

# Bluetooth LE マイコン/モジュール

## Bluetooth Low Energy プロファイル開発者ガイド

### 要旨

本ドキュメントでは、下記デバイスの Bluetooth LE プロファイルの開発者向けに、Bluetooth LE 開発ツールである QE for BLE を使用したプロファイルの開発方法をガイドします。

本書は以下のドキュメントを統合して再編集したものです。

RA4W1 Group Bluetooth Low Energy Profile Developer's Guide (R01AN5428)

RE01B Group Bluetooth Low Energy プロファイル開発者ガイド (R01AN5638)

RX23W Group Bluetooth Low Energy プロファイル開発者ガイド (R01AN4553)

### 対象デバイス

- RX23W グループ
- RA4W1 グループ
- RE01B 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>】
- Bluetooth® Security and Privacy Best Practices Guide 【<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)
- 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)
  
- RA4W1 Group User's Manual: Hardware (R01UH0883)
- RA Flexible Software Package Documentation
- RA4W1 Group BLE Sample Application (R01AN5402)
- RA4W1 Group Bluetooth Low Energy Application Developer's Guide (R01AN5653)
  
- RE01B グループ ユーザーズマニュアル ハードウェア編 (R01UH0903)
- RE01B グループ CMSIS パッケージを用いた開発スタートアップガイド (R01AN5310)
- RE01B グループ Bluetooth Low Energy サンプルコード(Using CMSIS Driver Package) (R01AN5606)
- RE01B グループ Bluetooth Low Energy アプリケーション開発者ガイド (R01AN5643)
  
- e<sup>2</sup> studio ユーザーズマニュアル 入門ガイド (R20UT4204)
- QE for BLE[RA,RE,RX] V1.5.0 リリースノート(R20UT5145)

Bluetooth® ワードマークおよびロゴは登録商標であり、Bluetooth SIG, Inc. が所有権を有します。ルネサス エレクトロニクス株式会社は使用許諾の下でこれらのマークおよびロゴを使用しています。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。

1. 概要	4
1.1 Bluetooth LE のデータ通信の概要	4
1.2 Bluetooth LE 通信プログラムの開発環境	5
1.2.1 e <sup>2</sup> studio	5
1.2.2 QE for BLE	6
1.2.3 Bluetooth LE 通信プロジェクト	8
1.3 プロファイルプログラムのソフトウェア構成	9
2. 開発環境の構築	11
2.1 QE for BLE のインストール	11
2.1.1 インストール済みの e <sup>2</sup> studio への QE for BLE の追加方法	11
2.1.2 e <sup>2</sup> studio のインストール時に QE for BLE を追加する方法	12
2.2 Bluetooth LE 通信プロジェクトの入手	13
2.2.1 RX23W	13
2.2.2 RA4W1	13
2.2.3 RE01B	13
3. QE for BLE によるプロファイル開発	14
3.1 QE for BLE の使用方法	14
3.2 プロファイルの設計	16
3.2.1 アプリケーションのロール設定	17
3.2.2 サービスの追加と設定	18
3.2.3 キャラクタースティックの追加と設定	23
3.2.4 ディスクリプタの追加と設定	26
3.3 ペリフェラルの設計	28
3.3.1 Advertising Data の設定	28
3.3.2 Scan Response Data の設定	30
3.3.3 Advertising Parameter の設定	30
3.4 セントラルの設計	31
3.4.1 Scan Parameter の設定	31
3.4.2 Scan Filter Data の設定	32
3.4.3 Connection Parameter の設定	33
4. プログラムの実装	34
4.1 サービス API プログラム (r_ble_xxs.c / r_ble_xxc.c)	37
4.1.1 encode/decode 関数の説明	40
4.1.2 エンコードデコード関数の自動生成機能	42
4.1.3 エンコードデコード関数の実装	45
4.2 アプリケーションフレームワーク (app_main.c)	47
4.2.1 セキュリティ要件への対応	49
4.2.1.1 Security Level 3 に設定した場合	49
4.2.1.2 Security Level 4 に設定した場合	49
4.2.2 MTU の変更	50
4.2.2.1 クライアントの実装	51
4.2.2.2 サーバーの実装	51
4.2.3 Write 動作の実装	52
4.2.3.1 クライアントの実装	54

4.2.3.2	サーバーの実装 .....	56
4.2.4	Write Without Response 動作の実装 .....	58
4.2.4.1	クライアントの実装 .....	59
4.2.4.2	サーバーの実装 .....	60
4.2.5	Read 動作の実装 .....	61
4.2.5.1	クライアントの実装 .....	63
4.2.5.2	サーバーの実装 .....	65
4.2.6	Notify 動作の実装 .....	66
4.2.6.1	クライアントの実装 .....	67
4.2.6.2	サーバーの実装 .....	68
4.2.7	Indicate 動作の実装 .....	69
4.2.7.1	クライアントの実装 .....	70
4.2.7.2	サーバーの実装 .....	71
4.3	GATT データベース (gatt_db.c / gatt_db.h) .....	73
5.	プログラムのビルドと実行 .....	75
5.1	RX23W .....	75
5.1.1	QE for BLE の移行方法 .....	75
5.2	RA4W1 .....	78
5.3	RE01B .....	79
6.	その他の実装例 .....	80
6.1	複数のサービスを実装する場合 .....	80
6.2	同一サービスを実装する場合 .....	80
6.3	セカンダリサービスを実装する場合 .....	82
6.4	インクルードサービスに対する discovery 動作を実装する場合 .....	86
6.5	コネクションアップデートの方法 .....	88
6.6	2つのMCUを接続してデータ通信を行う場合 .....	89
6.7	旧バージョンの認証情報(QDID:134484)を使用する場合 .....	94
6.7.1	QE for BLE の生成コード変更の設定 .....	94
6.7.2	プロファイル共通部の取得 .....	96
	改訂記録 .....	97

## 1. 概要

### 1.1 Bluetooth LE のデータ通信の概要

Bluetooth LE では、主に汎用アトリビュートプロファイル(GATT)に従ってデータ通信します。GATT はクライアント・サーバーアーキテクチャで通信します。

GATT を利用するアプリケーションの通信仕様をプロファイルと呼びます。プロファイルは様々なアプリケーションごとに策定されたプロトコルです。同一のプロファイルを持つデバイス間で通信が可能です。プロファイルは、アプリケーションの機能を表す「サービス」を一つ以上持ちます。サービスは、データ構造を表す「キャラクタリスティック」、キャラクタリスティックのデータに情報を追加する「ディスクリプタ」からなります。

サーバーは、サービスのデータを管理する GATT データベースを持ちます。クライアントとサーバーは GATT データベースを介してアプリケーションデータの送受信を行います。アプリケーションデータは、サービスのキャラクタリスティックとして GATT データベースに保持されます。GATT データベースでは、データの格納場所を示すアトリビュートハンドルを指定してアクセスします。

GATT では、サーバーからクライアントにデータを送信する Notify / Indicate 動作、クライアントがデータベースの読み書きするための Read / Write 動作などが定義されています。

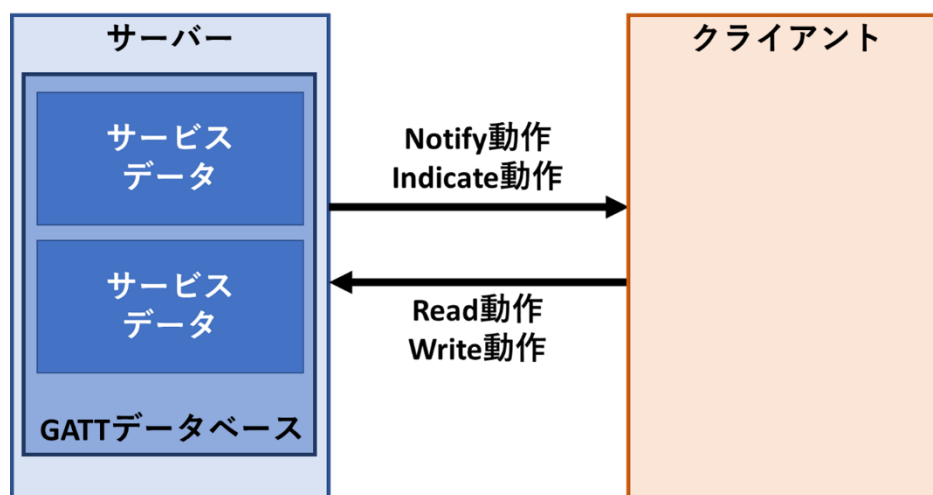


図 1.1 プロファイル通信の概要

プロファイルによる通信はピアツーピアで接続されたデバイス間で確立されます。デバイスの接続は、汎用アクセスプロファイル(GAP)で決められています。Advertising 動作を行うデバイス(ペリフェラル)と Scan 動作やコネクション動作を行うデバイス(セントラル)とで接続を確立します。

## 1.2 Bluetooth LE 通信プログラムの開発環境

ルネサス エレクトロニクスは、Bluetooth LE のアプリケーション開発を支援するための Bluetooth LE プロファイル開発環境を提供します。

プロファイル開発環境を図 1.2 に示します。QE for BLE は、GATT のプロファイル通信を実現するプログラムを生成します。生成されたプログラムは、Bluetooth LE 通信プロジェクト上で動作します。



図 1.2 QE for BLE によるプロファイル開発環境 (RE01B 環境)



図 1.3 QE for BLE によるプロファイル開発環境 (RX23W, RA4W1 環境)

プロファイル開発では、e<sup>2</sup> studio、QE for BLE、Bluetooth LE 通信プロジェクトを使用します。

### 1.2.1 e<sup>2</sup> studio

e<sup>2</sup> studio は、ルネサスマイコン用の統合開発環境です。コード編集機能に加え、様々な拡張を搭載しています。サンプルコードのダウンロードからデバッグまで、すべての開発プロセスを e<sup>2</sup> studio 上で実行できます。

[統合開発環境 e<sup>2</sup> studio | Renesas](#)

## 1.2.2 QE for BLE

QE for BLE は、GUI 上でプロファイルを設計するツールです。設計したプロファイルはソースコードとして生成されます。本製品は、e<sup>2</sup> studio の拡張機能として提供されます。

Bluetooth 仕様では、Bluetooth SIG 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作成できます。本書では、独自に定義したサービスをカスタムサービスと呼びます。

QE for BLE は、表 1.1 に示す SIG 標準サービスをサポートします。これらの多くは、認証を取得しています。また表 1.2 に QE for BLE が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 1.1 QE for BLE 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	1.1.1
Body Composition Service	BCS	1.0	Bond Management Service	BMS	1.0.1
Continuous Glucose Monitoring Service	CGMS	1.0.2	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	1.0.1
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	1.0.1	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	1.0.1	Reconnection Configuration Service	RCS	1.0.1
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	1.1	Weight Scale Service	WSS	1.0
GATT Service	GATS	-	GAP Service	GAPS	-

【注】 Object Transfer Service は認証を取得していません。

表 1.2 QE for BLE がサポートするプロファイル一覧

プロファイル名 [略称]	バージョン	プロファイルを構成するサービス			
Alert Notification Profile [ANP]	1.0	ANS			
Automation IO Profile [AIOP]	1.0	AIOS			
Blood Pressure Profile [BLP]	1.1.1	BLS	DIS		
Continuous Glucose Monitoring Profile [CGMP]	1.0.2	CGMS	DIS	(BMS)	
Cycling Power Profile [CPP]	1.1	CPS	(DIS)	(BAS)	
Cycling Speed and Cadence Profile [CSCP]	1.0	CSCS	(DIS)		
Environmental Sensing Profile [ESP]	1.0	ESS	(DIS)	(BAS)	
Find Me Profile [FMP]	1.0	IAS			
Fitness Machine Profile [FTMP]	1.0	FTMS	(DIS)	(UDS)	
Glucose Profile [GLP]	1.0.1	GLS	DIS		
Health Thermometer Profile [HTP]	1.0	HTS	DIS		
Heart Rate Profile [HRP]	1.0	HRS	DIS		
HID over GATT Profile [HOGP]	1.0	HIDS	DIS	BAS	(ScPS)
Insulin Delivery Profile [IDP]	1.0.1	IDS	DIS	(BAS)	(CTS)
		(BMS)	(IAS)		
Location and Navigation Profile [LNP]	1.0	LNS	(DIS)	(BAS)	
Phone Alert Status Profile [PASP]	1.0	PASS			
Proximity Profile [XP]	1.0.1	IAS	(LLS)	(TPS)	
Pulse Oximeter Profile [PLXP]	1.0.1	PLXS	DIS	(BAS)	(CTS)
		(BMS)			
Reconnection Configuration Profile [RCP]	1.0.1	RCS	(BMS)		
Running Speed and Cadence Profile [RSCP]	1.0	RSCS	(DIS)		
Scan Parameters Profile [ScPP]	1.0	ScPS			
Time Profile [TIP]	1.0	CTS	(NDCS)	(RTUS)	
Weight Scale Profile [WSP]	1.0	WSS	DIS	(BCS)	(BAS)
		(CTS)	(UDS)		

【注】 ()のないサービスは Mandatory のサービス。()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。

### 1.2.3 Bluetooth LE 通信プロジェクト

QE for BLE が生成するソースコードは、以下のプログラム上で動作します。各プログラムの入手方法は、「2.2 Bluetooth LE 通信プロジェクトの入手」をご覧ください。

1. Bluetooth LE Protocol Stack

Bluetooth LE Protocol Stack は、Bluetooth LE 機能を実現するためのプログラムです。Bluetooth LE Protocol Stack は、スタティックライブラリとして提供されます。プロファイル共通部から R\_BLE\_GATTC\_XXX API や R\_BLE\_GATTS\_XXX API が使用されます。

2. プロファイル共通部

プロファイル共通部は、プロファイルによるデータ通信の共通処理の部分をまとめたプログラムです。プロファイル共通部は、Bluetooth LE Protocol Stack 上で動作します。R\_BLE\_DISC\_XXX API、R\_BLE\_SERVC\_XXX API、R\_BLE\_SERVS\_XXX API が提供されます。

3. 抽象 API

4. 抽象 API は、Bluetooth LE 通信のうち接続やセキュリティに関する手続きを簡易に実現するためのプログラムです。Bluetooth LE 通信でよく使う機能をまとめています。R\_BLE\_ABS\_XXX API が提供されます。



### 1.3 プロファイルプログラムのソフトウェア構成

QE for BLE のコード生成機能によって生成されるプロファイルを実現するプログラムのソフトウェア構成を図 1.4 に示します。

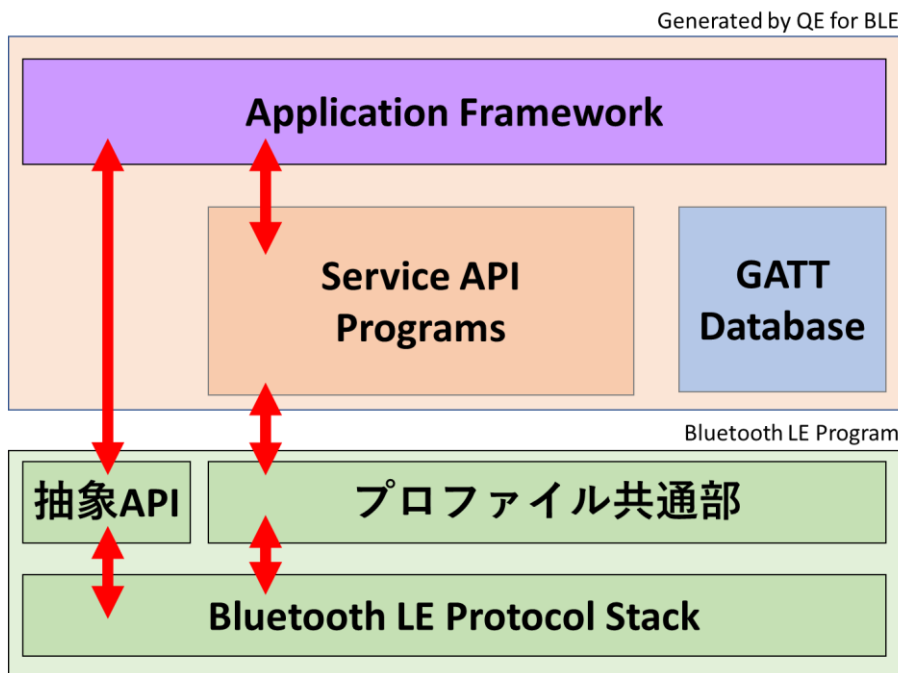


図 1.4 QE for BLE が生成するプログラム(RE01B 環境の場合)

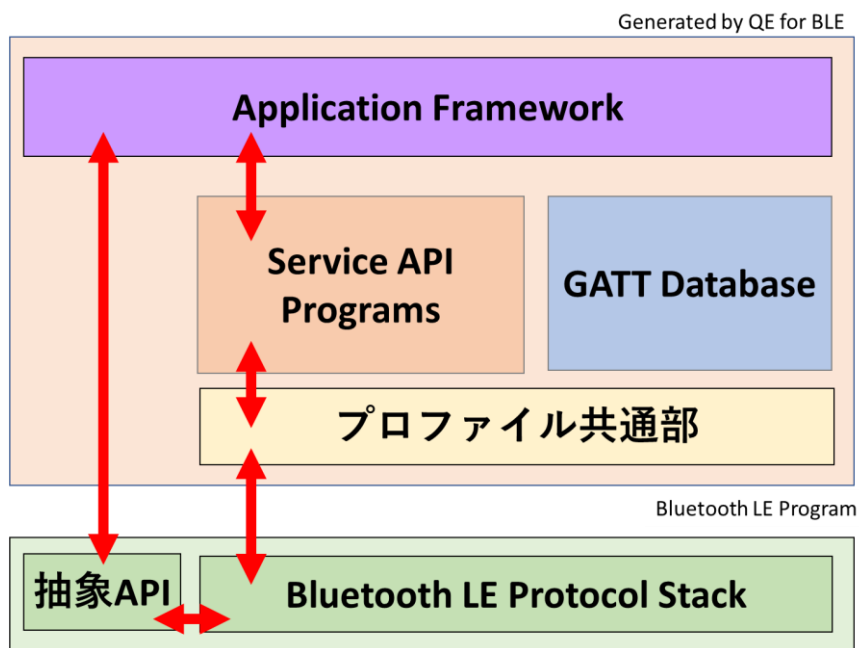


図 1.5 QE for BLE が生成するプログラム(RX23W, RA4W1 環境の場合)

QE for BLE から生成されるプロファイルプログラムは、プロファイル共通部を利用して Bluetooth LE 通信を実現します。プロファイルプログラムは以下の三つのプログラムからなります。プロファイルプログラムの生成方法については「3章 QE for BLE によるプロファイル開発」を参照してください。

1. アプリケーションフレームワーク

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です。ユーザーアプリケーションは本フレームワークをベースにしてサービス API を使用して実装します。詳細は「4.2 アプリケーションフレームワーク(app\_main.c)」を参照してください。

2. サービス API プログラム

GATT データベースに定義されたサービスのデータにアクセスするための API プログラムです。詳細は「4.1 サービス API プログラム (r\_ble\_xxs.c / r\_ble\_xxc.c)」を参照してください。

3. GATT データベースプログラム

サービスのデータ構造を反映した GATT データベースの実装です。詳細は「4.3 GATT データベース (gatt\_db.c / gatt\_db.h)」を参照してください。

## 2. 開発環境の構築

本章では、QE for BLE のインストール方法と、e<sup>2</sup> studio のワークスペースへの Bluetooth LE 通信プロジェクト追加方法を説明します。

### 2.1 QE for BLE のインストール

#### 2.1.1 インストール済みの e<sup>2</sup> studio への QE for BLE の追加方法

QE for BLE は以下のページからダウンロードできます。

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

インストールの方法は以下の通りです：

1. e<sup>2</sup> 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]]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. e<sup>2</sup> studio の再起動を促されるので再起動を行う。

### 2.1.2 e<sup>2</sup> studio のインストール時に QE for BLE を追加する方法

QE for BLE は、e<sup>2</sup> studio インストール時にインストールできます。

e<sup>2</sup> studio インストールウィザードの「追加のソフトウェア」の選択画面で QE for BLE[RA, RE, RX]を チェックします。

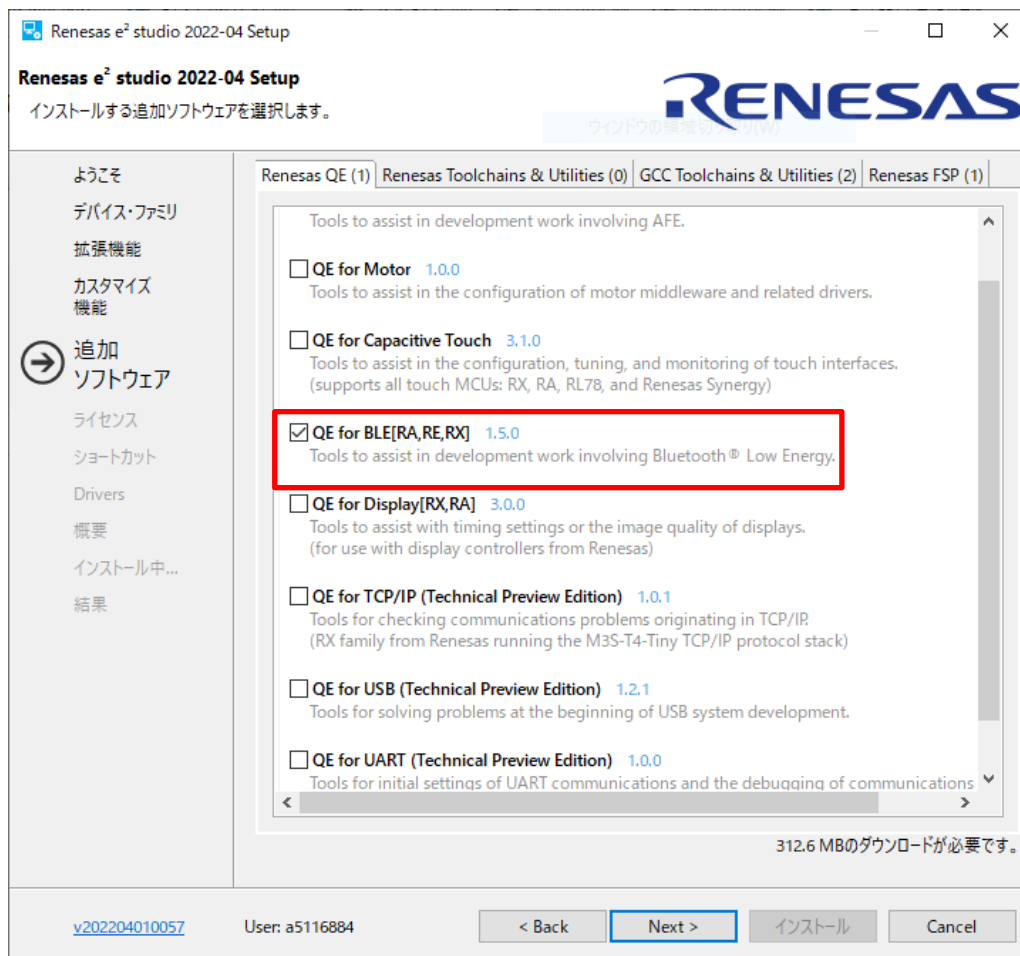


図 2.1 QE for BLE のインストール

## 2.2 Bluetooth LE 通信プロジェクトの入手

### 2.2.1 RX23W

RX23W では、Bluetooth LE Protocol Stack 及び抽象 API は、FIT 形式で提供されています。

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、Bluetooth LE 通信プロジェクトを e<sup>2</sup> studio のワークスペースに追加してください。

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

プロファイル共通部は、QE for BLE 1.60 以上かつ BLE FIT モジュール 2.50 以上を使用する場合、QE for BLE から生成されます。BLE FIT モジュール 2.40 以下は、プロファイル共通部を同梱していません。

### 2.2.2 RA4W1

RA4W1 では、Bluetooth LE Protocol Stack と抽象 API は FSP のモジュールとして提供されています。プロファイル共通部は、QE for BLE から生成されます。

以下のドキュメントの 2 章「How to use demo project」を参考にして、Bluetooth LE 通信プロジェクトを e<sup>2</sup> studio のワークスペースに追加してください。

- RA4W1 Group BLE Sample Application (R01AN5402)

### 2.2.3 RE01B

RE01B では、Bluetooth LE Protocol Stack、抽象 API 及びプロファイル共通部は、アプリケーションノートのサンプルプロジェクトとして提供されます。

以下のドキュメントの 4 章「プロジェクトの作成」を参考にして、Bluetooth LE 通信プロジェクトを e<sup>2</sup> studio のワークスペースに追加してください。

- RE01B グループ Bluetooth Low Energy サンプルコード(Using CMSIS Driver Package) (R01AN5606)

### 3. QE for BLE によるプロファイル開発

本章では、QE for BLE によるプロファイルの設計方法を説明します。QE for BLE では、GATT プロファイルの設計だけでなく、Bluetooth LE の GAP ロールとパラメータを設定できます。

#### 3.1 QE for BLE の使用方法

e<sup>2</sup> 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。

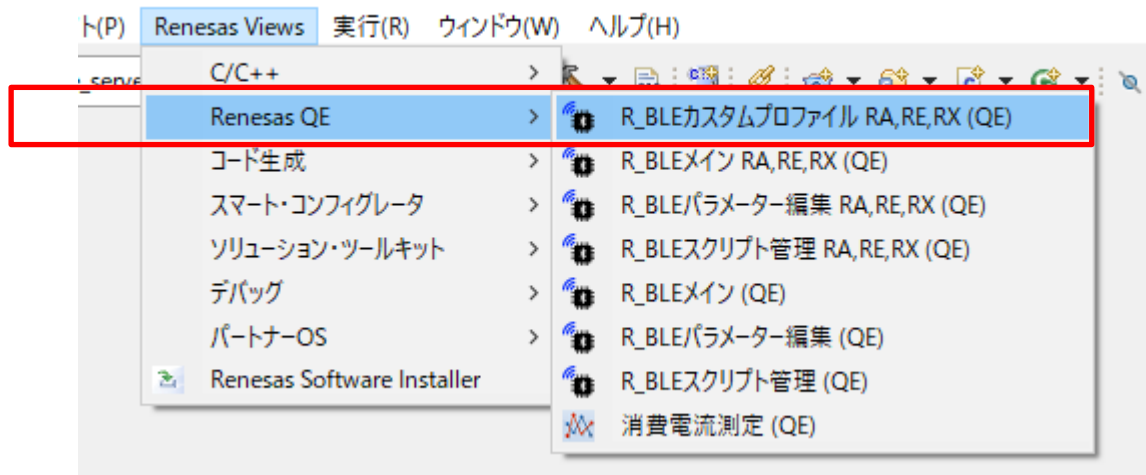


図 3.1 QE for BLE の開き方

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グレータのコンポーネントを削除します。

- [QE for BLE \[RA,RE,RX\] V1.5.0 リリースノート](#)

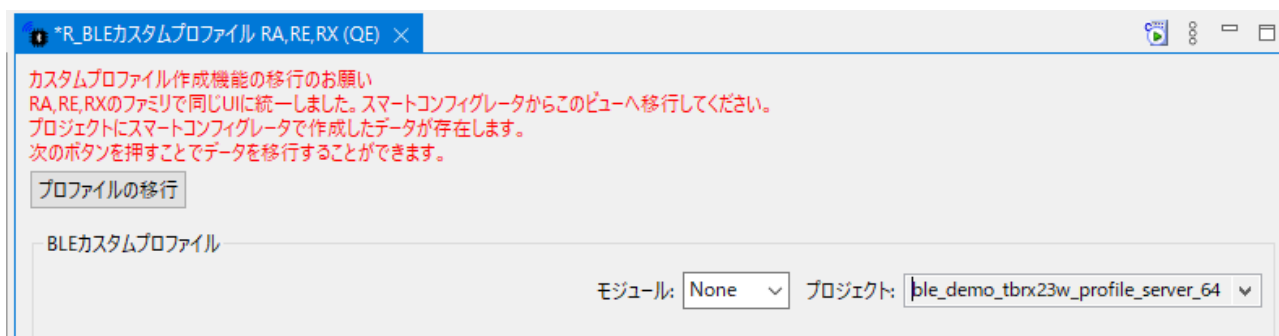


図 3.2 旧バージョンのプロジェクト使用時のプロファイルの移行

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。

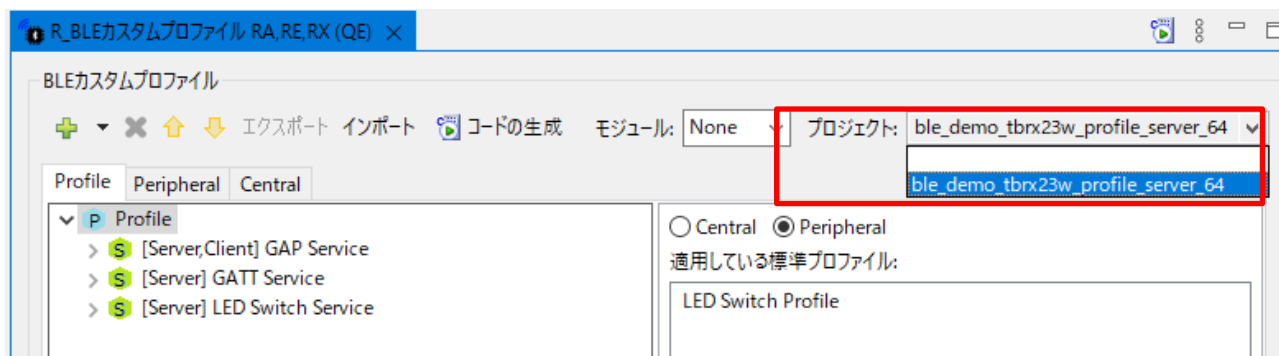


図 3.3 プロジェクトの選択

### 3.2 プロファイルの設計

本節では、アプリケーションが使用する Bluetooth LE プロファイルの設計方法を説明します。





図 3.4 QE for BLE の設定画面



プロファイルで扱うデータは、サービスおよびキャラクターリスティック、ディスクリプタで構成されるツリー構造を持ちます。このプロファイルツリーには設計中のプロファイル全体のデータ構成が表示されます。

プロファイルツリーの要素を選択すると、詳細設定エリアに選択した要素の設定項目が表示されます。詳細設定エリアでは、選択した要素を設計できます。

ツールバーからプロファイルツリーの要素を追加/削除します。ツールバーのアイコンと動作を以下に示します。

「」：選択している要素を追加します。

「」：選択している要素を削除します。

「」 「」：選択している要素を移動します。

エクスポート/インポートでは、プロファイルツリーで選択したサービスを保存/読み込みできます。



### 3.2.1 アプリケーションのロール設定

本節では、アプリケーションが使用する GAP ロールを選択します。プロファイルツリーのプロファイル「P」を選択し詳細設定エリアにプロファイルの設定を表示します。

生成するプログラムの GAP ロールを選択します。ここで選択したロールのスケルトンプログラムが生成されます。「Peripheral」の場合にはアダプタイズを行うプログラムが生成されます。「Central」を選択するとスキャンと接続要求を発行するプログラムが生成されます。



図 3.5 アプリケーションの GAP ロールの選択

詳細設定エリアには、追加されるサービスの組み合わせに応じて表 1.2 に示すプロファイル名が表示されます。



図 3.6 プロファイルの確認画面

## 3.2.2 サービスの追加と設定

プロファイル「P」を選択した状態でツールバーの「+」を押し、プロファイルにサービスを追加します。

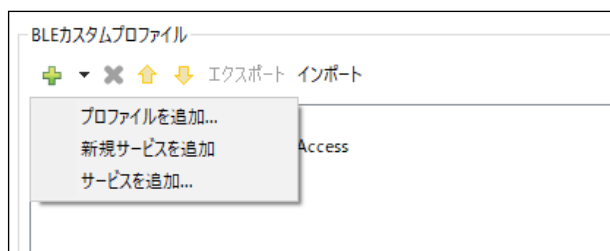


図 3.7 サービスの追加

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」を選択します。

Bluetooth SIG で定義されるプロファイルを作成する場合には、最初に「プロファイルを追加」を選択して Mandatory の SIG 標準サービスを追加してください。Optional の SIG 標準サービスが必要な場合は「サービスを追加」から個別に追加してください。

プロファイルツリーのサービス「S」を選択すると、詳細設定エリアにサービスの設定項目が表示されます。設定項目を図 3.8 に示します。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。削除しないようにしてください。

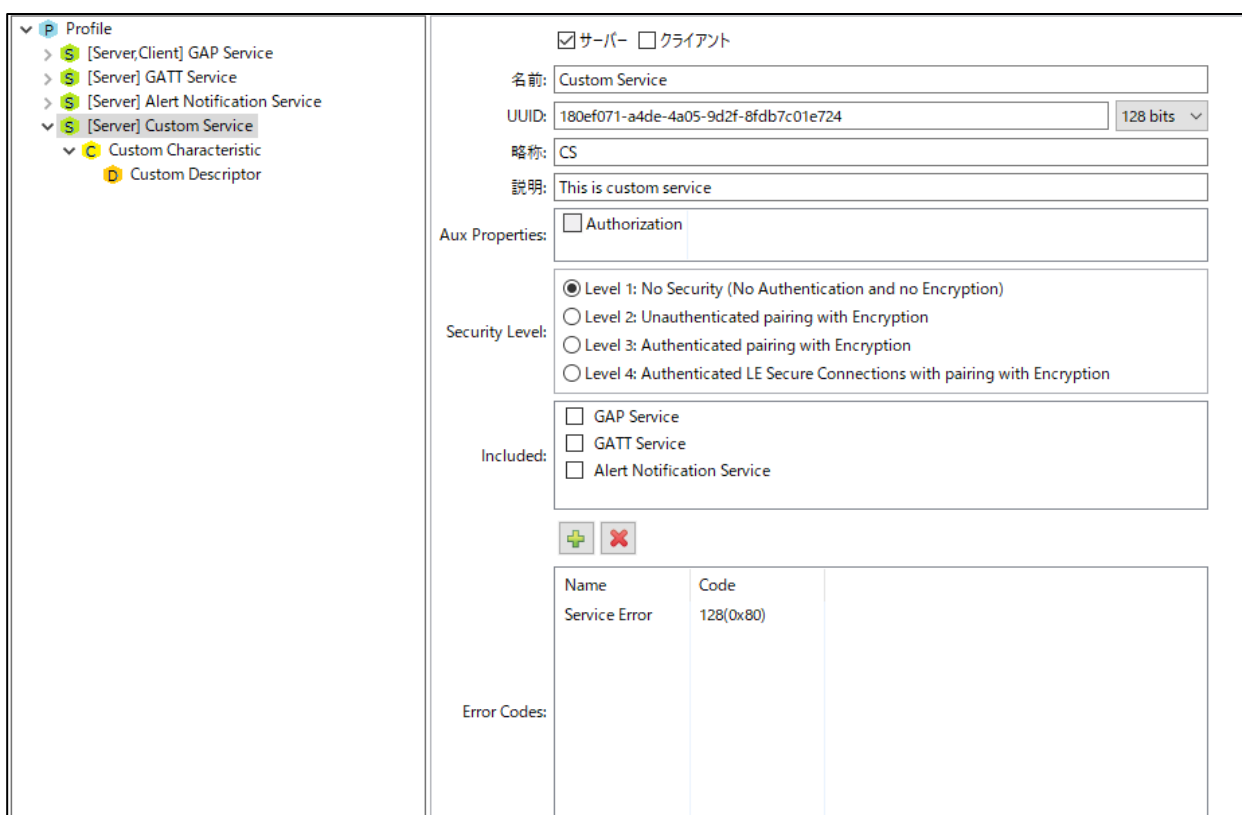


図 3.8 サービスの設定項目

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ーロールのサービス API を生成します。 また、GATT データベースにキャラクターリスティックとディスクリプタが追加されます。	
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ロールのサービス API を生成します。	
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入力されます。これは、任意の値に変更できます。 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ます。 他のサービスと重複しないようにしてください。 Example) cs	
説明 [任意]	サービスの説明を設定します。 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な条件を設定します。 以下の項目を設定できます。	
	Authorization	ユーザの承認が必要となります。「クライアント」を選択した場合の設定は無効です。 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のアクセスに必要なセキュリティのレベルを選択します。「クライアント」を選択した場合の設定は無効です。 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 MITM Protection 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 MITM Protection あり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 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 標準サービスを追加した場合には変更箇所が制限されます。この時のサービスの設定項目を図 3.9 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。

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は、「カスタマイズ」ボタンをクリックします。

Name	Code
Command not ...	160(0xa0)

図 3.9 SIG 標準サービスの設定項目

セキュリティレベルの設定は、GATT データベースのデータを保護するために重要な設定です。

- カスタムサービスを追加する場合

カスタムサービスを追加する場合は、表 3.2 を参考に開発する製品が必要とするセキュリティレベルに一致する「Security Level」を設定してください。

【注】 Bluetooth SIG が公開するガイド(Bluetooth® Security and Privacy Best Practices Guide)では、カスタムプロファイルについては Security Level 2 以上を使用するよう勧告しています。

表 3.2 セキュリティレベル概要

Security Mode 1	暗号化	ペアリング時 ユーザ認証 操作	MITM Protection のための IO Capability or OOB	LE Legacy Pairing による アクセス	LE Secure Connections Pairing による アクセス	特徴
Security Level 1	不要	不要	不要	許可	許可	ペアリングは省略可能。 暗号化なしでサービスにアクセスを許可する設定。
Level 2	必要	不要	不要	許可	許可	ペアリング時にエンドユーザによる認証操作が不要。 IO Capability を実装できない製品向け。
Level 3	必要	必要	必要	許可	許可	ペアリング時にエンドユーザによる認証操作が必要、ユーザが関与しないペアリングを抑制可能。 IO Capability を実装できる製品向け。
Level 4	必要	必要	必要	不許可	許可	ペアリング時にエンドユーザによる認証操作が必要、ユーザが関与しないペアリングを抑制可能。 LE Legacy Pairing を許可しないため、通信盗聴への対策強化となる。 IO Capability を実装でき、より安全なクライアントのみにアクセスを許可したい製品向け。 (LE Secure Connections をサポートしていないデバイスからはアクセスできない)

● SIG 標準サービスを追加する場合

表 3.3 は QE for BLE が提供する SIG 標準プロファイルを使用する場合に追加設定が必要な Security Level の一覧を示します。QE for BLE が提供する SIG 標準サービスでは Security Level が設定されていないため、使用するプロファイルで必要なサービスを追加した後で Security Level を設定する必要があります。

表 1.2 を参考に SIG 標準プロファイルで必要なサービスを追加し、プロファイルで必要なすべてのサービスの Security Level に対して表 3.3 で示すレベルのうち製品が必要とする値を設定してください。例えば、製品が必要とする IDP の Security Level を 3 と判断した場合、IDS 及び付随するサービス (IAS, CTS, BAS, etc) の Security Level には 3 を設定します。Security Level は数値が高いほど安全性が向上します。製品が必要とするセキュリティレベルについては表 3.2 を参考にしてください。

表 3.3 SIG 標準プロファイル使用時に追加設定が必要な Security Level 一覧

プロファイル仕様	Security Level	備考
Alert Notification Profile (ANP) 1.0	2 or 3	
Automation IO Profile (AIOP) 1.0	2 or higher	
Blood Pressure Profile (BLP) 1.1.1	2 or 3	
Continuous Glucose Monitoring Profile (CGMP) 1.0.2	2 or 3	Bond Management Service (BMS) には「Aux Properties」の「Authorization」にチェックが必要
Cycling Power Profile (CPP) 1.1	1, 2 or 3	
Cycling Speed and Cadence Profile (CSCP) 1.0	1, 2 or 3	
Environmental Sensing Profile (ESP) 1.0	2 or higher	Environmental Sensing Service (ESS)には「Aux Properties」の「Authorization」にチェックが必要
Find Me Profile (FMP) 1.0	2 or 3	
Fitness Machine Profile (FTMP) 1.0	2 or higher	
Glucose Profile (GLP) 1.0.1	2 or 3	
Health Thermometer Profile (HTP) 1.0	2 or 3	
Heart Rate Profile (HRP) 1.0	2 or 3	
HID over GATT Profile (HOGP) 1.0	2 or 3	
Insulin Delivery Profile (IDP) 1.0.1	3 or 4	Insulin Delivery Service (IDS) には「Aux Properties」の「Authorization」にチェックが必要 Bond Management Service (BMS) には「Aux Properties」の「Authorization」にチェックが必要
Location and Navigation Profile (LNP) 1.0	1, 2 or 3	
Phone Alert Status Profile (PASP) 1.0	2 or 3	
Proximity Profile (PXP) 1.0.1	2 or 3	
Pulse Oximeter Profile (PLXP) 1.0.1	2 or higher	
Reconnection Configuration Profile (RCP) 1.0.1	1 or higher	Bond Management Service (BMS) には「Aux Properties」の「Authorization」にチェックが必要
Running Speed and Cadence Profile (RSCP) 1.0	1, 2 or 3	
Time Profile (TIP) 1.0	2 or 3	
Weight Scale Profile (WSP) 1.0	2 or higher	

【注】 表 3.3 に記載された Security Level には Bluetooth LE 4.1 以前 (LE Secure Connections 機能未サポート) に策定された要件が含まれています。Bluetooth SIG が公開するガイド (Bluetooth® Security and Privacy Best Practices Guide) では、リモートデバイス側の制約がない限り、Security Level 4 または Security Level 2, 3 と LE Secure Connections ペアリングとの組み合わせのサポートを推奨していません。

## 3.2.3 キャラクターリスティックの追加と設定

サービス「S」を選択した状態でツールバーの「+」を押し、サービスにキャラクターリスティックを追加します。

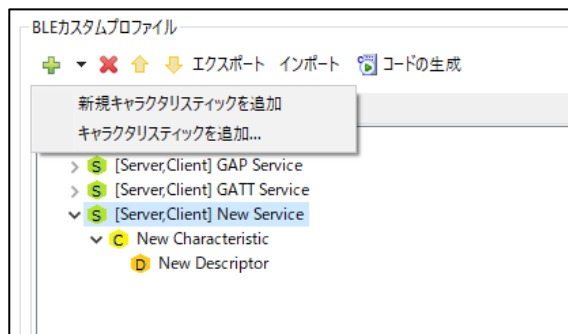


図 3.10 キャラクターリスティックの追加

カスタムキャラクターリスティックを追加する場合は「新規キャラクター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ーリスティックを追加する場合は「キャラクターリスティックを追加」を選択します。

プロファイルツリーのキャラクターリスティック「C」を選択すると、詳細設定エリアにキャラクターリスティックの設定項目が表示されます。設定項目を図 3.11 に示します。各設定項目の説明を表 3.4、表 3.5 に示します。

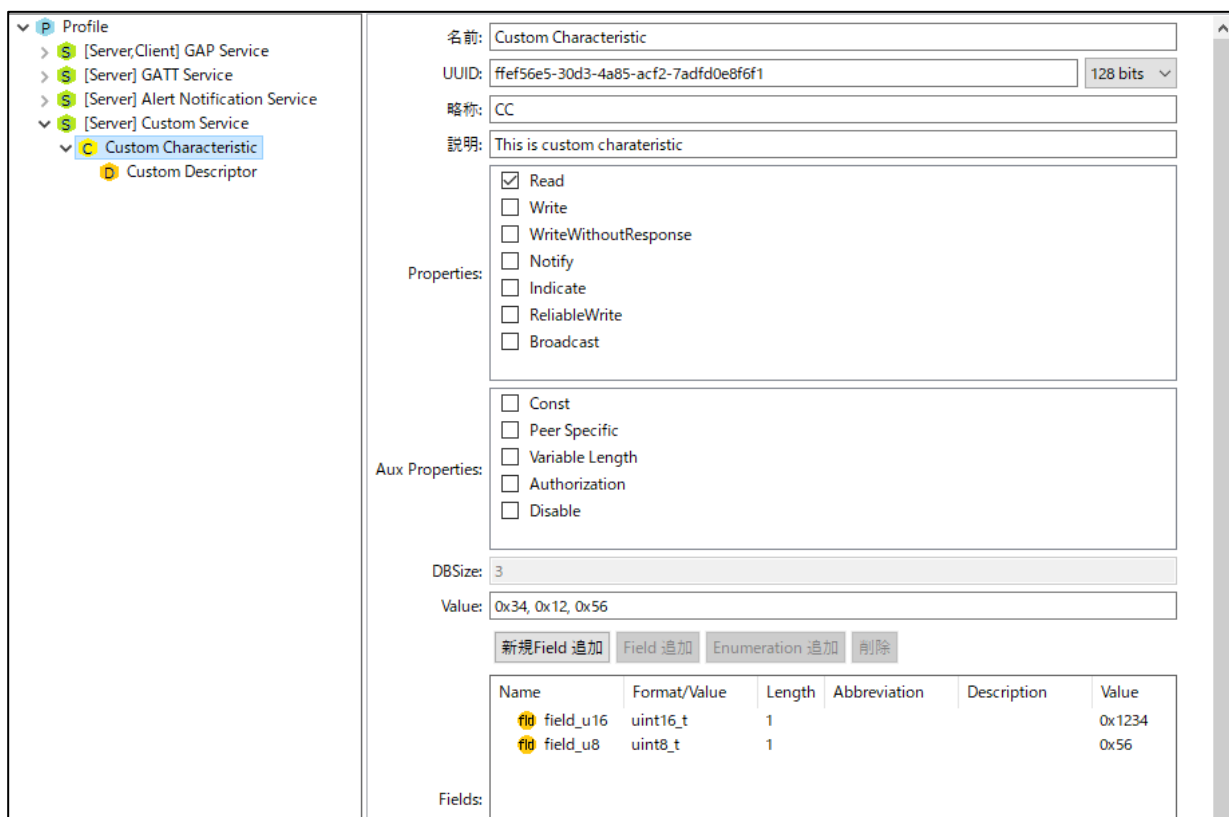


図 3.11 キャラクターリスティックの設定項目

表 3.4 キャラクターリスティックの設定項目の説明

項目名	説明	
名前 [必須]	キャラクターリスティックの名前を設定します。 Example) Custom Characteristic	
UUID [必須]	キャラクターリスティックの UUID を設定します。 カスタムキャラクター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ます。これは、任意の値に変更できます。 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]	キャラクターリスティックの名前の略称を設定します。 キャラクターリスティックの関数やデータの名称に使われます。 他のキャラクターリスティックと重複しないようにしてください。 Example) cc	
説明 [任意]	キャラクター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]	キャラクター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]	キャラクター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、すべての接続相手と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]	キャラクター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]	キャラクター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 : 0x12,0x34,0x56,0x78 文字列の場合 : "example"	
Field [必須]	キャラクター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 3.5 を参照してください。	



表 3.5 Field の設定項目

項目名	説明		
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ーシティック・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。		

## 3.2.4 ディスクリプタの追加と設定

キャラクタースティック「C」を選択した状態でツールバーの「+」を押し、キャラクタースティックにディスクリプタを追加します。

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」を選択します。



図 3.12 ディスクリプタの追加

プロファイルツリーのディスクリプタ「D」を選択すると、詳細設定エリアにディスクリプタの設定項目が表示されます。設定項目を図 3.13 に示します。各設定項目の説明を表 3.6 に示します。

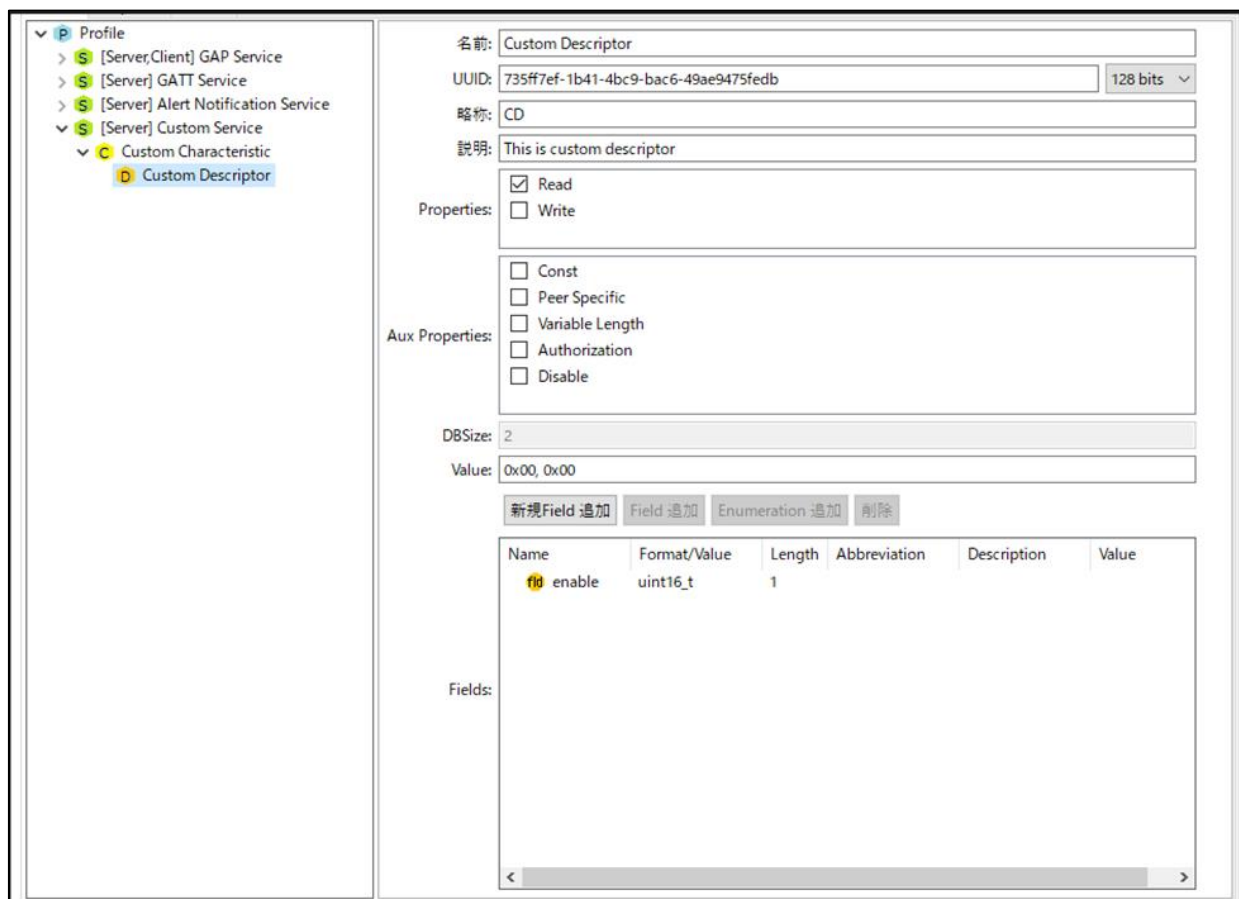


図 3.13 ディスクリプタの設定項目

表 3.6 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ます。これは任意の値に変更できます。 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ます。 他のディスクリプタと重複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、すべての接続相手と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 3.5 を参照してください。

### 3.3 ペリフェラルの設計

本節では、アプリケーションがペリフェラル動作する場合の GAP パラメータ設計方法を説明します。この設定はプロファイルタブの「Peripheral」を選択時に有効になります。

このタブでは以下の項目を設定できます。

表 3.7 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。

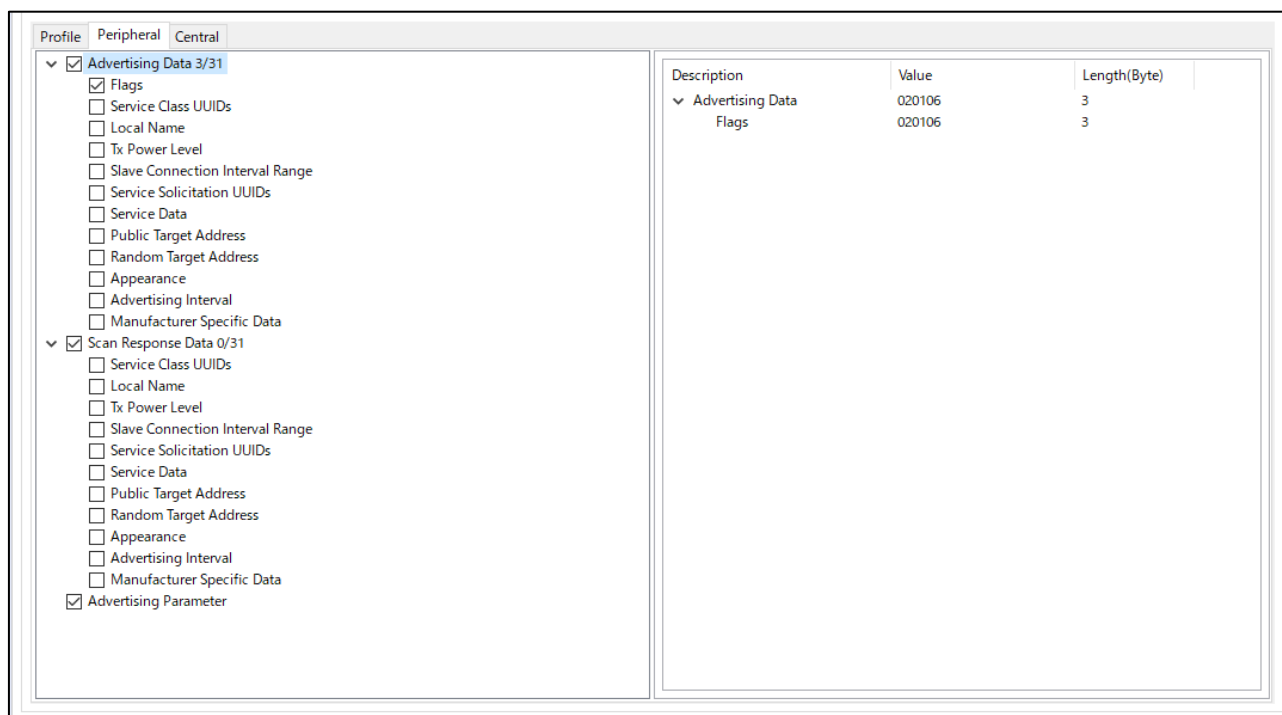


図 3.14 Peripheral タブの設定画面

#### 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 3.8 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 3.8 設定できるデータ型の一覧

データ型名	説明	
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の設定はできません。 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します。チェックしたサービスがリストに追加されます。	
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択できます。	
	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接続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します。チェックしたサービスがリストに追加されます。	
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。 <a href="https://www.bluetooth.com">https://www.bluetooth.com</a>	
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できます。ユーザが追加することが可能なデータ型を表 3.8 に示しています。

### 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できます。設定できるパラメータを表 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、「Advertising Interval」のパラメータを小さくします。

表 3.9 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せん。	
	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

本節では、アプリケーションがセントラル動作する場合の GAP パラメータ設計方法を説明します。この設定はプロファイルタブの「Central」を選択時に有効になります。

このタブでは以下の項目を設定できます。

表 3.10 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。

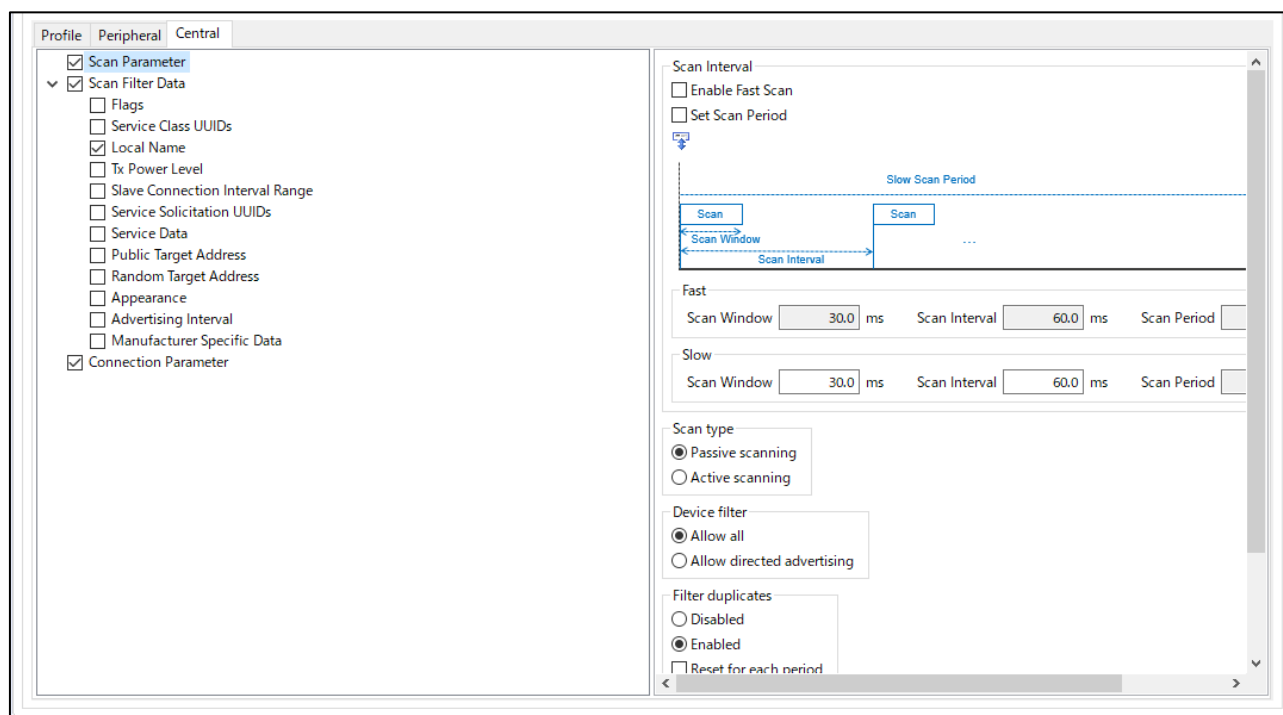


図 3.15 セントラルタブの設定画面

#### 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できます。設定できるパラメータは表 3.11 に示します。

【注】デフォルトの設定で接続しづらい場合、「Scan Interval」を小さく、「Scan Window」を長くします。

表 3.11 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ーを設定できます。 以下からフィルタポリシー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ィルタを無効にし、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できます。Scan Filter Data を設定すると、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が、アプリケーションに通知されます。

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 3.8 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 3.12 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できます。 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	ペリフェラル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場合のパラメータを設定できます。 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。

## 4. プログラムの実装

本章では、QE for BLE から生成されたソースコードをベースにユーザーアプリケーションを実装する方法を説明します。

QE for BLE から生成されるコードを表 4.1 に示します。

表 4.1 QE for BLE が生成するファイル

プログラム	ファイル名	説明
アプリケーション フレームワーク	app_main.c	ユーザーアプリケーションとプロファイルのフレームワークです。 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 データベース プログラム	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]にチェックしたサービスが定義されます。
サービス API プログラム	r_ble_[略称][s/c].c r_ble_[略称][s/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]がサーバー側、[略称][c]がクライアント側です。 Example) [略称]=[sig]、サーバー : 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
プロファイル共通部	profile_cmn discovery	プロファイル共通部のプログラムです。 RA4W1, RX23W 環境の場合に生成されます。

本章では、カスタムサービスの実装例を以下のプロファイルを用いて説明します。XX サービスと YYY キャラクタリスティック、ZZZ ディスクリプタを追加しています。

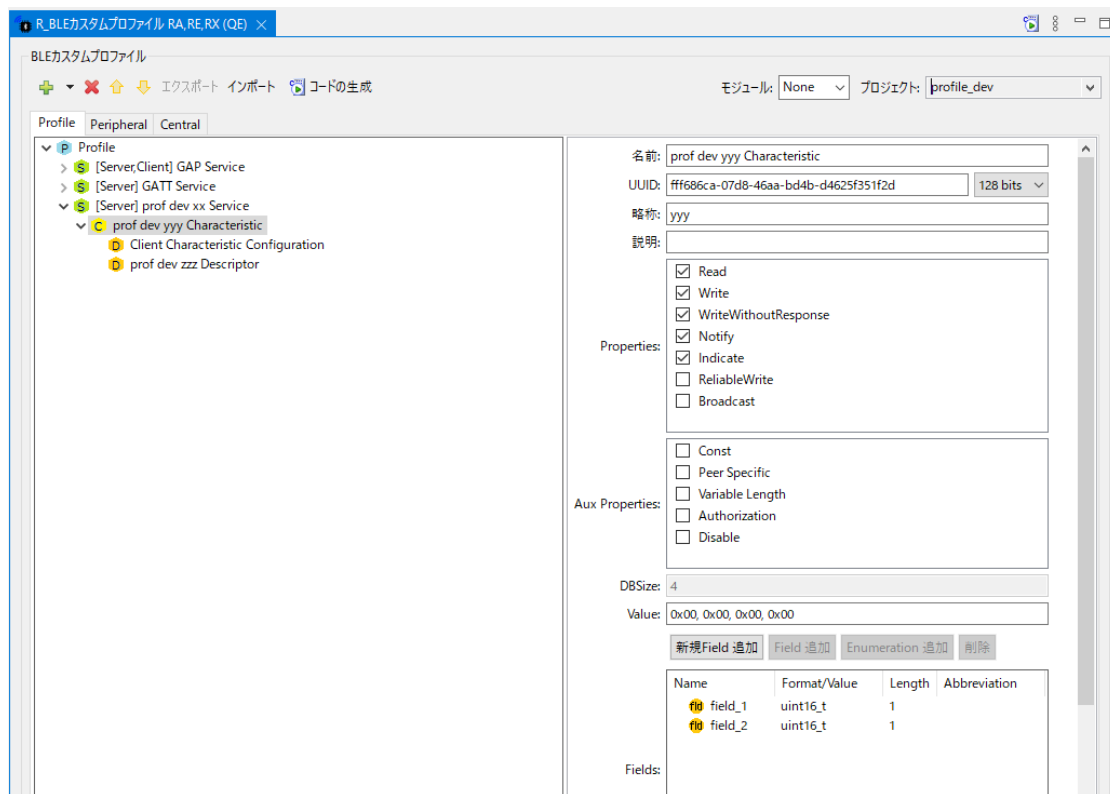


図 4.1 説明で使用するキャラクタリスティック

表 4.2 説明で使用するプロファイル

サービス	略称	ロール
GAP Service	gap	サーバー クライアント
GATT Service	gat	サーバー クライアント
prof dev XX Service	xx	サーバー クライアント
キャラクターリスティック名	略称	プロパティ
prof dev YYY Characteristic	yyy	Read Write WriteWithoutResponse Notify Indication
ディスクリプタ名	略称	プロパティ
Client Configuration Characteristic Descriptor	cli cnfg	Read Write
prof dev ZZZ Descriptor	zzz	Read Write

QE for BLE が生成するコード例を図 4.2 に示します。各プログラムは以下のフォルダに生成されます。

- ・ [プロジェクト名]/qe\_gen/ble

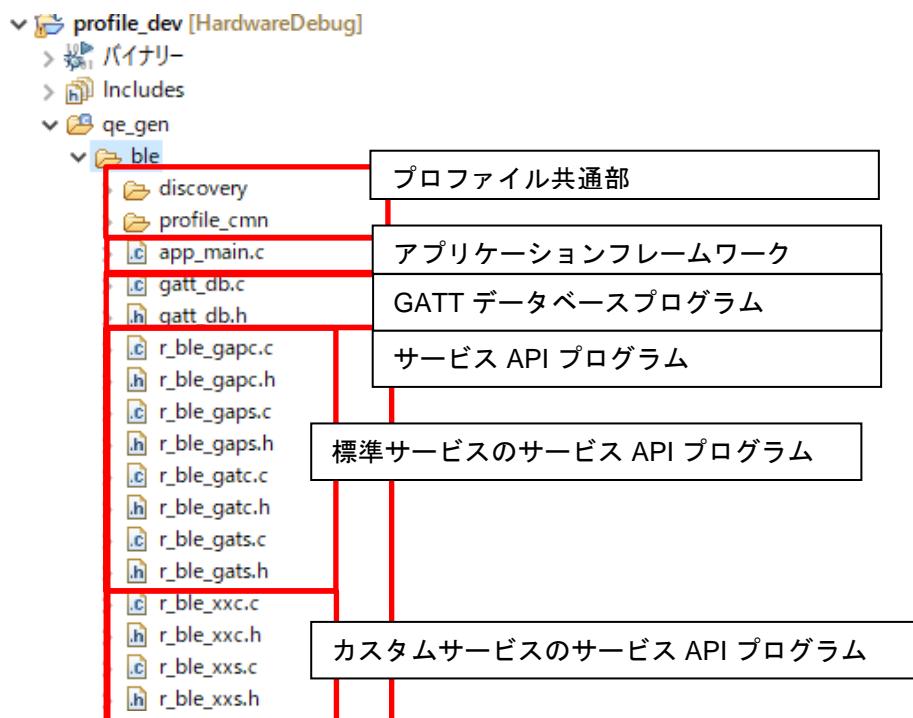


図 4.2 QE for BLE から生成されるソースコード例

QE for BLE には、コード生成時のユーザコード保護機能があります。以下に示すコードブロック内に追加されたコードはコード生成後も保持されます。各プログラムに実装する場合はこのコメント間に実装してください。

```

/* 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 4.3 ユーザコードの実装場所

## 4.1 サービス API プログラム (r\_ble\_xxs.c / r\_ble\_xxc.c)

サービス API プログラムはプロファイルによるデータ通信を簡略化するプログラムです。

本節では、生成される API の詳細と、API を利用するために必要な encode/decode 関数の実装方法について説明します。

各サービスはロールに関わらず、表 4.3 の関数が生成されます。表中の[xx]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ーの場合 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_XX[S or C]_Init	サービスの初期化関数です。プロファイル共通部にサービスを登録します。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。本関数で登録したコールバック関数に、GATT 動作の結果が通知されます。

サービス API プログラムは、GATT 動作を行うための API を提供します。QE for BLE で選択した Property に応じた GATT 動作に対応する API が生成されます。

表 4.4 クライアントロールに生成される GATT 動作 API

関数名	説明
R_BLE_XXC_WriteYyy R_BLE_XXC_WriteYyyZzz	クライアント側から GATT データベースを書き込みます。 書き込み完了時にサーバーから応答が返ります。 QE for BLE の Write プロパティを有効にした場合に生成されます。
R_BLE_XXC_ReadYyyZzz	サーバーの GATT データベースを読み出します。 QE for BLE の Read プロパティを有効にした場合に生成されます。
R_BLE_XXC_WritewithoutRespYyy	クライアント側から GATT データベースを書き込みます。 書き込み完了時にサーバーから応答が返りません。 QE for BLE の Write Without Response プロパティを有効にした場合に生成されます。
R_BLE_XXC_ServDiscCb	プロファイル共通部のディスカバリライブラリによるサービスディスカバリの結果を受け取り、その結果を保持するために使用します。
R_BLE_XXC_GetServAttrHdl	サービスディスカバリしたアトリビュートハンドルを取得します。

表 4.5 サーバーロールに生成される API

関数名	説明
R_BLE_XXS_NotifyYyy	サーバーからクライアントにデータを送信します。 QE for BLE の Notify プロパティを有効にした場合に生成されます。 この動作で、GATT データベースは書き込まれません。
R_BLE_XXS_IndicateYyy	サーバーからクライアントにデータを送信します。 クライアントから受信確認を受けます。 QE for BLE の Indicate プロパティを有効にした場合に生成されます。 この動作で、GATT データベースは書き込まれません。
R_BLE_XXS_SetYyy R_BLE_XXS_SetYyyZzz	GATT データベースに値を設定します。 QE for BLE の Read, Write, Write Without Response プロパティのいずれかが有効の場合に生成されます。
R_BLE_XXS_GetYyy R_BLE_XXS_GetYyyZzz	GATT データベースに値を取得します。 QE for BLE の Read, Write, Write Without Response プロパティのいずれかが有効の場合に生成されます。

GATT 動作中のイベントが、R\_BLE\_XX[S/C]\_Init に登録したコールバック関数に通知されます。サーバー側に通知されるイベントを表 4.6 に、クライアントに通知されるイベントを表 4.7 に示します。

表 4.6 サーバー側で発生するイベント

イベント	説明
BLE_XXS_EVENT_Yyy_WRITE_REQ	Write 動作による GATT データベースの書き込みリクエストを受け取った時に発生します。
BLE_XXS_EVENT_Yyy_WRITE_COMP BLE_XXS_EVENT_Yyy_Zzz_WRITE_COMP	Write 動作による GATT データベースの書き込みが完了した場合に発生します。
BLE_XXS_EVENT_WRITE_CMD	Write Without Response による GATT データベースの書き込みを受け取った場合に発生します。
BLE_XXS_EVENT_Yyy_READ_REQ	Read 動作による GATT データベースの読み出しリクエストを受け取った時に発生します。
BLE_XXS_EVENT_Yyy_HDL_VAL_CNF	Indicate 動作の受信確認パケットを受信したときに発生します。

表 4.7 クライアント側で発生するイベント

イベント	説明
BLE_XXS_EVENT_Yyy_WRITE_RSP BLE_XXS_EVENT_Yyy_Zzz_WRITE_RSP	Write 動作の、書き込みリクエストに対する応答の受信時に発生します。
BLE_XXC_EVENT_Yyy_READ_RSP BLE_XXC_EVENT_Yyy_Zzz_READ_RSP	Read 動作の、読み出し結果を受信した場合に発生します。
BLE_XXC_EVENT_Yyy_HDL_VAL_NTF	Notify 動作によるデータ受信時に発生します。
BLE_XXC_EVENT_Yyy_HDL_VAL_IND	Indicate 動作によるデータ受信時に発生します。

サービス API プログラムでは、キャラクターリックやディスクリプタは、QE for BLE の Field の設定を反映したデータ型で表されます。キャラクタースティックとディスクリプタのデータ型は、図 4.4 に示すようにキャラクタースティック定義構造体で確認できます。

```

/* prof_dev_yyy Characteristic characteristic definition */
static const st_ble_servc_char_info_t gs_yyy_char = {
    .uuid_128      = BLE_XXXC_YYY_UUID,
    .uuid_type     = BLE_GATT_128_BIT_UUID_FORMAT,
    .app_size      = sizeof(st_ble_xxc_yyy_t),
    .db_size       = BLE_XXC_YYY_LEN,
    .char_idx      = BLE_XXC_YYY_IDX,
    .p_attr_hdls  = gs_yyy_char_ranges,
    .decode        = (ble_servc_attr_decode_t)decode_st_ble_xxc_yyy_t,
    .encode        = (ble_servc_attr_encode_t)encode_st_ble_xxc_yyy_t,
    .num_of_descs = ARRAY_SIZE(gspp_yyy_descs),
    .pp_descs     = gspp_yyy_descs,
};

```

図 4.4 イベントに通知されるアプリケーションデータの型の参照(r\_ble\_xx[c/s].c)

QE for BLE の Field に設定した要素が複数ある場合には、キャラクタースティックやディスクリプタの構造体が定義されます。図 4.5 のようにヘッダファイルに定義されます。

```
/******//**
 * @brief prof dev yyy Characteristic value structure.
 *****/
typedef struct {
    uint16_t field_1; /**< field_1 */
    uint16_t field_2; /**< field_2 */
} st_ble_xxc_yyy_t;
```

図 4.5 キャラクタースティックのアプリケーションデータ構造体(r\_ble\_xx[c/s].h)

無線パケットの PDU に含まれるアプリケーションデータとこの構造体は、エンコードデコード関数によって相互に変換されます。

#### 4.1.1 encode/decode 関数の説明

サービス API プログラムは、キャラクタリスティックやディスクリプタの値を QE for BLE の「Field」で指定した型で扱います。一方、GATT データベースや、Bluetooth LE Protocol Stack では、これらの形式は 8bit データ配列として扱われます。

プロファイル共通部は、キャラクタリスティック毎に encode/decode 関数を使用してアプリケーション用のデータ構造と GATT データベース用の 8bit 配列データを変換します。

encode/decode 関数は図 4.6 のようにデータ形式を変換します。

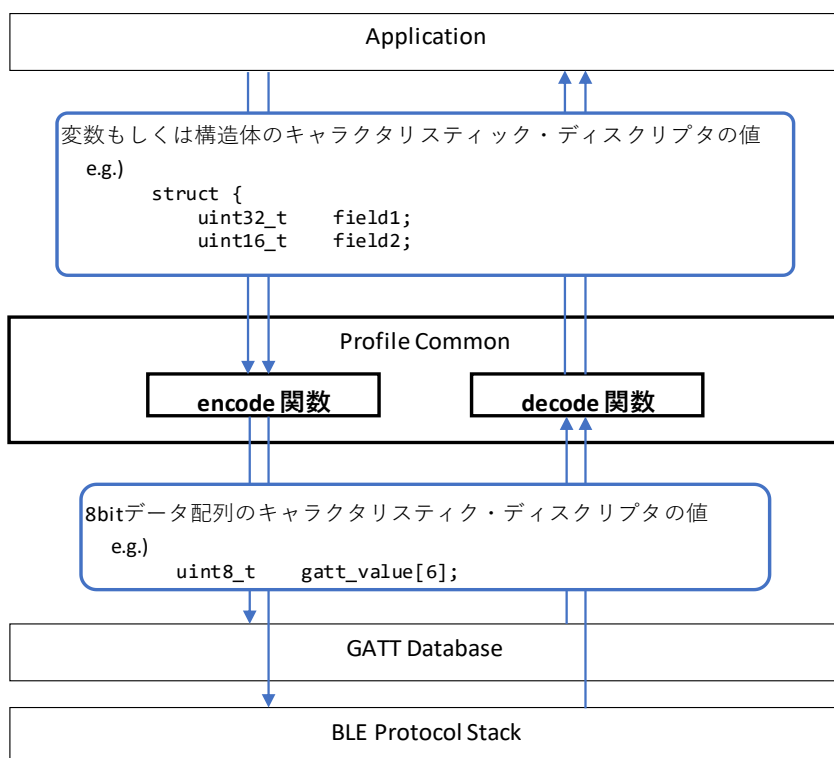


図 4.6 encode/decode 関数の関係

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プロファイル共通部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 サーバーに 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 4.7 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 サーバー側のサービス API プログラムで使用されます。

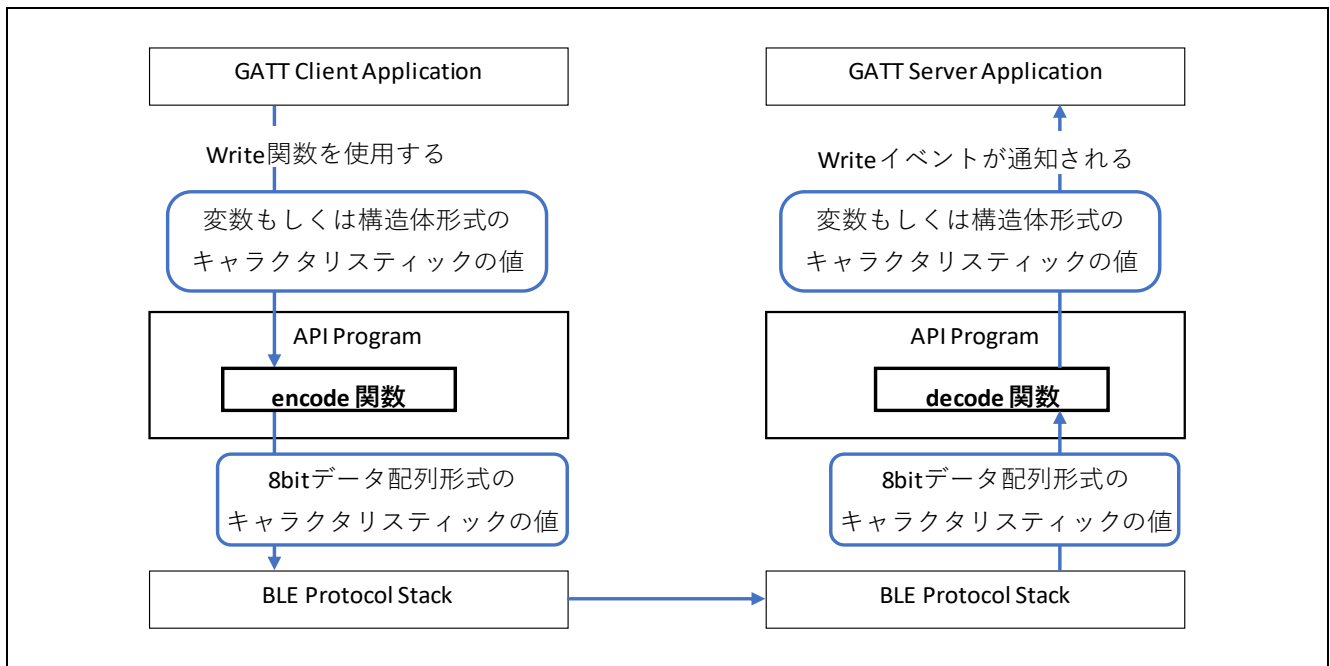


図 4.7 Write 動作時の encode/decode 関数のユースケース

同様に GATT サーバーが 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 4.8 に示します。encode 関数は GATT サーバー側のサービス 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。

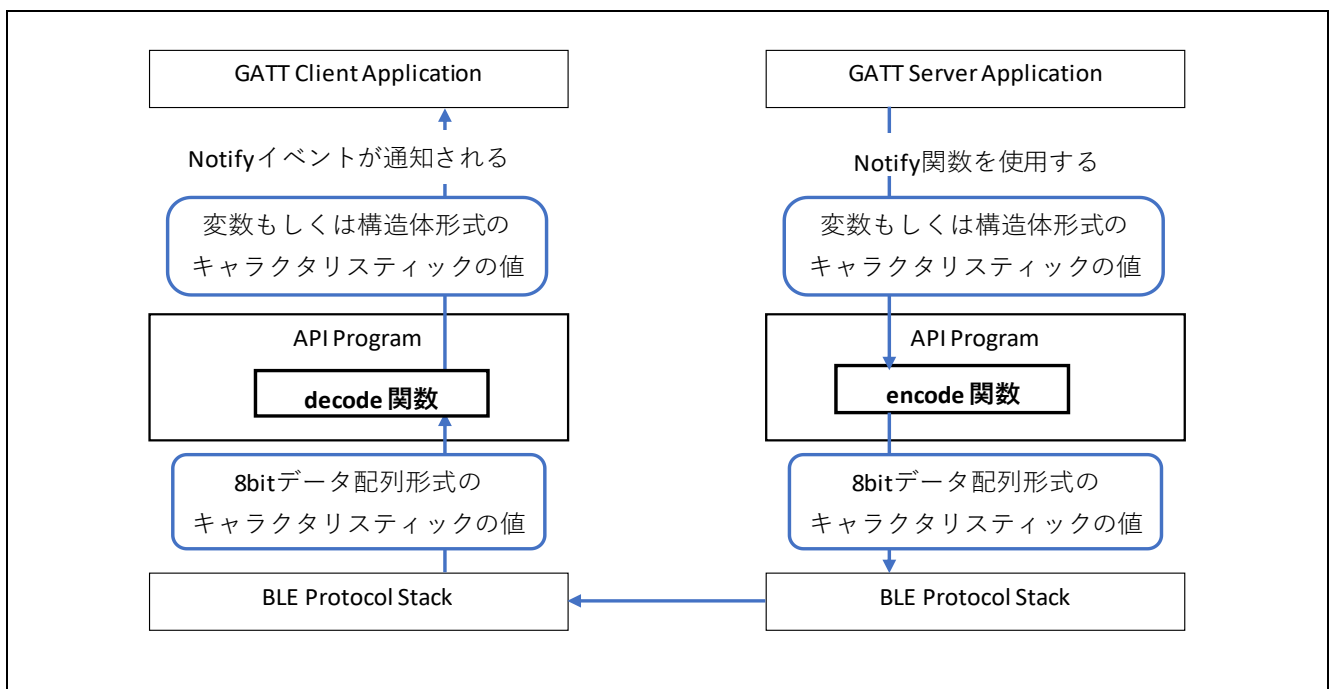


図 4.8 Notify 動作時の encode/decode 関数のユースケース

### 4.1.2 エンコードデコード関数の自動生成機能

QE for BLE Utility 1.60 以降からエンコードデコード関数が自動生成されます。本節では、生成されるエンコードデコード関数を説明します。エンコードデコード関数は、フィールドに設定されたデータ型を表 4.8 に示すバイト数になるようにエンコードデコードします。

表 4.8 QE for BLE で設定できる構造体とバイト数

データ型	バイト数
bool, char, uint8_t, int8_t	1
uint16_t, int16_t	2
uint32_t, int32_t	4
st_ble_ieee11073_sfloat_t	2
st_ble_ieee11073_float_t	3
st_ble_dev_addr_t	7
st_ble_date_time_t	7

st\_ble\_seq\_data\_t が複数のデータ型とともにフィールドに含まれる場合には、そのキャラクターリステック/ディスクリプタには空のエンコードデコード関数が生成されます。

図 4.9 にフィールドとバイト列の例を示します。各フィールドを上から順に送信データに詰めます。

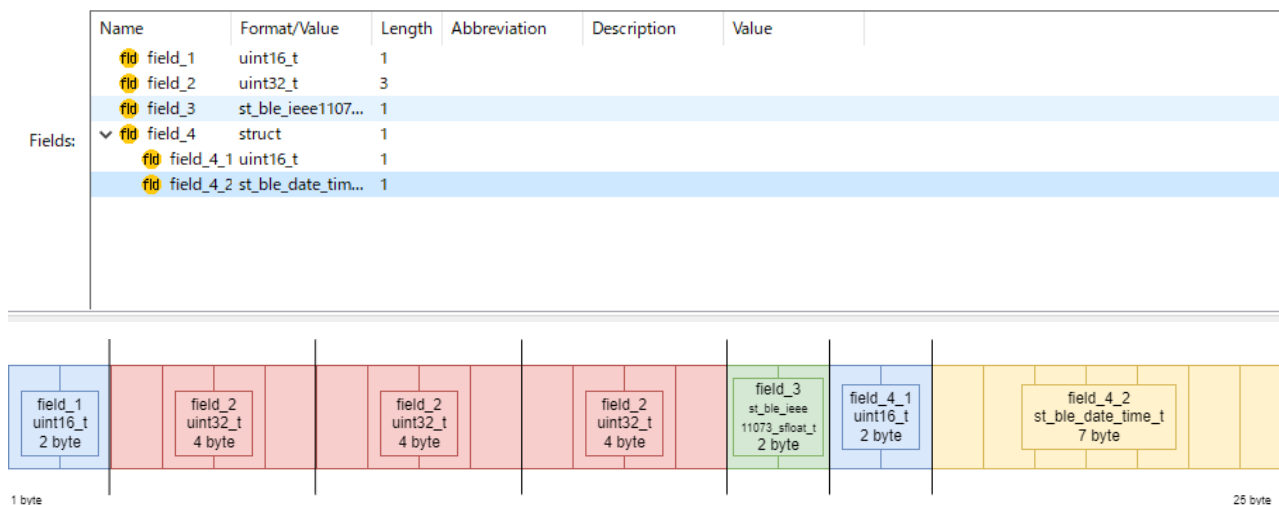


図 4.9 エンコードデコード関数生成機能の説明用フィールドとバイト列

また、図 4.10 に生成されるエンコード関数を示します。Length を 1 より大きく設定した場合には、for 文で送信データに詰められます。

```
static ble_status_t encode_st_ble_xxc_prof_t(
    const st_ble_xxc_prof_t *p_app_value,
    st_ble_gatt_value_t *p_gatt_value)
{
    /* Start user code for characteristic value encode function. */
    /* End user code. Do not edit comment generated here */
#ifdef BLE_XXC_DISABLE_ENCODE_DECODE

    uint32_t pos = 0;

    BT_PACK_LE_2_BYTE(&p_gatt_value->p_value[pos], &p_app_value->field_1);
    pos += 2;

    for (uint32_t i=0; i<3; i++)
    {
        BT_PACK_LE_4_BYTE(&p_gatt_value->p_value[pos], &p_app_value->field_2[i]);
        pos += 4;
    }

    pos += ble_pack_st_ble_ieee11073_sfloat_t(
        &p_gatt_value->p_value[pos],
        &p_app_value->field_3);

    pos += ble_pack_st_ble_xxc_prof_field_4_t(
        &p_gatt_value->p_value[pos],
        &p_app_value->field_4);

    p_gatt_value->value_len = (uint16_t)pos;
#endif /* BLE_XXC_DISABLE_ENCODE_DECODE */

    return BLE_SUCCESS;
}
```

図 4.10 生成されるエンコード関数の例

図 4.11 に生成されるデコード関数を示します。

```

static ble_status_t decode_st_ble_xxc_prof_t(
    st_ble_xxc_prof_t *p_app_value,
    const st_ble_gatt_value_t *p_gatt_value)
{
    /* Start user code for New Characteristic value decode function.  */
    /* End user code. Do not edit comment generated here */
    #ifndef BLE_XXC_DISABLE_ENCODE_DECODE

        uint32_t pos = 0;

        BT_UNPACK_LE_2_BYTE(&p_app_value->field_1,&p_gatt_value->p_value[pos]);
        pos += 2;

        for (uint32_t i=0;i<3;i++)
        {
            BT_UNPACK_LE_4_BYTE(&p_app_value->field_2[i],&p_gatt_value->p_value[pos]);
            pos += 4;
        }

        pos += ble_unpack_st_ble_ieee11073_sfloat_t(
            &p_app_value->field_3,
            &p_gatt_value->p_value[pos]);

        pos += ble_unpack_st_ble_xxc_prof_field_4_t(
            &p_app_value->field_4,
            &p_gatt_value->p_value[pos]);

    #endif /* BLE_XXC_DISABLE_ENCODE_DECODE */

    return BLE_SUCCESS;
}

```

図 4.11 生成されるデコード関数の例

これらのエンコードデコード関数は図 4.12 のようにユーザーコードブロックに以下のマクロを定義することで無効できます。

無効にした場合には 4.1.3 章を参考にしてエンコードデコード関数を実装してください。

```

/* Start user code for function prototype declarations and global variables. Do not
edit comment generated here */
#define BLE_XXC_DISABLE_ENCODE_DECODE
/* End user code. Do not edit comment generated here */

```

図 4.12 生成されたエンコードデコード関数の無効化

## 4.1.3 エンコードデコード関数の実装

QE for BLE Utility 1.60 より前のバージョンを使用もしくは、自動生成されるエンコードデコード関数を無効にした場合には、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 4.9 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 4.9 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 4.14 に示す Field を持つカスタムキャラクターリスティック(略称:yyy)の decode / encode 関数の実装例を図 4.13 に示します。この decode / encode 関数では表 4.9 で提供される関数を使用しています。

```

static ble_status_t decode_st_ble_xxxc_yyy_t(st_ble_xxxc_yyy_t *p_app_value, const st_ble_gatt_value_t
*p_gatt_value)
{
    /* Start user code for profile dev yyy Characteristic characteristic value decode function. Do not
    edit comment generated here */
    uint32_t pos = 0;
    BT_UNPACK_LE_2_BYTE(&p_app_value->field_1, &p_gatt_value->p_value[pos]);
    pos += 2;

    BT_UNPACK_LE_2_BYTE(&p_app_value->field_2, &p_gatt_value->p_value[pos]);
    pos +=2;
    /* End user code. Do not edit comment generated here */
    return BLE_SUCCESS;
}

static ble_status_t encode_st_ble_xxc_yyy_t(const st_ble_xxc_yyy_t *p_app_value, st_ble_gatt_value_t
*p_gatt_value)
{
    /* Start user code for profile dev yyy Characteristic characteristic value encode function. Do not
    edit comment generated here */
    uint32_t pos = 0;
    BT_PACK_LE_2_BYTE(&p_gatt_value->p_value[pos], &p_app_value->field_1);
    pos += 2;

    BT_PACK_LE_2_BYTE(&p_gatt_value->p_value[pos], &p_app_value->field_2);
    pos += 2;
    /* End user code. Do not edit comment generated here */
    return BLE_SUCCESS;
}

```

図 4.13 Field の実装例(encode 関数の実装例)

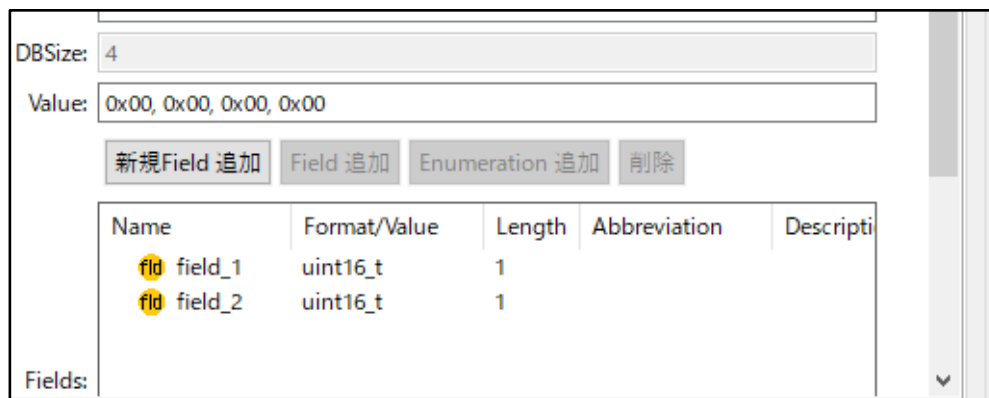


図 4.14 Field の設定例

SIG 標準サービスの場合には encode/decode 関数は既実装されています。したがって、この手順は不要です。

また、「Field」が以下の型一つだけの場合にも encode / decode 関数を実装する必要はありません。

uint8\_t, uint16\_t, uint32\_t, int8\_t, int16\_t, int32\_t,

st\_ble\_seq\_data\_t, st\_ble\_ieee11073\_sfloat, st\_ble\_date\_time\_t

## 4.2 アプリケーションフレームワーク(app\_main.c)

app\_main.c には、アプリケーションのロールに応じた Bluetooth LE 通信を実現するプログラムがあらかじめ実装されています。

図 4.15 にそれぞれのロールで生成した app\_main.c 同士を実行した場合のシーケンスチャートを示します。本ケースでは、セントラルデバイスを GATT クライアント、ペリフェラルデバイスを GATT サーバーとします。

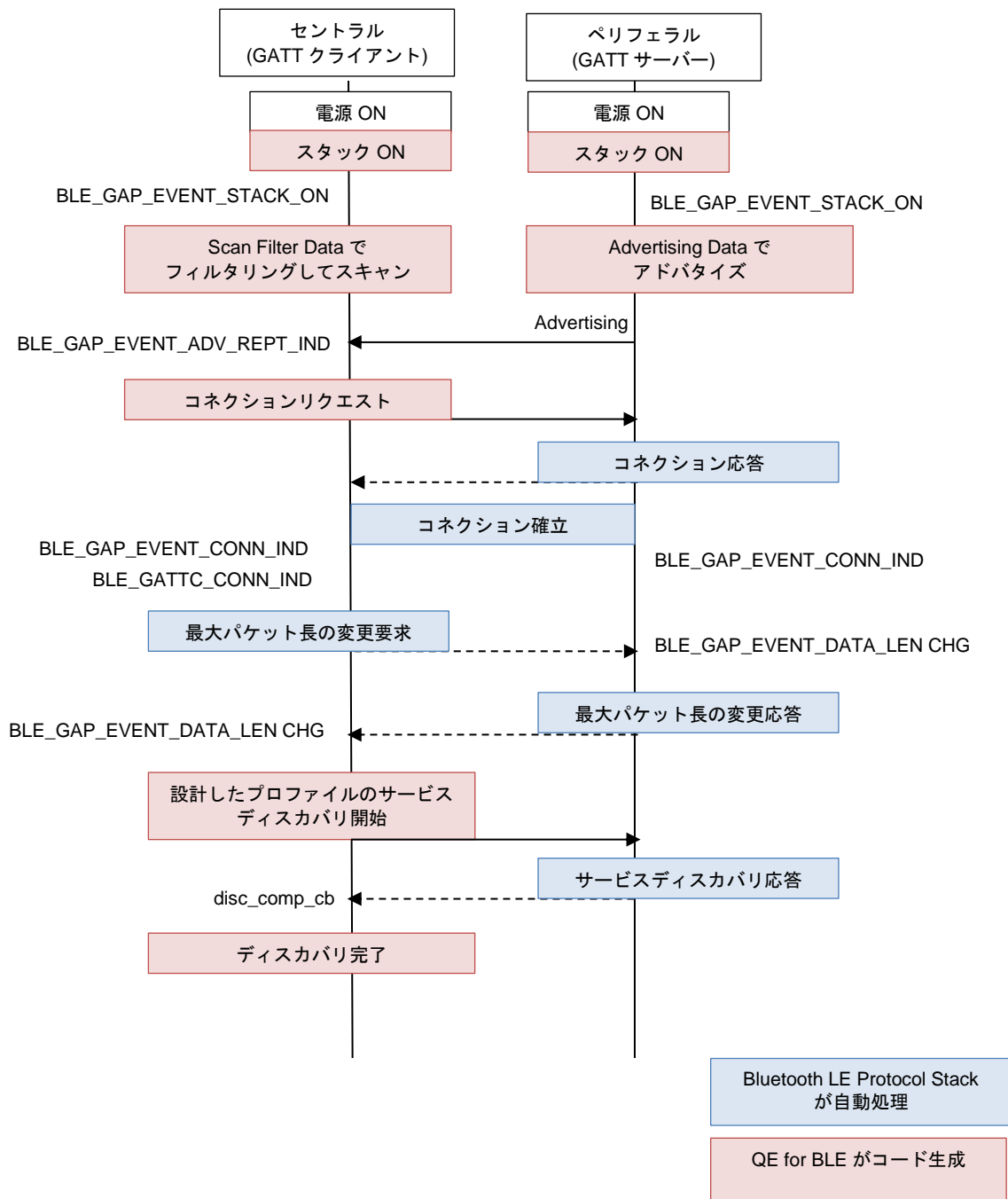


図 4.15 app\_main.c に実装されているプログラムの動作

サービスディスカバリが終了すると、disc\_comp\_cb が通知されます。クライアントは、このコールバック関数を呼び出し以降にプロファイル通信が可能になります。

app\_main.c は QE for BLE で設定したプロファイルを含むアプリケーションのスケルトンプログラムです。ソースコード上に定義された関数に処理を追加することで、プロファイルによるデータ通信を実現します。

ここでは、代表的な GATT 動作を例に、サービス API プログラムを利用したプロファイルのデータ通信の実装方法を説明します。その他の Bluetooth LE 機能を実装する場合には、アプリケーション開発者ガイドをご覧ください



#### 4.2.1 セキュリティ要件への対応

3.2.2 章でサービスの Security Level 3 以上を設定した場合、データ通信を行うためにはペアリングパラメータの変更が必要です。

##### 4.2.1.1 Security Level 3 に設定した場合

ペアリング時のユーザ操作と MITM Protection が必要です。これらを実現するためには、デバイスに入出力機能が必要です。デバイスの機能に合わせて io capability を変更してください。

RX23W 環境では app\_main.c のペアリングパラメータを変更します。

```

/* Pairing parameters */
static st_ble_abs_pairing_param_t gs_abs_pairing_param =
{
    .iocap          = BLE_GAP_IOCAP_NOINPUT_NOOUTPUT,
    .mitm           = BLE_GAP_SEC_MITM_BEST_EFFORT,
    .sec_conn_only  = BLE_GAP_SC_BEST_EFFORT,
    .loc_key_dist   = BLE_GAP_KEY_DIST_ENCKEY,
    .rem_key_dist   = 0,
    .max_key_size   = 16,
};
    
```

図 4.16 app\_main.c のペアリングパラメータの変更(RX23W)

RA4W1 環境では、ペアリングパラメータは RA Configurator から設定できます。

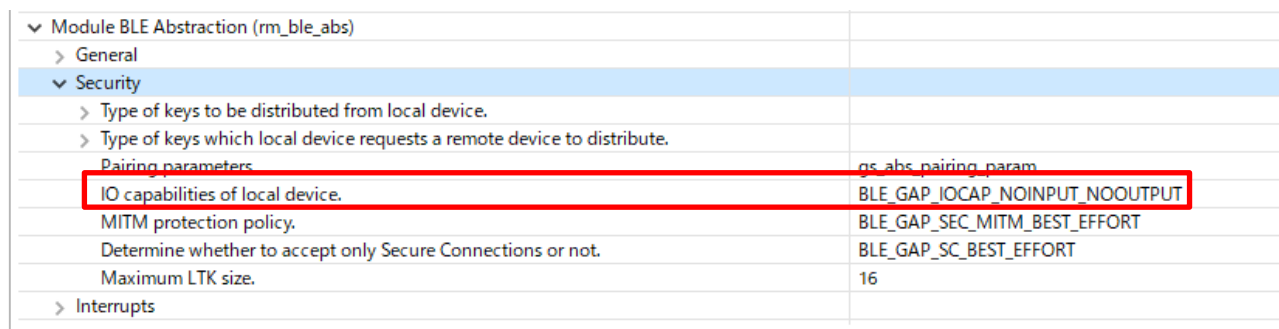


図 4.17 ペアリングパラメータの変更(RA4W1)

詳細は以下のドキュメントをご覧ください。

表 4.10 ペアリング実施に関するドキュメント

MCU	ドキュメント	章
RX23W	RX23W グループ Bluetooth Low Energy アプリケーション開発者ガイド (R01AN5504)	9.1 ペアリング
RA4W1	RA4W1 Group Bluetooth Low Energy Application Developer's Guide (R01AN5653)	8.1 Pairing

##### 4.2.1.2 Security Level 4 に設定した場合

4.2.1.1 の設定を行ったうえで、リモートデバイスが LE Secure Connection に対応している必要があります。RX23W, RE01B, RA4W1 はすべて LE Secure Connection に対応していますが、リモートデバイスが LE Secure Connection に対応していない場合には、GATT データベースへのアクセスが許可されません。

#### 4.2.2 MTU の変更

MTU は、一度の GATT 動作で送受信できる最大のデータ長です。接続時の MTU は 23 バイトです。MTU は接続中に一度だけクライアントから変更できます。

相手からの受信確認が不要な Notify や Write Without Response 動作は効率よく連続してデータを送信できますが、MTU-3 byte より大きいデータは送信できません。効率的なデータ送信を実現するには、キャラクタースティックのサイズが MTU-3 byte 以下になるように MTU を変更することが一つの解決策となります。

サポートする MTU サイズは、デバイスによって異なります。Bluetooth LE Protocol Stack は 247 byte までサポートしています。MTU は、お互いのデバイスがサポートする MTU のうち小さい方が設定されます。

MTU 交換のシーケンスチャートを示します。赤字が関数呼び出し、青字がアプリケーション通知されるイベントです。

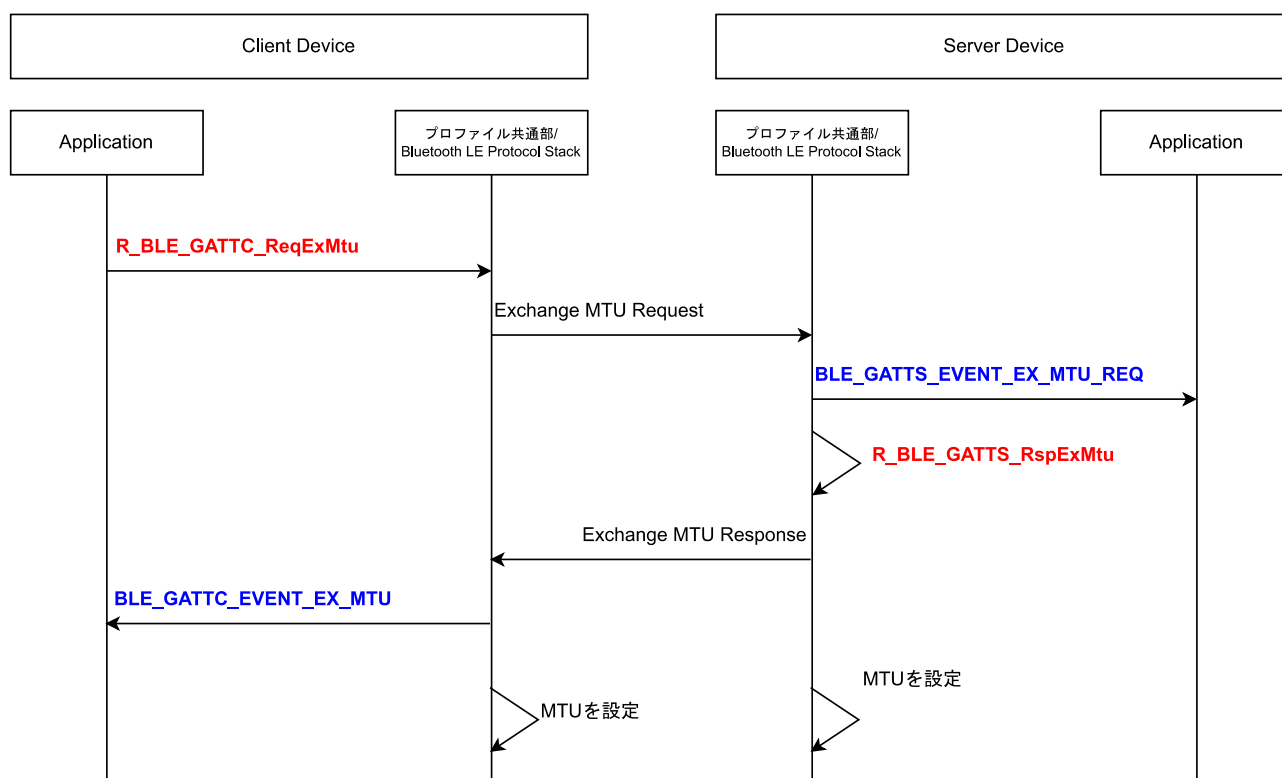


図 4.18 MTU 交換動作時のシーケンスチャート

## 4.2.2.1 クライアントの実装

MTUの変更はクライアントから行います。MTUの交換を行うAPIがBluetooth LE Protocol Stackに定義されています。

```
ble_status_t R_BLE_GATTC_ReqExMtu(uint16_t conn_hdl, uint16_t mtu);
```

図 4.19 MTU 交換 API

例えば、GATT Client API のイベントコールバック `gattc_cb` に、図 4.20 のように実装することで、サービスディスカバリーを効率的に実行できます。デバイスがサポートする最大の MTU サイズは、Bluetooth LE Protocol Stack のコンフィグ画面で設定できます。

```
static void gattc_cb(uint16_t type, ble_status_t result, st_ble_gattc_evt_data_t *p_data)
{
    R_BLE_SERVC_GattcCb(type, result, p_data);
    switch(type)
    {
        case BLE_GATTC_EVENT_CONN_IND:
        {
            uint16_t mtu = 247;
            R_BLE_GATTC_ReqExMtu(p_data->conn_hdl, mtu);
        } break;

        case BLE_GATTC_EVENT_EX_MTU_RSP:
        {
            /* Start discovery operation after mtu exchanged */
            R_BLE_DISC_Start(p_data->conn_hdl, gs_disc_entries,
                ARRAY_SIZE(gs_disc_entries), disc_comp_cb);
        } break;
    }
}
```

図 4.20 MTU 交換の実装例

## 4.2.2.2 サーバーの実装

サーバーは、BLE\_GATTS\_EVENT\_EX\_MTU\_REQ イベントで、サポートする MTU をクライアントに送信します。

本処理は、プロファイル共通部の、`profile_cmn/r_ble_servs_if.c` ファイルの `R_BLE_SERVS_GattsCb` 関数に実装されています。

追加の実装は不要です。

```
void R_BLE_SERVS_GattsCb(uint16_t type, ble_status_t result, st_ble_gatts_evt_data_t *p_data)
{
    static uint16_t s_write_long_attr_hdl = BLE_GATT_INVALID_ATTR_HDL_VAL;

    switch (type)
    {
        case BLE_GATTS_EVENT_CONN_IND:
        case BLE_GATTS_EVENT_DISCONN_IND:
        break;

        case BLE_GATTS_EVENT_EX_MTU_REQ:
        {
            R_BLE_GATTS_RspExMtu(p_data->conn_hdl, BLE_PRF_MTU_SIZE);
        }
        break;
    }
}
```

図 4.21 r\_ble\_servs\_if.c ファイルの MTU 交換処理の実装

### 4.2.3 Write 動作の実装

Write 動作は、クライアントからサーバーに対してデータを送信し、GATT データベースに値を書き込みます。サーバーは、書き込みの要求に対してレスポンスを返します。クライアントがサーバーへの送信完了を確認しながらデータを送信できます。

Write 動作 API では、送信するデータ長が MTU-3 よりも大きい場合には Write Long 動作、MTU-3 以下の場合には Write 動作を行います。

図 4.22 に Write 動作を行う場合のシーケンスチャートを示します。赤字が関数呼び出し、青字がアプリケーション通知されるイベントです。

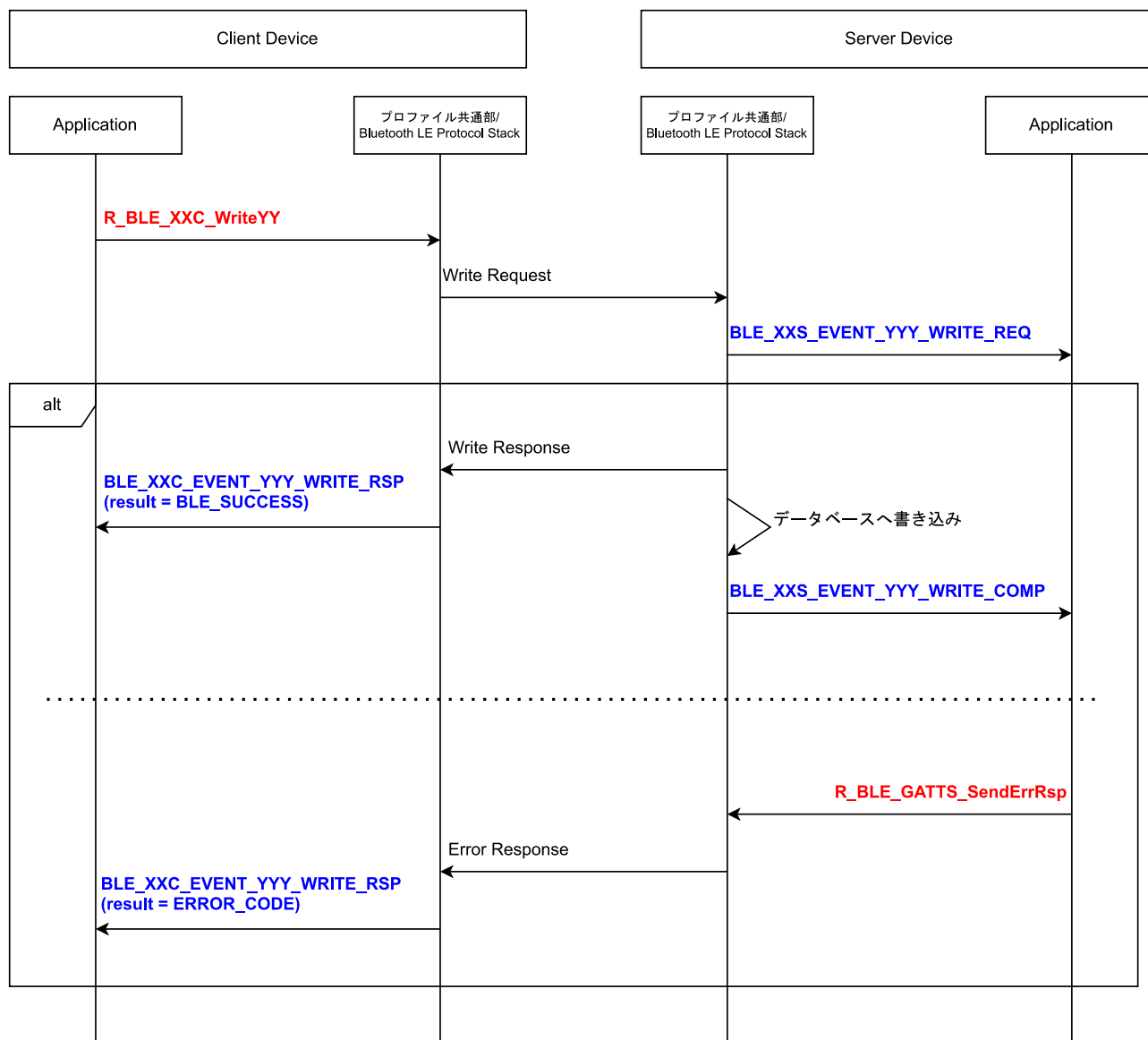


図 4.22 Write 動作時のシーケンスチャート

Write Long 動作では、クライアントは、サーバーに送信するデータを一度に送信できるサイズにデータを分割して送信します。その後、全データを送信後に GATT データベースに書き込みます。サーバーは受信したデータを Prepare Write Queue にため込みます。図 4.23 に Write Long 動作を行う場合のシーケンスチャートを示します。赤字が関数呼び出し、青字がアプリケーション通知されるイベントです。

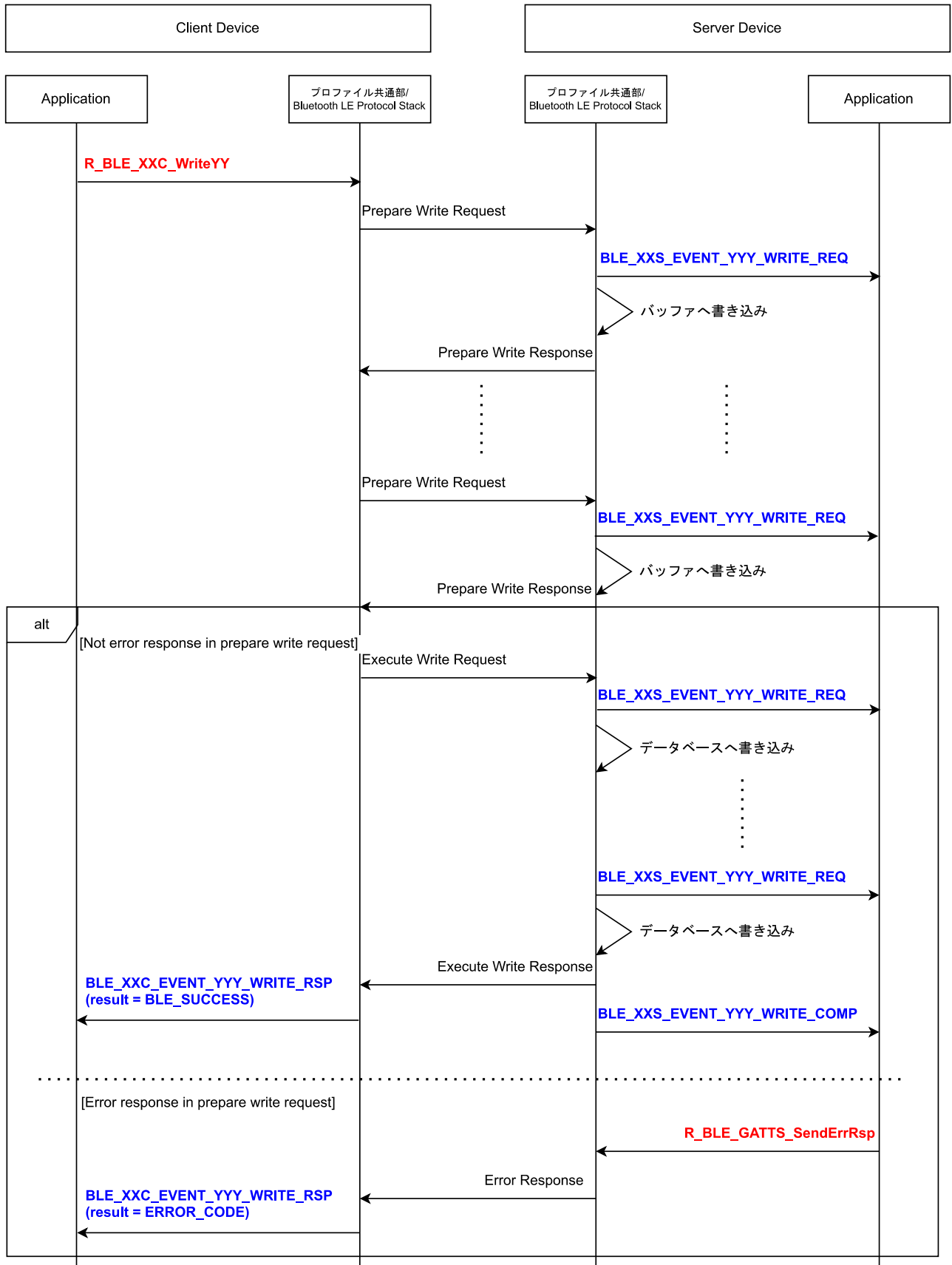


図 4.23 Write Long 動作のシーケンスチャート

## 4.2.3.1 クライアントの実装

## Write 動作 API

Write 動作は、クライアントから開始します。Write 動作には、サービス API プログラムに実装されている Write 動作 API を使用します。引数はコネクションハンドルと対象のキャラクタリスティックへの送信データです。

```
ble_status_t R_BLE_XXC_WriteYyy(uint16_t conn_hdl, const st_ble_xxc_yy_t *p_value);
ble_status_t R_BLE_XXC_WriteYyyZzz(uint16_t conn_hdl, const st_ble_xxc_yy_t *p_value);
```

図 4.24 サービス API プログラム(r\_ble\_xxc.h)での Write 動作 API の定義

引数に渡したキャラクタリスティックのエンコード関数が設定するデータ長によって Write 動作を行うか Write Long 動作を行うかが決まります。

表 4.11 クライアントの送信データ長と MTU と実行される動作の関係

クライアントの送信データ長と MTU	実行される動作
data_length <= MTU-3	Write 動作
data_length > MTU-3	Write Long 動作

Write 動作 API の送信データ長は、図 4.25 に示すキャラクタリスティックのエンコード関数の p\_gatt\_value->value\_len の値(黄色ハイライト部分)によって設定されます。Write Long 動作を意図しないキャラクタリスティックを設計する場合には、この設定値が MTU-3 よりも大きくならないように設計してください。

```
static ble_status_t encode_st_ble_xxs_yyy_t
    (const st_ble_xxs_yyy_t *p_app_value, st_ble_gatt_value_t *p_gatt_value)
{
    uint32_t pos = 0;

    BT_PACK_LE_16_BYTE(&p_gatt_value->p_value[pos], &p_app_value->field_1);
    pos += 2;

    BT_PACK_LE_16_BYTE(&p_gatt_value->p_value[pos], &p_app_value->field_2);
    pos += 2;

    p_gatt_value->value_len = pos;

    return BLE_SUCCESS;
}
```

図 4.25 encode 関数による送信データ長の設定

Write 動作 API は、連続して呼び出せません。BLE\_XXC\_EVENT\_YYY\_WRITE\_RSP イベントの受信後に再度呼び出せません。

**BLE\_XXC\_EVENT\_YYY\_WRITE\_RSP イベント**

クライアントは、Write 動作の結果をサーバーから受け取ります。

GATT データベースが書き込まれた場合には、result 変数に BLE\_SUCCESS が通知されます。サーバーが Error Response を送信した場合には、result 変数にエラーコードが通知されます。

```
static void xxc_cb(uint16_t type, ble_status_t result, st_ble_servc_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_XXC_EVENT_YYY_WRITE_RSP:
        {
            if(result == BLE_SUCCESS)
            {
                /* GATT Database value in server is written. */
            }
            else
            {
                /* Error Response (0x30XX) or BLE_ERR_RSP_TIMEOUT (0x0011) */
            }
        }
        break;
    }
}
```

図 4.26 クライアントの Write 動作時のイベントの実装例

## 4.2.3.2 サーバーの実装

## Prepare Write Queue の設定

データ長が  $MTU_{min} - 3$  byte (20 byte) を超えるキャラクタリスティックを設計した場合には、Write Long 動作による GATT データベースへの書き込みが行われる可能性があります。

クライアントからの Write Long 動作を受け入れる場合には、分割されたデータを保持するための一時領域(Prepare Write Queue)を用意します。この Prepare Write Queue を、Bluetooth LE Protocol Stack に登録することで Write Long 動作に対応します。

app\_main.c には、あらかじめ Prepare Write Queue に関する処理が追加されています。QE for BLE のコード生成では、同時接続台数 1 の場合に 245 byte のバッファに対して 14 回の Prepare Write Request を保持する設定になっています。

アプリケーションに合わせて、図 4.27 に示すマクロ定義を変更してください。

```
/* Queue for Prepare Write Operation. Change if needed. */
#define BLE_GATTS_QUEUE_ELEMENTS_SIZE (14)
#define BLE_GATTS_QUEUE_BUFFER_LEN (245)
#define BLE_GATTS_QUEUE_NUM (1)
```

図 4.27 Prepare Write Queue の設定

Write Long 動作を使用する可能性がない場合には、図 4.28 のようにマクロ定義を行うことで、この処理を無効にできます。

Prepare Write Queue が登録されていない状態で Prepare Write Request を受信した場合には、エラーレスポンスが Bluetooth LE Protocol Stack から自動で応答されます。

```
#define BLE_APP_PREPARE_WRITE_DISABLE (1)
```

図 4.28 Prepare Write Queue の無効化

## BLE\_XXS\_EVENT\_YYY\_WRITE\_REQ イベント

Write 動作によってクライアントが送信した Write のデータは、サーバーの BLE\_XXS\_EVENT\_YY\_WRITE\_REQ イベントに通知されます。キャラクタリスティックの構造体にキャストすることで、書き込まれた値を確認できます。

アプリケーションが通知された Write データを評価して GATT データベースへの書き込みを受け入れない場合には、R\_BLE\_GATTS\_SendErrRsp 関数を呼びます。この関数を呼ぶと、クライアントにエラーレスポンスが送信され、GATT データベースへ書き込みされません。



**BLE\_XXS\_EVENT\_YYY\_WRITE\_COMP イベント**

WRITE\_REQ イベントで R\_BLE\_GATTS\_SendErrRsp 関数が呼ばれなかった場合、Bluetooth LE Protocol Stack は、クライアントに Write Response を送信し GATT データベースへ値を書き込みます。書き込み完了後の GATT データベースの値が BLE\_XXS\_EVENT\_YY\_WRITE\_COMP イベントに通知されます。サーバーの Write 動作はこのイベントで完了です。

これらのイベントは、app\_main.c の xxs\_cb 関数に通知されます。この関数にアプリケーションの処理を追加します。以下に、実装例を示します。result 変数が BLE\_SUCCESS の場合に、書き込まれた値を取得できます。

```
static void xxs_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_XXS_EVENT_YYY_WRITE_REQ:
        {
            if( BLE_SUCCESS == result)
            {
                /* Write Request Data*/
                st_ble_xxs_yyy_t *event_data = (st_ble_xxs_yyy_t*)p_data->p_param;

                if(event_data->field_1 != 0x01)
                {
                    /* Application can send Error Response */
                    uint16_t error_code = 0x3081;
                    R_BLE_GATTS_SendErrRsp(error_code);
                }
            }
            } break;

        case BLE_XXS_EVENT_YYY_WRITE_COMP:
        {
            if( BLE_SUCCESS == result)
            {
                /* Cast Application data*/
                st_ble_xxs_yyy_t *event_data = (st_ble_xxs_yyy_t *)p_data->p_param;

                /* Implement process in Write Complete */
                /* Application can execute next write operation */
            }
            } break;
    }
}
```

図 4.29 サーバーの Write 動作時のイベントの実装例

## 4.2.4 Write Without Response 動作の実装

Write Without Response 動作は、クライアントからサーバーに対して GATT データベースに値を書き込みます。サーバーは、GATT データベースの書き込みに対してレスポンスを返しません。クライアントからサーバーへ高速にデータを送信する場合に便利です。Write Without Response では、MTU-3 バイトを超えるデータは送信できません。

Write Without Response 動作を行う場合のシーケンスチャートを示します。赤字が関数呼び出し、青字がアプリケーション通知されるイベントです。

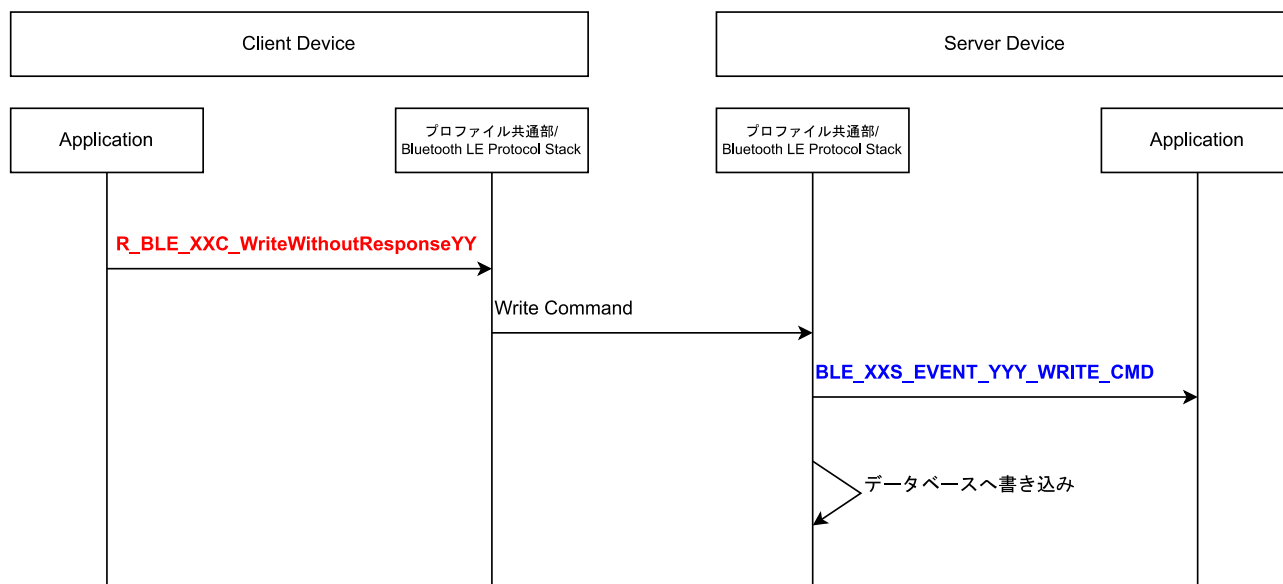


図 4.30 Write Without Response 動作時のシーケンスチャート

## 4.2.4.1 クライアントの実装

**Write Without Response 動作 API**

Write Without Response 動作は、クライアントから開始します。サービス API プログラムに実装されている Write Without Response 動作 API を使用します。Write Without Response 動作はキャラクターリスティックのみに定義されます。

引数はコネクションハンドルと書き込み対象のキャラクターリスティックの値です。

```
ble_status_t R_BLE_XXC_WriteWithoutResponseYyy  
(uint16_t conn_hdl, const st_ble_xxc_yy_t *p_value);
```

図 4.31 サービス API プログラム(r\_ble\_xxc.h)での Write Without Response 動作 API の定義

API の呼び出し以降に、クライアントに通知されるイベントはありません。

## 4.2.4.2 サーバーの実装

## BLE\_XXS\_EVENT\_YYY\_WRITE\_CMD イベント

クライアントが送信した Write Without Response 動作のデータは、サーバーの BLE\_XXS\_EVENT\_YY\_WRITE\_CMD イベントに通知されます。キャラクタリスティックの構造体にキャストすることで、書き込まれた値を確認できます。result 変数が BLE\_SUCCESS の場合に、書き込まれた値を取得できます。

```
static void xxs_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_XXS_EVENT_YYY_WRITE_CMD:
        {
            if(BLE_SUCCESS == result)
            {
                /* Cast Application data*/
                st_ble_xxs_yyy_t *event_data = (st_ble_xxs_yyy_t *)p_data->p_param;

                /* Implement process in Write Without Response */
                /* The GATT database value is not written when this event is notified. */
            }
        } break;
    }
}
```

図 4.32 サーバーの Write Without Response の受信イベント時の実装例

このイベント処理の終了後に GATT データベースに値が書き込まれます。

### 4.2.5 Read 動作の実装

Read 動作では、クライアントがサーバーの GATT データベースのデータを読み出します。サーバーは読み出し要求を受け入れないこともできます。

Read 動作 API は、GATT データベースのデータサイズが(MTU - 1)以下場合には、Read 動作を行います。GATT データベースのデータサイズが MTU-1 より大きい場合には、Read Long 動作を行います。

図 4.33 に Read 動作を行う場合のシーケンスチャートを示します。赤字が関数呼び出し、青字がアプリケーション通知されるイベントです。

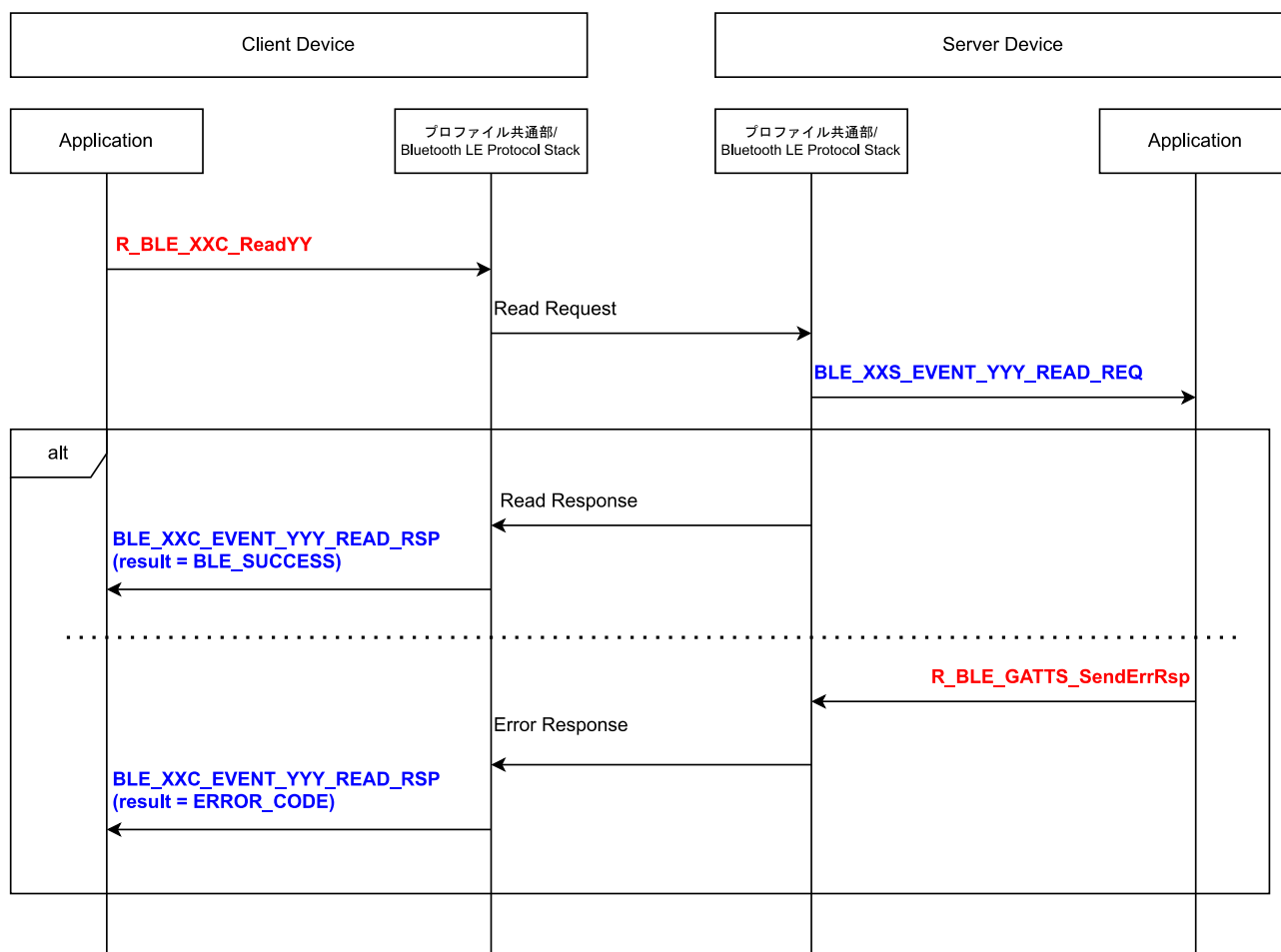


図 4.33 Read 動作時のシーケンスチャート

Read Long 動作では、キャラクターリスティックのデータを一度に送信できるサイズ毎にサーバーからクライアントに送信されます。

サーバーのアプリケーションでは、読み出し対象のキャラクターリスティックサイズ / (MTU - 1) (小数点切り上げ)回だけ読み出し要求イベントが発生します。

Read Long 動作中のシーケンスチャートを図 4.34 に示します。赤字が関数呼び出し、青字がアプリケーション通知されるイベントです。

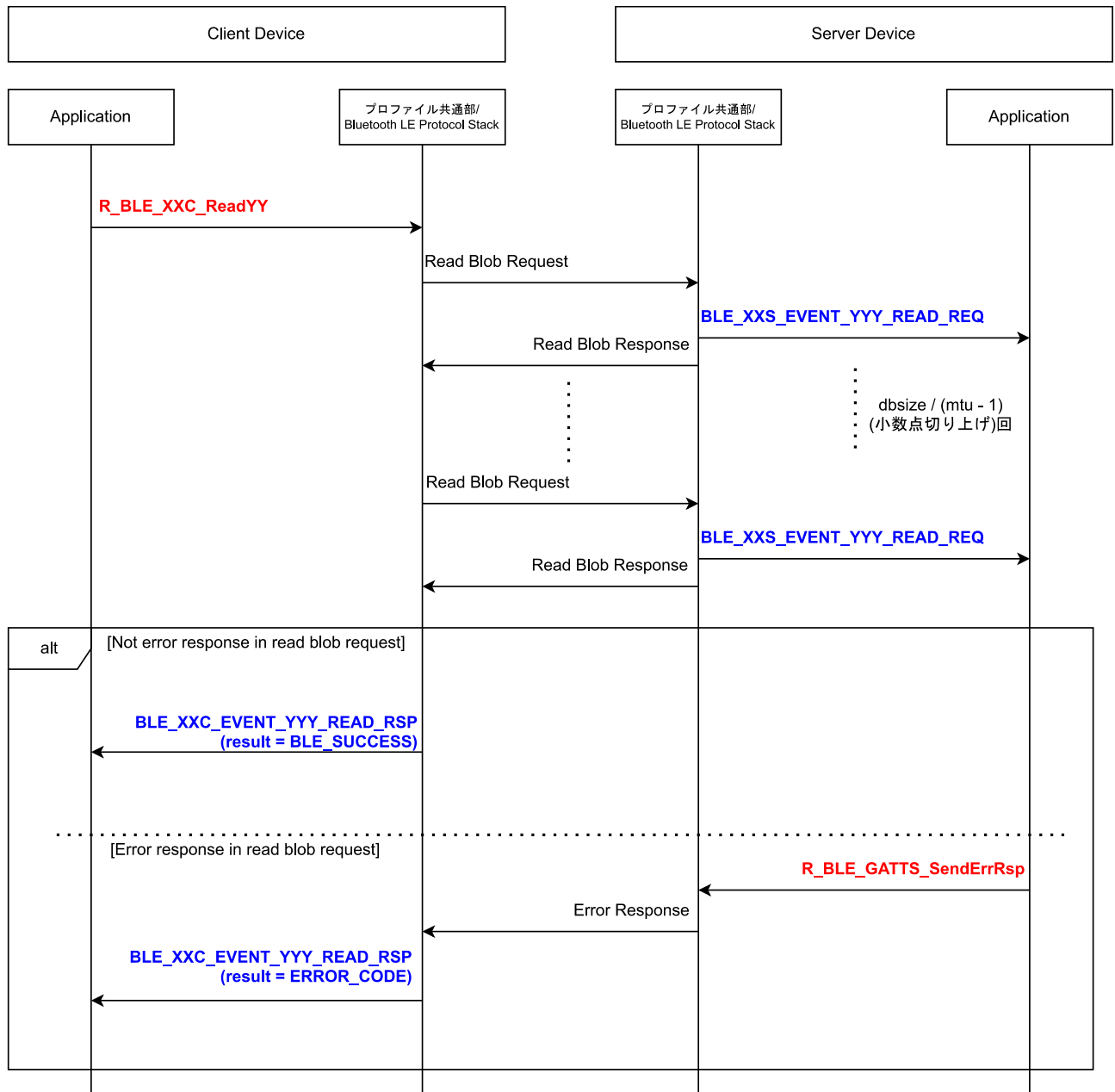


図 4.34 Read Long 動作のシーケンスチャート

## 4.2.5.1 クライアントの実装

## Read 動作 API

Read 動作は、クライアントから開始します。サービス API プログラムに実装されている Read 動作 API を使用します。キャラクタリスティックとディスクリプタ毎に実装されています。

引数はコネクションハンドルのみです。

```
ble_status_t R_BLE_XXC_ReadYyy(uint16_t conn_hdl);
ble_status_t R_BLE_XXC_ReadYyyZzz(uint16_t conn_hdl);
```

図 4.35 サービス API プログラム(r\_ble\_xxc.h)での Read 動作 API の定義

引数に渡したキャラクタリスティックの db\_size によって Read 動作を行うか Read Long 動作を行うかが決まります。表 4.12 に MTU と実行される動作の関係を示します。

表 4.12 db\_size と MTU と実行される動作の関係

dbsize と MTU	実行される動作
dbsize <= MTU-1	Read 動作
dbsize > MTU-1	Read Long 動作

キャラクタリスティックの db\_size は、図 4.36 のように r\_ble\_xxc.c ファイル内のキャラクタリスティック定義構造体に登録されている db\_size(黄色ハイライト)の値を参照してください。

```
static const st_ble_servc_char_info_t gs_yyy_char = {
    .uuid_128      = BLE_XXC_YYY_UUID,
    .uuid_type     = BLE_GATT_128_BIT_UUID_FORMAT,
    .app_size      = sizeof(st_ble_xxc_yyy_t),
    .db_size       = BLE_XXC_YYY_LEN,
    .char_idx      = BLE_XXC_YYY_IDX,
    .p_attr_hdls  = gs_yyy_char_ranges,
    .decode        = (ble_servc_attr_decode_t)decode_st_ble_xxc_yyy_t,
    .encode        = (ble_servc_attr_encode_t)encode_st_ble_xxc_yyy_t,
    .num_of_descs = ARRAY_SIZE(gspp_yyy_descs),
    .pp_descs     = gspp_yyy_descs,
};
```

図 4.36 r\_ble\_xxc.c ファイルのキャラクタリスティック定義構造体

**BLE\_XXC\_EVENT\_YYY\_READ\_RSP イベント**

Read 動作によるデータの読み出し結果は、BLE\_XXC\_EVENT\_YYY\_READ\_RSP イベントに通知されます。キャラクタースティックの構造体にキャストすることで、読み出した値を確認できます。

サーバーからエラーレスポンスを受け取った場合、プロファイル共通部でメモリの動的確保に失敗した場合、decode 関数の戻り値が BLE\_SUCCESS でなかった場合に result 変数にエラー内容が通知されます。

```
static void xxc_cb(uint16_t type, ble_status_t result, st_ble_servc_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_XXC_EVENT_YYY_READ_RSP:
        {
            If (result == BLE_SUCCESS)
            {
                st_ble_xxc_yyy_t *event_data = (st_ble_xxc_yyy_t *)p_data->p_param;

                /*Implement application process. */

            }
        } break;
    }
}
```

図 4.37 クライアントの Read 動作時のイベントの実装例



## 4.2.5.2 サーバーの実装

## BLE\_XXS\_EVENT\_YYY\_READ\_REQ イベント

クライアントから Read 動作による GATT データベースへ読み出しを受信した場合に通知されます。イベント処理後の GATT データベースの値がクライアントに送信されます。

クライアントの読み出しに対して任意のデータを渡す場合には、このイベントで GATT データベースを更新してください。GATT データベースの更新には、R\_BLE\_XXS\_SetYyy 関数を使用します。

```
static void xxs_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_XXS_EVENT_YYY_READ_REQ:
        {
            st_ble_xxs_yyy_t new_value;

            R_BLE_XXS_SetYyy(&new_value);
        } break;
    }
}
```

図 4.38 GATT データベースを更新する方法

一方で、クライアントからの読み出しを受け入れない場合には、R\_BLE\_GATTS\_SendErrRsp 関数を呼び、Error Response を送信します。

```
static void xxs_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_XXS_EVENT_YYY_READ_REQ:
        {
            bool is_readable = false;

            /* Application does not read GATT Database */
            if ( !is_readable )
            {
                /* Application can send Error Response */
                uint16_t error_code = 0x3081;
                R_BLE_GATTS_SendErrRsp(error_code);
            }
        } break;
    }
}
```

図 4.39 サーバーの Read 動作時のイベントの実装例

### 4.2.6 Notify 動作の実装

Notify 動作は、サーバーからクライアントにデータを送信します。

Notify 動作を行うために、クライアントはキャラクターリスティックの Client Configuration Characteristic Descriptor (CCCD)に Notify 動作を有効に設定します。

Notify 動作では、MTU-3 バイトより大きいデータは送信できません。

Notify 動作を行う場合のシーケンスチャートを示します。赤字が関数呼び出し、青字がアプリケーション通知されるイベントです。

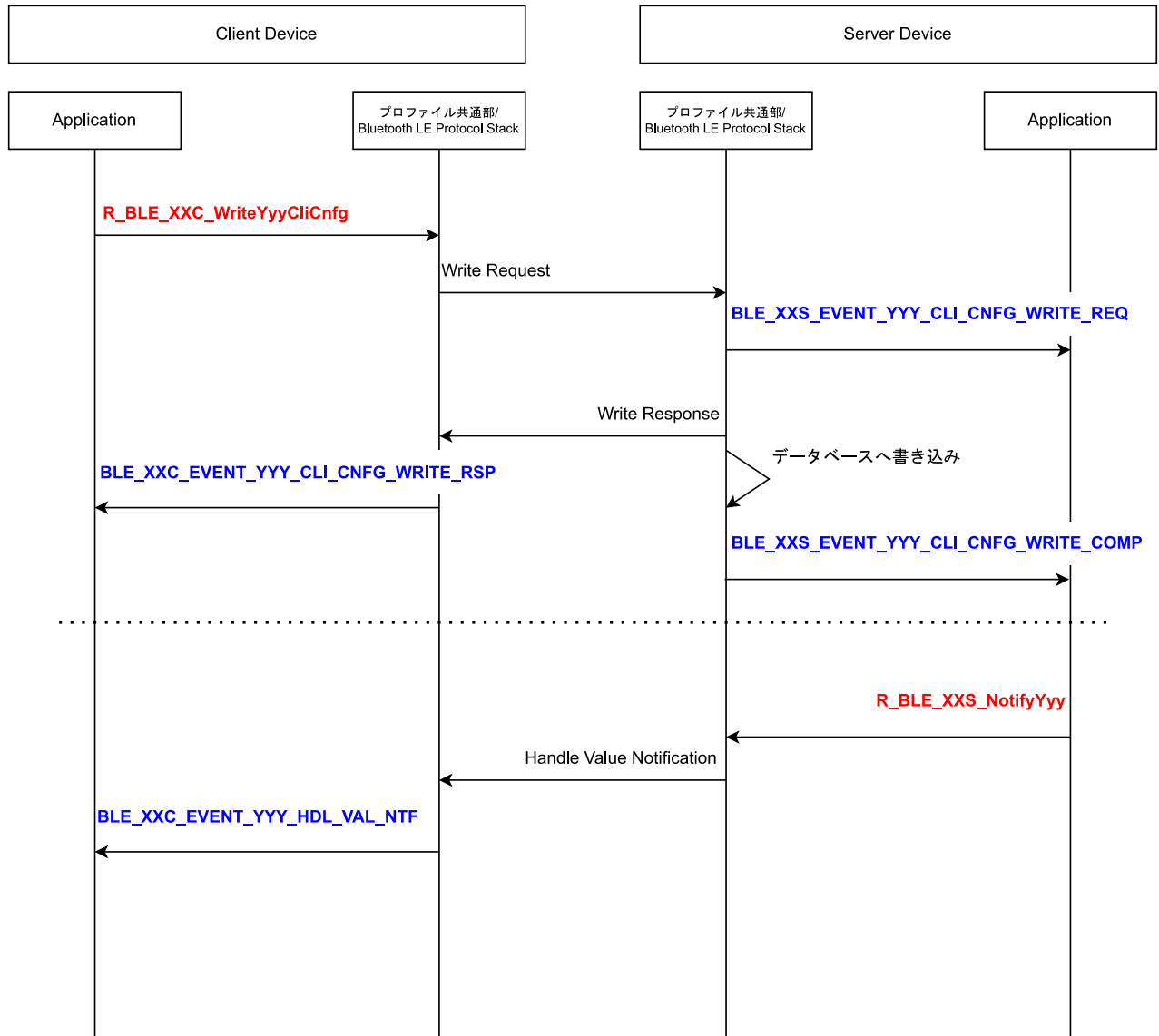


図 4.40 Notify 動作のシーケンスチャート

## 4.2.6.1 クライアントの実装

## CCCD への書き込み

まずは CCCD へ Write 動作を行います。CCCD は、16bit のビットフィールドであらわされます。Notify 動作を有効にするには、cccd に 0x0001 を書き込みます。Bluetooth LE Protocol Stack に BLE\_GATTS\_CLI\_CNFG\_NOTIFICATION でマクロ定義されています。

## 関数定義

```
ble_status_t R_BLE_XXC_WriteYyyCliCnfg(uint16_t conn_hdl, const uint16_t *p_value)
```

## 実装例

```
uint16_t cccd = BLE_GATTS_CLI_CNFG_NOTIFICATION;
R_BLE_XXC_WriteYyyCliCnfg(conn_hdl, &cccd);
```

図 4.41 サービス API プログラム(r\_ble\_xxs.h)での CCCD への Write 動作 API と実装例

## BLE\_XXC\_EVENT\_YYY\_HDL\_VAL\_NTF イベント

クライアントが Notify 動作を有効にすると、サーバーの任意のタイミングで Notify 動作によるデータ送信が行われます。データは BLE\_XXC\_EVENT\_YYY\_HDL\_VAL\_NTF に通知されます。

キャラクタリスティックの構造体にキャストすることで、受信した値を確認できます。

```
static void xxc_cb(uint16_t type, ble_status_t result, st_ble_servc_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_XXC_EVENT_YYY_HDL_VAL_NTF:
        {
            if( BLE_SUCCESS == result )
            {
                st_ble_xxc_yyy_t *event_data = (st_ble_xxc_yyy_t)p_data->p_param;
                /*Implement application process. */
            }
            } break;
    }
}
```

図 4.42 クライアントの Notify 動作時のイベントの実装例

## 4.2.6.2 サーバーの実装

## BLE\_XXS\_EVENT\_YYY\_CLI\_CNFG\_WRITE\_COMP イベント

クライアントが CCCD に書き込み完了後にこのイベントが発生します。CCCD への書き込み内容が通知されます。

以下の実装例では、CCCD の Notify の設定を確認し後述の Notify 動作 API を使用してクライアントにデータを送信しています。

```
static void xxs_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_XXS_EVENT_YYY_CLI_CNFG_WRITE_COMP:
        {
            uint16_t cccd = *(uint16_t *)p_data->p_param;

            if( (cccd & BLE_GATTS_CLI_CNFG_NOTIFICATION) == BLE_GATTS_CLI_CNFG_NOTIFICATION)
            {
                st_ble_xxs_yyy_t notify_value;
                R_BLE_XXS_NotifyYyy(p_data->conn_hdl, &notify_value);
            }
        } break;
    }
}
```

図 4.43 CCCD の書き込み完了イベントでの Notify の開始

## Notify 動作 API

Notify 動作は、CCCD の Notify が有効になっている場合にいつでもサーバーから送信できます。Notify 動作を行う API は以下の通りです。Notify 動作は、キャラクタリスティックのみ定義されています。

引数はコネクションハンドルと Notify 対象のキャラクタリスティックの値です。

```
ble_status_t R_BLE_XXS_NotifyYyy(uint16_t conn_hdl, const st_ble_xxs_yyy_t *p_value);
```

図 4.44 サービス API プログラム(r\_ble\_xxs.h)での Notify 動作 API の定義

CCCD の Notify が有効になっていない場合には、戻り値が BLE\_ERR\_INVALID\_OPERATION(0x0009)となり、Notify によるデータ送信は行われません。

API の呼び出し以降に、サーバーに通知されるイベントはありません。

### 4.2.7 Indicate 動作の実装

Indicate 動作は、サーバーからクライアントにデータを送信します。Indicate 動作では、クライアントからサーバーに、データを受信したことを伝える Handle Value Confirmation が送信されます。サーバーからクライアントへのデータ送信を確認しながら送信する場合に便利です。

Indicate 動作を行うために、クライアントはキャラクターリスティックの Client Configuration Characteristic Descriptor (CCCD)に Indicate 動作を有効に設定します。

Indicate 動作では、MTU-3 バイトより大きいデータは送信できません。

Indicate 動作を行う場合のシーケンスチャートを示します。赤字が関数呼び出し、青字がアプリケーション通知されるイベントです。



図 4.45 Indicate 動作のシーケンスチャート

## 4.2.7.1 クライアントの実装

## CCCD への書き込み

まずは CCCD へ Write 動作を行います。CCCD は、16bit のビットフィールドであらわされます。Indicate 動作を有効にするには、cccd に 0x0002 を書き込みます。Bluetooth LE Protocol Stack に BLE\_GATTS\_CLI\_CNFG\_INDICATION でマクロ定義されています。

## 関数定義

```
ble_status_t R_BLE_XXC_WriteYyyCliCnfg(uint16_t conn_hdl, const uint16_t *p_value)
```

## 実装例

```
uint16_t cccd = BLE_GATTS_CLI_CNFG_INDICATION;
R_BLE_XXC_WriteYyyCliCnfg(conn_hdl, &cccd);
```

図 4.46 サービス API プログラム(r\_ble\_xxs.h)での CCCD への Write 動作 API と実装例

## BLE\_XXS\_EVENT\_YYY\_HDL\_VAL\_IND イベント

クライアントが Indicate 動作を有効にすると、サーバーの任意のタイミングで Indicate 動作によるデータ送信が行われます。データは BLE\_XXS\_EVENT\_YYY\_HDL\_VAL\_IND に通知されます。

キャラクタリスティックの構造体にキャストすることで、受信した値を確認できます。

```
static void xxc_cb(uint16_t type, ble_status_t result, st_ble_servc_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_XXC_EVENT_YYY_HDL_VAL_IND:
        {
            if( BLE_SUCCESS == result )
            {
                st_ble_xxc_yyy_t *event_data = (st_ble_xxc_yyy_t)p_data->p_param;
                /*Implement application process. */
            }
        } break;
    }
}
```

図 4.47 クライアントの Indicate 動作時のイベントの実装例

受信確認通知の Handle Value Confirmation は、Bluetooth LE Protocol Stack が自動で応答します。

## 4.2.7.2 サーバーの実装

## BLE\_XXS\_EVENT\_YYY\_CLI\_CNFG\_WRITE\_COMP イベント

クライアントが CCCD に書き込み完了後にこのイベントが発生します。CCCD への書き込み内容が通知されます。

以下の実装例では、CCCD の Indicate の設定を確認し後述の Indicate 動作 API を使用してクライアントにデータを送信しています。

```
static void xxs_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_XXS_EVENT_YYY_CLI_CNFG_WRITE_COMP:
        {
            uint16_t cccd = *(uint16_t *)p_data->p_param;

            if( (cccd & BLE_GATTS_CLI_CNFG_INDICATION) == BLE_GATTS_CLI_CNFG_INDICATION)
            {
                st_ble_xxs_yyy_t notify_value;
                R_BLE_XXS_IndicateYyy(p_data->conn_hdl, &notify_value);
            }
        } break;
    }
}
```

図 4.48 CCCD の書き込み完了イベントでの Indicate の開始

## Indicate 動作 API

Indicate 動作は、CCCD の Indicate が有効になっている場合にサーバーから送信できます。Indicate 動作を行う API は以下の通りです。Indicate 動作は、キャラクタリスティックのみ定義されています。

引数は接続ハンドルと Indicate 対象のキャラクタリスティックの値です。

CCCD の Indicate が有効になっていない場合には、戻り値が BLE\_ERR\_INVALID\_OPERATION(0x0009) となり、Indicate によるデータ送信は行われません。

```
ble_status_t R_BLE_XXS_IndicateYyy(uint16_t conn_hdl, const st_ble_xxs_yyy_t *p_value);
```

図 4.49 サービス API プログラム(r\_ble\_xxc.h)での Indicate 動作 API の定義

Indicate 動作 API は、連続して呼び出せません。BLE\_XXS\_EVENT\_YYY\_HDL\_VAL\_CNF イベントの受信後に再度呼び出せます。

## BLE\_XXS\_EVENT\_YYY\_HDL\_VAL\_CNF イベント

Indicate 動作によってサーバーからクライアントにデータが送信されると、クライアントはサーバーに受信確認通知を送信します。このイベントは、BLE\_XXS\_EVENT\_YYY\_HDL\_VAL\_CNF イベントとして通知されます。

サーバーは、この受信確認によってクライアントにデータが届いたことを確認できます。

```
static void xxc_cb(uint16_t type, ble_status_t result, st_ble_servc_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_XXC_EVENT_YYY_HDL_VAL_IND:
        {
            if( BLE_SUCCESS == result )
            {
                st_ble_xxc_yyy_t *event_data = (st_ble_xxc_yyy_t)p_data->p_param;
                /*Implement application process. */
            }
        } break;
    }
}
```

図 4.50 サーバーの Indicate 動作時のイベントの実装例



### 4.3 GATT データベース (gatt\_db.c / gatt\_db.h)

「サーバー」として使用するサービスから構成される GATT データベースが実装されています。変更する必要はありません。

gatt\_db.h ファイルには、アトリビュートハンドルとキャラクタースティックとディスクリプタのアトリビュートバリューのサイズがマクロ形式で展開されています。

```
typedef enum
{
    BLE_INVALID_ATTR_HDL = 0x0000,
    BLE_GAPS_DECL_HDL = 0x0001,
    BLE_GAPS_DEV_NAME_DECL_HDL = 0x0002,
    BLE_GAPS_DEV_NAME_VAL_HDL = 0x0003,
    BLE_GAPS_APPEARANCE_DECL_HDL = 0x0004,
    BLE_GAPS_APPEARANCE_VAL_HDL = 0x0005,
    BLE_GAPS_PER_PREF_CONN_PARAM_DECL_HDL = 0x0006,
    BLE_GAPS_PER_PREF_CONN_PARAM_VAL_HDL = 0x0007,
    BLE_GAPS_CENT_ADDR_RSLV_DECL_HDL = 0x0008,
    BLE_GAPS_CENT_ADDR_RSLV_VAL_HDL = 0x0009,
    BLE_GAPS_RSLV_PRIV_ADDR_ONLY_DECL_HDL = 0x000A,
    BLE_GAPS_RSLV_PRIV_ADDR_ONLY_VAL_HDL = 0x000B,
    BLE_GATS_DECL_HDL = 0x000C,
    BLE_GATS_SERV_CHGED_DECL_HDL = 0x000D,
    BLE_GATS_SERV_CHGED_VAL_HDL = 0x000E,
    BLE_GATS_SERV_CHGED_CLI_CNFG_DESC_HDL = 0x000F,
    BLE_XXS_DECL_HDL = 0x0010,
    BLE_XXS_YYY_DECL_HDL = 0x0011,
    BLE_XXS_YYY_VAL_HDL = 0x0012,
    BLE_XXS_YYY_CLI_CNFG_DESC_HDL = 0x0013,
    BLE_XXS_YYY_ZZZ_DESC_HDL = 0x0014,
} e_ble_attr_hdl_t;

#define BLE_GAPS_DEV_NAME_LEN (128)
#define BLE_GAPS_APPEARANCE_LEN (2)
#define BLE_GAPS_PER_PREF_CONN_PARAM_LEN (8)
#define BLE_GAPS_CENT_ADDR_RSLV_LEN (1)
#define BLE_GAPS_RSLV_PRIV_ADDR_ONLY_LEN (1)
#define BLE_GATS_SERV_CHGED_LEN (4)
#define BLE_GATS_SERV_CHGED_CLI_CNFG_LEN (2)
#define BLE_XXS_YYY_LEN (4)
#define BLE_XXS_YYY_CLI_CNFG_LEN (2)
#define BLE_XXS_YYY_ZZZ_LEN (2)
```

図 4.51 GATT データベースのマクロ定義

gatt\_db.c ファイルには、Bluetooth LE Protocol Stack の仕様に則った GATT データベースが実装されています。また、QE for BLE で設計したプロファイルのサービス一覧を実装されている GATT データベースとしてコメント形式で表示しています。

```

/**
 * GATT DATABASE QUICK REFERENCE TABLE:
 * Abbreviations used for PROPERTIES:
 *   BC = Broadcast
 *   RD = Read
 *   WW = Write Without Response
 *   WR = Write
 *   NT = Notification
 *   IN = Indication
 *   RW = Reliable Write
 *
 * HANDLE | ATT_TYPE | PROPERTIES | ATT_VALUE | DEFINITION
 * =====
 * GAP Service
 * =====
 * 0x0001 | 0x28,0x00 | RD | 0x00,0x18 | GAP Service Declaration
 * -----+-----+-----+-----+-----
 * 0x0002 | 0x28,0x03 | RD | 0x0A,0x03,0x00,0x00,0x2A | Device Name characteristic...
 * -----+-----+-----+-----+-----
 * 0x0003 | 0x00,0x2A | RD,WR | 0x00,0x00,0x00,0x00,0x00... | Device Name characteristic ...
 * -----+-----+-----+-----+-----
 * 0x0004 | 0x28,0x03 | RD | 0x02,0x05,0x00,0x01,0x2A | Appearance characteristic ...
 * -----+-----+-----+-----+-----
 * 0x0005 | 0x01,0x2A | RD | 0x00,0x00 | Appearance characteristic ...
 * -----+-----+-----+-----+-----
 * 0x0006 | 0x28,0x03 | RD | 0x02,0x07,0x00,0x04,0x2A | Peripheral Preferred ...
 * -----+-----+-----+-----+-----
 * 0x0007 | 0x04,0x2A | RD | 0x00,0x00,0x00,0x00,0x00,0x00... | Peripheral Preferred ...
 * -----+-----+-----+-----+-----
 * 0x0008 | 0x28,0x03 | RD | 0x02,0x09,0x00,0xA6,0x2A | Central Address Resolution ...
 * -----+-----+-----+-----+-----
 * 0x0009 | 0xA6,0x2A | RD | 0x00 | Central Address Resolution ...
 * -----+-----+-----+-----+-----
 * 0x000A | 0x28,0x03 | RD | 0x02,0x0B,0x00,0xC9,0x2A | Resolvable Private Address ...
 * -----+-----+-----+-----+-----
 * 0x000B | 0xC9,0x2A | RD | 0x00 | Resolvable Private Address ...
 * =====
 * GATT Service
 * =====
 * 0x000C | 0x28,0x00 | RD | 0x01,0x18 | GATT Service Declaration
 * -----+-----+-----+-----+-----
 * 0x000D | 0x28,0x03 | RD | 0x20,0x0E,0x00,0x05,0x2A | Service Changed ...
 * -----+-----+-----+-----+-----
 * 0x000E | 0x05,0x2A | IN | 0x00,0x00,0x00,0x00 | Service Changed ...
 * -----+-----+-----+-----+-----
 * 0x000F | 0x02,0x29 | RD,WR | 0x00,0x00 | Client Characteristic ...
 * =====
 * prof dev xx Service
 * =====
 * 0x0010 | 0x28,0x00 | RD | 0x50,0x44,0x02,0xb7,0xaf,0xf8... | prof dev xx Service ...
 * -----+-----+-----+-----+-----
 * 0x0011 | 0x28,0x03 | RD | 0x3E,0x12,0x00,0x2d,0x1f,0x35... | prof dev yyy char...
 * -----+-----+-----+-----+-----
 * 0x0012 | 0x2d,0x1f,0x35... | RD,WW,WR... | 0x00,0x00,0x00,0x00 | prof dev yyy char value...
 * -----+-----+-----+-----+-----
 * 0x0013 | 0x02,0x29 | RD,WR | 0x00,0x00 | Client Characteristic...
 * -----+-----+-----+-----+-----
 * 0x0014 | 0xb9,0xfd,0x08... | RD,WR | 0x00,0x00 | prof dev zzz Descriptor ...
 * =====
 */

```

図 4.52 gatt\_db.c ファイルの GATT データベースの構造コメント例

## 5. プログラムのビルドと実行

本章では、MCU 毎に QE for BLE から生成されたコードをビルドする場合の注意点を説明します。

e<sup>2</sup> studio でプロジェクトをビルド・デバッグする場合は「e<sup>2</sup> studio ユーザーズマニュアル 入門ガイド (R20UT4204)」を参照してください。

### 5.1 RX23W

新規プロジェクトを作成した場合には、QE for BLE から生成したコードは、設定を変更せずに実行できます。

バージョン 2.50 以上の BLE FIT モジュールとバージョン 1.50 以前の QE for BLE の組み合わせで新規プロジェクトを作成した場合、プロファイル共通部がプロジェクトに追加されません。以下のサイトを参考に、QE for BLE を最新の環境にアップデートしてください。

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

バージョン 2.30 以前の BLE FIT モジュールに同梱されているサンプルプロジェクトをベースに開発した場合にファイルが競合する可能性があります。

プロジェクト内に以下のフォルダが存在する場合は削除をお願いいたします。

- src/smc\_gen/Config\_BLE\_PROFILE
- src/smc\_gen/r\_ble\_qe\_utility

#### 5.1.1 QE for BLE の移行方法

3.1 章の手順で、図 5.1 が表示される場合には、本節で説明するようにデータ移行を行ってください。

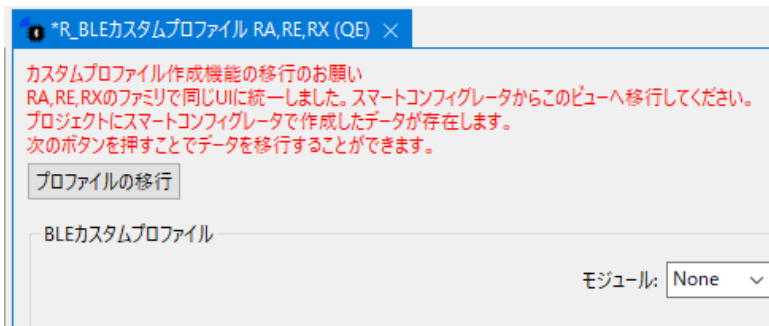


図 5.1 QE for BLE の移行のお願い

スマートコンフィグレータのデータ移行とコンポーネントの削除およびコード生成を行います。データの移行は自動的に行われます。コンポーネントの削除はポップアップするプロファイルの 移行ダイアログの指示に従って操作してください。

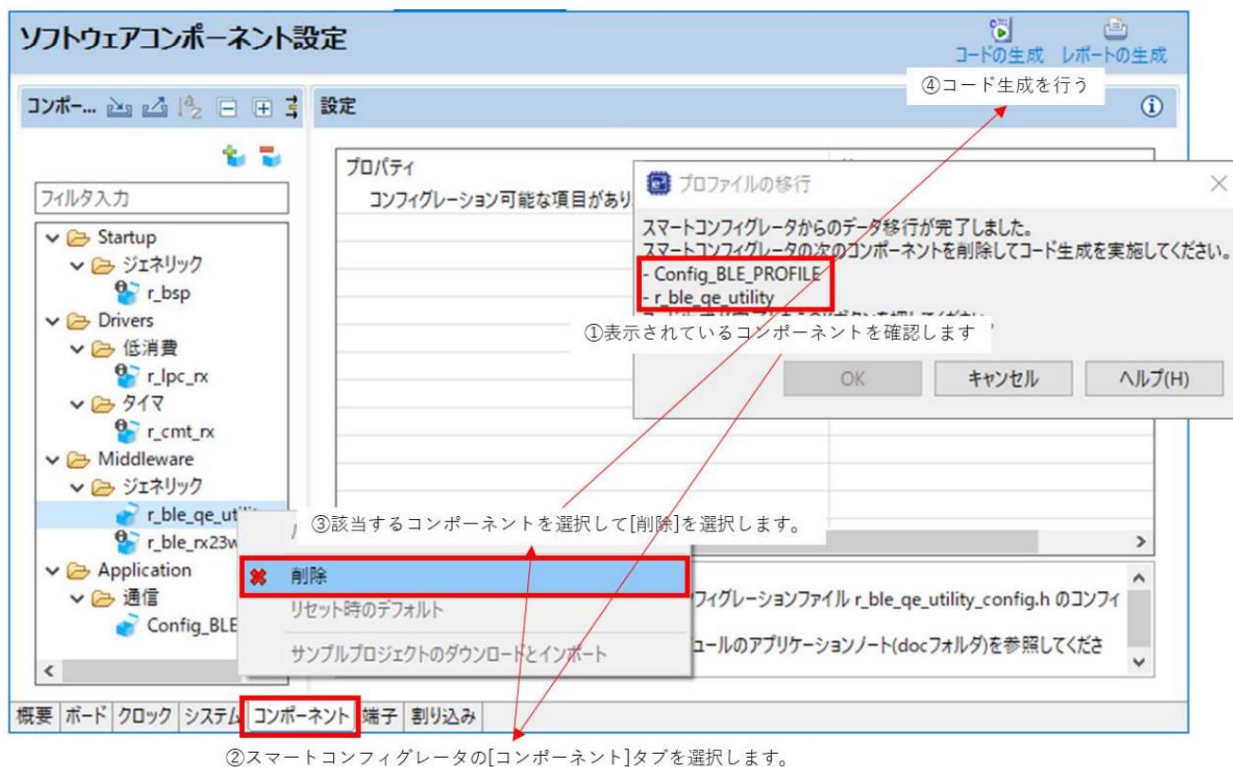


図 5.2 コンポーネントの削除手順

下図のメッセージが表示されたら移行完了です。

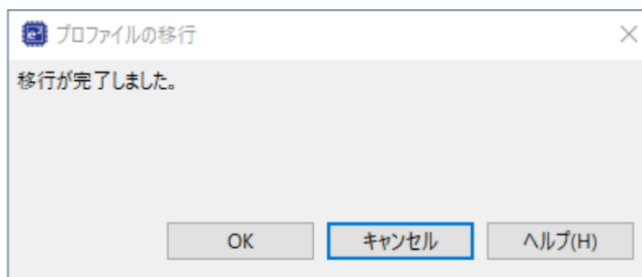


図 5.3 移行完了メッセージ

最後に、RX 向けプラグインのアンインストール QE for BLE[RX] V1.0.0 もしくは V1.1.0 は使用しませんのでアンインストールを行います。

1. [ヘルプ(H)]→[e<sup>2</sup> studio について(A)]メニューを選択し、[e<sup>2</sup> studio について]ダイアログを開きます。
2. [インストール詳細]ボタンを押下し、[e<sup>2</sup> studio のインストール詳細]ダイアログを開きます。
3. [インストールされたソフトウェア]タブに表示されている[QE for BLE[RX]]を選択し、[アンインストール(U)]ボタンを押下して、[アンインストール]ダイアログを開きます。
4. 表示された内容を確認し、[終了(F)]ボタンを押下する。e<sup>2</sup> studio の再起動を促されるので再起動を行います。

**注意事項**

- スマートコンフィグレータで削除したコンポーネントは再度追加しないでください。
- プロファイルの移行時に次のダイアログが表示される場合は、[R\_BLE カスタムプロファイル RA,RE,RX(QE)]でコード生成したデータをスマートコンフィグレータのコード生成で上書きします。

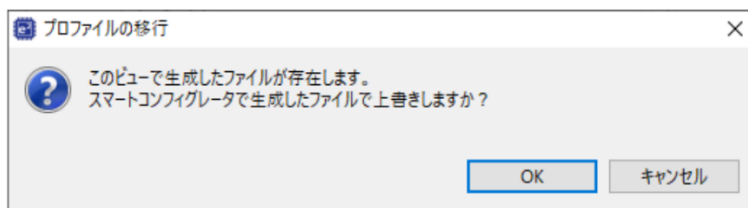


図 5.4 上書きの確認画面

## 5.2 RA4W1

QE for BLE から生成されたコードを実行する場合には、hal\_entry.c ファイルで app\_main 関数を実行します。以下に実装例を示します。

```
extern void app_main(void);

void hal_entry(void) {
    /* TODO: add your own code here */
    app_main();
}
```

図 5.5 hal\_entry.c での app\_main 関数の呼び出し

また、FreeRTOS、AzureRTOS を使用するプロジェクトをベースにプロファイル開発を行っている場合、QE for BLE から生成されたコードをビルドすると以下のビルドエラーが発生する可能性があります。

- Error [Pe020]: identifier "g\_ble\_abs0\_ctrl" is undefined.
- Error [Pe020]: identifier "g\_ble\_abs0\_cfg" is undefined.

これは、抽象 API モジュールの設定変数の外部参照宣言が行われていないことに起因します。

これらの宣言は、Bluetooth LE Protocol Stack の含むタスクのヘッダファイルで記述されています。サンプルプロジェクトでは ble\_core\_task です。

このヘッダファイルを app\_main.c ファイルでインクルードしてください。

```
#include "ble_core_task.h"
```

図 5.6 Bluetooth LE タスクのヘッダファイルのインクルード例 (サンプルプロジェクトの場合)

### 5.3 RE01B

Bluetooth LE 通信プロジェクトをベースにして QE for BLE から生成したコードを使用する場合は、設定を変更せずに実行できます。

## 6. その他の実装例

### 6.1 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ー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。

### 6.2 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。

1. 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。
2. 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。同様にキャラクタースティックやディスクリプタの[略称]も他のキャラクター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 6.1 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。

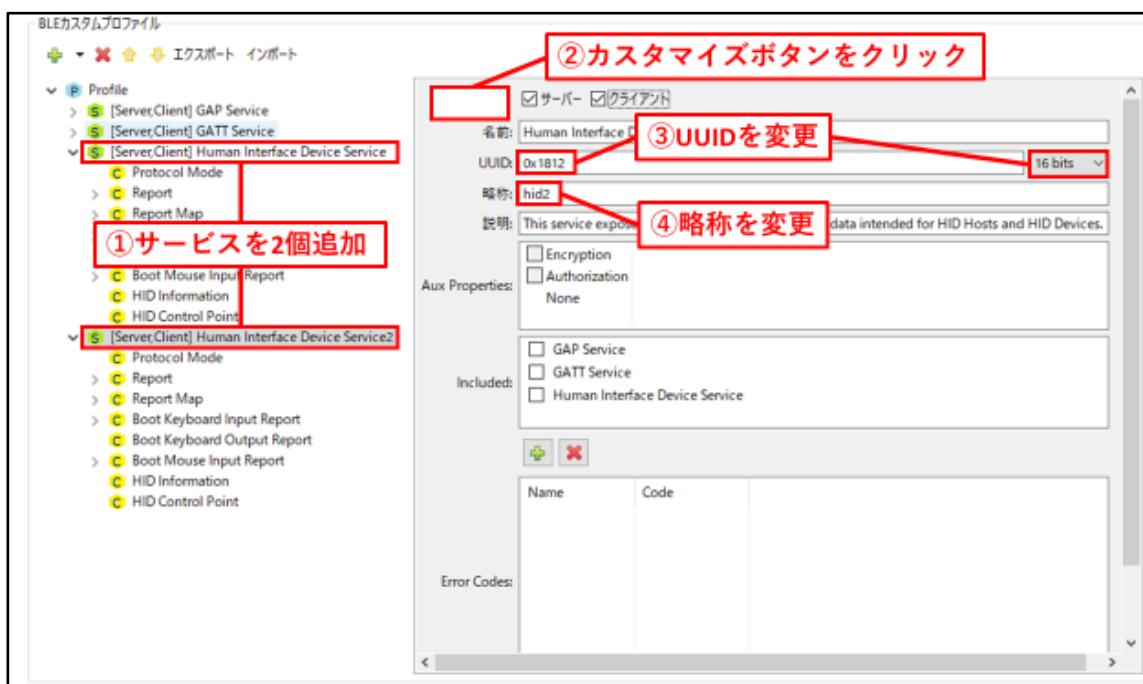


図 6.1 複数サービスの設定(QE for BLE のサービス設定画面)



カスタムサービスに変更した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異なりますが、多くの場合、以下の処理を追加します。

1. encode/decode 関数の中身を実装します。キャラクター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。
2. 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がキャラクター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があります。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 6.2 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 HIDS_UUID[] = { 0x12, 0x18 }; //HIDS specific service UUID
/* Human Interface Device Service2 UUID */
static uint8_t HIDS2C_UUID[] = { 0x12, 0x18 }; //Same service UUID

/* Service discovery parameters */
static st_ble_disc_entry_t gs_disc_entries[] = {
    {
        .p_uuid      = HIDS_UUID,
        .uuid_type   = BLE_GATT_16_BIT_UUID_FORMAT,
        .serv_cb     = R_BLE_HIDS_ServDiscCb,
        /* Add member [idx] */
        .idx         = 0, /* Set index number if service UUID is same */
    },
    {
        .p_uuid      = HIDS2C_UUID,
        .uuid_type   = BLE_GATT_16_BIT_UUID_FORMAT,
        .serv_cb     = R_BLE_HIDS2C_ServDiscCb,
        /* Add member [idx] */
        .idx         = 1, /* Set index number if service UUID is same */
    },
};
```

図 6.2 HIDS を 2 つ実装した場合のプログラムの実装例

### 6.3 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、サーバー側とクライアント側で異なります。

#### サーバー側の実装

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。

1. 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。
2. [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 6.3 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 6.3 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。

1. 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。
2. [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 6.4 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する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生成前にあらかじめ QE for BLE で設定してください。

```
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 6.4 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、プロファイル共通部の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。「6.4 インクルードサービスに対する discovery 動作を実装」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、Bluetooth LE 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、各 MCU 以下のドキュメントをご覧ください

表 6.1 GATT Client API に関するドキュメント

MCU	ドキュメント
RX23W	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)」
RA4W1	[RA Flexible Software Package Documentation].
RE01B	「Bluetooth Low Energy サンプルコード (R01AN5606)」に付属する「R_BLE API ドキュメント(r_ble_api_spec.chm)」

## 6.4 インクルードサービスに対する discovery 動作を実装する場合

## 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、プロファイル共通部の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 6.5 から図 6.6 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 6.5 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 6.6 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 6.7 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(
                    conn_hdl, /* Connection handle */
                    0, /* idx */
                    BLE_DISC_PRIM_SERV_FOUND, /* Notify as a primary service */
                    &serv_param); /* Service handle information */
            }
        }
    }
}
/* Generated code */
}
```

図 6.7 インクルードサービスのディスカバリの実装

## 6.5 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。通信頻度を変更するパラメータには以下があります。

### 1. コネクションインターバル

- データパケットを出すタイミングである接続イベントの間隔を設定します。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

### 2. ペリフェラルレイテンシ

- ペリフェラルは送信するデータがなければ設定された値の回数まで接続イベントでの応答を連続してスキップできます。
- 変数 : conn\_latency

### 3. スーパービジョンタイムアウト

- パケットを受信できなかった接続イベントが連続し、この時間を経過した場合、コネクションが切断されます。
- 変数 : sup\_to

図 6.8 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)
{
    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 6.8 コネクションアップデート関数の利用例

コネクションアップデートによるコネクションパラメータの変更は、要求先のデバイスが受け入れた場合に行われます。コネクションアップデートによるコネクションパラメータの変更についてはアプリケーション開発者ガイド「コネクションパラメータの更新」をご覧ください。



## 6.6 2つのMCUを接続してデータ通信を行う場合

QE for BLE のアプリケーションフレームワークを用いて2つのプロジェクトを接続してデータ通信を行う場合のプロファイル設計例を示します。

二つのプロジェクトを接続して通信する場合には、以下の設定が必要です。

- アドバタイズデータにスキャンフィルターデータが含まれている事
- 同一のプロファイルをサポートしている事

本章では、以下のプロジェクトを使用して、2つのプロジェクトを接続する場合の設計例を示します。

- セントラル(GATT クライアント) : prof\_dev\_central
- ペリフェラル(GATT サーバー) : prof\_dev\_peripheral

QE for BLE の「Profile」タブの GAP ロールを、それぞれセントラルとペリフェラルに設定します。

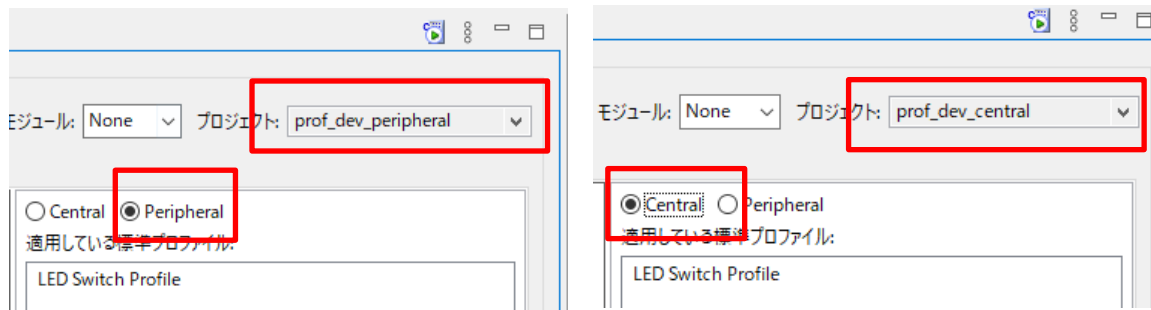


図 6.9 GAP ロールの設定例

次に、データ通信を行うプロファイルを設定します。サービスのインポート/エクスポート機能を利用して同一のサービスをそれぞれ追加します。今回はセントラルロールをクライアント、ペリフェラルロールをサーバーにします。使用するサービスの UUID が一致するようにします。

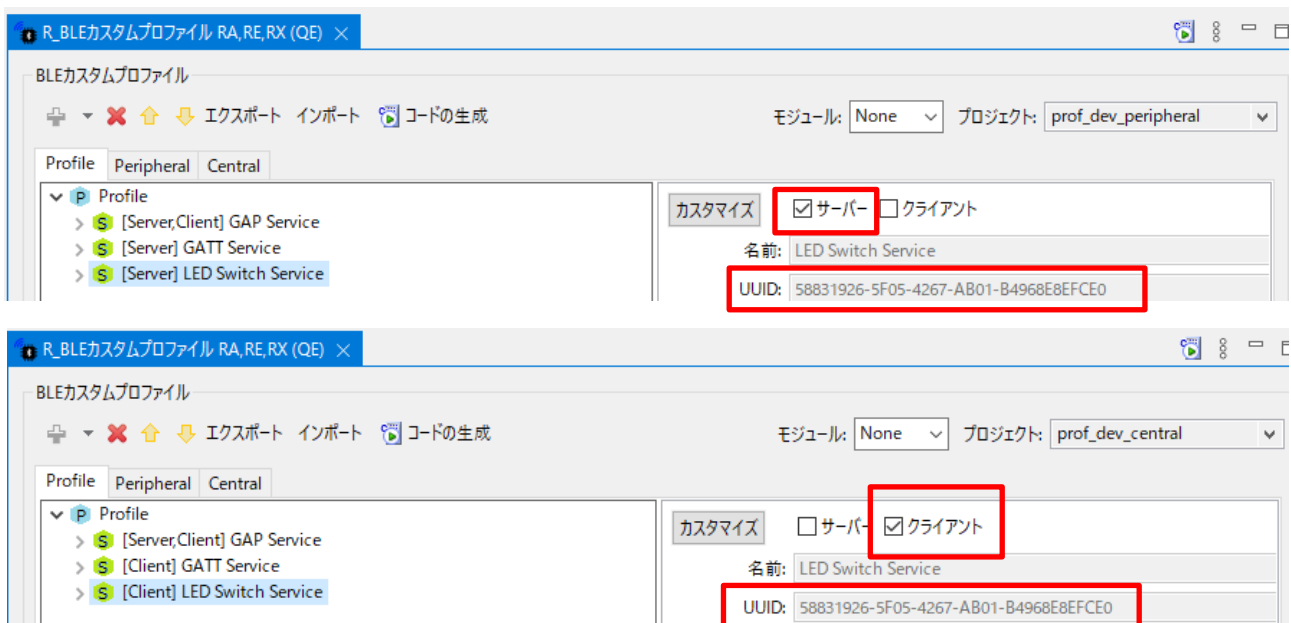


図 6.10 プロファイルの設定例

最後にそれぞれの Advertise Data と Scan Filter Data を設定します。セントラルは、Scan Filter Data に指定されたデータを持つアドバタイズを受信した場合に接続を行います。この値を一致させることで、二つのプロジェクトを接続できます。Advertise Data に「Local Name」を使用した場合の設定例を示します。

ペリフェラルロールでは、「Peripheral」タブから Advertise Data を設定します。

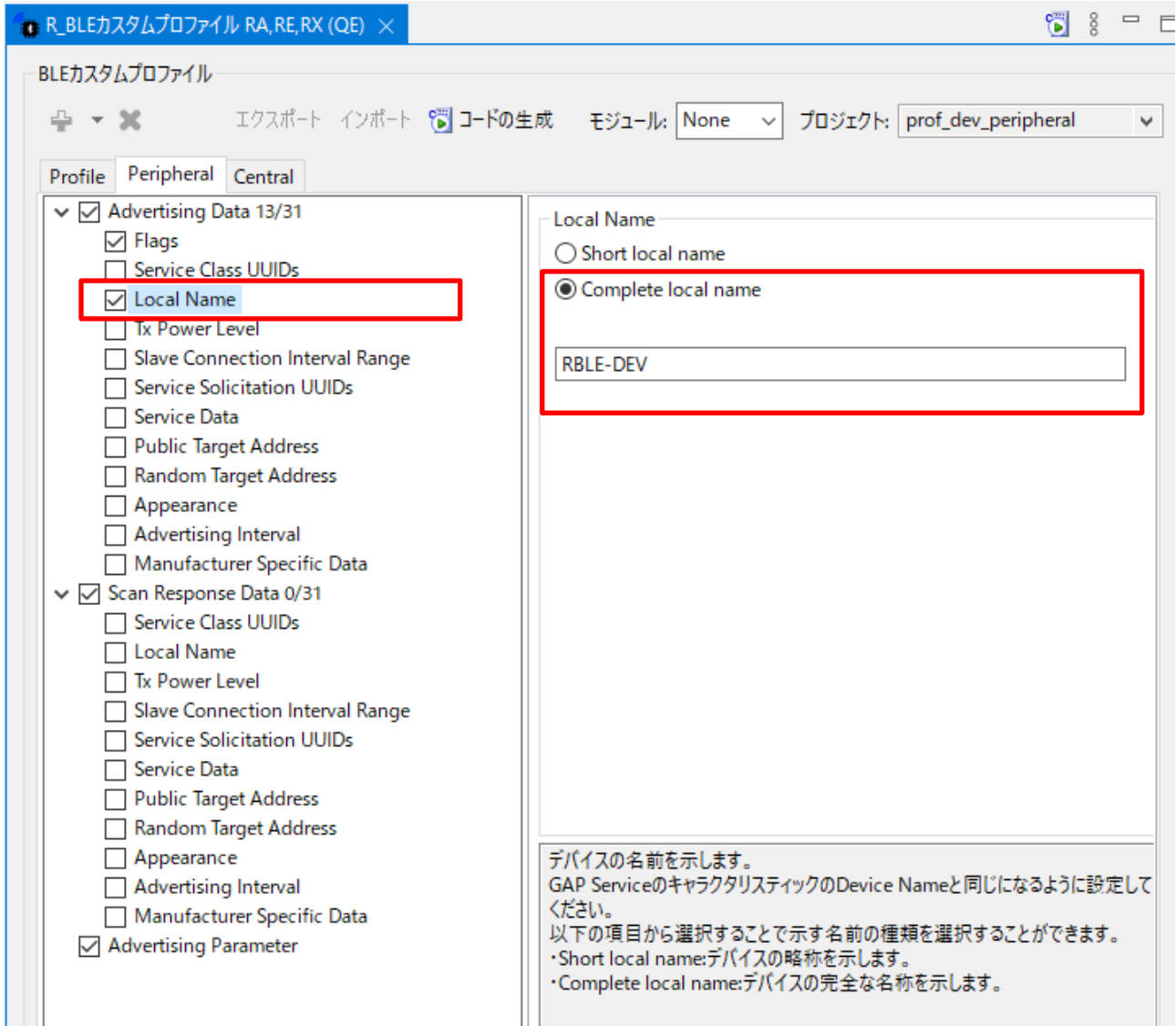


図 6.11 Advertise Data の設定例

セントラルロールでは、「Central」タブから Scan Filter Data を設定します。ペリフェラルの Advertise Data に設定したものと同様のデータにチェックをし、値を設定します。

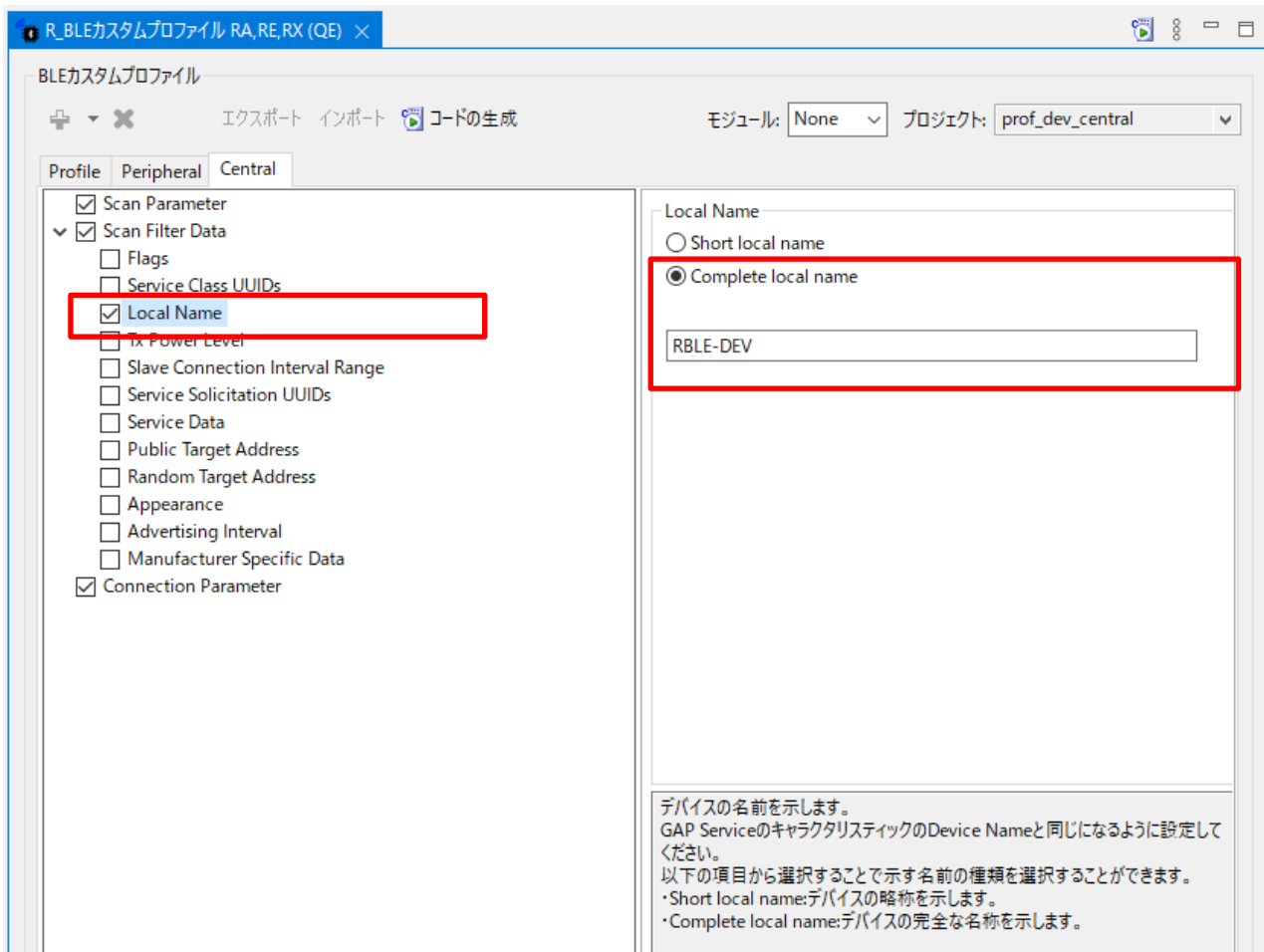


図 6.12 Scan Filter Data の設定例

以上で、二つのプロジェクトを接続するために必要な設定は完了です。

上記の設定後に接続できない場合には、アドバタイズ間隔とスキャン間隔を見直します。

QE for BLE の初期設定では、Advertising 動作と Scan 動作は「Slow」の設定を使用します。この設定では消費電力が抑えられますが、デバイスの検知が難しくなります。アプリケーションが素早い接続を求める場合には、「Fast」の設定を使用します。

ペリフェラルロールの「Fast」の設定例を示します。「Advertising Parameter」の設定を行います。「Fast」の設定を使用する場合は、「Enable Fast Advertising」をチェックします。本設定では起動から 30 秒間、30msec 毎にアドバタイズパケットを送信します。

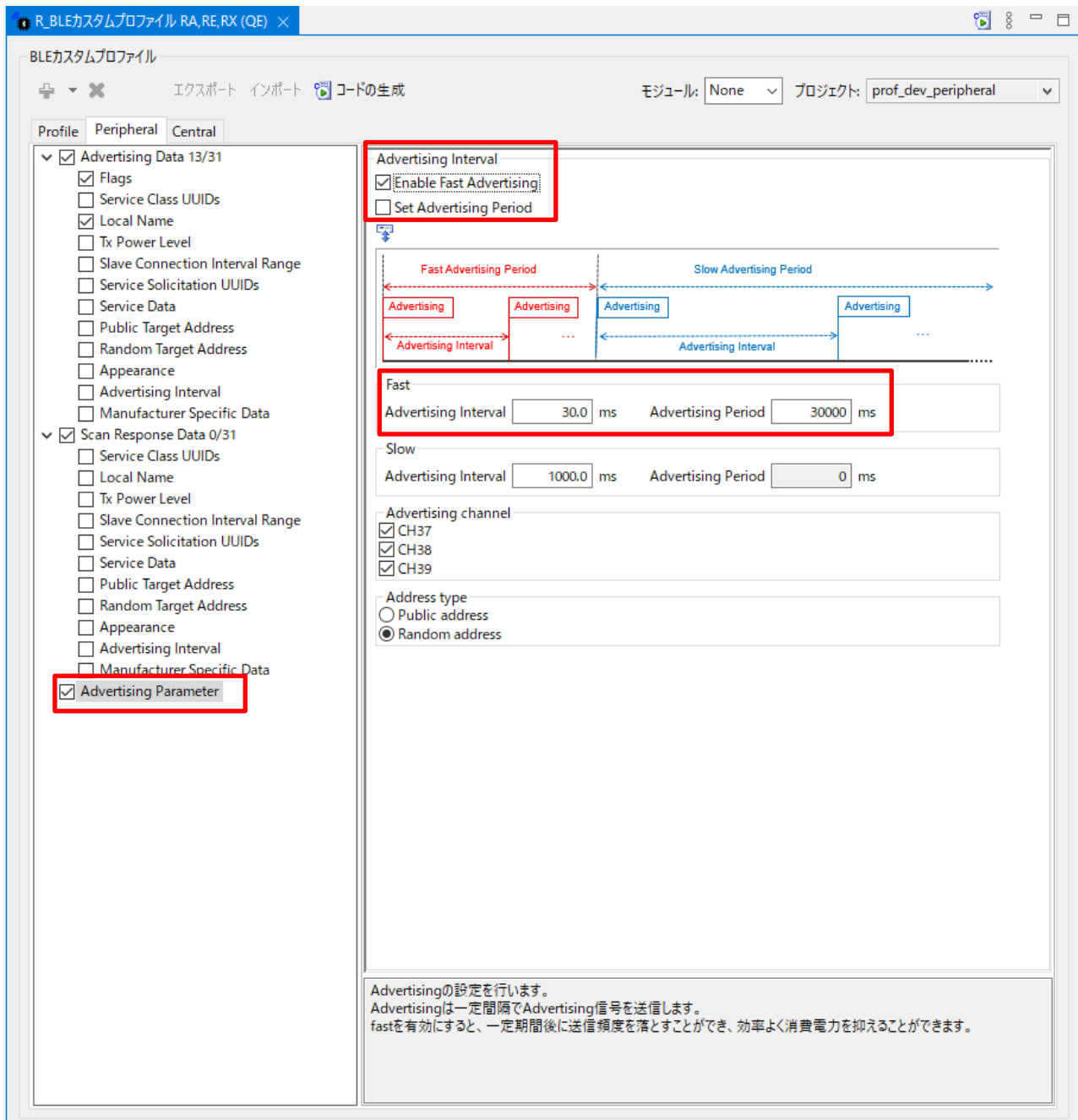


図 6.13 Advertise 動作に「Fast」の設定を使用する場合の設定例

セントラルロールの「Fast」の設定例を示します。「Scan Parameter」の設定を行います。「Fast」の設定を使用する場合は、「Enable Fast Scan」をチェックします。本設定では起動から 30 秒間、60msec 毎に 30msec の Scan Window を開き Scan 動作を実行します。

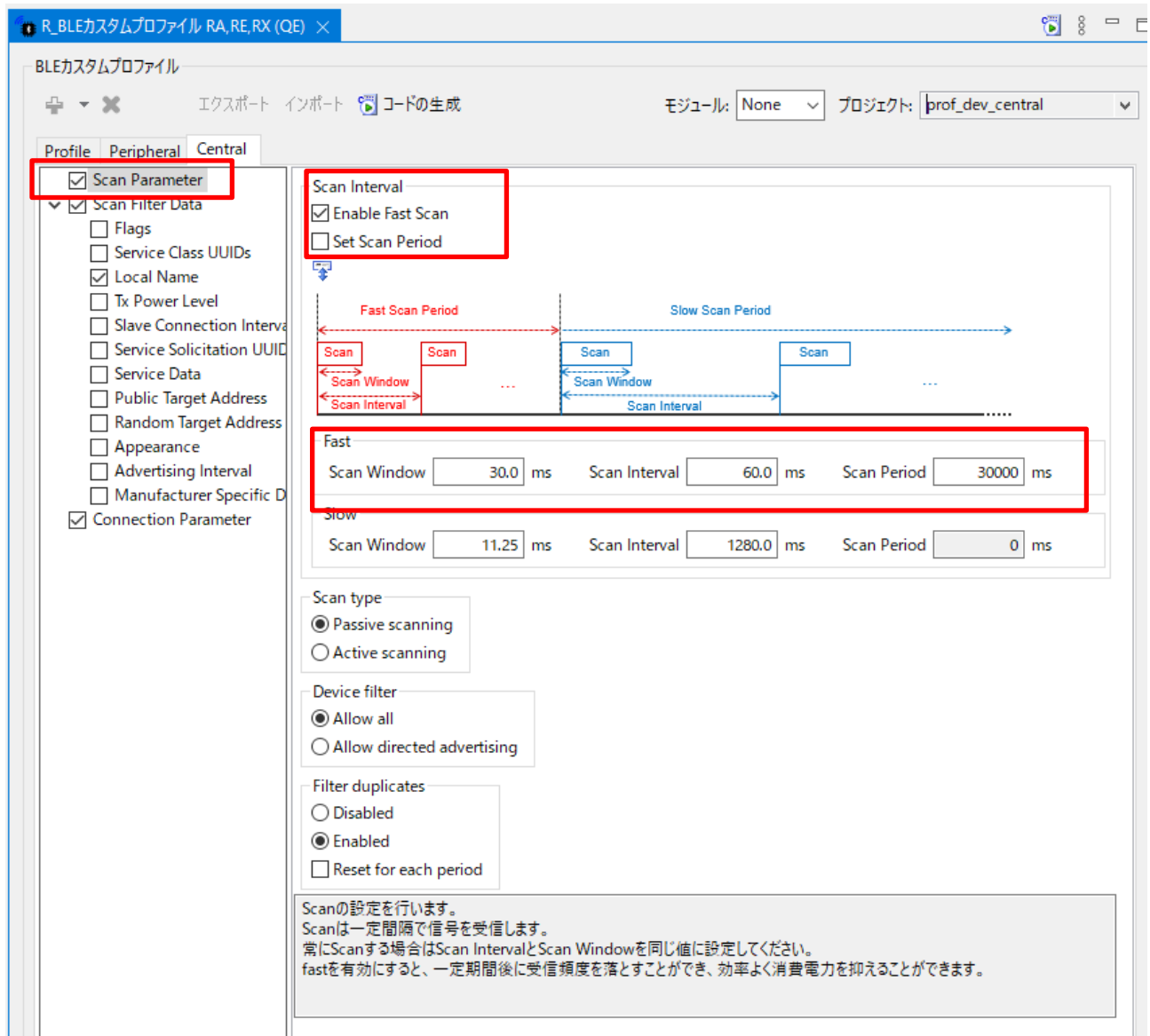


図 6.14 スキャン動作に「Fast」の設定を使用する場合の設定例

## 6.7 旧バージョンの認証情報(QDID:134484)を使用する場合

QE for BLE Utility 1.60 から QDID: 199248 の認証を取得したサービス API プログラムが生成されます。以前の QDID:134484 を使用して引き続き開発する場合には、以下の二つの手順を実施します。\*認証情報(QDID:134484)は 2023 年 2 月 1 日以降に Bluetooth 認定製品として新規に登録できなくなります。開発中の製品が既に認定済み製品の派生製品の場合は、2024 年 1 月 31 日まで製品の追加登録は可能です 2024 年 2 月 1 日以降は既存の認定製品の販売のみ可能です。

- QE for BLE の生成コード変更の設定
- プロファイル共通部の取得 (RX23W のみ)

### 6.7.1 QE for BLE の生成コード変更の設定

以前の認証情報を使用して開発する場合には QE for BLE の設定を変更します。

e<sup>2</sup> studio のメニューバーの"ウィンドウ"の"設定"を開きます。

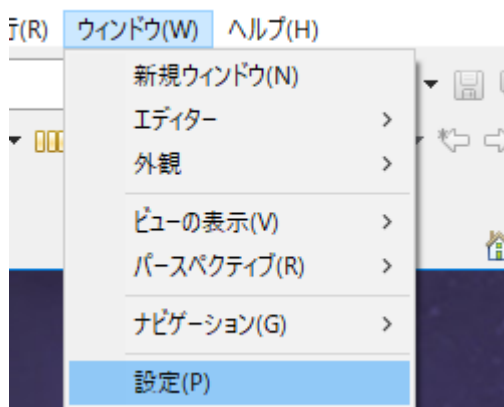


図 6.15 QE for BLE のオプションの開き方

左の一覧から"Renesas"→"Renesas QE"→ "QE for BLE"を選択し、"旧版のQE for BLE[RA, RE, RX] Utility を使用する。"にチェックを入れます。

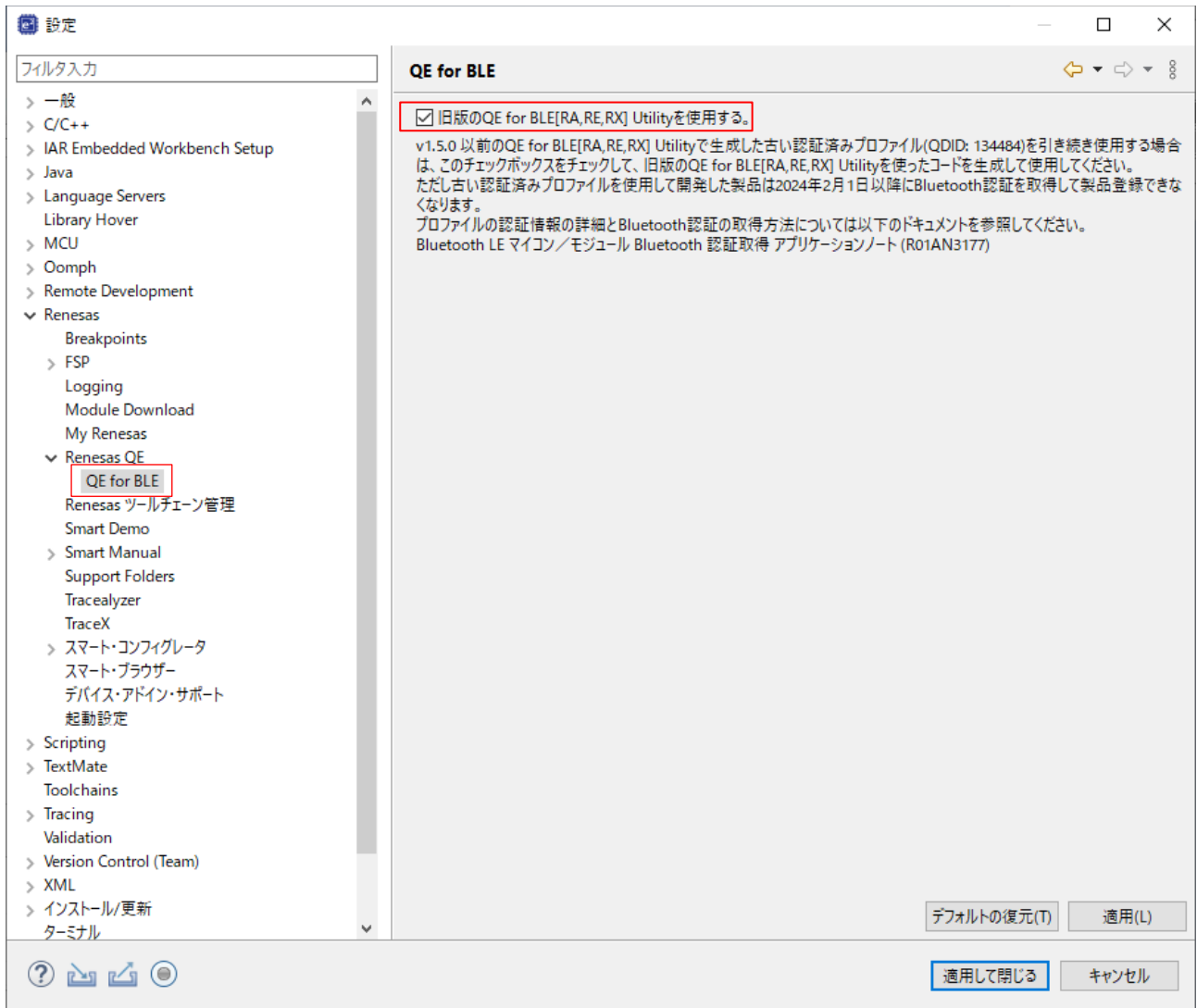


図 6.16 旧版のサービス API プログラムを使用する設定画面

設定を適用した後で QE for BLE からコード生成すると旧版のサービス API プログラムが生成されます。

## 6.7.2 プロファイル共通部の取得

この手順は RX23W 環境で BLE FIT Module 2.50 以上を使用する場合に実施します。BLE FIT 2.50 以上を使用して、旧版を使用する場合プロファイル共通部が QE for BLE から生成されないため、プロファイル共通部をプロジェクトに追加する必要があります。

以下のいずれかの方法を実施してください。

- QE for BLE から生成されるプロファイル共通部を追加する。
- trash フォルダから BLE FIT 2.40 に同梱されているプロファイル共通部を復元する。

### QE for BLE から生成されるプロファイル共通部を追加

trash フォルダにプロファイル共通部がない場合には、6.7.1 節の手順で「旧版の QE for BLE[RA,RE,RX]を使用する」のチェックを外してコード生成を行い、生成された以下のフォルダをプロジェクトのパスの通った場所にコピーします。

- qe\_gen/discovery
- qe\_gen/profile\_cmn

その後、もう一度 6.7.1 節を実施し、旧版のサービス API プログラムが生成されるように設定します。

### trash フォルダから BLE FIT 2.40 に同梱されているプロファイル共通部を復元

trash にある BLE FIT 2.40 の以下のフォルダをプロジェクトのパスの通った場所にコピーします。

- src/smc\_gen/r\_ble\_rx23w/src/discovery
- src/smc\_gen/r\_ble\_rx23w/src/profile\_cmn

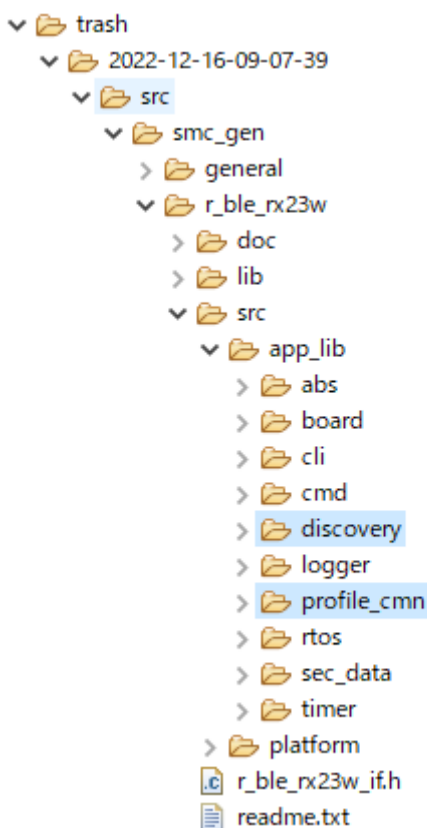


図 6.17 trash フォルダからの復元



## 改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2022.06.30	—	新規作成
1.10	2022.12.27	5	RX23W 環境でプロファイル共通部が QE for BLE から生成されることを明記
		6	QE for BLE がサポートするプロファイル/サービスのバージョン情報を更新
		20	カスタムおよび SIG 標準プロファイル/サービスのセキュリティ要件を追記
		42	エンコードデコード関数の自動生成機能についての説明を追記
		49	セキュリティ要件を設定した場合の実装方法を追記
		52, 61	Write Long 動作、Read Long 動作の記述を追記
		75	BLE FIT 2.50 と QE for BLE Utility 1.50 の組み合わせ時にプロファイル共通部がなくなることを追記
		94	6.7 旧バージョンの認証情報(QDID:134484)を使用する場合を追加

## 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しないでください。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、 $V_{IL}(\text{Max.})$  から  $V_{IH}(\text{Min.})$  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、 $V_{IL}(\text{Max.})$  から  $V_{IH}(\text{Min.})$  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えば、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、ノイズ輻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まれます。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、変更、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た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なります。
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](http://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/](http://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。