタムサービスの API プログラムで共通の仕様についてガイドします。

4.1.1　サービスで定義される API

SIG 標準サービスやカスタムサービスの API は一定の規則に従って命名されており、API の名前を確認するだけでどのような動作を行うかを判断することができます。各キャラクタリスティックやディスクリプタの値に対する動作は以下のような命名規則に従って生成されます。

R_BLE_[service][S or C]_[operation]

[service]は QE for BLE でサービスの[略称]に設定した文字列になります。[S or C]はサービスがサーバの場合 S、クライアントの場合 C になります。[operation]は QE for BLE でキャラクタリスティックやディスクリプタの[properties]に設定した Bluetooth LE の通信における各動作の名前になります。

[operation]には Bluetooth LE 通信におけるキャラクタリスティックやディスクリプタの値の送受信の動作が設定されます。表 4.1 にサーバ側の API プログラムで生成される API の[operation]の一覧、表 4.2 にクライアント側の API プログラムで生成される API の[operation]の一覧を示します。両表中の[characteristic]には QE for BLE でキャラクタリスティックの[略称]に設定した文字列、[descriptor]には QE for BLE でディスクリプタに設定した文字列が設定されます。

表 4.1 サーバ側で生成される API

operation	説明
Get[characteristic] Get[characteristic][descriptor]	キャラクタリスティックやディスクリプタの値を GATT データベースから取得します。 Write Characteristic Value 動作などで変更された値を確認することができます。
Set[characteristic] Set[characteristic][descriptor]	キャラクタリスティックやディスクリプタの値を GATT データベースへ格納します。 格納された値は Read Characteristic Value 動作などで使用されます。
Notify[characteristic]	Handle Value Notification を送信して Notification 動作を開始します。 この API を使用しても GATT データベースに値は格納されません。
Indicate[characteristic]	Handle Value Indication を送信して Indication 動作を開始します。 この API を使用しても GATT データベースに値は格納されません。

表 4.2 クライアント側で生成される API

operation	説明
Get[characteristic]AttrHdl	discovery 動作で取得した、キャラクタリスティックのアトリビュートハンドルを取得します。 キャラクタリスティックに含まれるディスクリプタのアトリビュートハンドルも取得することができます。 こちらの API を使用する前に discovery 動作を完了してください。
Write[characteristic] Write[characteristic][descriptor]	Write Request を送信して Write characteristic value 動作を開始します。 値のデータ長が MTU サイズを超過する場合は Write prepare request を送信して Write Long Characteristic Value 動作を開始します。
Read[characteristic] Read[characteristic][descriptor]	Read Request を送信して Read Characteristic Value 動作を開始します。 値のデータ長が MTU サイズを超過する場合は Read Blob request を送信して Read Long Characteristic Value 動作を開始します。

また各サービスにはその構成にかかわらず、表 4.3 に示す関数が定義されています。表中の[service]は QE for BLE でサービスの[略称]に設定した文字列、[S or C]はサービスがサーバの場合 S、クライアントの場合 C に設定されます。

表 4.3 各サービスの API プログラムに定義されている API

API 名	説明
R_BLE_[service][S or C]_Init	サービスの初期化関数です。 サービスの API プログラムを使用する場合には必ず呼び出してください。
R_BLE_[service][S or C]_GetServAttrHdl	discovery 動作で取得したサービスのアトリビュートハンドルを返します。 discovery 動作が完了してから使用してください。 クライアント側の API プログラムのみ実装されます。
R_BLE_[service][S or C]_ServDiscCb	discovery 動作を行うための関数です。 ディスカバリライブラリを使用する時にコールバック関数として使用されます。 クライアント側の API プログラムのみ実装されます。

4.1.2　サービスのイベントについて

カスタムサービスを含むすべてのサービスの API プログラムは、Bluetooth LE の通信における各送受信の動作に対してイベントが定義されています。BLE プロトコルスタックからアプリケーションのフレームワークにそれぞれのサービス単位でイベントが通知されます。ユーザは定義されたイベントに対応する動作をコールバック関数に実装することでアプリケーションを開発することができます。

各イベントは扱うデータの種類と通信時の動作に基づいて命名されています。

キャラクタリスティックのデータに関するイベントは以下のように命名されます。

BLE_[service][S or C]_EVENT_[characteristic]_[event type]

[service]には QE for BLE でサービスの[略称]に設定した文字列、[characteristic]には QE for BLE でキャラクタリスティックの[略称]に設定した文字列になります。[S or C]はサービスがサーバの場合 S、クライアントの場合 C になります。[event type]に入る文字列は後述するイベントの種類によって決定します。

ディスクリプタのデータに関するイベントは以下のように命名されます。

BLE_[service][S or C]_EVENT_[characteristic]_[descriptor]_[event type]

[service]には QE for BLE でサービスの[略称]に設定した文字列、[characteristic]には QE for BLE でキャラクタリスティックの[略称]に設定した文字列、[descriptor]には QE for BLE でディスクリプタに設定した文字列になります。[S or C]はサービスがサーバの場合 S、クライアントの場合 C になります。[event type]に入る文字列は後述するイベントの種類によって決定します。

[event type]には Bluetooth LE 通信における送受信のイベントが入力されます。発生するイベントの種類は GATT のサーバ側とクライアント側で異なります。サーバ側に発生するイベントの一覧を表 4.4、クライアント側に発生するイベントの一覧を表 4.5 に示します。

表 4.4 サーバ側で発生するイベント

イベント	説明
WRITE_REQ	Write Request または Prepare Write Request を受信した時に発生するイベントです。 Write Characteristic Value 動作及び Write Characteristic Long Value 動作で使用されます。 GATT イベント： BLE_GATTS_OP_CHAR_PEER_WRITE_REQ
WRITE_COMP	Write Response または Execute Write Response を送信した後に発生するイベントです。 Write Characteristic Value 動作及び Write Characteristic Long Value 動作で使用されます。 GATT イベント： BLE_GATTS_EVENT_CHAR_WRITE_RSP_COMP BLE_GATTS_EVENT_EXE_WRITE_RSP_COMP
WRITE_CMD	Write Command または Signed Write Command を受信した時に発生するイベントです。 Write Characteristic Without Response 動作及び Signed Write 動作で使用されます。 GATT イベント： BLE_GATTS_OP_CHAR_PEER_WRITE_CMD
READ_REQ	Read Request を受信した時に発生するイベントです。 Read Characteristic Value 動作及び Read Characteristic Long Value 動作で使用されます。 GATT イベント： BLE_GATTS_OP_CHAR_PEER_READ_REQ
HDL_VAL_CNF	Handle Value Confirmation を受信した時に発生するイベントです。 Indication 動作で使用されます。 GATT イベント： BLE_GATTS_EVENT_HDL_VAL_CNF

表 4.5 クライアント側で発生するイベント

イベント	説明
WRITE_RSP	Write Response または Execute Write Request を受信した時に発生するイベントです。 Write Characteristic Value 動作及び Write Characteristic Long Value 動作で使用されます。 GATT イベント： BLE_GATTC_EVENT_CHAR _WRITE_RSP BLE_GATTC_EVENT_LONG_CHAR_WRITE_COMP
READ_RSP	Read Response または Read Blob Response を受信した時に発生するイベントです。 Read Characteristic Value 動作及び Read Characteristic Long Value 動作で使用されます。 GATT イベント： BLE_GATTC_EVENT_CHAR_READ_RSP BLE_GATTC_EVENT_CHAR_PART_READ_RSP（失敗した場合） BLE_GATTC_EVENT_LONG_CHAR_READ_COMP
HDL_VAL_NTF	Handle Value Notification を受信した時に発生するイベントです。 Notification 動作で使用されます。 GATT イベント： BLE_GATTC_EVENT_HDL_VAL_NTF
HDL_VAL_IND	Handle Value Indication を受信した時に発生するイベントです。 Indication 動作で使用されます。 GATT イベント： BLE_GATTC_EVENT_HDL_VAL_IND

図 4.4 にカスタムサービスに定義されたイベントの例を示します。本例では BLE FIT モジュールに同梱されるサンプルプロジェクトで使用されるカスタムサービス「LED Switch Service (略称：ls)」のクライアント側に定義される「Switch State (略称：switch_state)」キャラクタリスティックと「LED Blink Rate (略称：blink_rate)」キャラクタリスティックのイベントを示します。

```
  /* LED Switch Service(Abbreviation:ls) Client Event Type Definition */
  typedef enum {

      /* Switch State Characteristic (Abbreviation:switch state) */
    /* Handle Value Notification */
      BLE_LSC_EVENT_SWITCH_STATE_HDL_VAL_NTF
       = BLE_SERVC_ATTR_EVENT(BLE_LSC_SWITCH_STATE_IDX, BLE_SERVC_HDL_VAL_NTF),

      /* Client Characteristic Configuration Descriptor (Abbreviation:cli_cnfg) */
      /* Read response */
    BLE_LSC_EVENT_SWITCH_STATE_CLI_CNFG_READ_RSP
       = BLE_SERVC_ATTR_EVENT(BLE_LSC_SWITCH_STATE_CLI_CNFG_IDX, BLE_SERVC_READ_RSP),
      /* Write response */
      BLE_LSC_EVENT_SWITCH_STATE_CLI_CNFG_WRITE_RSP
       = BLE_SERVC_ATTR_EVENT(BLE_LSC_SWITCH_STATE_CLI_CNFG_IDX,
BLE_SERVC_WRITE_RSP),

      /* LED Blink Rate Characteristic (Abbreviation:blink rate) */
      /* Read Response */
      BLE_LSC_EVENT_BLINK_RATE_READ_RSP
       = BLE_SERVC_ATTR_EVENT(BLE_LSC_BLINK_RATE_IDX, BLE_SERVC_READ_RSP),
      /* Write Response */
      BLE_LSC_EVENT_BLINK_RATE_WRITE_RSP
       = BLE_SERVC_ATTR_EVENT(BLE_LSC_BLINK_RATE_IDX, BLE_SERVC_WRITE_RSP),

  } e_ble_lsc_event_t;
```

図 4.4 カスタムサービスに定義されたイベントの例

4.2　カスタムサービスの実装

SIG 標準サービスで定義されていない機能を使用する場合、カスタムサービスを作成する必要があります。本章では QE for BLE で作成されたカスタムサービスの API プログラムを利用可能にするための方法をガイドします。

4.2.1　encode/decode 関数の実装

アプリケーションフレームワークでは、QE for BLE の「Field」で指定した形式でキャラクタリスティックやディスクリプタの値を扱います。一方、GATT データベースでは QE for BLE の「DBSize」の長さの 8bit データ配列として扱われ、BLE Protocol Stack によってビット列でデータの送受信が行われます。そのため、サービス API プログラムでは encode/decode 関数を使用してアプリケーション用の構造体データと GATT データベース用のデータを変換します。

encode/decode 関数は図 4.5 のようにデータ形式を変換します。

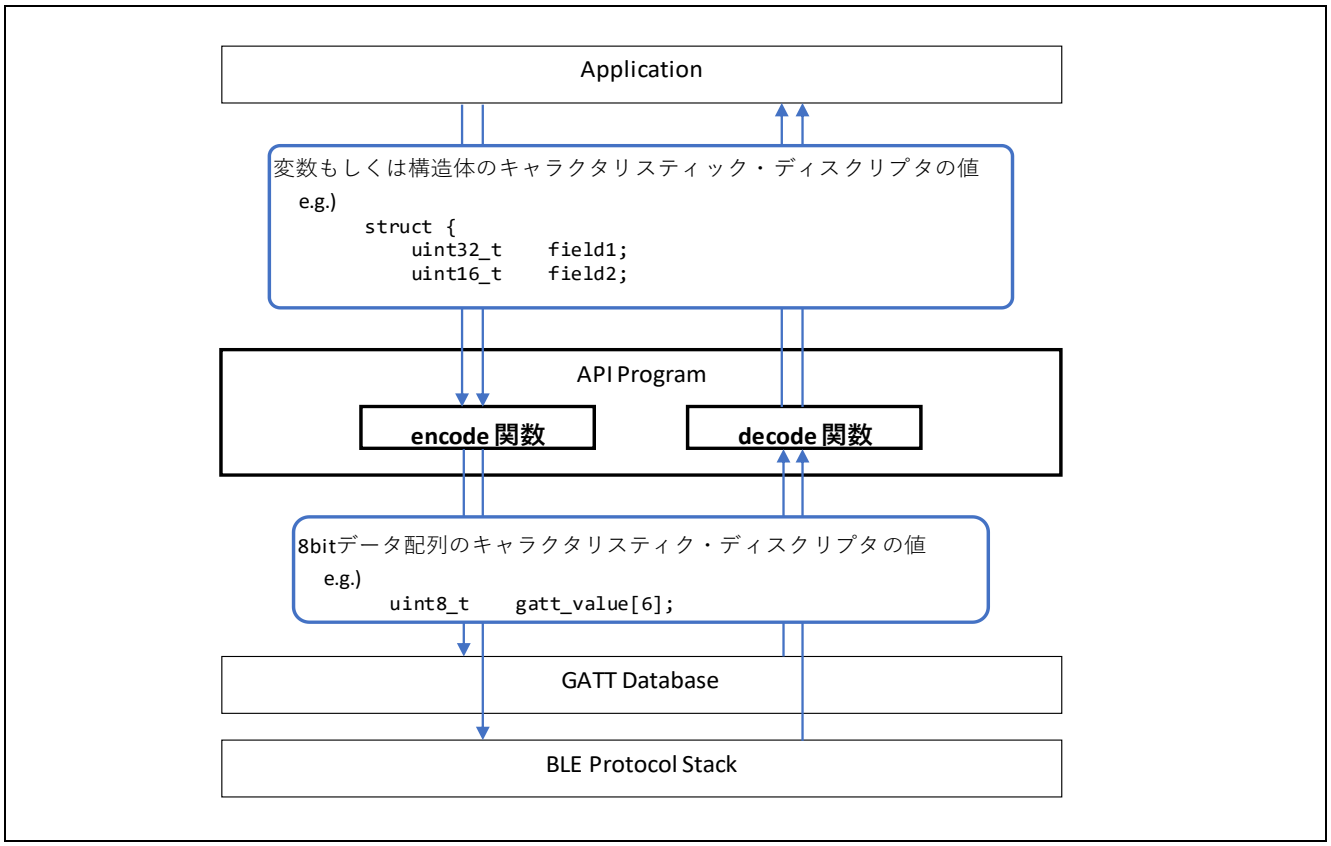

図 4.5 encode/decode 関数の関係

encode 関数は API を使用してキャラクタリスティックやディスクリプタのデータを送信するとき、または GATT データベースの値を変更するときなどでサービス API プログラムによって使用されます。decode 関数は受信したデータがコールバック関数を通じてアプリケーションに通知されるときなどで使用されます。

GATT クライアントが GATT サーバに Write 動作で新しい値を書き込む場合の encode/decode 関数のユースケースを図 4.6 に示します。encode 関数は GATT クライアント側のサービス API プログラムで使用され、decode 関数は GATT サーバ側のサービス API プログラムで使用されます。

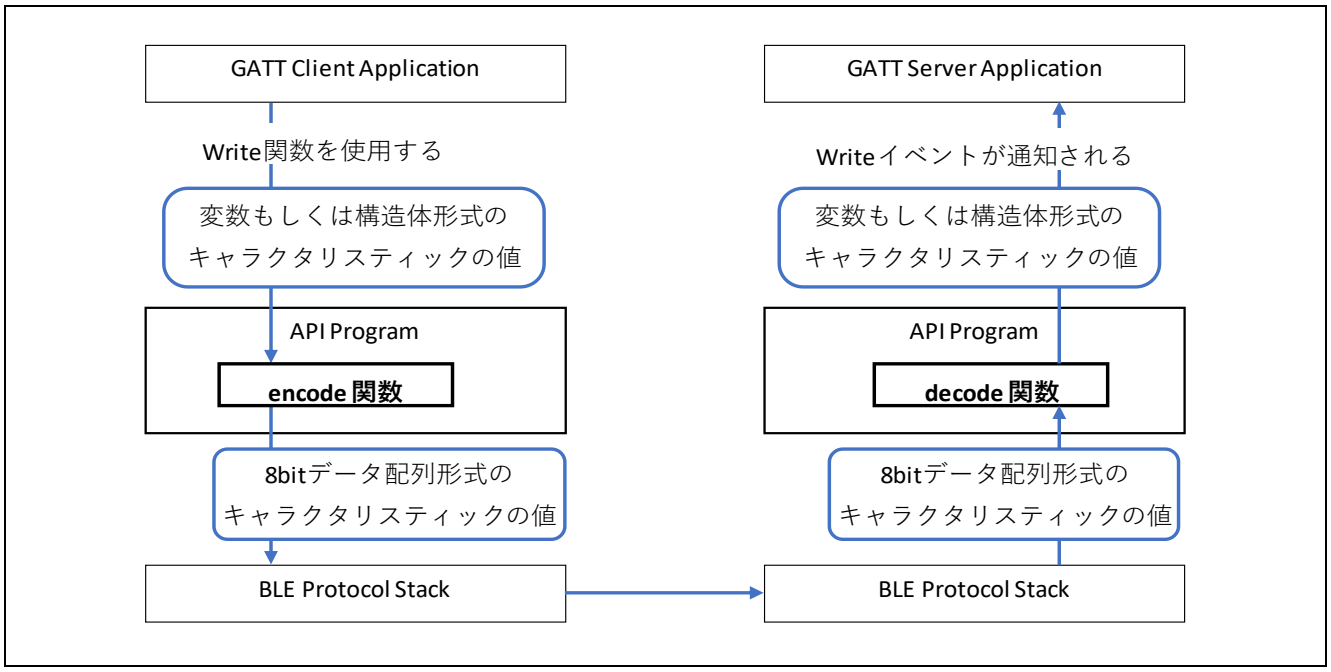

図 4.6 Write 動作時の encode/decode 関数のユースケース

同様に GATT サーバが GATT クライアントに Notify 動作で値を通知する場合の encode/decode 関数のユースケースを図 4.7 に示します。encode 関数は GATT サーバ側のサービス API プログラムで使用され、decode 関数は GATT クライアント側のサービス API プログラムで使用されます。

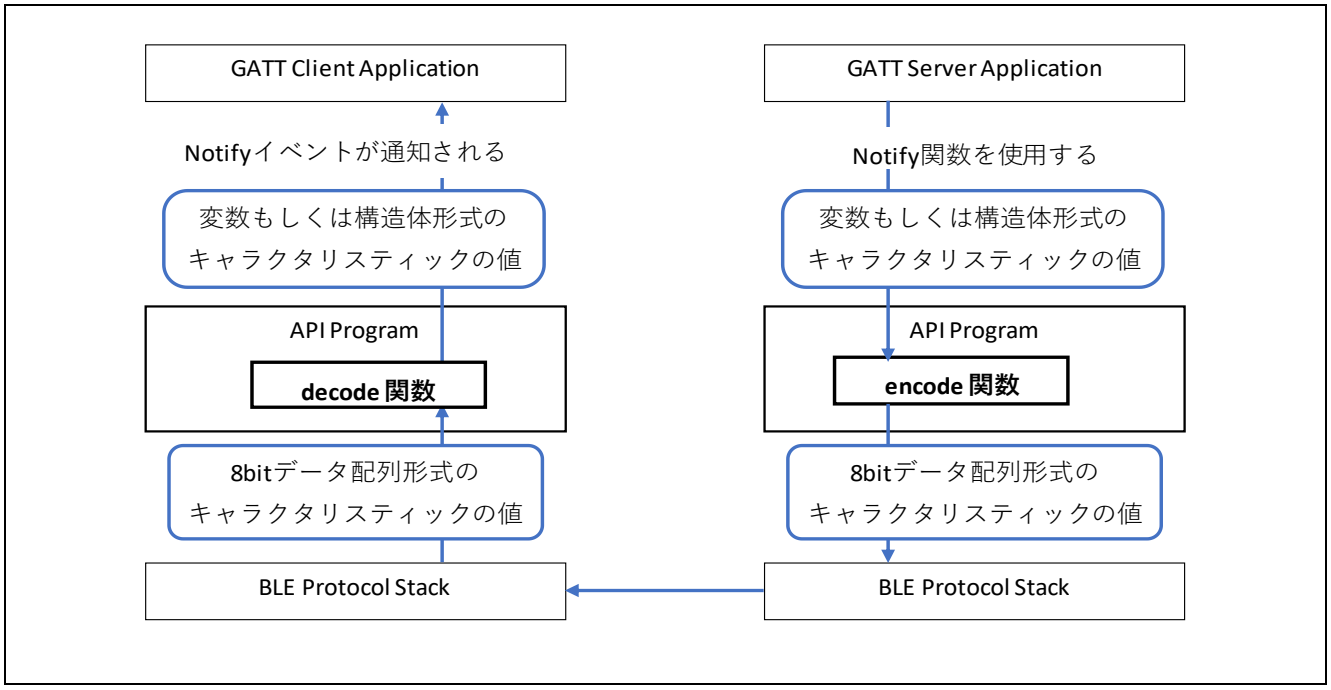

図 4.7 Notify 動作時の encode/decode 関数のユースケース

カスタムサービスでは各キャラクタリスティックやディスクリプタに対して encode/decode 関数が生成されますが、その中身は実装されていません。そのため、それぞれのデータ構造に対応した encode/decode 関数を実装します。uint8_t 型などの基本的なデータ構造や ieee11073 SFLOAT 型などサービスでよく使用されるデータ構造については、encode/decode 関数を実装するためのマクロおよび関数が定義されています。それらのマクロや関数を適宜呼び出すことで encode/decode 関数を簡単に実装することができます。表 4.6 に提供される encode/decode 関数を実装するためのマクロおよび関数の一覧を示します。

表 4.6 encode/decode 関数を簡単に実装するためのマクロおよび関数

field に設定した型	encode	decode
char uint8_t int8_t	BT_PACK_LE_1_BYTE(*dst, *src)	BT_UNPACK_LE_1_BYTE(*dst, *src)
uint16_t int16_t	BT_PACK_LE_2_BYTE(*dst, *src)	BT_UNPACK_LE_2_BYTE(*dst, *src)
uint32_t int32_t	BT_PACK_LE_4_BYTE(*dst, *src)	BT_UNPACK_LE_4_BYTE(*dst, *src)
st_ble_ieee11073_sfloat_t	pack_st_ble_ieee11073_sfloat_t(*p_dst, *p_src)	unpack_st_ble_ieee11073_sfloat_t(*p_dst, *p_src)
st_ble_date_time_t	pack_st_ble_date_time_t(*p_dst, *p_src)	unpack_st_ble_date_time_t(*p_dst, *p_src)

図 4.9 に示す Field を持つカスタムキャラクタリスティック(略称:cc)の encode 関数の実装例を図 4.8 に示します。この encode 関数では表 4.6 で提供される関数を使用しています。

```
  typedef struct {
      uint16_t field_u16; /**<field_u16 */
      uint8_t field_u8; /**< field_u8 */
      st_ble_date_time_t field_date; /**< field_date */
  } st_ble_css_cc_t;

  static ble_status_t encode_st_ble_css_cc_t(const st_ble_css_cc_t *p_app_value,
st_ble_gatt_value_t *p_gatt_value)
  {
      /* Start user code for Custom Characteristic characteristic value encode
function. Do not edit comment generated here */

       /* Value to check the correct position to pack value */
       uint8_t pos = 0;
       /*  Use BT_PACK_LE_2_BYTE macro for uint16_t type */
       BT_PACK_LE_2_BYTE(&p_gatt_value->p_value[pos], &p_app_value->field_u16);
       pos += 2;
       /*  Use BT_PACK_LE_1_BYTE macro for uint8_t type */
       BT_PACK_LE_1_BYTE(&p_gatt_value->p_value[pos], &p_app_value->field_u8);
       pos += 1;
       /*  Use pack_st_ble_date_time_t() function for st_ble_date_time_t type */
       pack_st_ble_date_time_t(&p_gatt_value->p_value[pos], &p_app_value->field_date);
       pos += 7
       /*  Store the total data size as value length */
       p_gatt_value->value_len = pos;
    /* End user code. Do not edit comment generated here */
      return BLE_SUCCESS;
  }
```

図　4.8 Field の実装例(encode 関数の実装例)

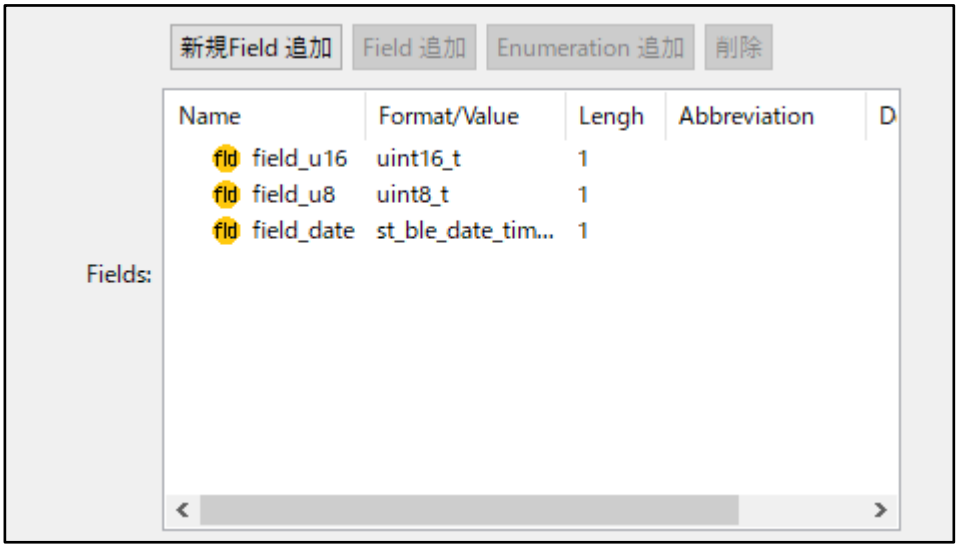

図　4.9 Field の実装例

4.2.2　サービス内でのコールバックの実装

Bluetooth LE ソフトウェアではデータの送受信やコネクションの確立などの通信に際してイベントを発生します。イベントに対応してプログラムを実装することでアプリケーションを実現することができます。イベントに対するコールバックは 2 種類の方法で実装することができます。

- サービス内でのコールバック
- アプリケーションでのコールバック

サービス内のコールバックを実装した場合、アプリケーションのコールバックは呼ばれなくなるのでご注意ください。ここではサービス内でのコールバックの実装についてガイドします。アプリケーションでのコールバックの実装については「4.3.1 アプリケーションでのコールバックの実装」でガイドします。

実装するカスタムサービスの機能によっては以下のような動作が求められます。

- キャラクタリスティックやディスクリプタに不正な値が書き込まれたときにエラーを返す。
- キャラクタリスティックやディスクリプタに書き込まれた特定の命令に対して別のキャラクタリスティックの値を返す。

このような動作を実装する場合、カスタムサービスの API プログラムに実装することで移植性が向上し、様々なアプリケーションに利用することができるようになります。

カスタムサービスの API プログラムの各キャラクタリスティックには図 4.10 のような構造体が定義されています。

```
static const st_ble_servs_char_info_t gs_nc_char = {
    .start_hdl    = BLE_CSS_CC_DECL_HDL,
    .end_hdl      = BLE_CSS_CC_CLI_CNFG_DESC_HDL,
    .char_idx     = BLE_CSS_CC_IDX,
    .app_size     = sizeof(st_ble_css_cc_t),
    .db_size      = BLE_CSS_CC_LEN,
    .decode       = (ble_servs_attr_decode_t)decode_st_ble_css_cc_t,
    .encode       = (ble_servs_attr_encode_t)encode_st_ble_css_cc_t,
    .pp_descs     = gspp_cc_descs,
    .num_of_descs = ARRAY_SIZE(gspp_cc_descs),
};
```

図 4.10 サービス API プログラムのキャラクタリスティック構造体

この構造体を図 4.11 に示すように編集することで、カスタムサービス内にキャラクタリスティックのイベントに対するコールバック関数を登録することができます。

```
  /* Callback function for write_req_cb */
  static void css_cc_write_req_cb(const void *p_attr, uint16_t conn_hdl, ble_status_t
result, const void *p_app_value)
  {
       /*…Implement program…*/
  }

  /* Callback function for write_comp_cb */
  static void css_cc_write_comp_cb(const void *p_attr, uint16_t conn_hdl,
ble_status_t result, const void *p_app_value)
  {
       /*…Implement program…*/
  }

  static const st_ble_servs_char_info_t gs_nc_char = {
      .start_hdl    = BLE_CSS_CC_DECL_HDL,
      .end_hdl      = BLE_CSS_CC_CLI_CNFG_DESC_HDL,
      .char_idx     = BLE_CSS_CC_IDX,
      .app_size     = sizeof(st_ble_css_cc_t),
      .db_size      = BLE_CSS_CC_LEN,
      .decode       = (ble_servs_attr_decode_t)decode_st_ble_css_cc_t,
      .encode       = (ble_servs_attr_encode_t)encode_st_ble_css_cc_t,
      .pp_descs     = gspp_cc_descs,
      .num_of_descs = ARRAY_SIZE(gspp_cc_descs),
      .write_req_cb = css_cc_write_req_cb,
      .write_comp_cb = css_cc_write_comp_cb,
  };
```

図 4.11 サービス API プログラムのコールバック関数の実装

登録できるコールバックはサーバ側とクライアント側で異なります。サーバ側が登録できるコールバック関数を表 4.7 に、クライアント側が登録できるコールバック関数を表 4.8 に示します。各イベントの詳細については BLE FIT モジュールに付属する「R_BLE API ドキュメント(r_ble_api_spec.chm)」をご確認ください。

表 4.7 サーバ側のキャラクタリスティックが登録できるコールバック

登録できるコールバック関数	対応するイベント
write_req_cb	このコールバック関数は Write Request もしくは Prepare Write Request を受信したときに呼び出されます。 Write 動作、Long Write 動作で使用できます。 GATT イベント： BLE_GATTS_OP_CHAR_PEER_WRITE_REQ
write_comp_cb	このコールバック関数は Write Response もしくは Execute Write Response を送信したときに呼び出されます。 Write 動作、Long Write 動作で使用できます。 GATT イベント： BLE_GATTS_EVENT_CHAR_WRITE_RSP_COMP BLE_GATTS_EVENT_EXE_WRITE_RSP_COMP
write_cmd_cb	このコールバック関数は Write Command もしくは Signed Write Command を受信したときに呼び出されます。 Write Without Response 動作、Signed Write 動作で使用できます。 GATT イベント： BLE_GATTS_OP_CHAR_PEER_WRITE_CMD
read_req_cb	このコールバック関数は Read Request を受信したときに呼び出されます。 Read 動作、Long Read 動作で使用できます。 GATT イベント： BLE_GATTS_OP_CHAR_PEER_READ_REQ
hdl_val_cnf_cb	このコールバック関数は Handle Value Confirmation を受信したときに呼び出されます。 Indication 動作で使用できます。 GATT イベント： BLE_GATTS_EVENT_HDL_VAL_CNF
flow_ctrl_cb	このコールバック関数は TX Flow イベントが通知されたときに呼び出されます。 VS イベント： BLE_VS_EVENT_TX_FLOW_STATE_CHG

表 4.8 クライアント側のキャラクタリスティックが登録できるコールバック

登録できるコールバック関数	対応するイベント
write_rsp_cb	このコールバック関数は Write Response もしくは Prepare Write Request を受信したときに呼び出されます。 Write 動作、Long Write 動作で使用できます。 GATT イベント： BLE_GATTC_EVENT_CHAR_WRITE_RSP BLE_GATTC_EVENT_LONG_CHAR_WRITE_RSP
read_rsp_cb	このコールバック関数は Read Response もしくは Read Blob Response を受信したときに呼び出されます。 Read 動作、Long Read 動作で使用できます。 GATT イベント： BLE_GATTC_EVENT_CHAR_READ_RSP BLE_GATTC_EVENT_LONG_CHAR_READ_COMP
hdl_val_ntf_cb	このコールバック関数は Handle Value Notification を受信したときに呼び出されます。 Notify 動作で使用することができます。 GATT イベント： BLE_GATTC_EVENT_HDL_VAL_NTF
hdl_val_ind_cb	このコールバック関数は Handle Value Indication を受信したときに呼び出されます。 Indicate 動作で使用できます。 GATT イベント： BLE_GATTC_EVENT_HDL_VAL_IND

各ディスクリプタでもキャラクタリスティックのものと同様に図 4.12 のような構造体が定義されています。この構造体を編集することでディスクリプタにもコールバック関数を登録することができます。

```
static const st_ble_servs_desc_info_t gs_cc_cd = {
    .attr_hdl = BLE_CSS_CC_CD_DESC_HDL,
    .app_size = sizeof(uint8_t),
    .desc_idx = BLE_CSS_CC_CD_IDX,
    .db_size  = BLE_CSS_CC_CD_LEN,
    .decode   = (ble_servs_attr_decode_t)decode_uint8_t,
    .encode   = (ble_servs_attr_encode_t)encode_uint8_t,
};
```

図 4.12 サービス API プログラムのディスクリプタ構造体

ディスクリプタでは登録できるコールバックの種類がキャラクタリスティックのものと異なります。サーバ側に登録できるコールバックを表 4.9、クライアント側に登録できるコールバックを表 4.10 に示します。各イベントの詳細については BLE FIT モジュールに付属する「R_BLE API ドキュメント (r_ble_api_spec.chm)」をご確認ください。

表 4.9 サーバ側のディスクリプタに登録できるコールバック

登録できるコールバック関数	対応するイベント
write_req_cb	このコールバック関数は Write Request もしくは Prepare Write Request を受信したときに呼び出されます。 Write 動作、Long Write 動作で使用できます。 GATT イベント： BLE_GATTS_OP_CHAR_PEER_CLI_CNFG_WRITE_REQ BLE_GATTS_OP_CHAR_PEER_SER_CNFG_WRITE_REQ BLE_GATTS_OP_CHAR_PEER_USR_CNFG_WRITE_REQ BLE_GATTS_OP_CHAR_PEER_HLD_CNFG_WRITE_REQ
write_comp_cb	このコールバック関数は Write Response もしくは Execute Write Response を受信したときに呼び出されます。 Write 動作、Long Write 動作で使用できます。 GATT イベント： BLE_GATTS_EVENT_WRITE_RSP_COMP BLE_GATTS_EVENT_EXE_WRITE_RSP_COMP
read_req_cb	このコールバック関数は Read Request を受信したときに呼び出されます。 Read 動作、Long Read 動作で使用できます。 GATT イベント： BLE_GATTS_OP_CHAR_PEER_CLI_CNFG_READ_REQ BLE_GATTS_OP_CHAR_PEER_SER_CNFG_READ_REQ BLE_GATTS_OP_CHAR_PEER_USR_CNFG_READ_REQ BLE_GATTS_OP_CHAR_PEER_HLD_CNFG_READ_REQ

表 4.10 クライアント側のディスクリプタに登録できるコールバック

登録できるコールバック関数	対応するイベント
write_rsp_cb	このコールバック関数は Write Response もしくは Prepare Write Response を受信したときに呼び出されます。 Write 動作、Long Write 動作で使用できます。 GATT イベント： BLE_GATTC_EVENT_CHAR_WRITE_RSP BLE_GATTC_EVENT_LONG_CHAR_WRITE_RSP
read_rsp_cb	このコールバック関数は Read Response もしくは Read Blob Response を受診したときに呼び出されます。 Read 動作、Long Read 動作で使用できます。 GATT イベント： BLE_GATTC_EVENT_CHAR_READ_RSP BLE_GATTC_EVENT_LONG_CHAR_READ_COMP

4.3　アプリケーションの実装

本章ではユーザアプリケーションやプロファイルのベースとなるフレームワークであるファイル app_main.c を実装する方法をガイドします。

4.3.1　アプリケーションでのコールバックの実装

Bluetooth LE ソフトウェアではデータの送受信やコネクションの確立などの通信に際してイベントを発生します。そのイベントに対してコールバックの処理を実装することでアプリケーションを実装します。Bluetooth LE ソフトウェアではイベントに対するコールバックを 2 種類の方法で実装することができます。

- サービス内でのコールバック
- アプリケーションでのコールバック

サービス内のコールバックを実装した場合、アプリケーションのコールバックは呼ばれなくなるのでご注意ください。ここではアプリケーションでのコールバックの実装についてガイドします。

アプリケーションでは Bluetooth LE 通信の基本的なイベントの処理を実装します。

プロファイルに含まれる各キャラクタリスティックやディスクリプタのデータを送受信する際のイベントについて説明します。コネクションの確立やペアリングの完了などのイベントについては「RX23W グループ Bluetooth Low Energy アプリケーション開発者ガイド (R01AN5504)」の「3. ユーザコードの実装方法」をご確認ください。

各イベントに対する処理はアプリケーションのフレームワーク上に生成されているコールバック関数に実装します。コールバック関数はプロファイルに含まれるサービスごとに生成されています。コールバック関数の命名ルールは以下の通りです。

[service][s or c]_cb

[service]は QE for BLE でサービスの[略称]に設定した文字列になります。[s or c]はサービスがサーバの場合 s、クライアントの場合 c になります。

サービスごとに発生するイベント名については、「4.1.2 サービスのイベントについて」を参照してください。図 4.13 にカスタムサービスのコールバック関数の実装例を示します。実装例は「RX23W グループ BLE モジュール Firmware Integration Technology(R01AN4860)」のサンプルプログラムで仕様される LED Switch Service(略称: LS)のサーバ側になります。クライアント側から Write 動作で送信されたデータを受信時、ソフトウェアタイマを更新する処理が実装されています。ソフトウェアタイマについては「RX23W グループ Bluetooth Low Energy アプリケーション開発者ガイド (R01AN5504)」の「4.1 ソフトウェアタイマ」をご確認ください

```
  static void lss_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t
*p_data)
  {
      switch (type)
      {
          case BLE_LSS_EVENT_BLINK_RATE_WRITE_REQ:
          {
              uint8_t rate = *(uint8_t *)p_data->p_param;
              R_BLE_TIMER_UpdateTimeout(gs_timer_hdl, rate*100);
          } break;

          default:
              break;
      }
  }
```

図 4.13 カスタムサービスのコールバック関数の実装例

4.4　その他の注意点

4.4.1　複数のサービスを実装する場合

複数のサービスを実装する場合、サービスに含まれるキャラクタリスティック及びディスクリプタのコードサイズに注意してください。コードサイズが対象デバイスの RAM/ROM サイズを超える場合、コンパイルできません。BLE Protocol Stack が使用する ROM/RAM サイズは「RX23W グループ BLE モジュール Firmware Integration Technology(R01AN4860)」のコードサイズの章を参照してください。

4.4.2　同一サービスを実装する場合

QE for BLE では同じ SIG 標準サービスを複数個プロファイルに追加した場合、ファイル名の競合などが原因で正しくプログラムを生成することができません。そのため SIG 標準サービスを複数個実装する場合は、1 つのサービスを SIG 標準サービスとして追加した後に、残りのサービスをカスタムサービスとして実装します。ここでは例として SIG 標準サービスである HIDS(Human Interface Device Service)を 2 個実装する場合を想定します。

QE for BLE で SIG 標準サービスとして HIDS を 2 個追加します。この 2 個の SIG 標準サービスの内、どちらかをカスタムサービスに設定します。SIG 標準サービスからカスタムサービスに変更するにはサービス設定画面のカスタマイズボタンをクリックします。カスタムサービスに変更したサービスには以下の変更を加えます。

- サービスの[UUID]が同じサービス間で一致するように変更します。カスタムサービスを SIG 標準サービスと同じサービスとして取り扱う場合は[UUID]を 16bit に設定したうえで変更してください。
- サービスの[略称]を他のサービスと競合しないように変更します。サービスの[略称]は変数名、関数名、ファイル名に使用されるため、これらの競合を防ぐためです。同様にキャラクタリスティックやディスクリプタの[略称]も他のキャラクタリスティックやディスクリプタと競合しないように設定します。

以上で QE for BLE での設定は終了です。図 4.14 に QE for BLE 上での複数の SIG 標準サービスを設定するための方法を示します。

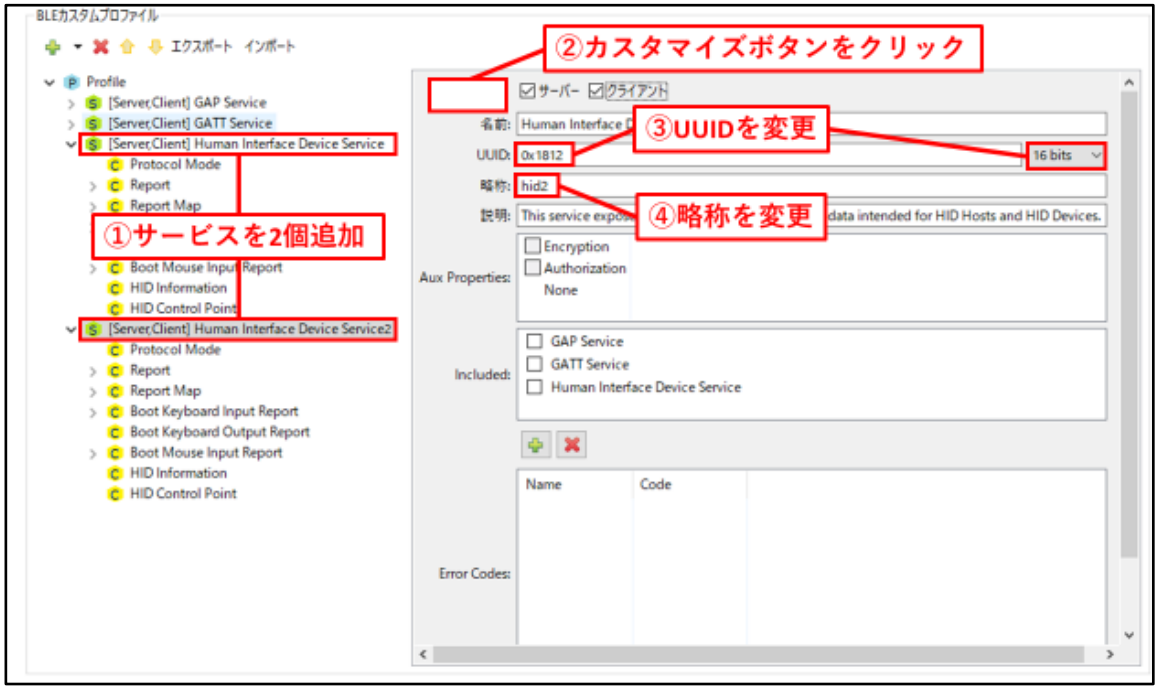

図 4.14 複数サービスの設定(QE for BLE のサービス設定画面)

カスタムサービスから生成される API プログラムはスケルトンプログラムであるため、アプリケーションとして利用するためには処理の実体を実装する必要があります。そこで同じ機能を持ち、定義された仕様に従って実装されている SIG 標準サービスの API プログラムを参考にしながら処理を追加します。追加しなければならない部分はサービスごとに違いますが、多くの場合、以下の処理を追加します。

- encode/decode 関数の中身を実装します。キャラクタリスティックやディスクリプタの構造は変わらないため、多くの部分を移植することができますが、変数名の違いや関数名の違いに注意してください。
- サービス内のコールバック関数を実装します。これが使用されるのは不正な値に対するエラー処理かキャラクタリスティックやディスクリプタに書き込まれた値に対して自動的にデータを返却する場合です。それぞれのサービスの機能に合わせて実装してください。

プロファイルに含まれるサービスの中で GAP Service 以外に[クライアント]に指定したサービスが一つ以上ある場合には、app_main.c には discovery 動作を行うためにディスカバリライブラリを使用したプログラムが実装されています。中でも配列 gs_disc_entries[ ]にはプロファイルに含まれるサービスの UUID と discovery 動作用の関数が定義されています。ここで、同じサービス UUID を持つサービスを実装する場合、要素 idx を追加して同じサービスごとのインデックス番号を追加してください。図 4.15 に HIDS を 2 つ実装した場合のプログラムの実装例を示します。

```
/* Human Interface Device Service UUID */
static uint8_t HIDC_UUID[] = { 0x12, 0x18 }; //HIDS specific service UUID
/* Human Interface Device Service2 UUID */
static uint8_t HID2C_UUID[] = { 0x12, 0x18 }; //Same service UUID

/* Service discovery parameters */
static st_ble_disc_entry_t gs_disc_entries[] = {
    {
            .p_uuid      = HIDC_UUID,
            .uuid_type   = BLE_GATT_16_BIT_UUID_FORMAT,
            .serv_cb     = R_BLE_HIDC_ServDiscCb,
     /* Add member [idx] */
.idx        = 0, /* Set index number if service UUID is same */
    },
    {
     .p_uuid      = HID2C_UUID,
            .uuid_type   = BLE_GATT_16_BIT_UUID_FORMAT,
     .serv_cb     = R_BLE_HID2C_ServDiscCb,
/* Add member [idx] */
.idx        = 1, /* Set index number if service UUID is same */
    },
};
```

図 4.15 HIDS を 2 つ実装した場合のプログラムの実装例

4.4.3　セカンダリサービスを実装する場合

QE for BLE ではすべてのサービスをプライマリサービスとして扱います。そのため、セカンダリサービスを使用する場合には生成されたプログラムを変更する必要があります。変更の方法については、サーバ側とクライアント側で異なります。

サーバ側の実装

プロファイルに含まれるサービスの中で、[サーバー]にチェックが入っているサービスはその情報が GATT データベースに追加された状態で QE for BLE から生成されます。QE for BLE ではすべてのサービスをプライマリサービスとして扱うため、GATT データベースにはプライマリサービスとして生成されます。そこで生成された GATT データベースのプログラムを変更することでセカンダリサービスを実装します。

ファイル gatt_db.c に定義されている配列 gs_gatt_type_table[ ]を変更します。この配列では以下の 2 点について変更してください。

- セカンダリサービスの定義を追加します。配列のほかの要素を参考に[UUID Offset]が 2 で、セカンダリサービスのアトリビュートハンドルを正しく示すようにします。
- [Primary Service Declaration]を変更します。正しいアトリビュートハンドルを指定するように変更してください。

図 4.16 に配列 gs_gatt_type_table[ ]の変更例を示します。

```
static const st_ble_gatts_db_uuid_cfg_t gs_gatt_type_table[] =
{
    /* 0 : Primary Service Declaration */
    {
        /* UUID Offset */
        0,
        /* First Occurrence for type */
      /* Change this value to proper handle */
        0x000C,
        /* Last Occurrence for type */
      /* Change this value to proper handle */
        0x0026,
    },

  /* Add from here */
    /* 2 : Secondary Service Declaration */
    {
        /* UUID Offset */
      /* set 2 for this value */
        2,
        /* First Occurrence for type */
      /* Change this value to proper handle */
        0x0010,
        /* Last Occurrence for type */
      /* Change this value to proper handle */
        0x0000,
    },
  /* Add until here */

}
```

図 4.16 セカンダリサービスの GATT データベースへの実装例(1)

また合わせて配列 gs_gatt_db_attr_table[ ]を変更します。GATT データベースの構造が定義されています。この配列では以下の 2 点について変更してください。

- セカンダリサービスに変更するサービスの[Primary Service Declaration]と示されている部分の[UUID_Offset]を変更します。ここではサービスのアトリビュートタイプを指定しており、0 を入力するとプライマリサービス、2 と入力するとセカンダリサービスとして定義することができます。そのため、[UUID_Offset]を 2 に変更してください。
- [Next Attribute Type Index]が正しいハンドル値を示すように変更します。[Next Attribute Type Index]には同じアトリビュートタイプを持つ次のデータのハンドル値を入力します。そのデータが同じアトリビュートタイプを持つデータの中で最後のデータの場合、0x0000 を入力します。そのため変更したサービスの[Primary Service Declaration]とその前の[Primary Service Declaration]を変更してください。

図 4.17 に gs_gatt_db_attr_table[ ]の変更例を示します。

【注】セカンダリサービスに変更したサービスは必ずいずれかのサービスのインクルードサービスになるように設定してください。

```
  static const st_ble_gatts_db_attr_cfg_t gs_gatt_db_attr_table[] =
  {
    /* Handle: 0x000C */
      /* GATT Service: Primary Service Declaration */
      {
          /* Properties */
          BLE_GATT_DB_READ,
          /* Auxiliary Properties */
          BLE_GATT_DB_FIXED_LENGTH_PROPERTY,
          /* Value Size */
          2,

          /* Next Attribute Type Index */
          /* change this value to handle of next primary service declaration */
          0x0026,    /* 0x0010 → 0x0026 */

          /* UUID Offset */
          0,
          /* Value */
          (uint8_t *)(gs_gatt_const_uuid_arr + 20),
      },



    /* Example: Secondary Service Declaration */
  /* Handle: 0x0010 */
  /* Human Interface Device Service: Primary Service Declaration */
      {
  /* Properties */
  BLE_GATT_DB_READ,

  /* Auxiliary Properties */
  BLE_GATT_DB_FIXED_LENGTH_PROPERTY,

       /* Value Size */
         2,

       /* Next Attribute Type Index */
  /* Change this value to proper handle */
       /* Last secondary service declared: 0x0000 */
       /* Not last secondary service declared: handle of next secondary service
declaration */
         0x0000,     /* 0x0026 → 0x0000 */

       /* UUID Offset */
       /* Change this value to proper Attribute type  */
       /* Primary service declaration: 0 */
       /* Secondary service declaration: 2 */
         2,   /* 0 → 2 */

         /* Value */
         (uint8_t *)(gs_gatt_const_uuid_arr + 26),
      },

      /* Handle: 0x0026 */
      /* Human Interface Device Service2: Primary Service Declaration */

  }
```

図 4.17 セカンダリサービスの GATT データベースへの実装例(2)

クライアント側の実装

プロファイルに含まれるサービスの中で GAP Service 以外に[クライアント]に指定したサービスが一つ以上ある場合には、[クライアント]に指定したサービスについて discovery 動作を行うためのコードが QE for BLE から生成されます。生成されたコードは、BLE Protocol Stack が提供するディスカバリライブラリを使用しており、プライマリサービスのみ discovery 動作を行います。セカンダリサービスはいずれかのサービスにインクルードされているため、インクルードサービスとして discovery 動作を行います。「4.4.4 インクルードサービスに対する discovery 動作の実装」をご参照ください。デバッグ等で、セカンダリサービスに対して discovery 動作を行う必要がある場合には、BLE Protocol Stack の GATT Client API にある R_BLE_GATTC_DiscAllSecondServ ()を使用してください。GATT Client API の詳細については BLE FIT モジュールに付属する「R_BLE API ドキュメント(r_ble_api_spec.chm)」を確認してください。

4.4.4　インクルードサービスに対する discovery 動作の実装

サービスのインクルードサービスの指定

プロファイルに含まれるサービスの中で GAP Service 以外に[クライアント]に指定したサービスが一つ以上あると、[クライアント]に指定したサービスに対して discovery 動作を行うためのコードが QE for BLE から生成されます。生成されたコードは、BLE Protocol Stack が提供するディスカバリライブラリを使用しており、プライマリサービスに対してのみ discovery 動作を行います。サービスが特定のサービスをインクルードサービスとして持つ場合は、その構造を考慮して discovery 動作する必要があり、ディスカバリライブラリは、この構造を考慮した discovery 動作の機能を提供します。app_main.c 内の gs_disc_entries 変数にインクルードサービスの UUID とディスカバリコールバック関数を指定すると、サービスのインクルードサービス宣言にあるハンドル範囲内に対して discovery 動作を行います。app_main.c の gs_disc_entries 変数を図 4.18 から図 4.19 のように変更します。

```
/*PRIMARY service entry */
static st_ble_disc_entry_t gs_disc_entries[] =
{
    {
        /*Weight Scale service disc entry */
        .p_uuid = (uint8_t *)BLE_WSC_UUID,
        .uuid_type = BLE_GATT_16_BIT_UUID_FORMAT,
        .serv_cb = R_BLE_WSC_ServDiscCb,
    },
    {   /*Body Composition service disc entry */
        .p_uuid = (uint8_t *)BLE_BCC_UUID,
        .uuid_type = BLE_GATT_16_BIT_UUID_FORMAT,
        .serv_cb = R_BLE_BCC_ServDiscCb,
    },
};
```

図 4.18 QE for BLE によって生成されるコード

```
/*Add INCLUDE service entry*/
static st_ble_disc_entry_t gs_disc_wsc_inc_entries[] =
{
/*Body Composition service disc entry AS A INCLUDE SERVICE IN WSS*/
{
            .p_uuid = (uint8_t *)BLE_BCC_UUID,
            .uuid_type = BLE_GATT_16_BIT_UUID_FORMAT,
       .serv_cb = R_BLE_BCC_ServDiscCb,
            .num_of_inc_servs = 0,
    },
};

/*PRIMARY service entry */
static st_ble_disc_entry_t gs_disc_entries[] =
{
/*Weight Scale service disc entry as a primary service*/
    {
            .p_uuid = (uint8_t *)BLE_WSC_UUID,
            .uuid_type = BLE_GATT_16_BIT_UUID_FORMAT,
            .serv_cb = R_BLE_WSC_ServDiscCb,

            /* Register include service entry*/
            .inc_servs = gs_disc_wsc_inc_entries,
            .num_of_inc_servs = 1

    },
};
```

図 4.19 インクルードサービスに対応したコード

インクルードサービスのアトリビュートハンドルの取得

ディスカバリライブラリを使用して discovery 動作を行いインクルードサービスがディスカバリされた時、インクルードサービスのアトリビュートハンドルはインクルードサービスではなくインクルードしている親となるサービスに通知されます。そのため、サービス XXX のインクルードサービスとしてディスカバリされたサービス YYY のアトリビュートハンドルを、API プログラムが提供する R_BLE_YYY_GetServAttrHdl( )によって取得できません。インクルードサービス YYY のアトリビュートハンドル範囲を取得する必要がある場合は、サービス XXX のプログラム API(r_ble_xxx.c)を編集し、インクルードサービスがディスカバリされた場合の通知をインクルードサービス YYY に届けます。図 4.20 に、サービス XXX が 16bitUUID のサービス YYY をインクルードサービスとして持つ場合の例を示します。128bit UUID の場合と 16bit UUID の場合とでは UUID を取り扱うデータ型が違うので注意してください。

```
  #include <string.h>
  #include "r_ble_XXX.h"
  #include "profile_cmn/r_ble_servc_if.h"

  /* ADD : including discovery library and include service yyy */
  #include "discovery/r_ble_disc.h"
  #include "r_ble_yyy.h"

  void R_BLE_XXX_ServDiscCb(uint16_t conn_hdl, uint8_t serv_idx, uint16_t type, void
*p_param)
  {
  /* ADD : */
       uint16_t YYY_UUID = 0x0000;
       if (type == BLE_DISC_INC_SERV_FOUND)
       {
              st_disc_inc_serv_param_t * evt_param =
                      (st_disc_inc_serv_param_t *)p_param;

              if (evt_param->uuid_type == BLE_GATT_16_BIT_UUID_FORMAT)
              {
                     if(YYY_UUID == evt_param->value.inc_serv_16.service.uuid_16)
                {
                           st_disc_serv_param_t serv_param = {
                                  .uuid_type            = BLE_GATT_16_BIT_UUID_FORMAT,
                                  .value.serv_16.range   =
                                         evt_param->value.inc_serv_16.service.range,
                                  .value.serv_16.uuid_16 =
                                 evt_param->value.inc_serv_16.service.uuid_16,
                  };
                     R_BLE_YYY_ServDiscCb(
                            /* Connection handle */
                             conn_hdl,
                             /* idx */
                             0,
                             /* Notify as a primary service */
                             BLE_DISC_PRIM_SERV_FOUND,
                             /* Service handle information */
                             &serv_param);
                }
              }
       }
  /* Generated code */
  }
```

図 4.20 インクルードサービスのディスカバリの実装

4.4.5　コネクションアップデートの方法

Bluetooth LE 通信ではコネクションアップデートを行うことで通信中に通信頻度を変更することができます。

関数「R_BLE_GAP_UpdConn」を使用することでコネクションアップデートを行うことができます。通信の頻度を変更する場合は以下のパラメータを変更してください。

- コネクションインターバル
  - 通信の頻度を設定します。最小値と最大値を設定することができます。(設定した値)×1.25ms で計算されます。
  - 変数：conn_intv_min、conn_intv_max
- スレイブレイテンシ
  - 設定された値の数だけ通信を無視します。5 と設定した場合、最初の受信のあと 6 回目の受信までの通信は無視されます。
  - 変数：conn_latency
- スーパービジョンタイムアウト
  - 設定した時間を過ぎた場合、コネクションが切断されます。通信頻度を下げる場合、合わせて変更してください。(設定した値)×10ms で計算されます。
  - 変数：sup_to

図 4.21 に disc_comp_cb 関数にコネクションアップデートを実装した例を示します。

```
  static void disc_comp_cb(uint16_t conn_hdl)
  {
  /* Hint: Input process such as GATT operation */
  /* Start user code for Discovery Complete callback function. Do not edit comment
generated here */
       st_ble_gap_conn_param_t conn_param = {
                     .conn_intv_min      = 0x0100,
                     .conn_intv_max      = 0x0100,
                     .conn_latency       = 0x0010,
                     .sup_to             = 0x0200,
                     .min_ce_length      = 0xFFFF,
                     .max_ce_length      = 0xFFFF,
       };
       R_BLE_GAP_UpdConn(conn_hdl, BLE_GAP_CONN_UPD_MODE_REQ, 0x00, conn_param);
  /* End user code. Do not edit comment generated here */
      return;
  }
```

図 4.21 コネクションアップデート関数の利用例

5.　作成したプロファイルのビルド及び実行

5.1　新規プロジェクトから作成した場合

新規プロジェクトを作成した場合、QE for BLE から生成されるプログラムはプロジェクト内の適切な位置に生成されます。コードの生成を行ったプロジェクトを使用して、ファイルやディレクトリの場所を変更することなくビルド及び実行することができます。

5.2　サンプルプロジェクトをインポートして使用する場合

BLE FIT モジュールには Bluetooth LE 通信機能を使用したサンプルプロジェクトが FITDemos という名前で同梱されています。ユーザはサンプルプロジェクトをインポートして、それをベースに開発を行うこともできます。

5.2.1　バージョン 2.31 以降の BLE FIT モジュールをベースに開発する場合

サンプルプロジェクトは QE for BLE を使用する構造で作成されています。そのため、プロジェクト内のファイルやフォルダを移動することなくビルド及び実行することができます。

5.2.2　バージョン 2.31 以前の BLE FIT　モジュールをベースに開発する場合

QE for BLE のコード生成先フォルダの変更に伴ってファイルの競合が発生します。以下のフォルダの削除をお願いいたします。

- src/smc_gen/Config_BLE_PROFILE

5.2.3　バージョン 1.10 以前の BLE FIT モジュールをベースに開発する場合

QE for BLE をサンプルプロジェクト内で使用した場合、サンプルプロジェクト内のファイルを置き換える形でプログラムを生成します。そのため、サンプルプロジェクトとは別に QE for BLE からプログラムを生成するためのプロジェクトを作成してください。次に、QE for BLE から生成したプログラムをサンプルプロジェクトにコピーします。

次に示すようにプロジェクトを変更してください。

- QE for BLE から生成された app_main.c → サンプルプロジェクトの app_main.c と入れ替えてください。
- QE for BLE から生成された gatt_db.c、gatt_db.h → サンプルプロジェクトの gatt_db.c、gatt_db.h と入れ替えてください。
- QE for BLE から生成された r_ble_[service] → すべてサンプルプロジェクトの src フォルダにコピーしてください。
- サンプルプロジェクトの services フォルダ → 中のファイルごと削除してください。

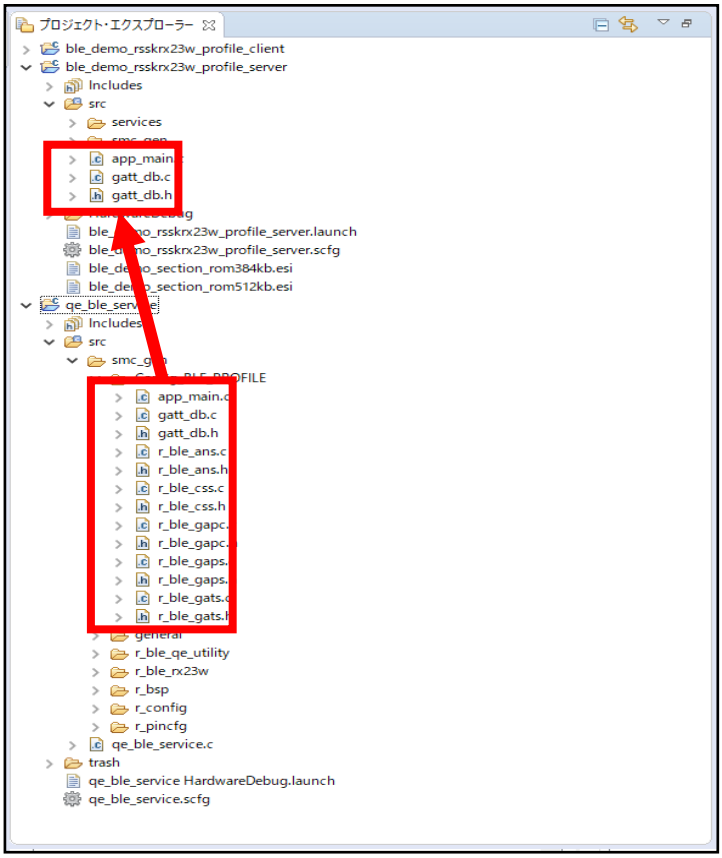

図 5.1 QE for BLE から生成したプログラムをサンプルプロジェクトに移動

サンプルプロジェクトで使用するために app_main.c を図 5.2 のように編集してください。

- app_main( )の名前を main( )に変更してください。

```
//void app_main(void)
void main(void)
{
    R_BLE_Open();
    ble_app_init();

    while (1)
    {
        R_BLE_Execute();
    }

    R_BLE_Close();
}
```

図 5.2 app_main の変更

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2019.11.27	—	新規作成
1.10	2021.3.12	1	関連ドキュメントに以下のドキュメントを追加 ● Bluetooth Low Energy プロトコルスタック基本パッケージ ユーザーズマニュアル (R01UW0205) ● RX23W グループ Bluetooth Low Energy アプリケーション開発者ガイド (R01AN5504)
		60	BLE FIT モジュールのバージョン 1.20 以降のサンプルプロジェクトを使用する場合の説明を追加
		33 22、25	QE Utility V1.10 で追加された機能に関する説明を追加 ● ユーザが実装したコードを保護するためのコードブロック機能の説明を追加 ● キャラクタリスティックやディスクリプタの Aux Properties の設定にて全ての設定が反映されるようになったため、「非対応」の文言を削除
		7	「2.1 ソフトウェア条件」を追加
		10	「2.2 BLE QE Utility モジュール」内に対応するプロファイルの一覧表を追加
		12	「3.1.2 FIT モジュールのダウンロード」を追加 またそれに合わせて前後の文章を変更
		15	「3.2.1 要素の追加」を追加
		35	「4.1 サービスのプログラム」の追加に合わせて「4.1.1 サービスで定義される API」、「4.1.2 サービスのイベントについて」を追加
		58	「4.4.5 GAT サービス、GAP サービスの利用方法」を追加
		59	「4.4.6 コネクションアップデートの方法」を追加
1.20	2021.8.18	 26 29 12	QE for BLE[RX] V1.10 の機能に関する説明を追加 ● 「3.3 ペリフェラルの設計」を追加 ● 「3.4 セントラルの設計」を追加 ● 上記の追加に合わせて「3.1.6 EQ for BLE の利用」の説明を変更
1.30	2022.4.13	6	QE for BLE v1.40 に合わせて内容を修正
		7	2 章を QE for BLE v1.40 環境の手順に変更
		12	3.1 章を QE for BLE v1.40 環境の手順に変更
		14	プロファイルの保存先をプロジェクトフォルダに変更
		60	BLE FIT 2.30 以前のプロジェクトを使用する場合を追記。
		-	QE for BLE 1.40 に組み込まれたため、 「GAT サービス、GAP サービスの利用方法」を削除。

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。