RX23W グループ

Bluetooth Low Energy アプリケーション開発者ガイド

要旨

本アプリケーションノートは、Bluetooth® Low Energy (以降、「Bluetooth LE」または「BLE」と表記)のアプリケーションの開発方法について説明します。

本文中の【BP】には、実装者がセキュリティおよびプライバシに対してベストプラクティスな選択ができるように Bluetooth SIG が公開するガイド(Bluetooth® Security and Privacy Best Practices Guide)を基に推奨事項やリスクについて説明していますのでご参考ください。

対象デバイス

- RX23W グループ

関連ドキュメント

- Bluetooth Core Specification (https://www.bluetooth.com)
- Bluetooth® Security and Privacy Best Practices Guide (https://www.bluetooth.com)
- Bluetooth Low Energy プロファイル開発者ガイド (R01AN6459)
- 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 BLE モジュール Firmware Integration Technology (R01AN4860)

Bluetooth® のワードマークおよびロゴは、Bluetooth SIG,Inc. が所有する登録商標であり、ルネサス エレクトロニクス株式会社はこれらのマークをライセンスに基づいて使用しています。その他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。

目次

1. 概要 7
1.1 Bluetooth Low Energy アプリケーションの開発 7
1.2 開発環境 9
1.2.1 ハードウェア条件 9
1.2.2 ソフトウェア条件 10
1.2.3 ツール 11
1.3 利用可能な通信機能 12
1.4 基本的な通信機能 14
1.4.1 デバイスの識別 16
1.5 Bluetooth LE Protocol Stack の動作概要 17
1.6 ソフトウェア構成 19
1.6.1 主要機能 20
1.6.2 周辺機能 24
1.7 開発の流れ 25
1.8 本ドキュメントの使用例 27
1.9 セクションの配置 28

2. コンフィグレーションオプション調整 29
2.1 コンフィグレーションオプション 29
2.2 RAM の調整方法 32
2.3 BD アドレスの設定方法 33
2.3.1 データ領域への書き込み 35
2.3.2 Random Address 使用方法 35
2.4 消費電流が最も小さくなる構成 37
2.4.1 MCU 低消費電力機能の使用 38

3. ユーザコードの実装方法 41
3.1 スケルトンプログラムの動作 42
3.2 app_main 関数 43
3.2.1 初期化処理(ble_app_init 関数) 44
3.2.2 メインループとスケジューラ(R_BLE_Execute) 47
3.2.3 終了処理 48
3.3 GAP のイベント(gap_cb 関数) 49
3.4 GATTS のイベント(gatts_cb 関数) 51
3.5 GATTC のイベント(gattc_cb 関数) 52
3.6 VS のイベント(vs_cb 関数) 53
3.7 サーバ側プロファイル API のイベント([サービス名]s_cb 関数) 54
3.8 クライアント側プロファイル API のイベント([サービス名]c_cb 関数) 55
3.9 L2CAP のイベント 56
3.10 イベント通知機能(R_BLE_SetEvent) 57
3.11 RF 通信タイミング通知 59

4. app_lib 63
4.1 ソフトウェアタイマ 63
4.2 コマンドライン 67

4.2.1 標準コマンドの使用手順 68
4.2.2 ユーザコマンドの作成手順 71
4.3 ロガー 75
4.4 セキュリティデータ管理 77
4.4.1 初期化処理 77
4.4.2 ローカルデバイスの鍵の取得 77
4.4.3 ローカルデバイスの鍵の保存 77
4.4.4 リモートデバイスの鍵の保存 78
4.5 ボード LED & Switch 79
4.5.1 お客様独自のボード用の設定 80
4.5.2 ボード LED & Switch 初期化 82
4.5.3 ボード LED の ON/OFF 82
4.5.4 スイッチ押下に割り当てるコールバック 82
4.6 抽象 API 83

5. Advertising 85
5.1 手順 85
5.2 GAP API による Advertising 86
5.2.1 Advertising パラメータ設定 86
5.2.2 Advertising Data / スキャンレスポンスデータの設定 / 更新 89
5.2.3 Advertising 開始 89
5.2.4 Advertising 停止 89
5.3 GAP API による Periodic Advertising 90
5.3.1 Non-Connectable Advertising パラメータ設定 91
5.3.2 Periodic Advertising パラメータ設定 91
5.3.3 Periodic Advertising Data 設定／更新 92
5.3.4 Periodic Advertising 開始 92
5.3.5 Periodic Advertising 停止 94
5.4 Advertising Data / スキャンレスポンスデータ / Periodic Advertising Data の設定 95
5.4.1 フォーマット 95
5.4.2 Advertising 中の更新 98
5.4.3 Periodic Advertising 中の更新 98
5.4.4 バッファサイズ 99
5.5 抽象 API による Advertising 100
5.5.1 ホワイトリストの使用(既知のデバイスに応答する) 100
5.5.2 プライバシ 101
5.6 スマートフォンとコネクション 102
5.7 ビーコンとして使用 103

6. スキャン 104
6.1 スキャンの開始／停止 104
6.2 スキャンパラメータ 104
6.2.1 プライバシ 106
6.3 スキャンで受信する情報 108
6.4 スキャンのフィルタリング 110
6.4.1 ホワイトリストの使用(既知のデバイスから受信する) 110
6.4.2 重複する Advertising を受信しない 111

6.4.3 Discoverable Mode によるフィルタリング 111
6.4.4 Advertising Data によるフィルタリング 111
6.5 Periodic Advertising Synchronization 112
6.5.1 スキャン開始 113
6.5.2 Periodic Advertising の確認 113
6.5.3 Periodic Advertiser List 登録 113
6.5.4 Periodic Advertising Sync 確立 113
6.5.5 Periodic Advertising の受信 115
6.5.6 Periodic Advertising Sync 喪失 115
6.5.7 Periodic Advertising Sync 終了 115

7. コネクション 116
7.1 セントラルのコネクション手順 116
7.1.1 ホワイトリストの使用(既知のデバイスにコネクションする) 117
7.1.2 プライバシ 118
7.2 コネクションキャンセル 119
7.3 複数台コネクション 120
7.3.1 複数のペリフェラルデバイスとコネクションする 121
7.3.2 複数のセントラルデバイスとコネクションする 126
7.3.3 マルチロール 130
7.4 切断 135

8. 通信 136
8.1 PHY の変更 136
8.2 最大送信パケット長の変更 139
8.3 コネクションパラメータの更新 141
8.4 MTU の変更 146
8.5 フロー制御 148
8.6 高スループット通信 149

9. セキュリティ 150
9.1 ペアリング 150
9.1.1 ペアリングパラメータ設定 152
9.1.2 鍵の生成・登録 156
9.1.3 OOB(Out of band)データの送受信 156
9.1.4 ペアリングの要求 157
9.1.5 ペアリング要求への応答 157
9.1.6 ペアリングメソッドの実施 158
9.1.7 鍵の交換 159
9.1.8 ペアリングの完了 160
9.2 ボンディング 161
9.2.1 リモートデバイスの鍵の保存 162
9.2.2 ローカルデバイスの鍵の保存 166
9.2.3 保存した鍵の再設定 166
9.2.4 保存した鍵の削除 166
9.2.5 ボンディング後のリモートデバイスのフィルタリング 167
9.3 暗号化 168

9.3.1　暗号化要求 168
9.3.2　暗号化要求への応答 170
9.4　プライバシ 173
9.4.1　ローカルデバイスの RPA 生成 175
9.4.2　リモートデバイスのアドレス解決 178

10.　プロファイル・サービス 185
10.1　標準プロファイル及びサービス 186
10.2　GATT プロシージャの API 192
10.2.1　Read 動作 192
10.2.2　Write 動作 193
10.2.3　WriteWithoutResponse 動作 194
10.2.4　Notification 動作 195
10.2.5　Indication 動作 197
10.2.6　ReliableWrites 動作 199
10.2.7　Broadcast 動作 201
10.3　GATT プロシージャの実装方法 203
10.3.1　クライアント側からデータを送信する実装例 203
10.3.2　サーバ側からデータを送信する実装例 206

11.　デバッグ 208
11.1　ロガー機能の使用 209
11.2　コマンドライン機能の使用 211
11.3　RF 通信タイミング通知機能の使用 213
11.4　サーバの動作確認 218
11.4.1　BTTS Beacon Scanning の使用 218
11.4.2　BTTS Data Comm Master の使用 219
11.4.3　GATT Browser の使用 219
11.5　クライアントの動作確認 220
11.5.1　BTTS Beacon Advertising の使用 220
11.5.2　BTTS Data Comm Slave の使用 221
11.6　その他 223
11.6.1　MCU パッケージ 223
11.6.2　MOT ファイルの生成 224
11.6.3　MAP ファイルの詳細出力 225
11.6.4　最適化 225

12.　付録 A：サンプルアプリケーション 226
12.1　Beacon サンプル 229
12.1.1　対向デバイス 229
12.1.2　動作 229
12.1.3　Advertising Data 229
12.1.4　コンフィグレーションオプション 231
12.1.5　変更可能なパラメータ 231
12.1.6　コマンド 231
12.2　Peripheral サンプル 232
12.2.1　対向デバイス 232

12.2.2 動作 232
12.2.3 コンフィグレーションオプション 234
12.2.4 変更可能なパラメータ 234
12.3 Central サンプル 235
12.3.1 対向デバイス 235
12.3.2 動作 235
12.3.3 コンフィグレーションオプション 235
12.3.4 変更可能なパラメータ 236
12.4 Multi-role サンプル 237
12.4.1 トポロジ 237
12.4.2 対向デバイス 238
12.4.3 動作 238
12.4.4 コンフィグレーションオプション 241
12.4.5 変更可能なパラメータ 242

改訂記録 243

1. 概要

1.1 Bluetooth Low Energy アプリケーションの開発

Bluetooth Low Energy を用いてデータ通信する方法はコネクションレス型とコネクション型があります。コネクションレス型の応用であるメッシュ通信を行う場合には、「Bluetooth メッシュスタックパッケージスタートアップガイド (R01AN4874)」や「Bluetooth メッシュスタック開発ガイド(R01AN4875)」をご参照ください。

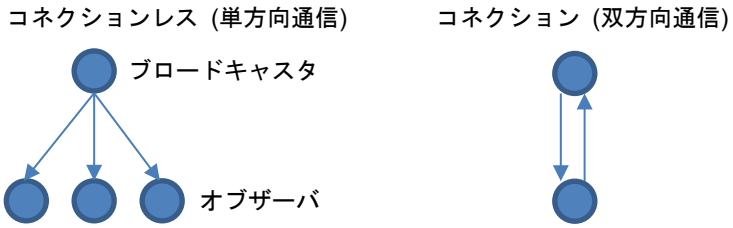


図 1-1　Bluetooth LE のデータ通信方式のイメージ

コネクションレス型では、アプリケーションデータをアドバタイズパケットにのせて送信します。受信デバイスはスキャンによってアドバタイズパケットを受け取ります。アプリケーションは、デバイスの検出やコネクションを行うための Generic Access Profile(GAP)によってこの通信を行います。この方式では、データは、ブロードキャスタからオブザーバへの単方向通信になります。デバイスのコネクションを行わないため、アドバタイズパケットはどのデバイスでも受信することができます。

双方向通信を行うためには、コネクション型による通信を行います。コネクション型は、デバイス同士を GAP によってコネクションします。アプリケーションデータは Generic Attribute Profile(GATT)によって送受信されます。GATT は、GAP の通信経路上におけるサーバクライアントアーキテクチャによる通信を提供します。GATT では、アプリケーションプロファイルに則ってデータ通信を行います。

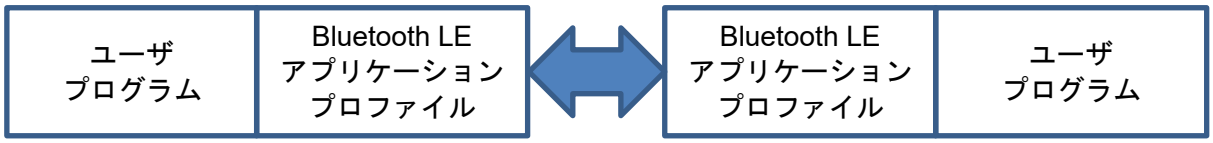


図 1-2　Bluetooth LE 双方向通信のイメージ

Bluetooth LE が想定するアプリケーションについては、Bluetooth SIG が仕様としてアプリケーションプロファイルを公開しています。このアプリケーションプロファイルを実装することで、既に動作している既存のデバイスと相互コネクションできます。新たに双方向通信アプリケーションを開発する際は、ユーザプログラムだけでなく、アプリケーションプロファイルを設計します。

アプリケーションプロファイルは、GATT のサーバクライアント間でやり取りされるアプリケーションデータの構造とデータベースへのアクセス方法と、GAP による通信パラメータの設定、デバイスのコネクション方法、セキュリティの設定等が決められています。

【BP】 独自設計のカスタムプロファイルを含め、変更可能な GATT キャラクタリスティック(例えばドアロックであればリモートデバイスが属性の値を変更することでロック状態を操作する)がある場合、認証と暗号化のサポートが推奨されています。

本ドキュメントでは、Bluetooth LE 通信を行うためのプログラムの実装方法と、アプリケーションプロファイル設計のヒントとなる情報を説明します。

Renesas では Bluetooth LE アプリケーション開発を補助するツールを提供しています。

(1) BLE FIT モジュール

Bluetooth SIG が規定する Bluetooth Core Specification version 5.0 に準拠した Bluetooth LE 機能を FIT モジュール形式で提供します。e²studio 上のスマートコンフィグレータからプロジェクトに追加し、Bluetooth LE アプリケーション開発を開始できます。

Bluetooth LE 機能は、Bluetooth LE Protocol Stack としてライブラリ形式で提供します。API を利用することで、Bluetooth LE の動作を行います。Bluetooth LE Protocol Stack では、消費電力削減のため Bluetooth LE に関するイベントをコールバック関数によりアプリケーションに通知します。

BLE FIT モジュールは、Bluetooth LE Protocol Stack に加えて、アプリケーション開発を補助するアプリケーションライブラリ(app_lib)を提供します。app_lib を利用することで、Bluetooth LE の基本的な動作を簡単に実現できます。

(2) QE for BLE, QE Utility モジュール

QE for BLE は、アプリケーションプロファイルを GUI で設計とコード生成を行うための QE ツールです。QE Utility モジュールが提供するテンプレートファイルに基づいてコード生成を行います。QE Utility モジュールは FIT モジュール形式または QE for BLE V1.40 以降では同梱形式で提供されます。

これらのツールを使用することで、アプリケーションプロファイルの GATT 部分を GUI から設計し、プロファイル実現のための API(サービス API)を生成します。設計したプロファイルだけでなく、Bluetooth SIG の仕様を満たす API も生成可能です。

最後に、Bluetooth LE アプリケーション開発プロセスの一例と、Renesas ツールの使用例を示します。

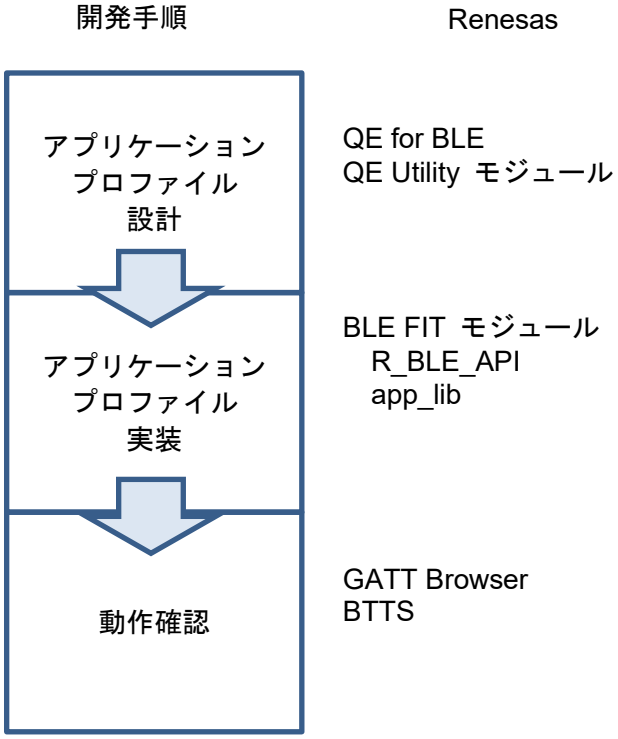


図 1-3　Bluetooth LE アプリケーションの開発手順と補助ツール

1.2　開発環境

1.2.1　ハードウェア条件

アプリケーションのビルドおよびデバッグに必要なハードウェア条件を表 1.1 に示します。

表 1.1　ハードウェア条件

ハードウェア	説明
ホスト PC	USB インタフェースを搭載した Windows PC
MCU ボード	RX23W を搭載したボード Target Board for RX23W [RTK5RX23W0C00000BJ] または RSSK RX23W [RTK5523W8AC00001BJ] または Target Board for RX23W module [RTK5RX23W0C01000BJ] 【注】本ドキュメントでは Target Board for RX23W を使って説明します。
オンチップデバッギング エミュレータ	E2 エミュレータ [RTE0T00020KCE00000R] または E2 エミュレータ Lite [RTE0T0002LKCE00000R] または E20 エミュレータ [R0E000200KCT00] または E1 エミュレータ [R0E000010KCE00] RSSK を使用する場合にはエミュレータが必要となります。 Target Board には E2 エミュレータ Lite 相当のオンボードデバッガが実装されているため、エミュレータを用意する必要はありません。 【注】E1 エミュレータは販売終了品です。 【注】本ドキュメントでは E2 エミュレータ Lite を使って説明します。
USB ケーブル	エミュレータおよび対象ボードとの接続に使用します。 E2 または E1 エミュレータ：USB A-miniB 1 本 Target Board：USB A-microB 2 本 RSSK： USB A-microB 1 本

1.2.2　ソフトウェア条件

アプリケーションのビルドおよびデバッグに必要なソフトウェア条件を表 1.2 に示します。

表 1.2　ソフトウェア条件

ソフトウェア		バージョン	説明
CC-RX 環境	統合開発環境 e² studio	v7.6.0 以降	ルネサス製デバイス用の統合開発環境 【注】本ドキュメントでは e²studio を使って説明します。
	統合開発環境 CS+ for CC	V8.02.00 以降	ルネサス製デバイス用の統合開発環境 【注】QE for BLE に対応していないため e²studio の使用を推奨します。
	CC-RX コンパイラ	V2.08.00 以降	C/C++コンパイラ (e² studio のインストーラからダウンロード選択できます)
	QE for BLE[RX]	v1.0.0 以降	アプリケーションとプロファイル開発用のスケルトンプログラムを生成するための e²studio 用プラグイン 【注】QE for BLE[RX]は QE for BLE[RA,RE,RX]に統合しました。
	QE for BLE[RA,RE,RX] QE for BLE[RA,RE,RX] Utility	v1.4.0 以降	
	BLE FIT モジュール (r_ble_rx23w)	v1.10 以降	ルネサス製マイコンで Bluetooth Low Energy のアプリケーションを開発するために必要です。 【注】 BLE FIT モジュール v2.50 以降を使用する場合は QE for BLE[RA,RE,RX] Utility v1.6.0 以降を使用してください。
	BSP FIT モジュール (r_bsp)	v5.40 以降	BLE FIT モジュールを使用するために必要です。 BLE FIT モジュールのバージョン 1.01 以降を使用する場合、r_bsp のバージョンを 5.40 以降への変更が必要となります。
	CMT FIT モジュール (r_cmt_rx)	v4.10 以降	BLE FIT モジュールを使用するために必要です。 Bluetooth LE Protocol Stack は CMT2, CMT3 を使用します。 app_lib/timer のソフトウェアタイマ機能を使用する場合は、さらに 1ch 使用します。
	LPC FIT モジュール (r_lpc_rx)	v1.42 以降	BLE FIT モジュールで MCU 低消費電力機能を使用する場合、v1.42 以降を使用します。
	フラッシュ FIT モジュール (r_flash_rx)	v4.10 以降	BLE FIT モジュールでオプション機能のデバイス固有データ管理機能を使用する場合、v4.10 以降を使用します。
IAR 環境	統合開発環境 IAR Embedded Workbench for Renesas RX	v4.12.1 以降	IAR 製の RX ファミリ用の統合開発環境 【注】BLE FIT モジュール v1.10 以降で対応
	IAR C/C++ Compiler for Renesas RX version	v4.12.1 以降	IAR 製の C/C++コンパイラ
	QE for BLE[RX]または QE for BLE[RA,RE,RX]	CC-RX 環境と同じ	e²studio で作成したプロジェクトを IAR のプロジェクトに変換して使用します。手順については「BLE モジュール Firmware Integration Technology (R01AN4860)」の「4.9 IAR 開発環境でのプロジェクト作成」を参照してください。
Renesas Flash Programmer		V3.06.00 以降	マイコンの内蔵フラッシュメモリ書き込みツール
整数型			ANSI C99 を使用しています。 これらの型は stdint.h で定義されています。
エンディアン			リトルエンディアン

1.2.3　ツール

アプリケーション開発は以下のツールによってサポートされます。

表 1.3　アプリケーション開発をサポートするツール

ツール	説明
GATT Browser	GATT サーバにアクセスするスマートフォンアプリケーションです。実装した Bluetooth Low Energy 通信の基本的な動作、GATT データベースの構造などをスマートフォンから確認できます。
BTTS	Windows PC と USB Serial で接続した RX23W を制御し、Bluetooth Core Specification 5.0 における RF、ビーコン通信、データ通信の 3 つの機能を評価するためのツールスイートです。デバイスが電波法認証を取得する際に使用することもできます。

1.3　利用可能な通信機能

RX23W は表 1.4 で示す Bluetooth LE(Low Energy)機能をサポートし、LE 機能を有するデバイスと通信することができます。

表 1.4　LE 機能

Bluetooth バージョン	LE 機能と説明	備考
5.0	LE 2M PHY (2 Msym/s PHY for LE) - 2Mbps PHY データレート	高データスループット 通信時間小による低消費電力化
5.0	LE Coded PHY (LE Long Range) - 500kbps/125kbps PHY データレート	通信距離拡大
5.0	LE Advertising Extensions - セカンダリチャネルによる Advertising が可能 (RX23W では 4 つまでの独立した Advertising の同時実行が可能) - Advertising Data/スキャンレスポンスデータの最大サイズを拡張(31 バイト → 1650 バイト) - Long Range による Advertising - Periodic Advertising が可能	無線干渉低減 ビーコン情報拡大 長距離でのコネクション確立 セカンダリチャネルの活用
5.0	LE Channel Selection Algorithm #2 - チャネルホッピングアルゴリズムの改良	無線干渉低減
5.0	High Duty Cycle Non-Connectable Advertising - 最小 Advertising Interval 短縮(100ms → 20ms)	コネクションまでの時間短縮 ビーコン送信の高頻度化
4.2	LE Data Packet Length Extension - データ通信のパケットサイズを拡張(27 バイト→251 バイト)	高データスループット 通信時間小による低消費電力化
4.2	LE Secure Connections - Elliptic curve Diffie-Hellman 鍵共有方式(ECDH)により、passive eavesdropping に対応したペアリングをサポート	セキュリティ強化
4.2	Link Layer Privacy - プライバシ機能のアドレス解決を Link Layer でできる機能	アドレス解決の高速化
4.2	Link Layer Extended Scanner Filter Policies	
4.1	Low Duty Cycle Directed Advertising - 既知のデバイスとの再コネクション用に Low Duty Cycle の Advertising をサポート	
4.1	32-bit UUID Support in LE - 32-bit の UUID をサポート(GATT で使用する場合は、128-bit に拡張)	
4.1	LE L2CAP Connection-Oriented Channel Support - L2CAP の credit based flow control チャネルを使った通信をサポート	
4.1	LE Link Layer Topology - セントラル、ペリフェラルの両方のロールをサポートし、あるリモートデバイスとのコネクションではセントラルとして、別のリモートデバイスとのコネクションではペリフェラルとして動作可能	トポロジの拡張
4.1	LE Ping - コネクションの暗号化後、MIC を含むパケットの送信要求により、コネクションが維持されているかどうかをチェックする機能	
Addendum 2	Appearance Data Type - GAP サービスで Appearance キャラクタリスティックを使用可能	

Bluetooth バージョン	LE 機能と説明	備考
4.0	Bluetooth Low Energy - Low Energy Controller Low Energy Physical Layer (PHY) Low Energy Link Layer (LL) - Low Energy Host Enhancements to L2CAP for Low Energy Security Manager (SM) - Enhancements to HCI for Low Energy - Low Energy Direct Test Mode - AES Encryption - Enhancements to GAP for Low Energy - Attribute Protocol (ATT) - Generic Attribute profile (GATT)	 Low Energy Controller は必須機能 Low Energy Host は必須機能 ATT は必須機能 GATT は必須機能

【注】BR/EDR(Basic Rate/Enhanced Data Rate)通信には非対応です。
【注】必須機能以外の機能の実装はオプション(ベンダ依存)となるため、スマートフォン等のデバイスでは対応していない可能性があります。

1.4　基本的な通信機能

LE 機能を有するデバイスが構築できる通信トポロジを図 1-4 に示します。

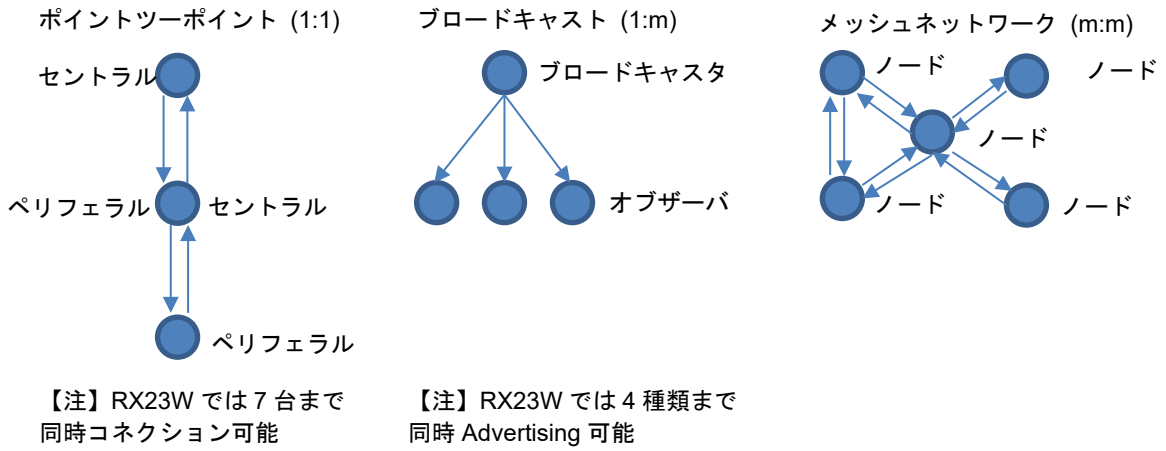
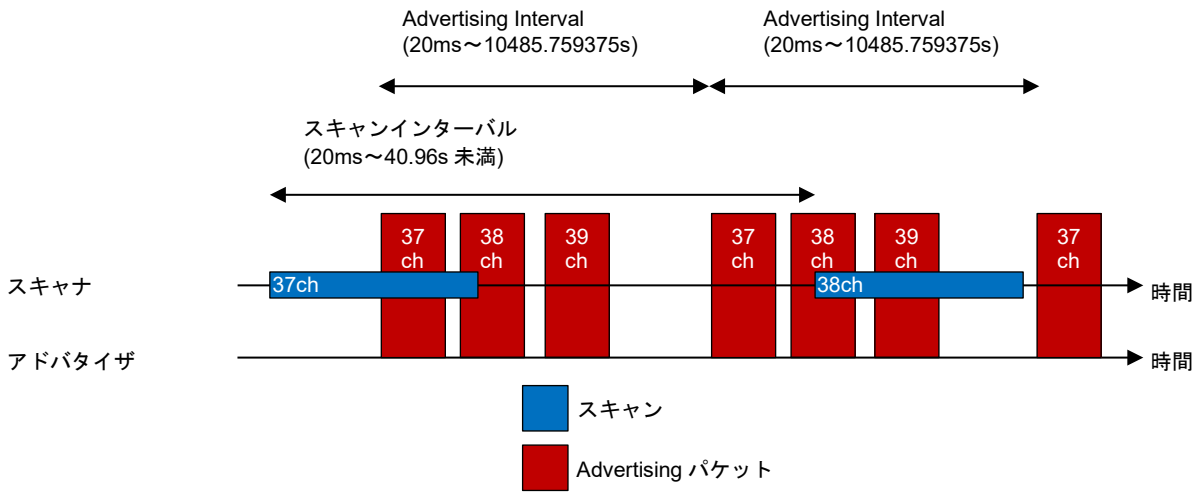


図 1-4　通信トポロジ

ブロードキャストではコネクションを確立しないで通信します。ブロードキャスタ(アドバタイザ)が Advertising を実行してパケットを送信し、オブザーバ(スキャナ)がスキャンを実行してパケットを受信します。



【注】Advertising Interval は実際には 0～10ms のランダムディレイが Advertising ごとに付加されます。

図 1-5　Advertising とスキャン

ポイントツーポイントではコネクションを確立して通信します。ペリフェラル(アドバタイザ)が Advertising を実行してパケットを送信し、セントラル(スキャナ)がスキャンを実行してパケットを受信します。一方のデバイスがイニシエータとしてコネクションしたいデバイスに対してコネクションを要求し、もう一方が受け入れるとコネクションが確立します。イニシエータがセントラル、もう一方がペリフェラルとなります。コネクションが確立すると、データ通信が可能になります。

Advertising とスキャンからコネクションの確立までは GAP (Generic Access Profile)のコマンドによって制御します。コネクションの確立後のデータ通信は GATT (Generic Attribute Profile)のコマンドによって制御します。GATT では、GATT データベースとしてセンサデータ等を保持してサービスを提供する側をサーバ、サービスを要求する側をクライアントと呼びます。クライアントがデータベースを持つサーバに対して Read と Write を実行できます。サーバはクライアントに対して Indication と Notification を実行できます。クライアントは Indication を受信したら Confirmation を実行して応答を返信します。以下はセントラルがクライアント、ペリフェラルがサーバとなった場合の例です。

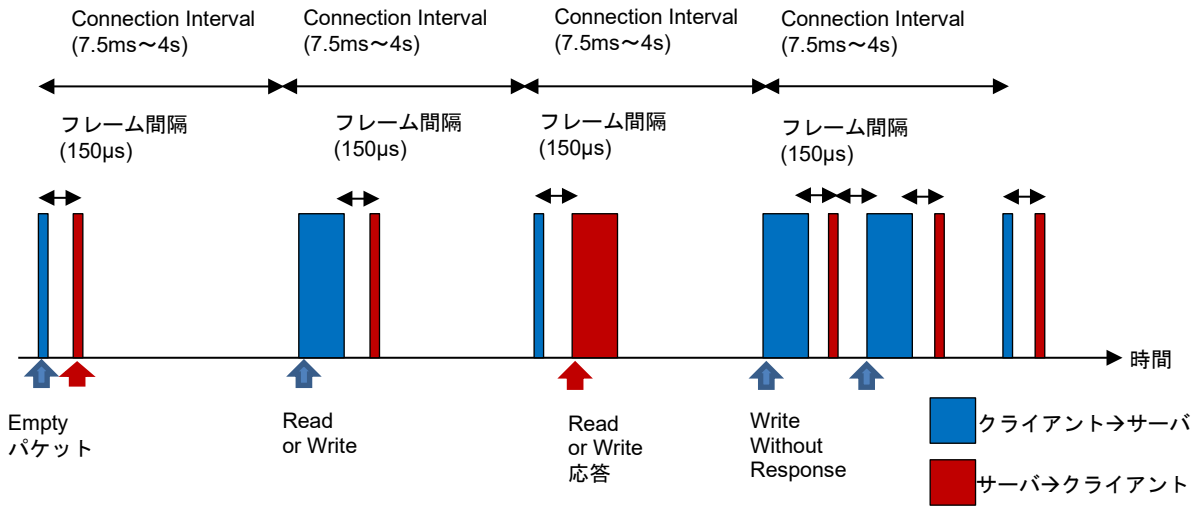


図 1-6　Read と Write

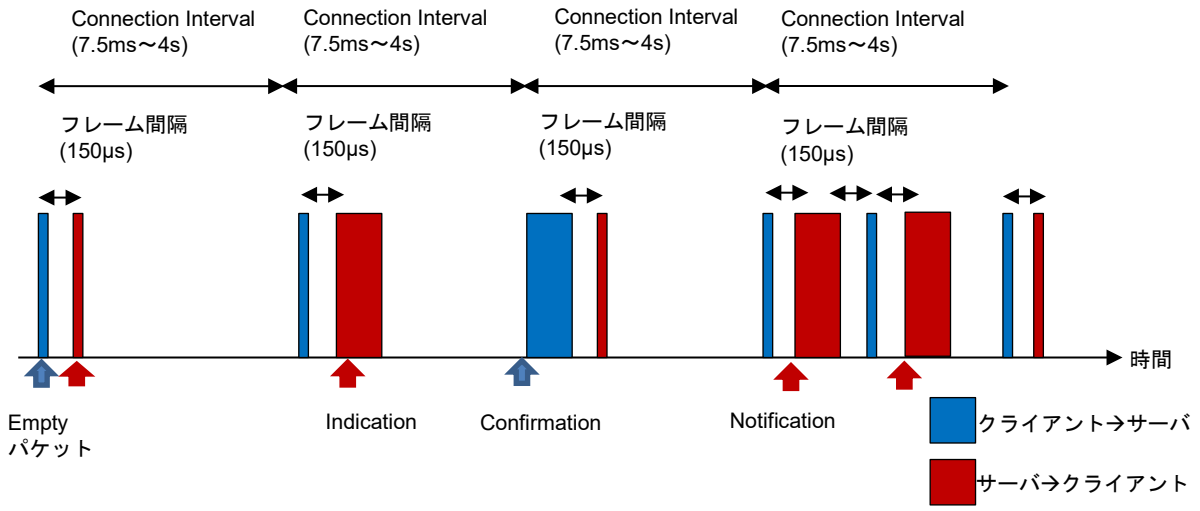


図 1-7　Indication と Notification

Advertising については「5 Advertising」で説明します。スキャンについては「6 スキャン」で説明します。コネクションについては「7 コネクション」で説明します。データ通信については「8 通信」で説明します。

【注】メッシュネットワークについては「Bluetooth メッシュスタックパッケージスタートアップガイド (R01AN4874)」を参照してください。

1.4.1　デバイスの識別

デバイスは Bluetooth Device アドレス(BD アドレス)を使って識別します。BD アドレスについては「2.3 BD アドレスの設定方法」で説明します。

追加で Advertising Data の Local Name や GAP サービスの Device Name を使用することもできます。Local Name を「表 5.7」に示します。Device Name を「表 10.2」に示します。

1.5　Bluetooth LE Protocol Stack の動作概要

Bluetooth LE Protocol Stack は BLE 周辺機能を制御し、RF イベントの実行を管理します。RF イベントとは、Bluetooth LE で規定される以下 4 つの動作状態におけるインターバルごとの 1 回の通信動作を指します。

- Advertising
- Scanning
- Initiating
- Connection

Bluetooth LE Protocol Stack は Bluetooth LE 動作の制御インタフェースを R_BLE API として提供します。BLE 周辺機能は RF イベントに応じて割り込み(BLEIRQ)を MCU へ発行します。BLEIRQ 発生時は R_BLE_Execute をコールし、RF イベントの状況に応じたタスク処理を行う必要があります。また、各種 R_BLE API をコールした場合も R_BLE_Execute をコールして Bluetooth LE Protocol Stack の API タスク処理を行う必要があります。

「2.1 コンフィグレーションオプション」の BLE_CFG_RF_DEEP_SLEEP_EN が 1 に設定されている場合、Bluetooth LE Protocol Stack が実行するタスクがなく、かつ次の RF イベント時間開始までに 80msec 以上の時間がある場合には消費電流を抑えるために R_BLE_Execute 内で RF 部をスリープモードに遷移させます。この時間は RF イベントのインターバル時間ではなく、RF イベント完了から次の RF イベント開始までの RF アイドル時間を指します。そのため、RF 部をスリープモードに遷移させるためには各レイヤの処理時間を考慮し RF イベントのインターバルを 100ms 以上に設定する必要があります。Scanning では、スキャンインターバルとスキャンウインドウとの時間差も 100ms 以上に設定する必要があります。

RF 部をスリープモードに遷移させるために、Bluetooth LE Protocol Stack は RF スリープ処理と RF ウェイクアップ処理を行います。図 1-8 に RF スリープを伴う場合の MCU/RF 動作概要を示します。

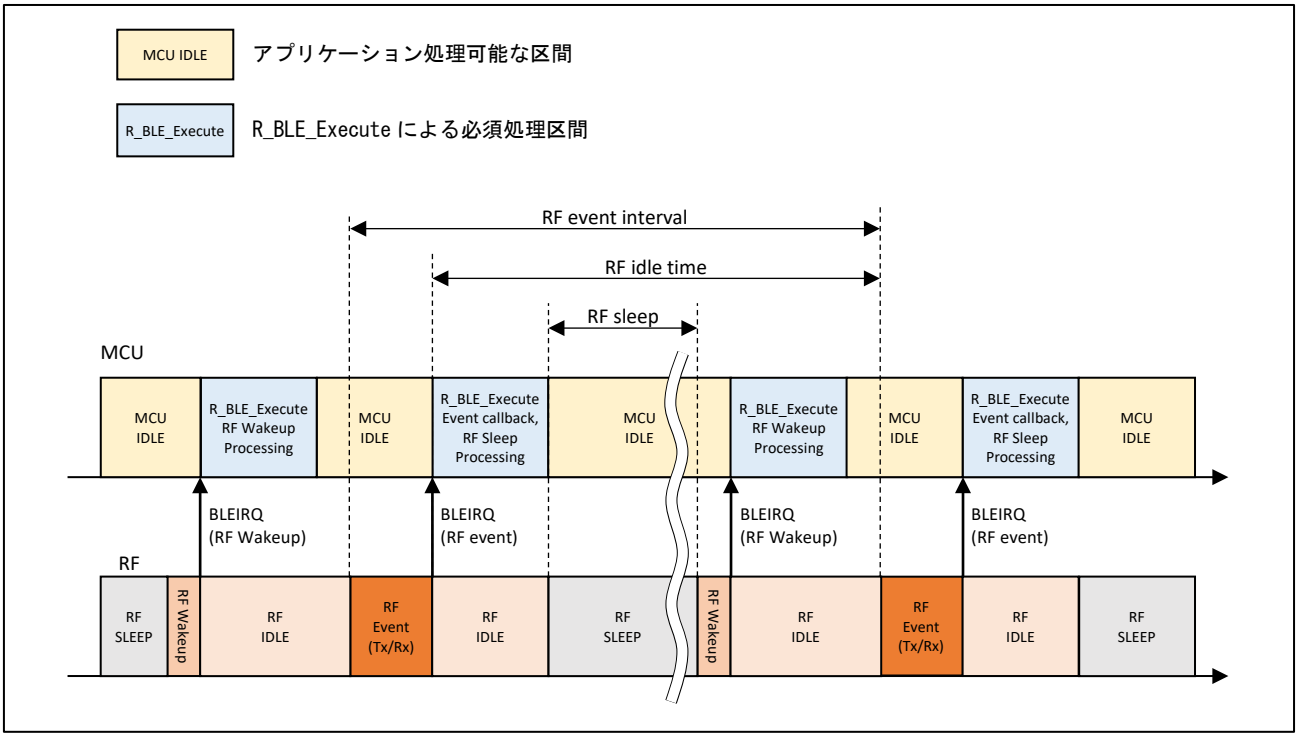


図 1-8　RF スリープを伴う場合の MCU/RF 動作概要

MCU アイドル中は、MCU を低消費電力モードに遷移させることや、アプリケーション処理を実行することが可能です。ただし、R_BLE_Execute による RF ウェイクアップ処理が RF イベント開始前までに行われない場合、RF イベントを実行することができません。そのため、アプリケーション処理は R_BLE_Execute コールを妨げないように実装する必要があります。

「2.1 コンフィグレーションオプション」の BLE_CFG_RF_DEEP_SLEEP_EN が 0 に設定されている場合、または BLE_CFG_RF_DEEP_SLEEP_EN が 1 に設定されていても RF スリープ遷移の条件を満たしていない場合、Bluetooth LE Protocol Stack は RF 部をスリープモードに遷移させません。この場合、RF アイドル時間中の消費電流は増加しますが、RF スリープ処理および RF ウェイクアップ処理を行わないためアプリケーションで使用可能な MCU アイドル時間が増加します。図 1-9 に RF スリープを伴わない場合の MCU/RF 動作概要を示します。

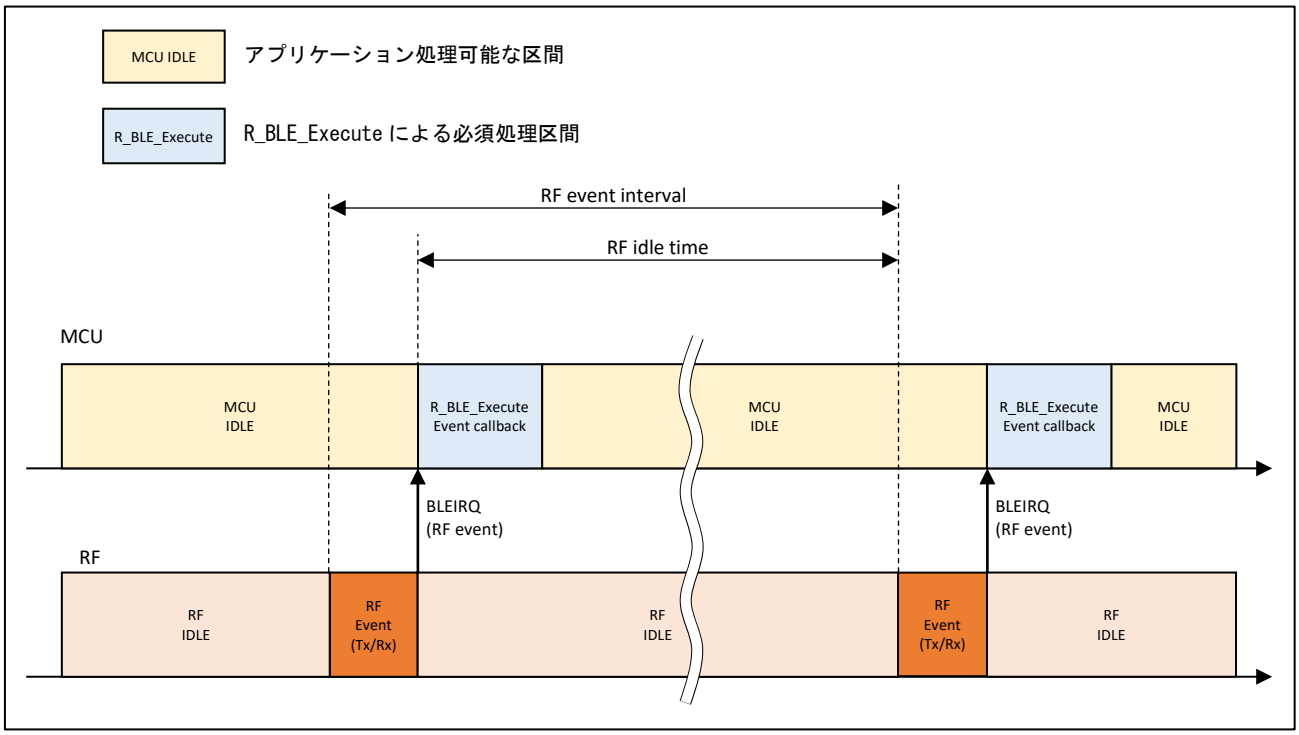


図 1-9　RF スリープを伴わない場合の MCU/RF 動作概要

RF スリープの状態に関わらず、アプリケーション処理が継続的に MCU を占有してしまい R_BLE_Execute がコールされない場合、コネクション維持が行えない場合があります。そのため、アプリケーション処理は短時間であることが推奨されます。長時間となる処理は「3.10 イベント通知機能 (R_BLE_SetEvent)」を参照して処理内容を複数回に分割して実行してください。

【注】Bluetooth LE Protocol Stack は R_BLE_Open によって RF ハードウェアの状態を初期化します。RF 通信動作中にソフトウェアリセットした場合は、R_BLE_Open をコールして RF ハードウェアの状態を初期化して RF 動作を停止させてください。

1.6　ソフトウェア構成

RX23W の Bluetooth LE のアプリケーションを開発するためには、図 1-10 で示すアプリケーション部分とプロファイル部分の開発が必要です。

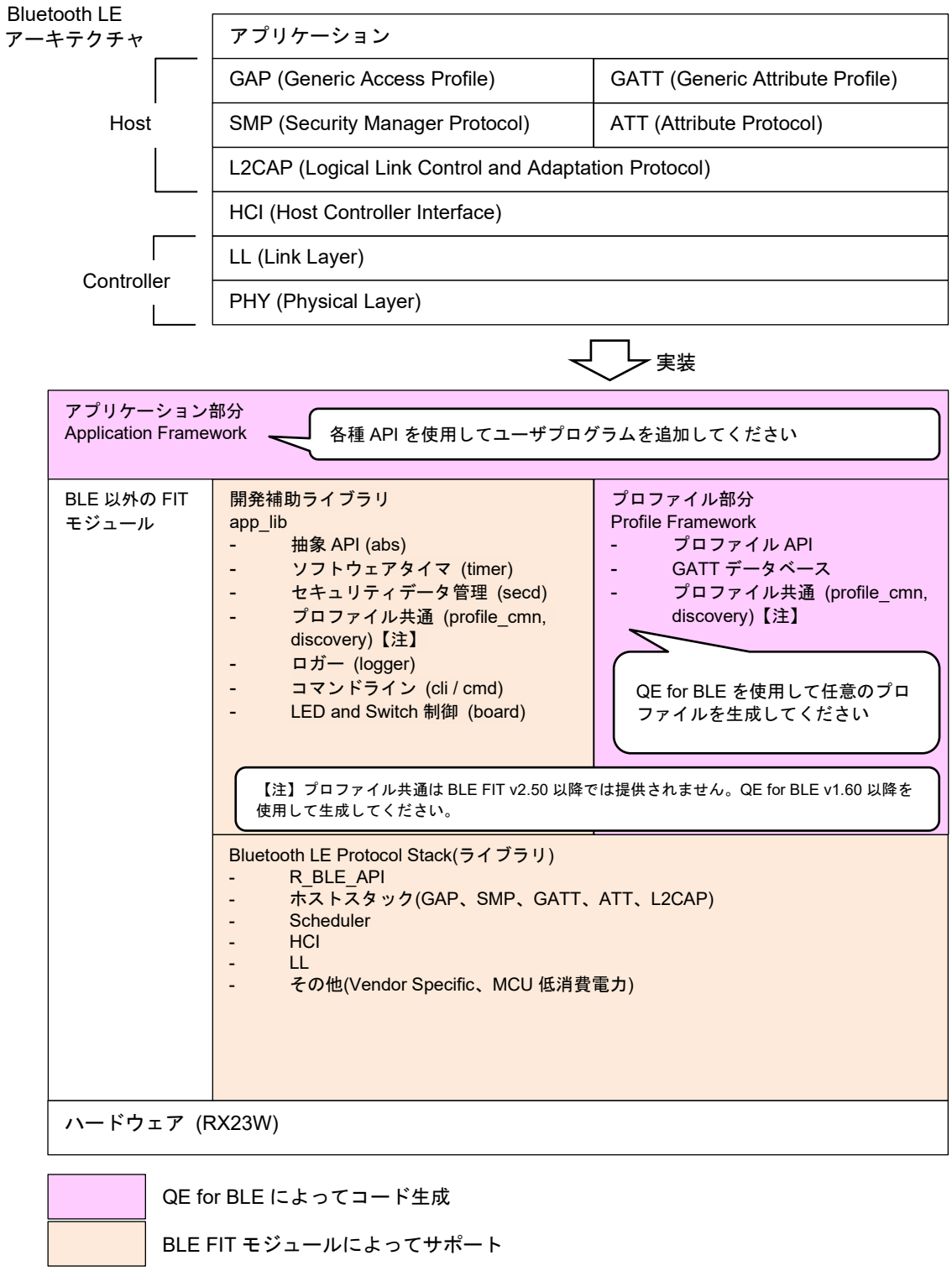


図 1-10　ソフトウェア構成

1.6.1　主要機能

統合開発環境 e²studio において BLE FIT モジュールをプロジェクトに組み込むと、Bluetooth LE のプロトコルやドライバをサポートするライブラリが使用可能になります。アプリケーション部分のスケルトンプログラム(Application Framework)とプロファイル部分(Profile Framework)は QE for BLE でコード生成することができます。各機能ブロックの詳細は表 1.5 のドキュメントを参照してください。

表 1.5　機能ブロック

機能ブロック	参照ドキュメント
BLE FIT モジュール	BLE モジュール Firmware Integration Technology (R01AN4860)
Bluetooth LE Protocol Stack	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)
app_lib	
Profile Framework	Bluetooth Low Energy プロファイル開発者ガイド(R01AN6459)
Application Framework	本ドキュメント
Mesh Stack	Bluetooth メッシュモジュール Firmware Integration Technology (R01AN4930) Bluetooth メッシュスタックパッケージスタートアップガイド (R01AN4874) Bluetooth メッシュスタックパッケージ開発ガイド (R01AN4875) 【注】参考まで

Bluetooth LE Protocol Stack ライブラリと開発補助ライブラリが提供する機能を表 1.6 に示します。

表 1.6　ライブラリが提供する機能

機能	API/マクロ名	インクルードヘッダと使用方法
Bluetooth LE	R_BLE_XXX R_BLE_GAP_XXX R_BLE_GATT_GetMtu R_BLE_GATTS_XXX R_BLE_GATTC_XXX R_BLE_L2CAP_XXX	#include "r_ble_rx23w_if.h" 必須 ● R_BLE_GAP_XXX R_BLE_GAP_Init でコールバック関数を登録すると API の結果を BLE_GAP_EVENT_XXX イベントとして受け取ることができます。 ● R_BLE_GATTS_XXX R_BLE_GATTS_RegisterCb でコールバック関数を登録すると API の結果を BLE_GATTS_EVENT_XXX イベントとして受け取ることができます。 ● R_BLE_GATTC_XXX R_BLE_GATTC_RegisterCb でコールバック関数を登録すると API の結果を BLE_GATTC_EVENT_XXX イベントとして受け取ることができます。 ● R_BLE_L2CAP_XXX R_BLE_L2CAP_RegisterCfPsm でコールバック関数を登録すると API の結果を BLE_L2CAP_EVENT_XXX イベントとして受け取ることができます。 R_BLE_XXX と R_BLE_GATT_GetMtu はコールバック関数の登録は必要ありません。API の結果をすぐに受け取ることができます。R_BLE_XXX_Init、R_BLE_XXX_RegisterCb、R_BLE_GAP_SetPairingParams も API の結果をすぐに受け取ることができます。

機能	API/マクロ名	インクルードヘッダと使用方法
Vendor Specific (VS)	R_BLE_VS_XXX	#include "r_ble_rx23w_if.h" ● フロー制御機能が利用可能 ● デバイス固有データ管理機能はデフォルト無効 (BLE_CFG_DEV_DATA_DF_BLOCK) 【注】自身の BD アドレスをデータフラッシュで管理する機能です。R_BLE_VS_SetBdAddr と R_BLE_VS_GetBdAddr で使用可能 R_BLE_VS_Init でコールバック関数を登録すると API の結果を BLE_VS_EVENT_XXX イベントとして受け取ることができます。

MCU 低消費電力(LPC)	R_BLE_LPC_XXX	#include "r_ble_rx23w_if.h"
		デフォルト有効(BLE_CFG_MCU_LPC_EN)
		コールバック関数の登録は必要ありません。API の結果をすぐに受け取ることができます。

抽象 API	R_BLE_ABS_XXX	#include "abs/r_ble_abs_api.h"
		デフォルト有効(BLE_CFG_ABS_API_EN)
		R_BLE_ABS_Init でコールバック関数を登録すると API の結果を BLE_GAP_EVENT_XXX / BLE_GATTS_EVENT_XXX / BLE_GATTC_EVENT_XXX / BLE_VS_EVENT_XXX イベントとして受け取ることができます。
		抽象 API のコードは変更しないでください。

ソフトウェアタイマ	R_BLE_TIMER_XXX	#include "timer/r_ble_timer.h"
		デフォルト有効(BLE_CFG_SOFT_TIMER_EN)
		抽象 API を使用する場合は、この機能を有効にしてください。
		R_BLE_TIMER_Create でコールバック関数を登録するとタイマの割り込み時にタイミング通知を受け取ることができます。
		【注】使用する方法
		app_main.c で R_BLE_TIMER_Init、R_BLE_TIMER_Create を呼ぶ

セキュリティデータ管理	R_BLE_SECD_XXX	#include "sec_data/r_ble_sec_data.h"
		デフォルト無効(BLE_CFG_EN_SEC_DATA)
		【注】ペアリング時のボンディング情報をデータフラッシュで管理する機能です。
		コールバック関数の登録は必要ありません。API の結果をすぐに受け取ることができます。
		【注】使用する方法
		スマートコンフィグレータの[コンポーネント]タブから r_flash_rx を追加
		[r_ble_rx23w] → [Store Security Data in DataFlash.] を"Enable"、[Data Flash Block for Security Data Management.]を 0～7 に設定([Device specific data block on E2 Data Flash.]とは別のブロックを設定)

プロファイル共通	R_BLE_DISC_XXX R_BLE_SERVC_XXX R_BLE_SERVS_XXX	#include "discovery/r_ble_disc.h" #include "profile_cmn/r_ble_servc_if.h" #include "profile_cmn/r_ble_servs_if.h" QE for BLE で生成される ● R_BLE_DISC_XXX R_BLE_DISC_Start でコールバック関数を登録すると Service Discovery の結果を受け取ることができます。 ● R_BLE_SERVC_XXX R_BLE_SERVC_GattcCb でコールバック関数を登録すると API の結果をイベントとして受け取ることができます。

機能	API/マクロ名	インクルードヘッダと使用方法
		● R_BLE_SERVS_XXX R_BLE_SERVS_GattsCb でコールバック関数を登録すると API の結果をイベントとして受け取ることができます。 ● SERVS で VS のイベントを受け取る機能 R_BLE_VS_Init または R_BLE_ABS_Init で登録された VS コールバック関数で受け取ったイベントデータをそのまま R_BLE_SERVS_VsCb に渡す必要があります。 【注】プロファイル共通は BLE FIT v2.50 以降では提供されません。QE for BLE v1.60 以降を使用して生成してください。

ロガー	BLE_BD_ADDR_STR
	BLE_UUID_STR
	BLE_LOG
	BLE_LOG_ERR
	BLE_LOG_WRN
	BLE_LOG_DBG
		#include "logger/r_ble_logger.h"
		デフォルト有効(BLE_CFG_LOG_LEVEL)
		コールバック関数の登録は必要ありません。

コマンドライン	R_BLE_CLI_XXX
	R_BLE_CMD_AbsGapCb
	R_BLE_CMD_VsCb
	R_BLE_CMD_SetResetCb
		#include "cli/r_ble_cli.h"
		#include "cmd/r_ble_cmd_abs.h"
		#include "cmd/r_ble_cmd_vs.h"
		#include "cmd/r_ble_cmd_sys.h"
		デフォルト無効(BLE_CFG_CMD_LINE_EN)
		R_BLE_CLI_RegisterCmds でコールバック関数を登録するとコマンド入力の割り込み時にイベントを受け取ることができます。
		● 抽象 API のログを出力する機能
		R_BLE_GAP_Init または R_BLE_ABS_Init で登録された GAP コールバック関数で受け取ったイベントデータをそのまま R_BLE_CMD_AbsGapCb に渡す必要があります。
		● VS のログを出力する機能
		R_BLE_VS_Init または R_BLE_ABS_Init で登録された VS コールバック関数で受け取ったイベントデータをそのまま R_BLE_CMD_VsCb に渡す必要があります。
		● リセット後のコールバック関数を登録する機能
		R_BLE_CMD_SetResetCb でコールバック関数を登録すると“ble reset”コマンドや R_BLE_ABS_Reset による Bluetooth LE Protocol Stack のリセット後にタイミング通知を受け取ることができます。
		【注】Target Board で使用する方法
		スマートコンフィグレータの[コンポーネント]タブから r_bsp が v5.40 以降であることを確認、r_sci_rx、r_byteq を追加
		[r_ble_rx23w] → [Enabled/Disabled command line function]を"Enabled"、[SCI CH for command line function]を"8"に設定(SCI8 を使用するため)
		[r_sci_rx] → [Include software support for channel 8]を”Include”、[ASYNC mode TX queue buffer size for channel 8]を”160”、[Transmit end interrupt]を”Enable”、[リソース] → [SCI] → [SCI8] → [RXD8/SMISO8 端子]と[TXD8/SMOSI8 端子]を”使用する”に設定
		app_main.c で gsp_cmds を定義する
		app_main 関数で R_BLE_CLI_Init、R_BLE_CLI_RegisterCmds、

機能	API/マクロ名	インクルードヘッダと使用方法
		R_BLE_CMD_SetResetCb、メインループで R_BLE_CLI_Process を呼ぶ

LED and Switch 制御	R_BLE_BOARD_XXX	#include "board/r_ble_board.h"
		デフォルト無効(BLE_CFG_BOARD_LED_SW_EN)
		R_BLE_BOARD_RegisterSwitchCb でコールバック関数を登録するとスイッチ等の割り込み時にタイミング通知を受け取ることができます。
		【注】Target Board で使用する方法
		スマートコンフィグレータの[コンポーネント]タブから r_gpio_rx、r_irq_rx を追加
		[r_ble_rx23w] → [Enabled/Disabled board LED and Switch control support.]を"Enable"、[Board Type]を"Target board"に設定
		[r_irq_rx] → [Filter for IRQ5]を"Enable"、[Filter clock divisor for IRQ5]を"Divisor1"、[リソース] → [ICU] → [IRQ5 端子]を"使用する"に設定
		app_main 関数で R_BLE_BOARD_Init と R_BLE_BOARD_RegisterSwitchCb を呼ぶ

プロファイル API	R_BLE_[サービス名]_XXX	#include "r_ble_[サービス名].h" QE for BLE で生成される R_BLE_[サービス名]_Init でコールバック関数を登録するとリモートデバイスからの Write、Read、Indication、Notification の受信時にイベントを受け取ることができます。

アプリケーションで使用する機能に応じて Bluetooth LE Protocol Stack ライブラリのタイプを選択することができます。「2.1 コンフィグレーションオプション」で選択できます。機能が制限されたタイプを選択することで ROM/RAM コードサイズを削減することができます。各タイプがサポートする機能を表 1.7 に示します。

表 1.7　Bluetooth LE Protocol Stack のタイプとサポートする機能

Bluetooth LE Feature	Bluetooth LE Protocol Stack のタイプ		
	All features	Balance	Compact
LE 2M PHY	Yes	Yes	No
LE Coded PHY	Yes	Yes	No
LE Advertising Extensions	Yes	No	No
LE Channel Selection Algorithm #2	Yes	Yes	No
High Duty Cycle Non-Connectable Advertising	Yes	Yes	Yes
LE Data Packet Length Extension	Yes	Yes	Yes
LE Secure Connections	Yes	Yes	Yes
Link Layer privacy	Yes	Yes	Yes
Link Layer Extended Scanner Filter policies	Yes	Yes	No
Low Duty Cycle Directed Advertising	Yes	Yes	Yes
32-bit UUID Support in LE	Yes	Yes	Yes
LE L2CAP Connection Oriented Channel Support	Yes	No	No
LE Link Layer Topology	Yes	Yes	No
LE Ping	Yes	Yes	Yes
Bluetooth Low Energy - Enhancements to GAP for Low Energy - - GAP Role	セントラル ペリフェラル オブザーバ ブロードキャスタ	セントラル ペリフェラル オブザーバ ブロードキャスタ	ペリフェラル ブロードキャスタ
Bluetooth Low Energy - Generic Attribute profile (GATT) - - GATT Role	サーバ クライアント	サーバ クライアント	サーバ クライアント

1.6.2　周辺機能

BLE 以外の FIT モジュールを組み込むと、BLE 以外の MCU 機能をより簡単に使用することができます。主に使用する FIT モジュールを表 1.8 に示します。

表 1.8　FIT モジュール

FIT モジュール名	コンポーネント名	コメント
BLE	r_ble_rx23w	Bluetooth LE 基本機能 Bluetooth LE ソフトウェアのために**必須** BLEIRQ, CMT2, CMT3 など Bluetooth LE Protocol Stack が使用する割り込み優先度は 14 固定(SC による変更不可)
BLE QE Utility	r_ble_qe_utility	プロファイル生成機能 QE for BLE のために**必須** **【注】QE for BLE V1.40 以降では FIT モジュールではなく QE for BLE に同梱される**
ボードサポートパッケージ(BSP)	r_bsp	MCU の基本設定 クロック設定等のために**必須** RX23W のフラッシュメモリプロテクト機能を使用可能(フラッシュメモリ書き込みツールでファームウェア書き込みする際にデバイス固有データの書かれているブロックを消去せずに保護できる)
IRQ	r_irq_rx	割り込みイベントの設定・通知 LED and Switch 制御機能で使用される スイッチやセンサ等による割り込みを検知してアプリケーションに通知できる IRQ の割り込み優先度はデフォルト 15 (SC による変更可能)
GPIO	r_gpio_rx	汎用入出力端子の設定・利用 LED and Switch 制御機能で使用される Pin に割り当てられた LED やスイッチ等の入出力に使用できる
LPC	r_lpc_rx	消費電力低減 MCU 低消費電力機能で使用される
フラッシュ	r_flash_rx	内蔵フラッシュメモリの書き換え セキュリティデータ管理機能で使用される デバイス固有データ管理機能で使用される HCI モードのためのオプション(v4.10 以降)
SCI	r_sci_rx	SCI の動作モード設定・利用 コマンドライン機能で使用される SCI の割り込み優先度はデフォルト 15 (SC による変更可能)
CMT	r_cmt_rx	タイマイベントの生成 H/W(RF)の制御のために**必須** ソフトウェアタイマ機能でも使用される CMT0, CMT1 の割り込み優先度はデフォルト 15 (SC による変更可能)
バイト型キューバッファ(BYTEQ)	r_byteq	バイト型リングバッファの設定・管理 コマンドライン機能で使用される
Bluetooth メッシュ	r_mesh_rx23w	メッシュトポロジのサポート メッシュ機能で使用される

1.7　開発の流れ

以下の手順で開発します。QE for BLE を使用し、最小構成のアプリケーション(Advertising を実行するアプリケーション)を作成する手順を説明します。標準の手順については「BLE モジュール Firmware Integration Technology (R01AN4860)」の「4. BLE FIT モジュールのプロジェクト」を参照してください。

(1)　統合開発環境 e²studio、スマートコンフィグレータ(SC)、QE for BLE をインストール
(2)　e²studio 上でプロジェクトを作成
　Target Board for RX23W では R5F523W8AxNG を指定してください。
　Target Board for RX23W module では R5F523W8CxLN を指定してください。
(3)　SC で FIT モジュールと QE for BLE のコンポーネントを追加、設定変更し、コード生成
　最小構成のコンポーネント設定を表 1.9 に示します。

表 1.9　最小構成のコンポーネント設定

手順	標準の手順	最小構成の手順
クロック設定	**必須**	**必須**
r_ble_rx23w 追加	**必須**	**必須**
r_ble_rx23w 変更	実施	下記を無効のままにする場合は不要 コマンドライン機能(BLE_CFG_CMD_LINE_EN) セキュリティデータ管理機能(BLE_CFG_EN_SEC_DATA) デバイス固有データ管理機能(BLE_CFG_DEV_DATA_DF_BLOCK) LED and Switch 制御機能(BLE_CFG_BOARD_LED_SW_EN) **【注】MCU 低消費電力機能(BLE_CFG_MCU_LPC_EN)を無効に変更**
r_ble_qe_utility 追加	オプション	**必須** **【注】QE for BLE V1.40 以降では不要**
BLE Profile Creation 追加	オプション	**必須** **【注】QE for BLE V1.40 以降では不要**
BLE Profile Creation 変更	オプション	Advertising を実行する設定がデフォルトのため不要
r_bsp 変更	実施	RX23W のフラッシュメモリプロテクト機能使用しない場合は不要 **【注】v5.40 以降に変更**
r_irq_rx 追加・変更	実施	LED and Switch 制御機能(BLE_CFG_BOARD_LED_SW_EN)を無効のままにする場合は不要
r_gpio_rx 追加	実施	LED and Switch 制御機能(BLE_CFG_BOARD_LED_SW_EN)を無効のままにする場合は不要
r_lpc_rx 追加	実施	MCU 低消費電力機能(BLE_CFG_MCU_LPC_EN)を無効に変更する場合は不要
r_flash_rx 追加	実施	セキュリティデータ管理機能(BLE_CFG_EN_SEC_DATA)とデバイス固有データ管理機能(BLE_CFG_DEV_DATA_DF_BLOCK)を無効のままにする場合は不要
r_sci_rx 追加・変更	実施	コマンドライン機能(BLE_CFG_CMD_LINE_EN)を無効のままにする場合は不要
r_cmt_rx 追加	**必須**	**必須**
r_byteq 追加	実施	コマンドライン機能(BLE_CFG_CMD_LINE_EN)を無効のままにする場合は不要

(4)　コード生成後の設定

CMT FIT モジュールが v4.50 より古い場合、r_cmt_rx.c 内の CMT_RX_NUM_CHANNELS の定義の後に以下を追加します。

```
#if defined(BSP_MCU_RX23W)
#undef CMT_RX_NUM_CHANNELS
#define CMT_RX_NUM_CHANNELS (2)
#endif /* BSP_MCU_RX23W */
```

コード 1-1　CMT_RX_NUM_CHANNELS の変更

BLE FIT モジュールの app_lib/board/r_ble_board.c の変更は LED and Switch 制御機能(BLE_CFG_BOARD_LED_SW_EN)を無効のままにする場合は不要です。

(5)　e²studio 上でリンカ設定とデバッグ設定

[プロジェクト] → [プロパティ] → [C/C++ Build] → [Settings] → [Tool Settings] → [Linker] → [セクション] → [...]のセクション設定画面に下記を追加します。

RAM : BLE_B*、BLE_R*

ROM : BLE_C*、BLE_D*、BLE_W*、BLE_L、BLE_P

[プロジェクト] → [プロパティ] → [C/C++ Build] → [Settings] → [Tool Settings] → [Linker] → [セクション] → [シンボル・ファイル] → [ROM から RAM へマップするセクション]に下記を追加します。

BLE_D=BLE_R

BLE_D_1=BLE_R_1

BLE_D_2=BLE_R_2

[プロジェクト] → [プロパティ] → [C/C++ Build] → [Settings] → [Build Steps] → [Pre-build steps] → [Command(s):]に以下を入力します。

..¥src¥smc_gen¥r_ble_rx23w¥lib¥ble_fit_lib_selector.bat

RX23W のフラッシュメモリプロテクト機能が有効のファームウェアが書かれているボードに接続する場合、[実行] → [デバッグの構成] → [Renesas GDB Hardware Debugging] → [(プロジェクト名) HardwareDebug] → [Debugger] → [Connection Settings] → [フラッシュ] → [ID コード] を"45FFFFFFFFFFFFFFFFFFFFFFFFFFFF"に変更します。

Target Board for RX23W では[実行] → [デバッグの構成] → [Renesas GDB Hardware Debugging] → [(プロジェクト名) HardwareDebug] → [Debugger] → [Connection Settings] → [電源] → [エミュレーターから電源を供給する(MAX 200mA)]の変更は不要です。

(6)　生成されたコードの使用

プロジェクトの src¥[プロジェクト名].c の main()から app_main()を呼ぶようにしてください。

```
#include "r_smc_entry.h"

void main(void);
void app_main(void);

void main(void)
{
        app_main();
}
```

コード 1-2　main 関数内での app_main のコール

(7)　コード追加・変更

次章以降を参考にして任意のアプリケーションを開発してください。

1.8　本ドキュメントの使用例

PC やスマートフォン等のセントラルからコネクションされてペリフェラルとなり、GATT サーバとして動作するアプリケーションが一般的です。以下が基本的なアプリケーションとその処理です。

表 1.10　基本的なアプリケーションとその処理

アプリケーション	処理	説明
GATT サーバ	Advertising	「5 Advertising」を参照してください。
	コネクション	「3.3 GAP のイベント(gap_cb 関数)」を参照してください。 セントラルからのコネクションリクエストを受信すると Bluetooth LE Protocol Stack によって自動的にコネクションが確立され、BLE_GAP_EVENT_CONN_IND が通知されます。
	ペアリング	「9 セキュリティ」を参照してください。
	データ通信 (Notification)	「8 通信」を参照してください。
GATT クライアント	スキャン	「6 スキャン」を参照してください。
	コネクション	「7 コネクション」を参照してください。
	ペアリング	「9 セキュリティ」を参照してください。
	データ通信 (Read、Write)	「8 通信」を参照してください。

その他、RX23W で各種 FIT モジュールと Bluetooth LE の機能を利用するアプリケーションの例を以下に示します。

産業機器の稼働ログや健康機器のセンサデータを収集し、PC やスマートフォン等のクライアントにアップロードする GATT サーバアプリケーション
→「2.4 消費電流が最も小さくなる構成」「7.3 複数台コネクション」「9 セキュリティ」を参照してください。

PC やスマートフォン等のクライアントからダウンロードされたデータを転送し、ファームウェアアップデートする GATT サーバアプリケーション
→「8.6 高スループット通信」「9 セキュリティ」を参照してください。

プリンタやスキャナ等のイメージデータや録音機器のボイスデータやオーディオデータを PC やスマートフォン等のクライアントにアップロードする、クライアントから設定データをダウンロードする GATT サーバアプリケーション
→「8.6 高スループット通信」を参照してください。

スマートフォン等の複数のクライアントから操作される電子錠、OA 機器、民生機器等の GATT サーバアプリケーション
→「7.3 複数台コネクション」「9 セキュリティ」を参照してください。

複数のセンサデータを周期的に発信するビーコンアプリケーション
→「5.7 ビーコンとして使用」を参照してください。

1.9　セクションの配置

FW アップデートなどの使用目的時に Bluetooth LE Protocol Stack とアプリケーションのセクション配置を分割できるようにするために、Bluetooth LE Protocol Stack のライブラリおよびライブラリから使用する公開ソースコードのセクションは接頭文字"BLE_"のセクション名に変更しています。

デモプロジェクトでの RX23W(R5F523W8Axxx)の RAM、Data Flash ROM(DF)、Code Flash ROM(CF)のメモリマップと e² studio 上でリンカに設定されるセクション配置を以下に示します。

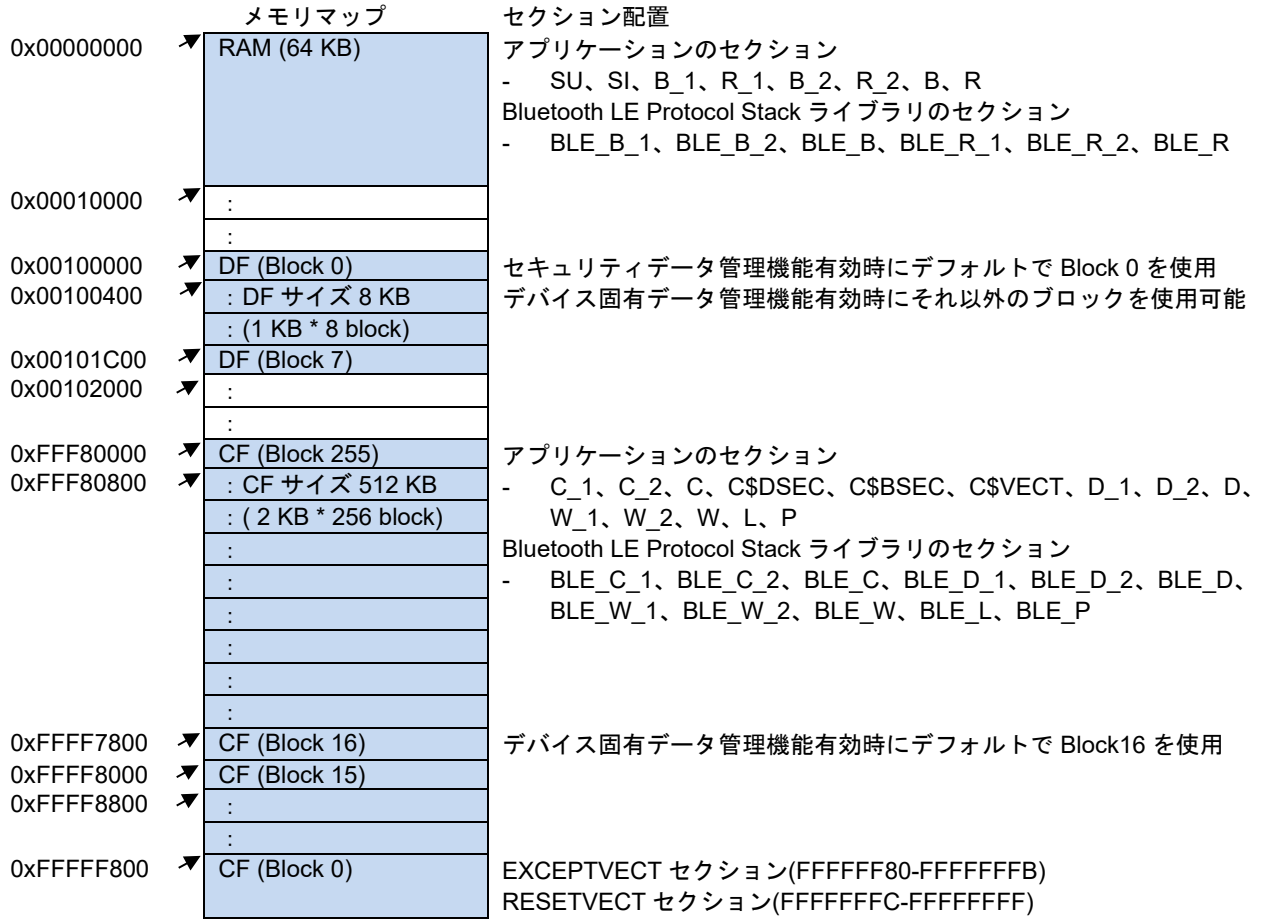


図 1-11　セクションの配置

リンカ設定については「1.7 開発の流れ」を参照してください。

実際のセクション配置は MAP ファイルで確認できます。MAP ファイルについては「11.6.3 MAP ファイルの詳細出力」を参照してください。

【注】RX23W のスタートアッププログラム保護機能を使用する場合、CF の Block 0 から Block 15 までが保護されます。そのため、BD アドレスなどのデバイス固有データを書き込むブロック(BLE_CFG_DEV_DATA_CF_BLOCK)はデフォルトで 16 が指定されています。BD アドレスについては「2.3 BD アドレスの設定方法」を参照してください。

2. コンフィグレーションオプション調整

2.1 コンフィグレーションオプション

BLE FIT モジュールのコンフィグレーションオプションの設定は r_ble_rx23w_config.h で行います。スマートコンフィグレータ(SC)を使用する場合は、ソフトウェアコンポーネント設定画面でコンフィグレーションオプションを設定できます。設定値はモジュールを追加する際に、自動的に r_ble_rx23w_config.h に反映されます。マクロ名および、SC 表示名と設定範囲を表 2.1 に示します。
【注】r_ble_rx23w_config.h を直接編集する場合、[プロジェクト]-[プロパティ]-[ビルダー]-[SC Code Generation Builder]の設定によっては、プロジェクトをビルドした時のコード生成によって編集内容が上書きされることがあります。SC からの設定変更を推奨します。

表 2.1　コンフィグレーションオプション

マクロ名 (SC 表示)	設定範囲 (デフォルト値)	説明
BLE_CFG_LIB_TYPE (Type of Bluetooth LE Protocol Stack library)	0: All features, 1: Balance, 2: Compact (0)	Bluetooth LE Protocol Stack のタイプ
BLE_CFG_RF_DBG_PUB_ADDR (Initial Public Address)	任意の値を設定します。 ({0xFF,0xFF,0xFF,0x50,0x90,0x74})	BLE FIT モジュールに設定される Public Address の初期値
BLE_CFG_RF_DBG_RAND_ADDR (Initial Static Address)	任意の値を設定します。 ({0xFF,0xFF,0xFF,0xFF,0xFF,0xFF})	BLE FIT モジュールに設定される Static Address の初期値
BLE_CFG_RF_CONN_MAX (Maximum number of connections)	1～7 (7)	同時最大コネクション数
BLE_CFG_RF_CONN_DATA_MAX (Maximum connection data length)	27～251 (251)	最大パケットデータサイズ(バイト)
BLE_CFG_RF_ADV_DATA_MAX (Maximum advertising data length)	31～1650 (1650)	最大 Advertising データサイズ(バイト)
BLE_CFG_RF_ADV_SET_MAX (Maximum advertising set number)	1～4 (4)	最大 Advertising セット数
BLE_CFG_RF_SYNC_SET_MAX (Maximum advertising set number)	1～2 (2)	最大 Periodic Sync セット数
BLE_CFG_EVENT_NOTIFY_CONN_START (Connection event start notify)	0～1 (0)	コネクションイベントの開始割り込み発生通知機能の設定
BLE_CFG_EVENT_NOTIFY_CONN_CLOSE (Connection event close notify)	0～1 (0)	コネクションイベントの完了割り込み発生通知機能の設定
BLE_CFG_EVENT_NOTIFY_ADV_START (Advertising event start notify)	0～1 (0)	Advertising イベントの開始割り込み発生時通知機能の設定
BLE_CFG_EVENT_NOTIFY_ADV_CLOSE (Advertising event close notify)	0～1 (0)	Advertising イベントの完了割り込み発生時通知機能の設定
BLE_CFG_EVENT_NOTIFY_SCAN_START (Scanning event start notify)	0～1 (0)	スキャンイベントの開始割り込み発生通知機能の設定
BLE_CFG_EVENT_NOTIFY_SCAN_CLOSE (Scanning event close notify)	0～1 (0)	スキャンイベントの完了割り込み発生通知機能の設定
BLE_CFG_EVENT_NOTIFY_INIT_START (Initiating event start notify)	0～1 (0)	Initiator イベントのスキャン開始割り込み発生通知機能の設定
BLE_CFG_EVENT_NOTIFY_INIT_CLOSE (Initiating event close notify)	0～1 (0)	Initiator イベントのスキャン完了割り込み発生通知機能の設定

マクロ名 (SC 表示)	設定範囲 (デフォルト値)	説明
BLE_CFG_EVENT_NOTIFY_DS_START (RF_DEEP_SLEEP start notify)	0～1 (0)	RF_DEEP_SLEEP start 通知機能の設定
BLE_CFG_EVENT_NOTIFY_DS_WAKEUP (RF_DEEP_SLEEP wakeup notify)	0～1 (0)	RF_DEEP_SLEEP wakeup 通知機能の設定
BLE_CFG_RF_CLVAL (Capacity adjustment of 32MHz crystal resonator)	0～15 (6)	32MHz 水晶振動子の調整値
BLE_CFG_RF_DDC_EN (DC-DC converter configuration for RF part)	0～1 (0)	RF 部の DC-DC の設定
BLE_CFG_RF_EXT32K_EN (Slow clock source for RF part)	0～1 (0)	RF 部への slow clock source の設定
BLE_CFG_RF_MCU_CLKOUT_PORT (MCU CLKOUT port)	0～1 (0)	MCU CLKOUT のポートの設定
BLE_CFG_RF_MCU_CLKOUT_FREQ (MCU clock frequency)	0～1 (0)	MCU の CLKOUT からの出力周波数の設定
BLE_CFG_RF_SCA (Sleep Clock Accuracy(SCA) for RF slow clock)	0～500 (250)	RF slow clock 用の Sleep Clock Accuracy(SCA)の設定
BLE_CFG_RF_MAX_TX_POW (Transmission power maximum value)	0～1 (1)	最大送信パワーの設定
BLE_CFG_RF_DEF_TX_POW (Default transmit power)	0～1 (0)	デフォルトの送信パワーの設定
BLE_CFG_RF_CLKOUT_EN (CLKOUT_RF output setting)	以下のいずれかの値を選択 0: No output 5: 4MHz output 6: 2MHz output 7: 1MHz output (0)	CLKOUT_RF 出力の設定
BLE_CFG_RF_DEEP_SLEEP_EN (RF_DEEP_SLEEP transition)	0～1 (1)	RF Deep Sleep 機能の設定
BLE_CFG_MCU_MAIN_CLK_KHZ (MCU Main Clock Frequency (kHz))	● HOCO の場合、この値は無視されます。 ● メインクロックの場合、1000～20000 ● PLL Circuit の場合、4000～12500 (4000)	MCU メインクロック周波数(kHz)
BLE_CFG_DEV_DATA_CF_BLOCK (Device specific data block on Code Flash (ROM))	-1～255 (16)	デバイス固有データを格納する Code Flash(ROM)の Block
BLE_CFG_DEV_DATA_DF_BLOCK (Device specific data block on E2 Data Flash)	-1～7 (-1)	デバイス固有データを格納する Data Flash の Block
BLE_CFG_GATT_MTU_SIZE (MTU Size configured by GATT MTU exchange procedure)	23～247 (247)	GATT 通信で使用する MTU サイズ(バイト)
BLE_CFG_NUM_BOND (Number of remote device bonding information)	1～7 (7)	保存するボンディング情報の数
BLE_CFG_EN_SEC_DATA (Store Security Data in DataFlash)	0～1 (0)	セキュリティデータ管理機能の設定

マクロ名 (SC 表示)	設定範囲 (デフォルト値)	説明
BLE_CFG_SECD_DATA_DF_BLOCK (Data Flash Block for Security Data Management)	0～7 (0)	ボンディング情報の保存する Data Flash Block
BLE_CFG_CMD_LINE_EN (Enabled/Disabled command line function)	0～1 (0)	コマンドライン機能の設定
BLE_CFG_CMD_LINE_CH (SCI CH for command line function)	1 or 5 or 8 (1)	コマンドライン機能で使用する SCI のチャネル
BLE_CFG_BOARD_LED_SW_EN (Enabled/Disabled board LED and Switch control support)	0～1 (0)	ボードの LED と SW の制御コードの設定
BLE_CFG_BOARD_TYPE (Board Type)	0～3 (0)	ボードのタイプ
BLE_CFG_LOG_LEVEL (Log level)	0～3 (3)	ログレベル
BLE_CFG_ABS_API_EN (Abstraction API support)	0～1 (1)	抽象 API の設定
BLE_CFG_SOFT_TIMER_EN (Software Timer support)	0～1 (1)	app_lib が提供するソフトウェアタイマの設定
BLE_CFG_MCU_LPC_EN (MCU low power consumption control support)	0～1 (1)	MCU の低消費電力機能の設定
BLE_CFG_HCI_MODE_EN (HCI mode support)	0～1 (0)	HCI モードでの起動の設定

2.2　RAM の調整方法

コンフィグレーションオプションによっては、RAM サイズに影響を与えるものがあります。表 2.2 にコンフィグレーションオプションの値を 1 加算したときの RAM サイズの追加サイズを示します。

表 2.2　コンフィグレーションオプション設定時の RAM の追加サイズ

コンフィグレーションオプション		設定範囲 （デフォルト値）	ライブラリ	追加サイズ （バイト）
設定内容	マクロ名(SC 表示)			
最大コネクション数	BLE_CFG_RF_CONN_MAX (Maximum number of connections)	1～7 (7)	All features	1094
			Balance	1086
			Compact	1074
最大データパケット長	BLE_CFG_RF_CONN_DATA_MAX (Maximum connection data length)	27 ～ 251 (251)	全ライブラリ	9
最大 Advertising data 長	BLE_CFG_RF_ADV_DATA_MAX (Maximum advertising data length)	31 ～ 1650 (1650)	All features	表 2.3 に記載
最大 Advertising set 数[*1]	BLE_CFG_RF_ADV_SET_MAX (Maximum advertising set number)	1 ～ 4 (4)	All features	308
最大 sync set 数[*2]	BLE_CFG_RF_SYNC_SET_MAX (Maximum periodic sync set number)	1 ～ 2 (2)	All features	66

[*1]：同時に実行可能な Advertising 数
[*2]：periodic synchronization の最大確立数

最大 Advertising data 長は最大 Advertising set 数の値によって、追加サイズが異なります。表 2.3 は最大 Advertising data 長が 0-252 に設定したときのサイズをベースに、最大 Advertising data 長を変更した場合の追加サイズを示しています。

表 2.3　最大 Advertising data 長と最大 Advertising set 数の設定による RAM の追加サイズ

最大 Advertising set 数									
	1	BLE_CFG_RF_ADV_DATA_MAX	0-252	253-504	505-756	757-1008	1009-1260	1261-1512	1513-1650
		追加サイズ(バイト)	0	512	1024	1536	2048	2560	3072
	2	BLE_CFG_RF_ADV_DATA_MAX	0-252	253-504	505-756	757-1008	1009-1260	1261-1512	1513-1650
		追加サイズ(バイト)	0	1024	2048	3072	4096	5120	6144
	3	BLE_CFG_RF_ADV_DATA_MAX	0-252	253-504	505-756	757-1008	1009-1260	1261-1650	
		追加サイズ(バイト)	0	1536	3072	4608	6144	7680	
	4	BLE_CFG_RF_ADV_DATA_MAX	0-252	253-504	505-756	757-1008	1009-1650		
		追加サイズ(バイト)	0	2048	4096	6144	7168		

最大 Advertising data 長と最大 Advertising set 数の値は以下の範囲に収まるように設定してください。

4250 >= 最大 Advertising data 長 × 最大 Advertising set 数

2.3　BD アドレスの設定方法

Bluetooth Device アドレス(BD アドレス)には以下の種類があります。

表 2.4　BD アドレスの種類

BD アドレスの種類			説明
Public device address			Public Address は上位 24 ビットを IEEE から取得します。 【注】Bluetooth Core Specification Vol 2, PartB, "1.2 Bluetooth Device Addressing"を参照してください。MAC アドレスと同様の手順で取得可能です。
Random device address	Static address		最上位ビットが 11 で始まり、残りのビットをランダムに設定して使用できる Random Address Cx:xx:xx:xx:xx:xx or Dx:xx:xx:xx:xx:xx or Ex:xx:xx:xx:xx:xx or Fx:xx:xx:xx:xx:xx 【注】Bluetooth Core Specification Vol 6, PartB, "1.3.2 Random Device Address"を参照してください。 【注】Bluetooth LE Protocol Stack ではフォーマットチェックは行っていません。 【注】スタティックアドレスは乱数値で構成されるため、他デバイスと値が重複する可能性はゼロではありません。
	Private address	Non-resolvable private address	最上位ビットが 00 で始まり、残りのビットが動的に再生成される Random Address 0x:xx:xx:xx:xx:xx or 1x:xx:xx:xx:xx:xx or 2x:xx:xx:xx:xx:xx or 3x:xx:xx:xx:xx:xx
		Resolvable private address (RPA)	最上位ビットが 01 で始まり、残りのビットが動的に再生成され、プライバシ機能によってセキュリティを高めた Random Address 4x:xx:xx:xx:xx:xx or 5x:xx:xx:xx:xx:xx or 6x:xx:xx:xx:xx:xx or 7x:xx:xx:xx:xx:xx

Bluetooth デバイスは Identity Address を持ちます。Identity Address は Public device address か Static Address のいずれかになります。プライバシ機能を使うデバイスは Identity Address が必要となります。

RX23W では、静的な BD アドレスである Public device address と Static Address を内蔵 ROM のデータ領域とユーザ領域に格納する機能(デバイス固有データ管理機能)を提供します。データ領域としてデータフラッシュ(DF)、ユーザ領域としてコードフラッシュ(CF)が使用可能です。デフォルトでは[r_ble_rx23w]のコンフィグレーションオプションによって以下の設定になっています。

表 2.5　BD アドレスの設定

コンフィグレーションオプション	初期値
BLE_CFG_DEV_DATA_DF_BLOCK	-1 (DF を使用しない)
BLE_CFG_DEV_DATA_CF_BLOCK	16 (CF のブロック 16 を使用する)
BLE_CFG_RF_DBG_PUB_ADDR	74:90:50:FF:FF:FF (Public Address のファームウェア初期値)
BLE_CFG_RF_DBG_RAND_ADDR	FF:FF:FF:FF:FF:FF (Random Address のファームウェア初期値)

BD アドレスは Advertising 開始時などに Public Address と Random Address のいずれかを選んで使用可能です。設定した Random Address の使用方法は「2.3.2 Random Address 使用方法」を参照してください。

BD アドレスはアプリケーション起動時に R_BLE_Open 内で「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)」の「5.4.6　BD アドレスの決定方法」によって以下のように決定され、Bluetooth LE Protocol Stack の管理 RAM に格納されます。

表 2.6　BD アドレスの決定方法

優先度	BD アドレスの決定方法	初期値	説明
1	DF を使用する (BLE_CFG_DEV_DATA_DF_BLOCK を 0～7 に設定する) 【注】 BLE_CFG_SECD_DATA_DF_BLOCK(デフォルト 0)で指定した Block と別の Block を指定してください。 【注】”-1”を指定すると DF を使用しません。	フラッシュ初期化状態 [Public Address] FF:FF:FF:FF:FF:FF [Random Address] FF:FF:FF:FF:FF:FF 【注】ALL 0x00 or 0xFF の場合は無効	製品出荷後に UART 等の外部 IF 経由で BD アドレスを書き込む場合に使用してください。 以下のいずれかの方法で書き換えできます。 - R_BLE_VS_SetBdAddr で DF を選択して書き換える - HCI モードファームウェアの場合は BDAddrWriter で書き換える 【注】フラッシュ FIT モジュールを有効にしてください。 【注】RX23W をリセットすると変更が反映されます。
2	CF を使用する (BLE_CFG_DEV_DATA_CF_BLOCK を 0～255 を設定する) 【注】"0"から"15"は Start-Up Program Protection block となっていますので、Start-Up Program Protection 機能を使用する場合は、”0”から”15”を指定しないでください。 【注】”-1”を指定すると CF を使用しません。	フラッシュ初期化状態 [Public Address] FF:FF:FF::FF:FF:FF [Random Address] FF:FF:FF:FF:FF:FF 【注】ALL 0x00 or 0xFF の場合は無効	製品出荷時にファームウェアとともに BD アドレスを書き込む場合に使用してください。 以下の方法で書き換えできます。 - Renesas Flash Programmer(RFP)のユニークコード機能を使用してファームウェアとともに BD アドレスをを書き込む 【注】RX23W のメモリプロテクション機能を使用して、第三者から書き換えられないようにガードすることができます。
3	ファームウェア初期値を使用する BLE_CFG_RF_DBG_PUB_ADDR BLE_CFG_RF_DBG_RAND_ADDR	[Public Address] 74:90:50:FF:FF:FF [Random Address] FF:FF:FF:FF:FF:FF 【注】ALL 0x00 or 0xFF の場合は無効	デバッグ時の一時的な BD アドレスを変更する場合に使用してください。
4	固定値を使用する	[Public Address] 74:90:50:FF:FF:FF [Random Address] XX:XX:XX:XX:XX:XX	上記がすべて無効(ALL 0x00 or 0xFF)の場合に使用されます。 Random Address は MCU のユニーク ID から自動生成します。
その他	Bluetooth LE Protocol Stack の管理 RAM を書き換えて使用する	[Public Address] XX:XX:XX:XX:XX:XX [Random Address] XX:XX:XX:XX:XX:XX	アプリケーションで BD アドレスを管理する場合に使用してください。 BLE_GAP_EVENT_STACK_ON 以降、R_BLE_VS_SetBdAddr で Current register を選択して書き換えが行えます。

データ領域の BD アドレスの書き換えの詳細は「2.3.1 データ領域への書き込み」を参照してください。

ユーザ領域の書き換えと RX23W のメモリプロテクション機能の詳細は「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)」の「5.4.3 ユーザ領域(ROM)への書き込み」と「5.4.5 RX23W のフラッシュメモリプロテクト機能」を参照してください。

2.3.1 データ領域への書き込み

データ領域へは R_BLE_VS_SetBdAddr()を使用して書き込みます。HCI モードファームウェアの場合はパブリック BD アドレス書き込みツール(BDAddrWriter)を使用して書き込みます。BDAddrWriter については「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)」の「5.4.4.2 BDAddrWriter による書き込み」を参照してください。RX23W を一度リセットすることで書き込まれた BD アドレスが使用されます。

2.3.2 Random Address 使用方法

以下は R_BLE_Open で決定した Random Address で Advertising するサンプルです。
R_BLE_VS_GetBdAddr で選択された Random Address を取得し、BLE_VS_EVENT_GET_ADDR_COMP イベントで取得した Random Address を使用して R_BLE_ABS_StartLegacyAdv をコールします。

```
static st_ble_abs_legacy_adv_param_t gs_adv_param =
{
    (省略)
    .o_addr_type     = BLE_GAP_ADDR_RAND,
};

static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch (type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            R_BLE_VS_GetBdAddr(BLE_VS_ADDR_AREA_REG, BLE_GAP_ADDR_RAND);
        } break;
        (省略)

static ble_status_t ble_app_init(void);
static void vs_cb(uint16_t type, ble_status_t result, st_ble_vs_evt_data_t *p_data)
{
    (省略)
    switch (type)
    {
        case BLE_VS_EVENT_GET_ADDR_COMP:
        {
            st_ble_vs_get_bd_addr_comp_evt_t * p_get_addr =
                (st_ble_vs_get_bd_addr_comp_evt_t *)p_data->p_param;
            memcpy(gs_adv_param.o_addr, p_get_addr->addr.addr, BLE_BD_ADDR_LEN);
            R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
        } break;
        (省略)
```

コード 2-1　Random Address 使用のサンプル

コマンドライン機能を使用すると、”vs addr get df”コマンドと”vs addr set df”で DF の BD アドレスの確認と書き換えができます。”vs addr get curr”と”vs addr set curr”で Bluetooth LE Protocol Stack の管理 RAM の BD アドレスの確認と書き換えができます。

```
$ vs addr get curr pub
$ BLE_VS_EVENT_GET_ADDR_COMP result:0x0000, param_len:8
  addr:36:35:34:33:32:31 pub on current register

$ vs addr get df pub
$ BLE_VS_EVENT_GET_ADDR_COMP result:0x0000, param_len:8
  addr:36:35:34:33:32:31 pub on data flash

$ vs addr get curr rnd
$ BLE_VS_EVENT_GET_ADDR_COMP result:0x0000, param_len:8
  addr:D9:7C:E6:81:83:35 rnd on current register

$ vs addr get df rnd
$ BLE_VS_EVENT_GET_ADDR_COMP result:0x0000, param_len:8
  addr:FF:FF:FF:FF:FF:FF rnd on data flash
```

2.4　消費電流が最も小さくなる構成

以下の構成で消費電流が最も小さくなります。

表 2.7　消費電流が最も小さくなる構成

コンフィグレーションオプション		コメント
MCU クロック設定	HOCO クロック：有効 周波数 32MHz	【注】スマートコンフィグレータの[クロック設定]タブで設定してください。 【注】使用しないクロックは無効にするか、またはクロック周波数を最小値まで下げてください。 【注】R_BLE_Open でクロック周波数に合わせて R_BLE_Execute の動作を最適化しています。クロック周波数を動的に変更した場合は、R_BLE_Close 後に R_BLE_Open を再度呼んでください。
	FCLK：x1 (32MHz)	
	ICLK：x1 (32MHz)	
	PCLKB：x1 (32MHz)	
r_ble_rx23w コンポーネント設定	RF 部の DC-DC コンバータ：有効 (BLE_CFG_RF_DDC_EN=1)	【注】「Bluetooth 基板設計ガイドライン アプリケーションノート (R01AN4534)」を参照してください。
	RF 部のスリープ機能：有効 (BLE_CFG_RF_DEEP_SLEEP_EN=1)	
	MCU 低消費電力機能：有効 (BLE_CFG_MCU_LPC_EN=1)	【注】メインループの R_BLE_Execute API コール後に R_BLE_LPC_EnterLowPowerMode API をコールする必要があります。
	CLKOUT_RF：出力なし (BLE_CFG_RF_CLKOUT_EN=0)	
	コマンドライン機能：無効 (BLE_CFG_CMD_LINE_EN=0)	
	LED/スイッチ制御機能：無効 (BLE_CFG_BOARD_LED_SW_EN=0)	
	RF 最大送信パワー：+4dBm → +0dBm (BLE_CFG_RF_MAX_TX_POW=0)	
	RF デフォルト送信パワー：High → Mid → Low (BLE_CFG_RF_DEF_TX_POW=2)	【注】RF 送信パワーを下げることで送信電流を下げることができますが、その分通信到達距離が短くなります。

2.4.1　MCU 低消費電力機能の使用

MCU の低消費電力状態への移行は BLE 機能を使用時においても可能です。低消費電力状態への移行の基本方針は以下の通りです。

- R_BLE_Execute()の実行完了後、次の R_BLE_Execute()を実行するまでの期間に、Bluetooth LE Protocol Stack は MCU の低消費電力状態への移行を阻害しません。
- BLE 機能を含め、使用するすべてのコンポーネントが MCU を低消費電力状態に移行しても問題ないことを確認した上で、アプリケーションが MCU を低消費電力状態に移行します。

低消費電力のサンプルとして、以下の機能を持ったプログラム(r_ble_pf_lowpower.c)を提供します。

- MCU の低消費電力状態への移行は、LPC FIT モジュールを使用します。
- 低消費電力状態として、スリープモード、ディープスリープモード、ソフトウェアスタンバイモードを用意しています。
- 低消費電力機能の初期化は R_BLE_LPC_Init()で行います。
- 低消費電力状態への移行は R_BLE_LPC_EnterLowPowerMode()で行います。
  この関数は R_BLE_Execute()の実行完了後にコールし、以下を実行します。
  - 割り込みの無効化
  - 各コンポーネントが低消費電力状態に移行しても問題ないことを確認
  - 各コンポーネントの低消費電力状態への移行処理を実施
  - MCU を低消費電力状態に移行
  - MCU が低消費電力状態から復帰後、各コンポーネントの通常状態への復帰処理を実施
- Bluetooth LE 通信が発生した場合、RF からの割り込みにより低消費電力状態から復帰します。
  ただし、割り込みの無効化処理中に RF からの割り込みが発生する可能性があるため、割り込み無効化後に一度 BLE タスクの状態のチェックを行い、BLE タスクがある場合は MCU の低消費電力状態への移行をスキップするようにしてください。

各低消費電力状態における各コンポーネントの動作状態は関連ドキュメント” R01UH0823 RX23W グループ ユーザーズマニュアル ハードウェア編”の”11. 消費電力低減機能”の表 11.2 に記載しています。

BLE 機能以外のコンポーネントについて、低消費電力状態への移行、復帰の処理を追加したい場合、r_ble_pf_lowpower.c の以降に示す処理を変更してください。

(1)　低消費電力状態への移行チェック処理

- ソフトウェアスタンバイモード

  check_software_standby()関数内にコンポーネントがソフトウェアスタンバイモードに移行しても問題ない状態かどうかをチェックする処理を追加します。コード 2-2 の" /* add check for other components */"の位置に処理を追加してください。

```
static bool check_software_standby(void)
{
    if (g_inhibit_software_standby)
    {
        return false;
    }

    /* 省略  */

    /* If DTC/DMAC/DataFlash is in active, MCU can not enter software standby.
       This code is copied from r_lpc_rx23w.c lpc_lowpower_activate_check. */
    if ((0x0000 != (FLASH.FENTRYR.WORD & 0x0081)) ||
        ((0 == SYSTEM.MSTPCRA.BIT.MSTPA28) &&
         ((1 == DTC.DTCST.BIT.DTCST) || (1 == DMAC.DMAST.BIT.DMST))))
    {
        return false;
    }

    /* add check for other components */

    return true;
}
```

コード 2-2　ソフトウェアスタンバイモードへの移行をチェックする場所

- ディープスリープモード

  check_deep_sleep()関数内にウォッチドックタイマ(WDT)使用時にディープスリープモードに移行しても問題ない状態かどうかをチェックする処理を追加します。ウォッチドックタイマ(WDT)以外のコンポーネントは追加する必要はありません。コード 2-3 の" /* add check for other components */"の位置に処理を追加してください。

```
static bool check_deep_sleep(void)
{
    /* If DTC/DMAC/DataFlash is in active, MCU can not enter deep sleep.
       This code is copied from r_lpc_rx23w.c lpc_lowpower_activate_check. */
    if ((0x0000 != (FLASH.FENTRYR.WORD & 0x0081)) ||
        (0 == SYSTEM.MSTPCRA.BIT.MSTPA28))
    {
        return false;
    }

    /* add check for other components */

    return true;
}
```

コード 2-3　ディープスリープモードへの移行をチェックする場所

(2) 低消費電力状態への移行準備処理

suspend_peripherals()関数内に各コンポーネントの低消費電力状態への移行準備処理を追加します。コード 2-4 の"/* add implementation for transiting xxx mode */"の位置に各消費電力状態に応じて移行準備処理を追加してください。

```
static void suspend_peripherals(lpc_low_power_mode_t mode)
{
    if (LPC_LP_SW_STANDBY == mode)
    {
        R_BLE_CLI_Terminate();

        /* add implementation for transiting the software standby mode. */
    }
    else if(LPC_LP_DEEP_SLEEP == mode)
    {
        /* add implementation for transiting the deep sleep mode. */
    }
    else if(LPC_LP_SLEEP == mode)
    {
        /* add implementation for transiting the sleep mode. */
    }
    (省略)
}
```

コード 2-4　各低消費電力状態への移行準備処理を追加する場所

(3) 低消費電力状態からの復帰処理

resume_peripherals()関数内にコンポーネントの低消費電力状態からの復帰処理を追加します。コード 2-5 の"/* add implementation for transiting the active state. */"の位置に各低消費電力状態に応じて復帰処理を追加してください。

```
static void resume_peripherals(lpc_low_power_mode_t mode)
{
    if (LPC_LP_SW_STANDBY == mode)
    {
        R_BLE_CLI_Init();

        /* add implementation for transiting the active state. */
    }
    else if(LPC_LP_DEEP_SLEEP == mode)
    {
        /* add implementation for transiting the active state. */
    }
    else if(LPC_LP_SLEEP == mode)
    {
        /* add implementation for transiting the active state. */
    }
    (省略)
}
```

コード 2-5　各低消費電力状態からの復帰処理を追加する場所

3. ユーザコードの実装方法

QE for BLE は設計したプロファイル部分のほかに GAP ロール(セントラル/ペリフェラル)に応じたアプリケーション処理を src¥smc_gen¥Config_BLE_Profile 内にコード生成します。GAP ロールごとにプロジェクトを作成してコード生成するだけで、相互にコネクション可能なプログラムが app_main.c として生成されます。生成されるプログラムの動作については「3.1 スケルトンプログラムの動作」を参照してください。

app_main.c の以下の関数内で各種の API を使用してユーザコードを追加・変更することでアプリケーションを開発できます。一部の API は即時実行されて関数の戻り値として結果が返りますが、ほとんどの API はイベントとしてスケジューラにキューイング・実行され、実行結果がイベント通知としてイベントハンドラに返されます。

表 3.1　app_main.c の関数

app_main.c の関数		説明
app_main 関数		app_main 関数はメインループを持っています。 以下の API はメインループ前で使用します。 - 初期化処理 API 必須機能です。Bluetooth LE Protocol Stack のスケジューラにイベントハンドラや割り込みハンドラをコールバック関数として登録します。GATT データベースも登録します。「3.2.1 初期化処理(ble_app_init 関数)」で説明します。 以下の API はメインループ内で使用します。 - R_BLE_Execute 必須機能です。スケジューラを実行してイベントを処理し、結果をコールバック関数に返します。「3.2.2 メインループとスケジューラ(R_BLE_Execute)」で説明します。 - R_BLE_CLI_Process コマンドライン機能が有効の場合に使用します。 - R_BLE_LPC_EnterLowPowerMode MCU 低消費電力機能が有効の場合に使用します。
コールバック関数	イベントハンドラ	GAP / GATTS / GATTC / VS / サーバ側プロファイル API / クライアント側プロファイル API / L2CAP / DISC のイベントが発生すると呼び出されます。 【注】RF 通信タイミング通知は RF の割り込みが発生すると LL→スケジューラから呼び出されます。 【注】ソフトウェアタイマはタイマの割り込みが発生すると CMT FIT モジュール→Bluetooth LE Protocol Stack のスケジューラから呼び出されます。 【注】LED and Switch 制御の LED は通知を使用しません。Switch は Switch 押下の割り込みが発生すると IRQ FIT モジュール→Bluetooth LE Protocol Stack のスケジューラから呼び出されます。
	割り込みハンドラ	コマンドライン機能が有効の場合、UART 送受信の割り込みが発生すると SCI FIT モジュールから呼び出されます。

app_main.c の処理を別のファイルに分割したい場合、e²studio 上で Config_BLE_Profile フォルダを右クリックし、以下の手順で別のファイルを追加してください。
[新規] → [ソース・ファイル]から[任意の名前].c を追加してください。
[新規] → [ヘッダー・ファイル]から[任意の名前].h を追加してください。

API のパラメータの詳細については R_BLE API ドキュメント(r_ble_api_spec.chm)を参照してください。

3.1　スケルトンプログラムの動作

以下に、QE for BLE が生成するスケルトンプログラムの動作と通知される主なイベントを示します。点線の応答や動作は Bluetooth LE Protocol Stack が自動処理するためコードの記述は必要ありません。

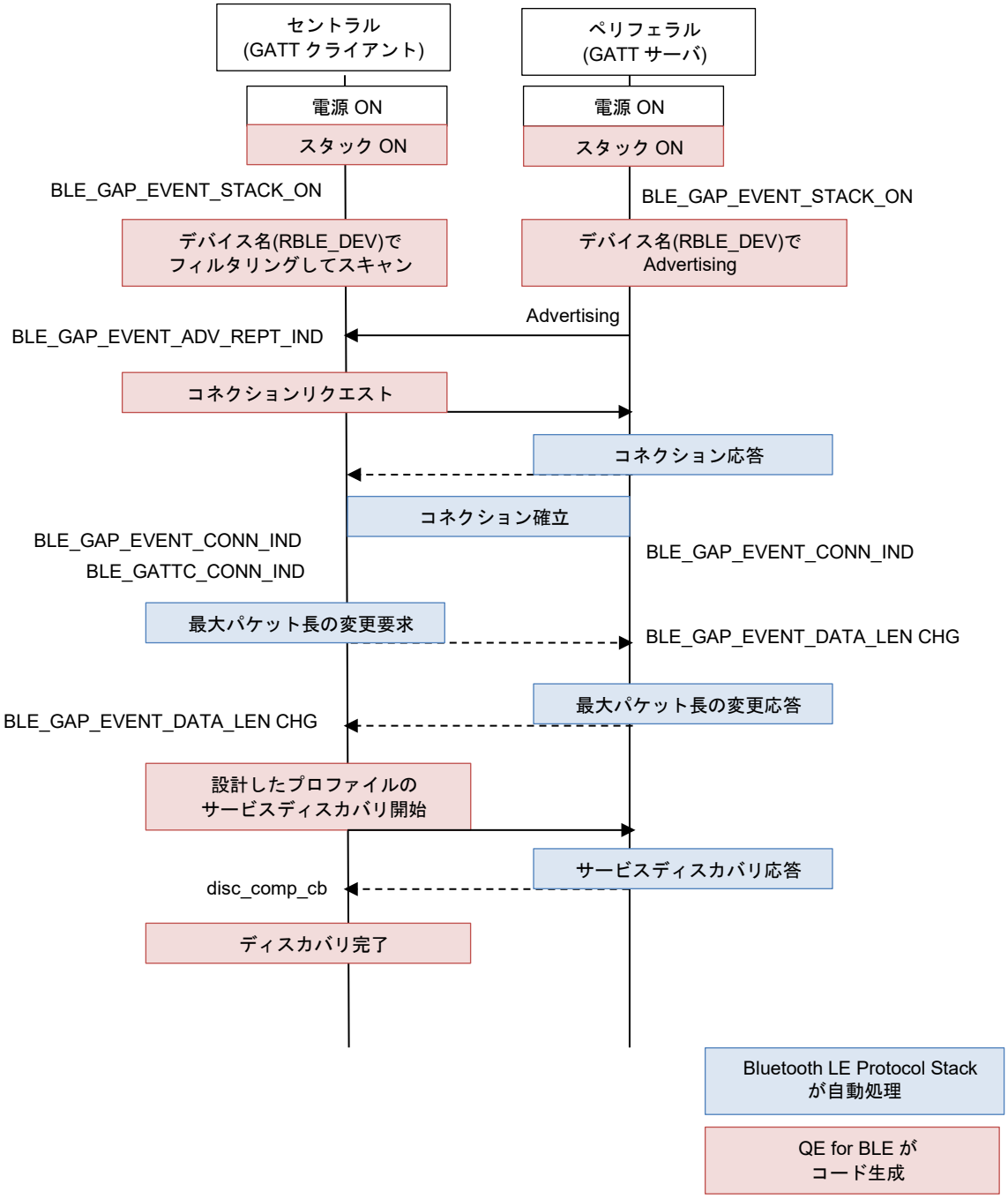


図 3-1　QE for BLE が生成するスケルトンプログラムの動作

3.2　app_main 関数

app_main 関数では、初期化処理、メインループの実装を行います。また、タイマ、ボード設定、コマンドラインなど使用する場合、 app_main 関数で初期化処理を行います。

【注】QE for BLE を使用する場合、app_main 関数のソースコードは自動的に生成されます。

app_main 関数の例をコード 3-1 に示します。

```
/* CommandLine parameters */
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
    &g_abs_cmd,
    &g_vs_cmd,
    &g_sys_cmd,
    &g_ble_cmd
};

void app_main(void)
{
    /* Initialize BLE */
    R_BLE_Open();

    /* Configure the board */
    R_BLE_BOARD_Init();

    /* Initialize the Low Power Control function */
    R_BLE_LPC_Init();

    /* Initialize timer for ABS & LED blink */
    R_BLE_TIMER_Init();

    /* Configure CommandLine */
    R_BLE_CLI_Init();
    R_BLE_CLI_RegisterCmds(gsp_cmds, ARRAY_SIZE(gsp_cmds));
    R_BLE_CMD_SetResetCb(ble_app_init);

    /* Initialize BLE host stack and profiles */
    ble_app_init();

    /* main loop */
    while (1)
    {
        /* Process Command Line */
        R_BLE_CLI_Process();
        /* Process Event */
        R_BLE_Execute();
        /* Enter the Lower Power Mode */
        R_BLE_LPC_EnterLowPowerMode();
    }
}
```

Bluetooth LE Protocol Stack の初期化(R_BLE_Open)
【注】必ず app_main 関数の最初でコールしてください。

ボードの初期化(R_BLE_BOARD_Init)

MCU 低消費電力機能の初期化(R_BLE_LPC_Init)

アプリケーション用の timer の初期化(R_BLE_TIMER_Init)

コマンドラインの初期化(R_BLE_CLI_Init)

ホストスタック、プロファイルの初期化(ble_app_init)

メインループ(R_BLE_Execute を呼び出す、R_BLE_LPC_EnterLowPowerMode で MCU 低消費電力状態への移行)

コード 3-1　app_main 関数の例

3.2.1　初期化処理(ble_app_init 関数)

ble_app_init 関数では、ホストスタック、プロファイルの初期化処理を行います。コールバック関数の登録や GATT データベースの登録などを行います。

【注】QE for BLE を使用する場合、ble_app_init 関数のソースコードは自動的に生成されます。

ble_app_init 関数の例をコード 3-2 に示します。

```
static ble_status_t ble_app_init(void)
{
    ble_status_t status;

    g_conn_hdl  = BLE_GAP_INVALID_CONN_HDL;
    gs_timer_hdl = BLE_TIMER_INVALID_HDL;

    /* Initialize host stack */
    status = R_BLE_ABS_Init(&gs_abs_init_param);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Initialize GATT Database */
    status = R_BLE_GATTS_SetDbInst(&g_gatt_db_table);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Initialize GATT Server */
    status = R_BLE_SERVS_Init();
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Initialize GATT client */
    status = R_BLE_SERVC_Init();
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Initialize GATT Discovery Library */
    status = R_BLE_DISC_Init();
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Initialize LED and Switch Service */
    status = R_BLE_LSC_Init(lss_cb);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Create timer for LED blink */
    status = R_BLE_TIMER_Create(&gs_timer_hdl, 1, BLE_TIMER_PERIODIC, timer_cb);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    return status;
}
```

ホストスタックの初期化(R_BLE_ABS_Init)
【注】抽象 API を使わない場合、以下を使用します。
R_BLE_GAP_Init
R_BLE_VS_Init
R_BLE_GATTS_Init
R_BLE_GATTC_Init

GATT データベースの登録(R_BLE_GATTS_SetDbInst)
【注】QE for BLE で GATT ロールをサーバ/クライアントのどちらにしてもコード生成されます。

GATT サーバ機能の初期化(R_BLE_SERVS_Init)
【注】QE for BLE で GATT ロールをサーバ/クライアントのどちらにしてもコード生成されます。

GATT クライアント機能の初期化(R_BLE_SERVC_Init)
【注】QE for BLE で GATT ロールをサーバ/クライアントのどちらにしてもコード生成されます。

Service Discovery 機能の初期化(R_BLE_DISC_Init)
【注】QE for BLE で GAP ロールをセントラルにするとコード生成されます。

サービスの初期化
(R_BLE_[サービス名]S_Init または R_BLE_[サービス名]C_Init)
【注】QE for BLE で GATT ロールをサーバにすると R_BLE_[サービス名]S_Init がコード生成されます。
【注】QE for BLE で GATT ロールをクライアントにすると R_BLE_[サービス名]C_Init)がコード生成されます。

ソフトウェアタイマ生成(R_BLE_TIMER_Create)

コード 3-2　ble_app_init 関数の例

3.2.1.1　コールバック関数の登録

アプリケーションにコールバック関数を登録することで、各種イベントの受信タイミングで処理を行うことが可能です。各機能ブロックのコールバック登録 API を表 3.2 に示します。

表 3.2　コールバック登録 API

機能ブロック	コールバック登録 API	コメント
GAP	R_BLE_ABS_Init or R_BLE_GAP_Init	登録されたコールバック関数は Advertising、スキャン、Connection 確立など、R_BLE_GAP_XXX の結果を受信時に呼び出されます。
GATT サーバ (プロファイル共通)	R_BLE_ABS_Init or R_BLE_GATTS_RegisterCb	登録されたコールバック関数は GATT クライアントからアクセス時に呼び出されます。
GATT クライアント (プロファイル共通)	R_BLE_ABS_Init or R_BLE_GATTC_RegisterCb	登録されたコールバック関数は GATT サーバからのアクセス時に呼び出されます。
Service Discovery (プロファイル共通)	R_BLE_DISC_Start()	登録されたコールバック関数は Service Discovery が完了したときに呼び出されます。
Vendor Specific	R_BLE_ABS_Init or R_BLE_VS_Init	登録されたコールバック関数は R_BLE_VS_XXX の結果を受信時に呼び出されます。
L2CAP	R_BLE_L2CAP_RegisterCfPsm()	登録されたコールバック関数は L2CAP Credit-Based Flow Control のリクエストの応答が返った時、L2CAP Credit-Based Flow Control の受信時など、R_BLE_L2CAP_XXX の結果を受信時に呼び出されます。 【注】QE for BLE でコード生成されません。
LED and Switch 制御	R_BLE_BOARD_RegisterSwitchCb()	登録されたコールバック関数は Board の Switch が押下された場合など、R_BLE_BOARD_XXX の結果を受信時に呼び出されます。 【注】QE for BLE でコード生成されません。
ソフトウェアタイマ	R_BLE_TIMER_Create()	登録されたコールバック関数は指定した時間が経過した場合など、R_BLE_TIMER_XXX の結果を受信時に呼び出されます。 【注】QE for BLE でコード生成されません。
サーバ側 プロファイル API	R_BLE_XXXS_Init() (XXX は Service の名前)	登録されたコールバック関数はクライアントからアクセスされた場合に呼び出されます。
クライアント側 プロファイル API	R_BLE_XXXC_Init() (XXX は Service の名前)	登録されたコールバック関数はサーバからアクセスされた場合に呼び出されます。

【注】R_BLE_ABS_Init は GAP、GATT サーバ、GATT クライアント、VS のコールバック関数をまとめて登録できます。

3.2.1.2 GATT データベースの登録(R_BLE_GATTS_SetDbInst)

GATT サーバとして動作させる GATT サービスアプリケーションを作成する場合、QE for BLE が以下のファイルにサービスのデータベースをコード生成します。

- gatt_db.c
- gatt_db.h

この GATT データベースは R_BLE_GATTS_SetDbInst によってアプリケーションに登録されます。

3.2.2　メインループとスケジューラ(R_BLE_Execute)

Bluetooth LE Protocol Stack はアプリケーションから呼ばれる API を処理するためにスケジューラを使用します。スケジューラを動作させるためにメインループ内にて R_BLE_Execute をコールしてください。発生したイベントは登録したコールバック関数に通知されます。

スケジューラは R_BLE_Execute により、Bluetooth LE Protocol Stack の各レイヤのタスク宛に送信されたメッセージキューに従ってタスクを処理します。図 3-2 に Bluetooth LE Protocol Stack の基本シーケンスチャートを示します。

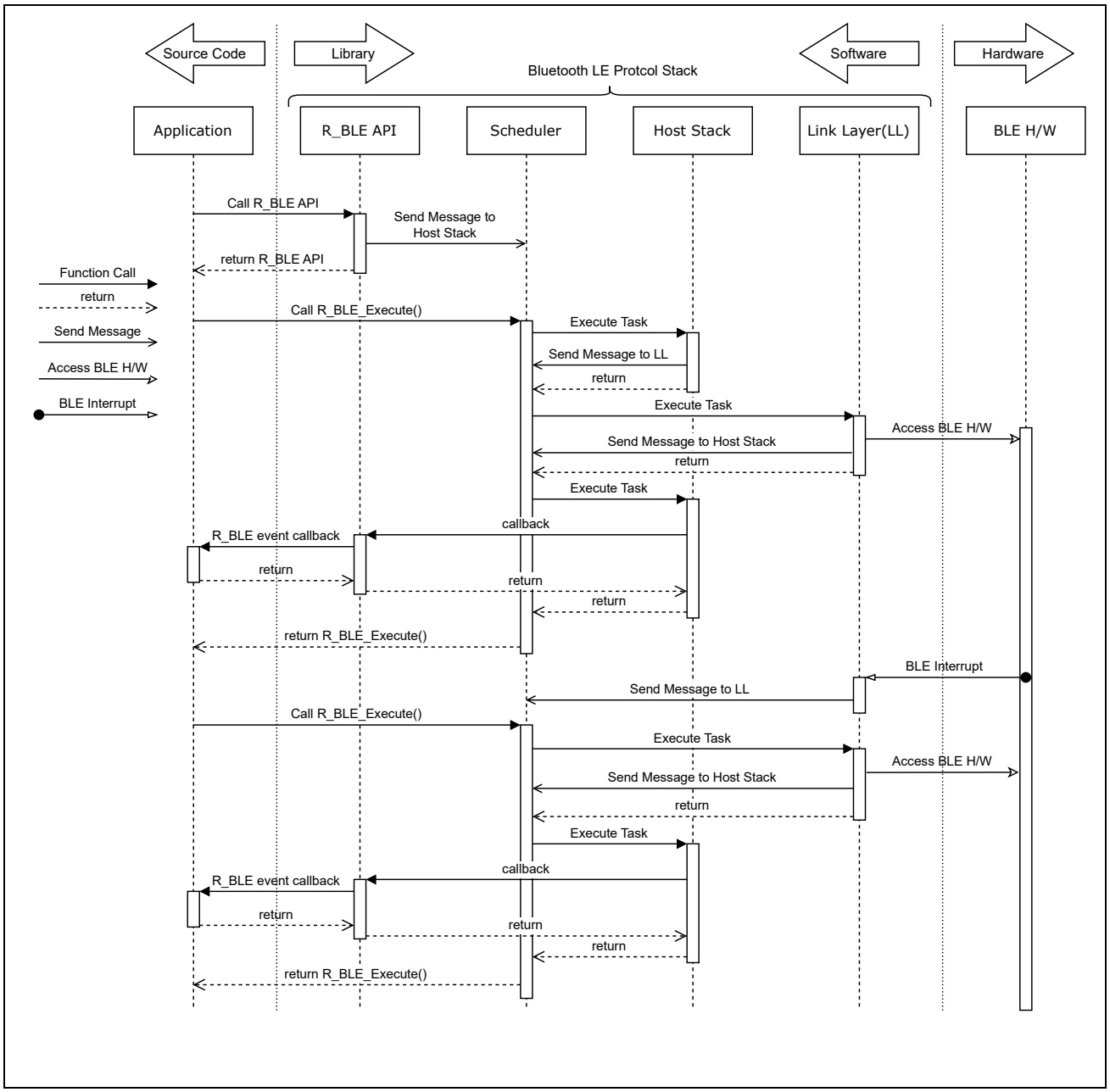


図 3-2　Bluetooth LE Protocol Stack の基本シーケンスチャート

3.2.3　終了処理

メインループから抜ける場合は、Bluetooth LE Protocol Stack、app_lib の終了処理を行うために以下の API をコールしてください。

[Bluetooth LE Protocol Stack]
R_BLE_Close()
R_BLE_GAP_Terminate()

[ソフトウェアタイマ]
R_BLE_TIMER_Terminate()

[コマンドライン]
R_BLE_CLI_Terminate()

終了処理のサンプルとして、抽象 API の R_BLE_ABS_Reset()を用意しています。この API 内にて上記の API をコールしています。

【注】R_BLE_Close()で RF H/W の停止をしているため、RF 通信中にソフトウェアリセットする場合は、リセットする前に必ず R_BLE_Close()を呼んでください。

3.3　GAP のイベント(gap_cb 関数)

GAP のコールバック関数は以下のイベントを受信します。

```
enum    e_ble_gap_evt_t {
    BLE_GAP_EVENT_INVALID = 0x1001,
    BLE_GAP_EVENT_STACK_ON,
    BLE_GAP_EVENT_STACK_OFF,
    BLE_GAP_EVENT_LOC_VER_INFO,
    BLE_GAP_EVENT_HW_ERR,
    BLE_GAP_EVENT_CMD_ERR = 0x1101,
    BLE_GAP_EVENT_ADV_REPT_IND,
    BLE_GAP_EVENT_ADV_PARAM_SET_COMP,
    BLE_GAP_EVENT_ADV_DATA_UPD_COMP,
    BLE_GAP_EVENT_ADV_ON,
    BLE_GAP_EVENT_ADV_OFF,
    BLE_GAP_EVENT_PERD_ADV_PARAM_SET_COMP,
    BLE_GAP_EVENT_PERD_ADV_ON,
    BLE_GAP_EVENT_PERD_ADV_OFF,
    BLE_GAP_EVENT_ADV_SET_REMOVE_COMP,
    BLE_GAP_EVENT_SCAN_ON,
    BLE_GAP_EVENT_SCAN_OFF,
    BLE_GAP_EVENT_SCAN_TO,
    BLE_GAP_EVENT_CREATE_CONN_COMP,
    BLE_GAP_EVENT_CONN_IND,
    BLE_GAP_EVENT_DISCONN_IND,
    BLE_GAP_EVENT_CONN_CANCEL_COMP,
    BLE_GAP_EVENT_WHITE_LIST_CONF_COMP,
    BLE_GAP_EVENT_RAND_ADDR_SET_COMP,
    BLE_GAP_EVENT_CH_MAP_RD_COMP,
    BLE_GAP_EVENT_CH_MAP_SET_COMP,
    BLE_GAP_EVENT_RSSI_RD_COMP,
    BLE_GAP_EVENT_GET_REM_DEV_INFO,
    BLE_GAP_EVENT_CONN_PARAM_UPD_COMP,
    BLE_GAP_EVENT_CONN_PARAM_UPD_REQ,
    BLE_GAP_EVENT_AUTH_PL_TO_EXPIRED,
    BLE_GAP_EVENT_SET_DATA_LEN_COMP,
    BLE_GAP_EVENT_DATA_LEN_CHG,
    BLE_GAP_EVENT_RSLV_LIST_CONF_COMP,
    BLE_GAP_EVENT_RPA_EN_COMP,
    BLE_GAP_EVENT_SET_RPA_TO_COMP,
    BLE_GAP_EVENT_RD_RPA_COMP,
    BLE_GAP_EVENT_PHY_UPD,
    BLE_GAP_EVENT_PHY_SET_COMP,
    BLE_GAP_EVENT_DEF_PHY_SET_COMP,
    BLE_GAP_EVENT_PHY_RD_COMP,
    BLE_GAP_EVENT_SCAN_REQ_RECV,
    BLE_GAP_EVENT_CREATE_SYNC_COMP,
    BLE_GAP_EVENT_SYNC_EST,
    BLE_GAP_EVENT_SYNC_TERM,
    BLE_GAP_EVENT_SYNC_LOST,
    BLE_GAP_EVENT_SYNC_CREATE_CANCEL_COMP,
    BLE_GAP_EVENT_PERD_LIST_CONF_COMP,
    BLE_GAP_EVENT_PRIV_MODE_SET_COMP,
    BLE_GAP_EVENT_PAIRING_REQ = 0x1401,
    BLE_GAP_EVENT_PASSKEY_ENTRY_REQ,
    BLE_GAP_EVENT_PASSKEY_DISPLAY_REQ,
    BLE_GAP_EVENT_NUM_COMP_REQ,
    BLE_GAP_EVENT_KEY_PRESS_NTF,
    BLE_GAP_EVENT_PAIRING_COMP,
    BLE_GAP_EVENT_ENC_CHG,
    BLE_GAP_EVENT_PEER_KEY_INFO,
    BLE_GAP_EVENT_EX_KEY_REQ,
    BLE_GAP_EVENT_LTK_REQ,
    BLE_GAP_EVENT_LTK_RSP_COMP,
    BLE_GAP_EVENT_SC_OOB_CREATE_COMP
}
```

主なイベントの受信タイミングを以下に示します。

表 3.3　GAP コールバックの主なイベント

イベント	受信タイミング
BLE_GAP_EVENT_STACK_ON(0x1001)	R_BLE_GAP_Init の完了
BLE_GAP_EVENT_ADV_PARAM_SET_COMP(0x1003)	R_BLE_GAP_SetAdvParam の完了
BLE_GAP_EVENT_ADV_DATA_UPD_COMP (0x1004)	R_BLE_GAP_SetAdvSresData の完了
BLE_GAP_EVENT_ADV_ON (0x1005)	Advertising の開始
BLE_GAP_EVENT_ADV_OFF (0x1006)	Advertising の終了
BLE_GAP_EVENT_SCAN_ON (0x1111)	スキャンの開始
BLE_GAP_EVENT_SCAN_OFF (0x1112)	スキャンの終了
BLE_GAP_EVENT_CONN_IND (0x1115)	コネクションの完了
BLE_GAP_EVENT_CONN_IND (0x1115)	コネクションの完了
BLE_GAP_EVENT_DISCONN_IND (0x1116)	コネクションの切断

以下が GAP のコールバック関数です。

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    ble_app_gapcb(type, result, p_data);

    switch(type)
    {
        /* TODO: Set callback events of GAP. Check BLE API reference for events. */
        【注】ここにイベント受信時の処理を追加します。
```

コード 3-3　GAP のコールバック関数

【注】result が BLE_SUCCESS 以外の場合、p_data で通知されるデータは不定値となります。

3.4　GATTS のイベント(gatts_cb 関数)

GATT サーバ(GATTS)のコールバック関数は以下のイベントを受信します。

```
enum    e_r_ble_gatts_evt_t {
    BLE_GATTS_EVENT_EX_MTU_REQ = 0x3002,
    BLE_GATTS_EVENT_READ_BY_TYPE_RSP_COMP = 0x3009,
    BLE_GATTS_EVENT_READ_RSP_COMP = 0x300B,
    BLE_GATTS_EVENT_READ_BLOB_RSP_COMP = 0x300D,
    BLE_GATTS_EVENT_READ_MULTI_RSP_COMP = 0x300F,
    BLE_GATTS_EVENT_WRITE_RSP_COMP = 0x3013,
    BLE_GATTS_EVENT_PREPARE_WRITE_RSP_COMP = 0x3017,
    BLE_GATTS_EVENT_EXE_WRITE_RSP_COMP = 0x3019,
    BLE_GATTS_EVENT_HDL_VAL_CNF = 0x301E,
    BLE_GATTS_EVENT_DB_ACCESS_IND = 0x3040,
    BLE_GATTS_EVENT_CONN_IND = 0x3081,
    BLE_GATTS_EVENT_DISCONN_IND = 0x3082,
    BLE_GATTS_EVENT_INVALID = 0x30FF
}
```

主なイベントの受信タイミングを以下に示します。

表 3.4　GATTS コールバックの主なイベント

イベント	受信タイミング
BLE_GATTS_EVENT_CONN_IND(0x3081)	コネクションの確立
BLE_GATTS_EVENT_EX_MTU_REQ(0x3002)	コネクション後に GATT クライアントからの MTU 変更要求あり
BLE_GATTS_EVENT_DB_ACCESS_IND(0x3040)	GATT データベースへのアクセスあり
BLE_GATTS_EVENT_READ_BY_TYPE_RSP_COMP(0x3009)	Read By Type Response の送信完了
BLE_GATTS_EVENT_WRITE_RSP_COMP(0x3013)	Write Response の送信完了
BLE_GATTS_EVENT_HDL_VAL_CNF(0x301E)	GATT クライアントからの Confirmation の受信完了
BLE_GATTS_EVENT_DISCONN_IND(0x3082)	コネクションの切断

以下が GATTS のコールバック関数です。

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

    switch(type)
    {
        /* TODO: Set callback events of GATTS. Check BLE API reference for events. */
        【注】ここにイベント受信時の処理を追加します。
```

コード 3-4　GATTS のコールバック関数

【注】result が BLE_SUCCESS 以外の場合、p_data で通知されるデータは不定値となります。

3.5　GATTC のイベント(gattc_cb 関数)

GATT クライアント(GATTC)のコールバック関数は以下のイベントを受信します。

```
enum    e_r_ble_gattc_evt_t {
    BLE_GATTC_EVENT_ERROR_RSP = 0x4001,
    BLE_GATTC_EVENT_EX_MTU_RSP = 0x4003,
    BLE_GATTC_EVENT_CHAR_READ_BY_UUID_RSP = 0x4009,
    BLE_GATTC_EVENT_CHAR_READ_RSP = 0x400B,
    BLE_GATTC_EVENT_CHAR_PART_READ_RSP = 0x400D,
    BLE_GATTC_EVENT_MULTI_CHAR_READ_RSP = 0x400F,
    BLE_GATTC_EVENT_CHAR_WRITE_RSP = 0x4013,
    BLE_GATTC_EVENT_CHAR_PART_WRITE_RSP = 0x4017,
    BLE_GATTC_EVENT_HDL_VAL_NTF = 0x401B,
    BLE_GATTC_EVENT_HDL_VAL_IND = 0x401D,
    BLE_GATTC_EVENT_CONN_IND = 0x4081,
    BLE_GATTC_EVENT_DISCONN_IND = 0x4082,
    BLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND = 0x40E0,
    BLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND = 0x40E1,
    BLE_GATTC_EVENT_ALL_PRIM_SERV_DISC_COMP = 0x40E2,
    BLE_GATTC_EVENT_PRIM_SERV_DISC_COMP = 0x40E3,
    BLE_GATTC_EVENT_SECOND_SERV_16_DISC_IND = 0x40E4,
    BLE_GATTC_EVENT_SECOND_SERV_128_DISC_IND = 0x40E5,
    BLE_GATTC_EVENT_ALL_SECOND_SERV_DISC_COMP = 0x40E6,
    BLE_GATTC_EVENT_INC_SERV_16_DISC_IND = 0x40E7,
    BLE_GATTC_EVENT_INC_SERV_128_DISC_IND = 0x40E8,
    BLE_GATTC_EVENT_INC_SERV_DISC_COMP = 0x40E9,
    BLE_GATTC_EVENT_CHAR_16_DISC_IND = 0x40EA,
    BLE_GATTC_EVENT_CHAR_128_DISC_IND = 0x40EB,
    BLE_GATTC_EVENT_ALL_CHAR_DISC_COMP = 0x40EC,
    BLE_GATTC_EVENT_CHAR_DISC_COMP = 0x40ED,
    BLE_GATTC_EVENT_CHAR_DESC_16_DISC_IND = 0x40EE,
    BLE_GATTC_EVENT_CHAR_DESC_128_DISC_IND = 0x40EF,
    BLE_GATTC_EVENT_ALL_CHAR_DESC_DISC_COMP = 0x40F0,
    BLE_GATTC_EVENT_LONG_CHAR_READ_COMP = 0x40F1,
    BLE_GATTC_EVENT_LONG_CHAR_WRITE_COMP = 0x40F2,
    BLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP = 0x40F3,
    BLE_GATTC_EVENT_RELIABLE_WRITES_COMP = 0x40F4,
    BLE_GATTC_EVENT_INVALID = 0x40FF
}
```

主なイベントの受信タイミングを以下に示します。

表 3.5　GATTC コールバックの主なイベント

イベント	受信タイミング
BLE_GATTC_EVENT_CONN_IND(0x4081)	コネクションの確立
BLE_GATTC_EVENT_EX_MTU_RSP(0x4003)	コネクション後に GATT サーバに MTU 変更要求して正常な応答あり
BLE_GATTC_EVENT_ERROR_RSP(0x4001)	GATT サーバからエラー応答あり
BLE_GATTC_EVENT_HDL_VAL_NTF(0x401B)	Notification の受信完了
BLE_GATTC_EVENT_HDL_VAL_IND(0x401D)	Indication の受信完了
BLE_GATTC_EVENT_DISCONN_IND(0x4082)	コネクションの切断

以下が GATTC のコールバック関数です。

```
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)
    {
        /* TODO: Set callback events of GATTC. Check BLE API reference for events. */
        【注】ここにイベント受信時の処理を追加します。
```

コード 3-5　GATTC のコールバック関数

【注】result が BLE_SUCCESS 以外の場合、p_data で通知されるデータは不定値となります。

3.6　VS のイベント(vs_cb 関数)

Vendor Specific (VS)のコールバック関数は以下のイベントを受信します。

```
enum    e_r_ble_vs_evt_t {
    BLE_VS_EVENT_SET_TX_POWER = 0x8001,
    BLE_VS_EVENT_GET_TX_POWER = 0x8002,
    BLE_VS_EVENT_TX_TEST_START = 0x8003,
    BLE_VS_EVENT_TX_TEST_TERM = 0x8004,
    BLE_VS_EVENT_RX_TEST_START = 0x8005,
    BLE_VS_EVENT_TEST_END = 0x8006,
    BLE_VS_EVENT_SET_CODING_SCHEME_COMP = 0x8007,
    BLE_VS_EVENT_RF_CONTROL_COMP = 0x8008,
    BLE_VS_EVENT_SET_ADDR_COMP = 0x8009,
    BLE_VS_EVENT_GET_ADDR_COMP = 0x800A,
    BLE_VS_EVENT_GET_RAND = 0x800B,
    BLE_VS_EVENT_TX_FLOW_STATE_CHG = 0x800C,
    BLE_VS_EVENT_FAIL_DETECT = 0x800D,
    BLE_VS_EVENT_SET_SCAN_CH_MAP = 0x800E,
    BLE_VS_EVENT_GET_SCAN_CH_MAP = 0x800F,
    BLE_VS_EVENT_INVALID = 0x80FF
}
```

主なイベントの受信タイミングを以下に示します。

表 3.6　VS コールバックの主なイベント

イベント	受信タイミング
BLE_VS_EVENT_SET_TX_POWER(0x8001)	R_BLE_VS_SetTxPower の完了
BLE_VS_EVENT_GET_TX_POWER(0x8002)	R_BLE_VS_GetTxPower の完了
BLE_VS_EVENT_SET_ADDR_COMP(0x8009)	R_BLE_VS_SetBdAddr の完了
BLE_VS_EVENT_GET_ADDR_COMP(0x800A)	R_BLE_VS_GetBdAddr の完了

以下が VS のコールバック関数です。

```
static void vs_cb(uint16_t type, ble_status_t result, st_ble_vs_evt_data_t *p_data)
{
    R_BLE_SERVS_VsCb(type, result, p_data);

    switch(type)
    {
        /* TODO: Set callback events of VS. Check BLE API reference for events. */
        【注】ここにイベント受信時の処理を追加します。
```

コード 3-6　VS のコールバック関数

【注】result が BLE_SUCCESS 以外の場合、p_data で通知されるデータは不定値となります。

3.7　サーバ側プロファイル API のイベント([サービス名]s_cb 関数)

サーバ側プロファイル API のコールバック関数は以下のイベントを受信します。

```
enum    e_ble_servs_event_t {
    BLE_SERVS_WRITE_REQ = 0x00,
    BLE_SERVS_WRITE_CMD = 0x01,
    BLE_SERVS_WRITE_COMP = 0x02,
    BLE_SERVS_READ_REQ = 0x03,
    BLE_SERVS_HDL_VAL_CNF = 0x04
}

enum    e_ble_[サービス名]s_event_t {
    BLE_[サービス名]S_EVENT_[キャラクタリスティック名]_WRITE_REQ = 0xXX00,
    BLE_[サービス名]S_EVENT_[キャラクタリスティック名]_WRITE_CMD= 0xXX01,
    BLE_[サービス名]S_EVENT_[キャラクタリスティック名]_WRITE_COMP = 0xXX02,
    BLE_[サービス名]S_EVENT_[キャラクタリスティック名]_READ_REQ = 0xXX03,
    BLE_[サービス名]S_EVENT_[キャラクタリスティック名]_HDL_VAL_CNF = 0xXX04,
    BLE_[サービス名]S_EVENT_[キャラクタリスティック名]_[ディスクリプタ名]_WRITE_REQ = 0xYY00,
    BLE_[サービス名]S_EVENT_[キャラクタリスティック名]_[ディスクリプタ名]_READ_REQ = 0xYY03,
    :
    :
}

【注】10～15bit 目はアトリビュート(キャラクタリスティックやディスクリプタ)を区別する連番。XX と YY は 00、04、08、10、…、FC になります。
```

主なイベントの受信タイミングを以下に示します。

表 3.7　サーバ側プロファイル API コールバックの主なイベント

イベント	受信タイミング
XXX_WRITE_REQ (0xXXX0)	Write Request の受信完了
XXX_WRITE_CMD (0xXXX1)	Write Without Response の受信完了
XXX_WRITE_COMP (0xXXX2)	Write Response の送信完了
XXX_READ_REQ (0xXXX3)	Read Request の受信完了
XXX_HDL_VAL_CNF (0xXXX4)	Confirmation の受信完了

以下がサーバ側プロファイル API のコールバック関数です。(ls サービスの例)

```
static void lss_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t *p_data)
{
    switch(type)
    {
        【注】ここにイベント受信時の処理を追加します。
```

コード 3-7　サーバ側プロファイル API のコールバック関数

【注】result が BLE_SUCCESS 以外の場合、p_data で通知されるデータは不定値となります。

3.8　クライアント側プロファイル API のイベント([サービス名]c_cb 関数)

クライアント側プロファイル API のコールバック関数は以下のイベントを受信します。

```
enum    e_ble_servc_event_t {
    BLE_SERVC_WRITE_RSP,
    BLE_SERVC_READ_RSP,
    BLE_SERVC_HDL_VAL_NTF,
    BLE_SERVC_HDL_VAL_IND
}

enum    e_ble_[サービス名]c_event_t {
    BLE_[サービス名]C_EVENT_[キャラクタリスティック名]_WRITE_RSP = 0xXX00,
    BLE_[サービス名]C_EVENT_[キャラクタリスティック名]_READ_RSP= 0xXX01,
    BLE_[サービス名]C_EVENT_[キャラクタリスティック名]_HDL_VAL_NTF = 0xXX02,
    BLE_[サービス名]C_EVENT_[キャラクタリスティック名]_HDL_VAL_IND = 0xXX03,
    BLE_[サービス名]C_EVENT_[キャラクタリスティック名]_[ディスクリプタ名]_WRITE_RSP = 0xYY00,
    BLE_[サービス名]C_EVENT_[キャラクタリスティック名]_[ディスクリプタ名]_READ_RSP = 0xYY01,
    :
    :
}

【注】10～15bit 目はアトリビュート(キャラクタリスティックやディスクリプタ)を区別する連番。XX と YY は 00、04、08、10、…、FC になります。
```

主なイベントの受信タイミングを以下に示します。

表 3.8　クライアント側プロファイル API コールバックの主なイベント

イベント	受信タイミング
XXX_WRITE_RSP (0xXXX0)	Write Response の受信完了
XXX_READ_RSP (0xXXX1)	Read Response の受信完了
XXX_HDL_VAL_NTF (0xXXX2)	Notification の受信完了
XXX_HDL_VAL_IND (0xXXX3)	Indication の受信完了

以下がクライアント側プロファイル API のコールバック関数です。(ls サービスの例)

```
static void lsc_cb(uint16_t type, ble_status_t result, st_ble_servc_evt_data_t *p_data)
{
    (void)result;
    (void)p_data;

    switch (type)
    {
        /* TODO: Set callback events of lsc. Check BLE API reference or e_ble_lsc_event_t for events. */
        【注】ここにイベント受信時の処理を追加します。
```

コード 3-8　クライアント側プロファイル API のコールバック関数

【注】result が BLE_SUCCESS 以外の場合、p_data で通知されるデータは不定値となります。

3.9　L2CAP のイベント

L2CAP のコールバック関数は以下のイベントを受信します。

```
enum    e_r_ble_l2cap_cf_evt_t {
    BLE_L2CAP_EVENT_CF_CONN_CNF = 0x5001,
    BLE_L2CAP_EVENT_CF_CONN_IND = 0x5002,
    BLE_L2CAP_EVENT_CF_DISCONN_CNF = 0x5003,
    BLE_L2CAP_EVENT_CF_DISCONN_IND = 0x5004,
    BLE_L2CAP_EVENT_CF_RX_DATA_IND = 0x5005,
    BLE_L2CAP_EVENT_CF_LOW_RX_CRD_IND = 0x5006,
    BLE_L2CAP_EVENT_CF_TX_CRD_IND = 0x5007,
    BLE_L2CAP_EVENT_CF_TX_DATA_CNF = 0x5008,
    BLE_L2CAP_EVENT_CMD_REJ = 0x5009
}
```

以下が L2CAP のコールバック関数です。

```
static void l2cap_cb(uint16_t type, ble_status_t result, st_ble_l2cap_cf_evt_data_t *p_data)
{
    switch (type)
    {
        【注】ここにイベント受信時の処理を追加します。
```

コード 3-9　L2CAP のコールバック関数

【注】result が BLE_SUCCESS 以外の場合、p_data で通知されるデータは不定値となります。

3.10 イベント通知機能(R_BLE_SetEvent)

R_BLE_SetEvent で Bluetooth LE Protocol Stack のスケジューラにイベントをキューイングすることができます。

Bluetooth LE Protocol Stack は R_BLE_Execute でイベントの状態を確認し、イベントがキューイングされている場合は、R_BLE_SetEvent で登録したコールバック関数を呼び出します。

【注】キューイングできるイベント数の上限は 8 です。

本機能は主に以下の場合に使用します。

- 割り込みハンドラで時間のかかる処理を割り込みハンドラ外で実行する。
  【注】Bluetooth LE Protocol Stack の RF 制御処理は高優先度で MCU に処理されます。RF 制御処理への影響を低減するため、アプリケーションの処理は短時間であることが推奨されます(推奨時間は RF イベント処理完了後から RF アイドル時間の 30%以内)。長時間となる処理は R_BLE_SetEvent を使用して複数回のコールバックに分割して実行してください。
- 割り込みハンドラ内で実行できない関数を割り込みハンドラ外で呼び出す。

以下に R_BLE_SetEvent のシーケンスチャートを示します。

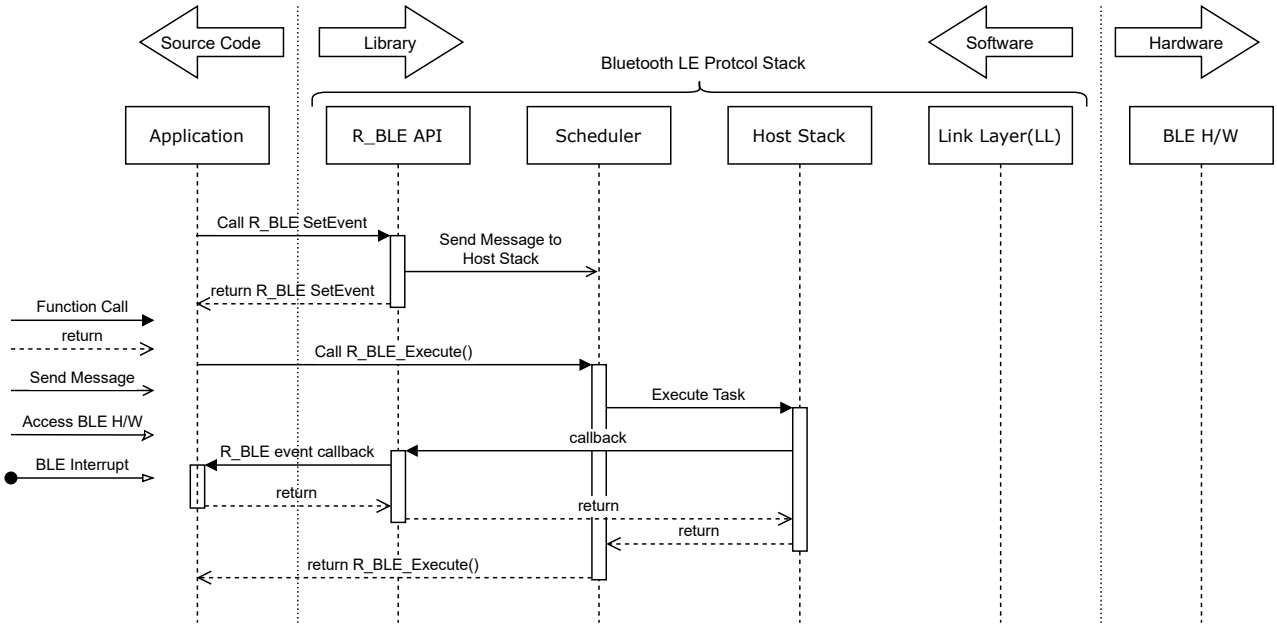


図 3-3　R_BLE_SetEvent のシーケンスチャート

RF 制御処理(Advertising)の合間に LED を点灯・消灯するサンプルをコード 3-10 に示します。次の Advertising への影響を低減するために Advertising 終了時にイベントをキューイングして LED を点灯・消灯しています。

```
[src¥smc_gen¥r_ble_rx23w¥src¥platform¥r_ble_pf_functions.c]
extern void sw_ntf_recv_event(void);
BLE_SECTION_P void r_ble_rf_notify_event_close(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( (uint16_t)(param>>16) )
    {
        case BLE_EVENT_TYPE_ADV:
        {
            R_BLE_SetEvent( sw_ntf_recv_event );
        } break;
    }
}

[app_main.c]
#include "board/r_ble_board.h"

void sw_ntf_recv_event(void)
```

```
{
    R_BLE_BOARD_ToggleLEDState(BLE_BOARD_LED1);
}

void app_main(void)
{
    /* Configure the board */
    R_BLE_BOARD_Init();
```

【注】LED and Switch 制御機能を使用します。「1.6.1 主要機能」を参照して機能を有効にしてコード生成してください。
【注】RF 通信タイミング通知機能を使用します。e²studio で[プロジェクト名].scfg をダブルクリックし、[コンポーネント] → [Middleware] → [ジェネリック] → [r_ble_rx23w] → [Advertising event close notify.]を Enable に変更してコード生成してください。

コード 3-10　イベント通知の例 (1)

SW1 押下時に割り込みハンドラ sw_cb からイベントをキューイングして LED を点灯・消灯するサンプルをコード 3-11 に示します。

```
[app_main.c]
#include "board/r_ble_board.h"

void sw_ntf_recv_event(void)
{
    R_BLE_BOARD_ToggleLEDState(BLE_BOARD_LED1);
}

static void sw_cb(void)
{
    R_BLE_SetEvent( sw_ntf_recv_event );
}

void app_main(void)
{
    /* Configure the board */
    R_BLE_BOARD_Init();
    R_BLE_BOARD_RegisterSwitchCb(BLE_BOARD_SW2, sw_cb);
```

【注】LED and Switch 制御機能を使用します。「1.6.1 主要機能」を参照して機能を有効にしてコード生成してください。

コード 3-11　イベント通知の例 (2)

3.11 RF 通信タイミング通知

「1.5 Bluetooth LE Protocol Stack の動作概要」で示した RF イベントと同期したアプリケーション開発を行うためには、RF 通信タイミング通知機能と「3.10 イベント通知機能(R_BLE_SetEvent)」を使用する必要があります。RF 通信タイミング通知機能の使用方法を以下に示します。

以下の設定から通知したい通信タイミングを選んで”Enable”にしてください。

表 3.9　RF 通信タイミング通知の設定

コンフィグレーションオプション	設定値
BLE_CFG_EVENT_NOTIFY_CONN_START	1: Enable
BLE_CFG_EVENT_NOTIFY_CONN_CLOSE	1: Enable
BLE_CFG_EVENT_NOTIFY_ADV_START	1: Enable
BLE_CFG_EVENT_NOTIFY_ADV_CLOSE	1: Enable
BLE_CFG_EVENT_NOTIFY_SCAN_START	1: Enable
BLE_CFG_EVENT_NOTIFY_SCAN_CLOSE	1: Enable
BLE_CFG_EVENT_NOTIFY_INIT_START	1: Enable
BLE_CFG_EVENT_NOTIFY_INIT_CLOSE	1: Enable
BLE_CFG_EVENT_NOTIFY_DS_START	1: Enable
BLE_CFG_EVENT_NOTIFY_DS_WAKEUP	1: Enable

以下に RF 通信タイミング通知のシーケンスチャートを示します。

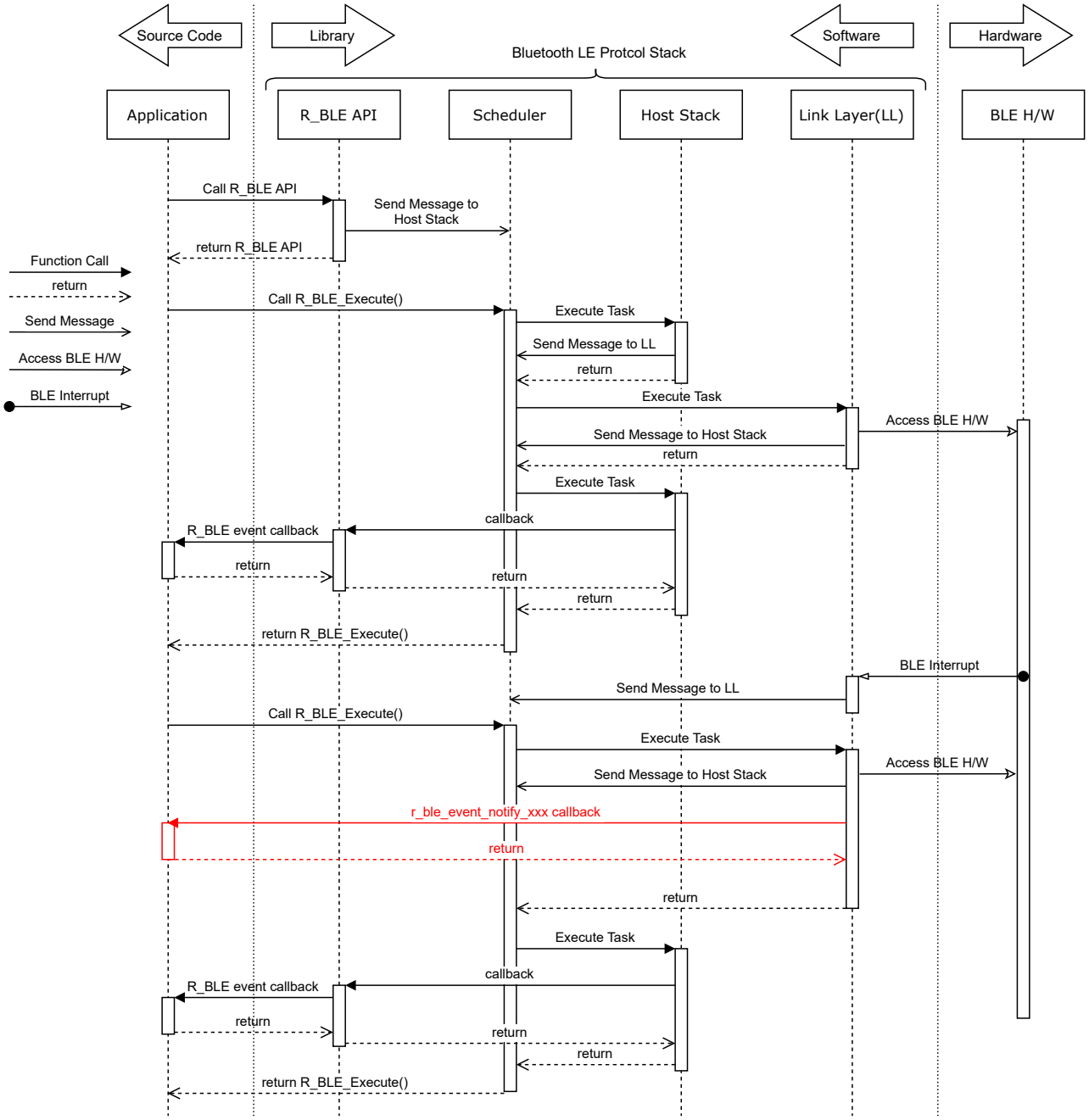


図 3-4　RF 通信タイミング通知のシーケンスチャート

以下は RF 通信タイミングを受けて R_BLE_SetEvent を使ってコマンドラインにログ表示するサンプルです。このサンプルはコマンドライン機能を使用します。「1.6.1 主要機能」を参照して機能を有効にしてコード生成してください。

以下のコードで RF 通信タイミング通知をログ表示します。

```
[src¥smc_gen¥r_ble_rx23w¥src¥platform¥r_ble_pf_functions.c]
(省略)

#include "cli/r_ble_cli.h"
#define pf R_BLE_CLI_Printf
void rf_ntf_recv_event(void)
{
        pf("RF event has come!!¥n");
}

(省略)

BLE_SECTION_P void r_ble_rf_notify_event_start(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( (uint16_t)(param>>16) )
    {
        case 0x0000:/*BLE_EVENT_TYPE_CONN*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case 0x0001:/*BLE_EVENT_TYPE_ADV*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case 0x0002:/*BLE_EVENT_TYPE_SCAN*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case 0x0003:/*BLE_EVENT_TYPE_INITIATOR*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
    }
}

(省略)

BLE_SECTION_P void r_ble_rf_notify_event_close(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( (uint16_t)(param>>16) )
    {
        case 0x0000:/*BLE_EVENT_TYPE_CONN*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case 0x0001:/*BLE_EVENT_TYPE_ADV*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case 0x0002:/*BLE_EVENT_TYPE_SCAN*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case 0x0003:/*BLE_EVENT_TYPE_INITIATOR*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
    }
}

(省略)

BLE_SECTION_P void r_ble_rf_notify_deep_sleep(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( param )
```

```
    {
        case BLE_EVENT_TYPE_RF_DS_START:
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case BLE_EVENT_TYPE_RF_DS_CLOSE:
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
    }
}

(省略)
```

コード 3-12　RF 通信タイミング通知のログ表示のサンプル(r_ble_pf_functions.c)

以下のコードでコマンドライン機能の入出力のみ動作させます。

```
[app_main.c]
(省略)

#include "cli/r_ble_cli.h"

(省略)

void app_main(void)
{
    (省略)
    /* Configure CommandLine */
    R_BLE_CLI_Init();
    (省略)
    while (1)
    {
        /* Process Command Line */
        R_BLE_CLI_Process();
        (省略)
```

コード 3-13　RF 通信タイミング通知のログ表示のサンプル(app_main.c)

4. app_lib

app_lib はアプリケーション開発を補助するライブラリです。app_lib を利用することで、Bluetooth LE の基本的な動作を簡単に実現できます。

4.1 ソフトウェアタイマ

ソフトウェアタイマは、アプリケーションにタイマ機能を提供します。

ソフトウェアタイマの特徴を以下に示します。

- Compare Match Timer(CMT)の 1 チャネルを使用します。CMT の制御には CMT FIT モジュール(r_cmt_rx)を使用します。使用するチャネルは r_cmt_rx が動的に確保し、アプリケーションによるソフトウェアタイマの利用が終了すると、CMT は解放されます。
- ソフトウェアタイマのタイムアウト時間はミリ秒で指定します。ソフトウェアタイマを起動し、タイムアウト時間が経過すると、コールバック関数で通知されます。
- ソフトウェアタイマは 2 つの動作モードを持ちます。
  - 周期通知モード(BLE_TIMER_PERIODIC)：ソフトウェアタイマのチャネルを起動すると、タイムアウト時間の経過を周期的に通知します。
  - 単発通知モード(BLE_TIMER_ONE_SHOT)：ソフトウェアタイマのチャネルを起動すると、タイムアウト時間の経過を 1 回だけ通知します。
- ソフトウェアタイマは複数のチャネルを持ちます。各チャネルには独立してタイムアウト時間、動作モード、コールバック関数を登録できます。
- ソフトウェアタイマのチャネル数は BLE_TIMER_NUM_OF_SLOT マクロ(デフォルト 10)で定義されており、変更できます。なお 1 チャネルにつき、RAM 上に 24byte の管理領域を必要とします。

ソフトウェアタイマ利用時の注意点を以下に示します。

- 長いタイムアウト時間を指定した場合、または複数のチャネルを使用した場合、CMT の停止と再起動を繰り返すため、タイムアウトが指定したタイムアウト時間より遅延します。
- CMT は CPU の低消費電力モードであるソフトウェアスタンバイでは停止します。ソフトウェアタイマの実行中は、ソフトウェアスタンバイに遷移しないでください。
- 抽象 API は本ソフトウェアタイマを使用します。抽象 API を使用する場合は、抽象 API の初期化関数である R_BLE_ABS_Init()の実行前に、ソフトウェアタイマの初期化関数である R_BLE_TIMER_Init()を実行してください。

ソフトウェアタイマは以下の API を提供します。API 仕様の詳細は、R_BLE API ドキュメントを参照してください。

表 4.1　ソフトウェアタイマの API

ソフトウェアタイマ API	概要
R_BLE_TIMER_Init	ソフトウェアタイマを初期化
R_BLE_TIMER_Terminate	ソフトウェアタイマを終了
R_BLE_TIMER_Create	ソフトウェアタイマのチャネルを確保し、動作パラメータを登録
R_BLE_TIMER_Delete	ソフトウェアタイマのチャネルを解放
R_BLE_TIMER_Start	ソフトウェアタイマのチャネルを起動
R_BLE_TIMER_Stop	ソフトウェアタイマのチャネルを停止
R_BLE_TIMER_UpdateTimeout	ソフトウェアタイマのチャネルのタイムアウト時間を更新して起動

ソフトウェアタイマの状態遷移を図 4-1 に示します。

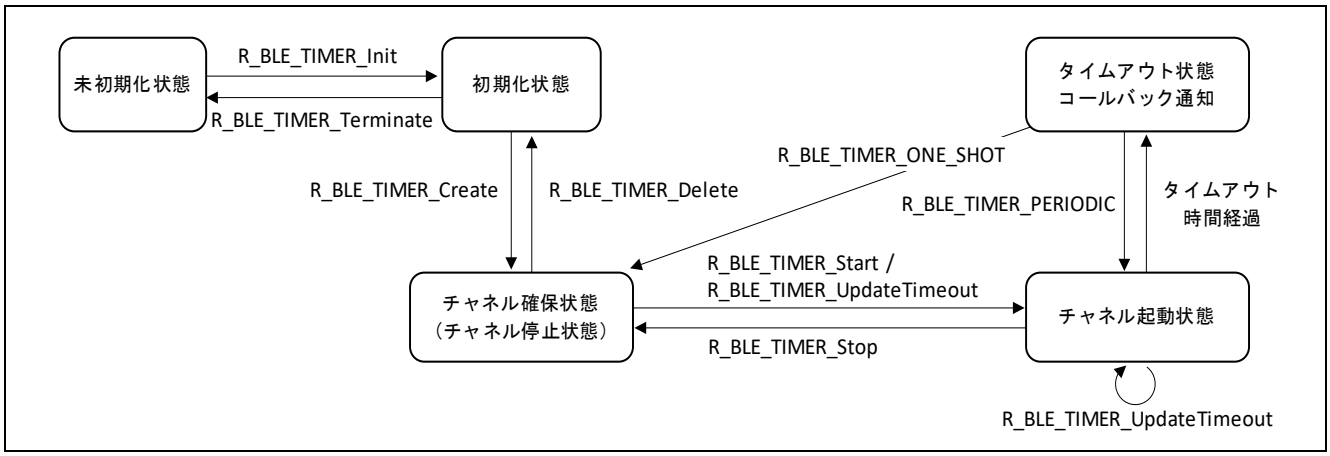


図 4-1　ソフトウェアタイマの状態遷移

- R_BLE_TIMER_UpdateTimeout()はタイムアウト状態からも実行できます。
- R_BLE_TIMER_Delete()はチャネル起動状態またはタイムアウト状態からも実行できます。

ソフトウェアタイマの利用例を以下に示します。

- ソフトウェアタイマのヘッダファイルをインクルードし、R_BLE_TIMER_Init()でソフトウェアタイマを初期化します。

```
/* ソフトウェアタイマ ヘッダファイルのインクルード */
#include "timer/r_ble_timer.h"

{
        /* ソフトウェアタイマの初期化 */
        R_BLE_TIMER_Init();
}
```

コード 4-1　ソフトウェアタイマの初期化

- R_BLE_TIMER_Create()で以下の動作パラメータを指定し、ソフトウェアタイマのチャネルを生成します。また R_BLE_TIMER_Create()はチャネルを識別するためのチャネルハンドルを返します。
  - タイムアウト時間(ms)
  - タイムアウト通知コールバック関数
  - 動作モード：BLE_TIMER_PERIODIC または BLE_TIMER_ONE_SHOT

注：生成可能なチャネル数(BLE_TIMER_NUM_OF_SLOT)を超えた場合、R_BLE_TIMER_Create()は BLE_ERR_LIMIT_EXCEEDED エラーを返します。

```
static void timer_cb(uint32_t timer_hdl)
{
}

{
        /* ソフトウェアタイマのチャネル確保 */
        ble_status_t status;
        status = R_BLE_TIMER_Create(&gs_timer_hdl, 1000, BLE_TIMER_PERIODIC, timer_cb);

        /* ソフトウェアタイマのチャネル起動 */
        R_BLE_TIMER_Start(gs_timer_hdl);
}
```

コード 4-2　ソフトウェアタイマのチャネル確保と起動

- R_BLE_TIMER_Start()または R_BLE_TIMER_UpdateTimeout()でソフトウェアタイマのチャネルを起動します。タイムアウト時間が経過すると、R_BLE_TIMER_Create()で登録したコールバック関数で通知されます。
- チャネル動作は R_BLE_TIMER_Stop()で停止できます。
- 確保したチャネルは何度でも再利用できます。

```
/* ソフトウェアタイマ チャネルハンドル */
static uint32_t gs_timer_hdl;

{
        /* ソフトウェアタイマのチャネル起動 */
        R_BLE_TIMER_Start(gs_timer_hdl);

        /* ソフトウェアタイマのチャネルタイムアウト更新と起動 */
        R_BLE_TIMER_UpdateTimeout(gs_timer_hdl, 500);

        /* ソフトウェアタイマのチャネル停止 */
        R_BLE_TIMER_Stop(gs_timer_hdl);
}
```

コード 4-3　ソフトウェアタイマのチャネル起動、更新、停止

- 生成したチャネルが不要となった場合、R_BLE_TIMER_Delete()で解放します。

```
{
        /* ソフトウェアタイマのチャネル解放 */
        R_BLE_TIMER_Delete(&gs_timer_hdl);
}
```

コード 4-4　ソフトウェアタイマのチャネル解放

- ソフトウェアタイマが不要となった場合、R_BLE_TIMER_Terminate()で終了します。

注：R_BLE_TIMER_Terminate()は必ず全チャネルを解放後に実行してください。

```
{
        /* ソフトウェアタイマの終了 */
        R_BLE_TIMER_Terminate();
}
```

コード 4-5　ソフトウェアタイマの終了

4.2　コマンドライン

コマンドラインは、VT100 エミュレーションに対応した端末エミュレータを通じて、任意のコマンド処理を実行する機能を提供します。コマンドライン機能を使用する場合、表 1.8 の SCI FIT モジュールとバイト型キューバッファ FIT モジュールを追加します。BLE FIT モジュールの表 4.2 のコンフィグレーションオプションを設定します。

表 4.2　コマンドラインのコンフィグレーションオプション

コンフィグレーションオプション	設定値
BLE_CFG_CMD_LINE_EN	1: Enable
BLE_CFG_CMD_LINE_CH	コマンドライン機能で使用する SCI のチャネルを設定します。以下のいずれかの値を設定してください。 1:　SCI1 5:　SCI5 8:　SCI8 12:　SCI12 (BGA 85pin のみ)

デフォルトで用意されているコマンドは以下の通りです。各コマンドの詳細な仕様については、”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)”をご参照ください。

表 4.3　サポートするコマンド一覧

コマンド名	サブコマンド名	コマンドの内容
gap	adv	Advertising を開始します。
	scan	スキャンを開始します。
	conn	コネクションします。
	disconn	切断します。
	device	コネクション中のデバイスを表示します。
	priv	ローカルデバイスのプライバシ機能を ON にします。
	conn_cfg	コネクションの設定を行います。
	wl	White List にリモートデバイスを登録します。
	auth	ペアリング／暗号化を行います。
	sync	Periodic Sync を確立します。
	ver	バージョン情報を表示します。
vs	txp	送信パワーの設定、取得を行います。
	scheme	Coded PHY の Coding Scheme を設定します。
	test	無線をテストするために Direct Test Mode(DTM)の操作を行います。
	addr	BD_ADDR の設定、取得を行います。
	rand	乱数を生成します。
sys	stby	ソフトウェアスタンバイモードの操作を行います。
ble	reset	Bluetooth LE Protocol Stack のリセットを行います。
	close	Bluetooth LE Protocol Stack を停止します。

以降にコマンドラインをアプリケーションに追加するときのコードへの追加内容を説明します。

4.2.1　標準コマンドの使用手順

(1)　ヘッダファイルのインクルード

標準コマンド用に、以下のヘッダファイルをインクルードします。

```
/* 標準コマンド用ヘッダファイルのインクルード */
#include "cmd/r_ble_cmd_abs.h"
#include "cmd/r_ble_cmd_vs.h"
#include "cmd/r_ble_cmd_sys.h"
```

コード 4-6　標準コマンド用ヘッダファイルのインクルード

(2)　初期化処理とコマンドの登録

コマンドライン機能を使うために、表 4.4 の API をアプリケーションの初期化処理の中でコールします。

表 4.4　コマンドライン機能の初期化時にコールする API

API	説明
R_BLE_CLI_Init	SCI FIT モジュールの初期化処理を行う。
R_BLE_CLI_RegisterCmds	コマンドを登録する。
R_BLE_CMD_SetResetCb	リセットコマンド時に Bluetooth LE Protocol Stack を再起動するコールバックを登録する。

表 4.4 のコマンドライン機能の API をアプリケーションの初期化処理に追加したサンプルを以下に示します。

```
/* 省略 */

/* CommandLine parameters */
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
    &g_abs_cmd,
    &g_vs_cmd,
    &g_sys_cmd,
    &g_ble_cmd
};

/* 省略 */
/* Reset BLE Protocol Stack */
static void ble_host_stack_init(void)
{
    ble_app_init();
}

/* 省略 */

/* Initialize BLE Protocol Stack */
static ble_status_t ble_app_init(void)
{
    ble_status_t status;

    /* Initialize host stack */
    status = R_BLE_ABS_Init(&gs_abs_init_param);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* 省略 */
}
/* 省略 */
```

```
void app_main(void)
{
    /* Initialize BLE */
    R_BLE_Open();

    /* 省略  */

    /* Configure CommandLine */
    R_BLE_CLI_Init();
    R_BLE_CLI_RegisterCmds(gsp_cmds, ARRAY_SIZE(gsp_cmds));
    R_BLE_CMD_SetResetCb(ble_host_stack_init);
    /* 省略  */
}
```

コード 4-7　コマンドラインの初期化を追加した例

(3)　コールバック関数

コマンド実行中にイベント処理を行うために、コールバック関数の先頭に表 4.5 の関数を追加します。

表 4.5　各コールバック関数に追加するコマンドライン関数

追加するコールバック関数	関数	説明
GAP コールバック関数	R_BLE_CMD_AbsGapCb	gap コマンド実行により発生した、 gap イベントを受信した後の処理を行う。
VS コールバック関数	R_BLE_CMD_VsCb	vs コマンド実行により発生した、 vs イベントを受信した後の処理を行う。

以下にコールバック関数に表 4.5 コマンドライン用の関数を追加したサンプルを示します。

```
/* 省略  */
/* GAP Callback */
void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    R_BLE_CMD_AbsGapCb(type, result, p_data);
    /* 省略  */
}

/* 省略  */
/* Vendor Specific Callback */
void vs_cb(uint16_t type, ble_status_t result, st_ble_vs_evt_data_t *p_data)
{
    R_BLE_CMD_VsCb(type, result, p_data);
    /* 省略  */
}
/* 省略  */
```

コード 4-8　コールバック関数へのコマンドライン用関数の追加例

(4)　メインループ

コマンドを実行するために、アプリケーションのメインループに以下の関数を追加します。

表 4.6　メインループに追加するコマンドライン関数

API	説明
R_BLE_CLI_Process	端末エミュレータから入力された文字を処理する。

以下にメインループに表 4.6 のコマンドライン用の関数を追加したサンプルを示します。

```
/* main loop */
void app_main(void)
{
    /* 省略  */
    /* main loop */
    while (1)
    {
        /* Process Command Line */
        R_BLE_CLI_Process();
        /* Process Event */
        R_BLE_Execute();
        /* 省略  */
     }
}
```

コード 4-9　メインループへのコマンドライン用関数の追加例

4.2.2　ユーザコマンドの作成手順

コマンドライン機能では、st_ble_cli_cmd_t 型の変数にコマンドを定義することでユーザ独自のコマンドを作成することができます。ここでは、デモプロジェクトで提供しているカスタムプロファイル LED Switch service の Client(以降は lsc と記します)動作を行うためのコマンドを新規に作成する場合を例に説明します。

(1)　ヘッダファイルのインクルード

コマンドを実装するソースコードにコマンドライン用ヘッダファイル、r_ble_cmd.h と r_ble_cli.h をインクルードします。

```
/* コマンドライン用ヘッダファイルのインクルード */
#include "cmd/r_ble_cmd.h"
#include "cli/r_ble_cli.h"
```

コード 4-10　コマンドライン用ヘッダファイルのインクルード

(2)　コマンドの定義

コマンドの名前、サブコマンド群、サブコマンドの数、help で出力する文字列を定義します。 ここではサンプルとして、以下のような lsc 用のコマンドを定義します。

```
/* コマンドの定義 */
const st_ble_cli_cmd_t g_lsc_cmd =
{
    .p_name      = "lsc",                      /* コマンド名 */
    .p_cmds      = lsc_sub_cmds,               /* サブコマンド群 */
    .num_of_cmds = ARRAY_SIZE(lsc_sub_cmds), /* サブコマンドの数 */
    .p_help       = "Sub Command: set_switch_state_ntf, write_led_blink_rate¥n"
                   "Try 'lsc sub-cmd help' for more information", /* helpで表示する内容 */
};
```

コード 4-11　コマンドの定義の例

(3)　サブコマンドの定義

サブコマンドを定義します。lsc 用として、下記にサブコマンドのサンプルを示します。コネクションコマンドのようにコネクションの完了を待ちたい場合や、スキャンコマンドのように処理を継続し、手動で中断するコマンドを作成する場合、abort ハンドラを登録します。abort ハンドラが登録されているコマンドは、Ctrl+C キーによりコマンドの実行を中断するまで、他の入力を受け付けなくなります。

```
/* サブコマンドの定義 */
static const st_ble_cli_cmd_t lsc_set_switch_state_ntf_cmd =
{
    .p_name = "set_switch_state_ntf",                      /* サブコマンドの名前 */
    .exec = cmd_lsc_set_switch_state_ntf,                  /* サブコマンドの関数 */
    .p_help = "Usage: lsc set_switch_state_ntf conn_hdl value",  /* helpで表示する内容 */
};

/* 省略  */

/* サブコマンドの定義 */
static const st_ble_cli_cmd_t lsc_write_led_blink_rate_cmd =
{
    .p_name = "write_led_blink_rate",                        /* サブコマンドの名前 */
    .exec = cmd_lsc_write_led_blink_rate,                    /* サブコマンドの関数 */
    .p_help = "Usage: lsc write_led_blink_rate conn_hdl blink_rate", /* helpで表示する内容 */
};

/* 省略  */

/* サブコマンドの定義 */
static const st_ble_cli_cmd_t lsc_conn_lss_cmd =
{
    .p_name = "conn_lss",                                    /* サブコマンドの名前 */
    .exec = cmd_lsc_conn_lss,                                /* サブコマンドの関数 */
    .abort = abort_lsc_conn,                                  /* Abortハンドラ */
    .p_help = "Usage: lsc conn_lss XX:XX:XX:XX:XX:XX addr_type", /* helpで表示する内容 */
};

/* 省略  */

/* サブコマンド群 */
static const st_ble_cli_cmd_t * const lsc_sub_cmds[] =
{
    &lsc_set_switch_state_ntf_cmd,  /* サブコマンド */
    &lsc_write_led_blink_rate_cmd,  /* サブコマンド */
    &lsc_conn_lss_cmd,              /* サブコマンド */
};
```

コード 4-12　サブコマンドの定義の例

(4)　サブコマンド関数の定義

サブコマンド実行時に動作する関数を定義します。

以下にサブコマンド関数のサンプルを示します。

```
/*-----------------------------------------------------------------
lsc set_switch_state_ntf command
-----------------------------------------------------------------*/
static void cmd_lsc_set_switch_state_ntf(int argc, char *argv[])
{
    if (argc != 3)
    {
        pf("lsc %s: unrecognized operands¥n", argv[0]);
        return;
    }

    uint16_t conn_hdl;
    conn_hdl = (uint16_t)strtol(argv[1], NULL, 0);

    long value = strtol(argv[2], NULL, 0);
    ble_status_t ret;
    ret = R_BLE_LSC_WriteSwitchStateCliCnfg(conn_hdl, (uint16_t *)&value);

    if (ret != BLE_SUCCESS)
    {
        pf("lsc %s: failed with 0x%04X¥n", argv[0], ret);
        return;
    }
}
```

コード 4-13　サブコマンド関数の定義の例

(5)　Abort ハンドラ

サブコマンド実行時に Ctrl+C キーにより中断するための関数を定義します。

以下に Abort ハンドラのサンプルを示します。

```
/*-----------------------------------------------------------------
lsc connect lss abort handler
-----------------------------------------------------------------*/
static void abort_lsc_conn(void)
{
    R_BLE_GAP_CancelCreateConn();
}
```

コード 4-14　Abort ハンドラの例

(6)　初期化処理とコマンドの登録

コマンド、サブコマンドを定義した後、アプリケーション独自のコマンドとして使えるように、下記のサンプルのように R_BLE_CLI_RegisterCmds()によりコマンド登録を行います。

```
/* コマンドの登録 */
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
    &g_abs_cmd,
    &g_vs_cmd,
    &g_lsc_cmd  /* 追加するコマンド */
};

/* 省略  */

void app_main(void)
{
    /* 省略  */

    R_BLE_CLI_Init();                                          /* コマンドライン機能の初期化 */
    R_BLE_CLI_RegisterCmds(gsp_cmds, ARRAY_SIZE(gsp_cmds));  /* コマンドの登録 */

    /* 省略  */
    /* main loop */
    while (1)
    {
        /* Process Command Line */
        R_BLE_CLI_Process();
        /* Process Event */
        R_BLE_Execute();
        /* 省略  */
     }

}
```

コード 4-15　初期化処理とコマンド登録の例

4.3　ロガー

ロガー機能は以下のログ出力機能を提供します。

- 3 段階のログレベル（ERROR, WARNING, DEBUG）
- ログ出力は printf と同様のフォーマットで記述
- BD アドレスおよび UUID を文字列に変換する関数の提供

ログメッセージは e² studio の Renesas Debug Virtual Console(デバッグ・コンソール)上に出力されます。そのため、コマンドライン機能のような端末エミュレータを使用しないアプリケーション環境であっても任意文字列を表示することが可能になります。ただし、デバッグ・コンソールへの出力が多い場合、MCU 処理を占有してしまい Bluetooth LE 通信が正常に行えなくなる場合があります。ロガー機能使用中に Bluetooth LE 通信が切断するなどの現象が発生した場合はロガー機能を無効にするか、出力情報を削減してください。

ログレベルの設定は、プロジェクト全体のログレベルを指定する、BLE_CFG_LOG_LEVEL コンフィグレーションオプションにより行います。BLE_CFG_LOG_LEVEL の設定値とログ出力の設定は表 4.7 の通りです。

表 4.7　BLE_CFG_LOG_LEVEL の設定

BLE_CFG_LOG_LEVEL 設定値	説明
0	ログ出力なし
1	ERROR ログ出力
2	ERROR & WARNING ログ出力
3	ERROR & WARNING & DEBUG　ログ出力

ログの内容は r_ble_logger.h で定義されている、表 4.8 のログ出力用マクロを使用します。

表 4.8　ログ出力用マクロ

ログ出力マクロ	LOG_LABEL	説明
BLE_LOG_ERR	ERR	ERROR ログ出力用
BLE_LOG_WRN	WRN	WARNING ログ出力用
BLE_LOG_DBG	DBG	DEBUG ログ出力用

ログ出力用マクロにより、printf と同じフォーマットで下記のようにログを設定します。

```
BLE_LOG_DBG("BLE_GAP_EVENT_STACK_ON ¥n");
```

ログは以下のフォーマットで出力されます。

```
module_tag: [LOG_LABEL] (function:line) log_body ¥n
```

“module_tag”はモジュール毎にログに付加するタグを” BLE_LOG_TAG”マクロにより指定することができます。タグの指定は r_lib_logger.h をインクルードする前に行ってください。

```
#define BLE_LOG_TAG “app_main”
#include “logger/r_lib_logger.h”
```

コード 4-16　ロガーヘッダのインクルード

先程の例では、ログは下記のように出力されます。”module_tag”は”app_main”を設定しています。

```
app_main: [DBG] (ble_app_gapcb:238) BLE_GAP_EVENT_STACK_ON
```

また、BD アドレス、16bit/128bit UUID のログ出力用に、BLE_BD_ADDR_STR(), BLE_UUID_STR()関数を用意しています。BLE_BD_ADDR_STR()は BD アドレスバイト配列とアドレスタイプをパラメータとして渡すと、BD アドレスを示す文字列を返します。BLE_UUID_STR()は UUID バイト配列とタイプをパラメータとして渡すと、UUID を示す文字列を返します。詳細については R_BLE API ドキュメント(r_ble_api_spec.chm)を参照してください。

BLE_ADDR_STR()および BLE_UUID_STR()は、以下のように使用します。

```
BLE_LOG_DBG(“Connected to %s¥n”, BLE_ADDR_STR(addr, addr_type));
BLE_LOG_DBG(“UUID: %s¥n”, BLE_UUID_STR(uuid, uuid_type));
```

コード 4-17　BLE_ADDR_STR()および BLE_UUID_STR()の使用例

4.4　セキュリティデータ管理

アプリケーションはペアリングで交換する鍵の管理を簡潔に行うために、セキュリティデータ管理機能を使うことが可能です。本機能では RX23W の Data Flash に鍵を保存します。セキュリティデータ管理機能を使用する場合、表 1.8 のフラッシュ FIT モジュールを追加します。表 4.9 のコンフィグレーションオプションを設定します。

表 4.9　セキュリティデータ管理のコンフィグレーションオプション

コンフィグレーションオプション	設定値
BLE_CFG_EN_SEC_DATA	1: Enable
BLE_CFG_SECD_DATA_DF_BLOCK	セキュリティデータ管理機能で Data Flash へのボンディング情報の保存機能で使用する Data Flash Block を"0"～"7"の範囲で設定します。他の用途の Data Flash Block と異なる値を設定します。
BLE_CFG_NUM_BOND	保存するボンディング情報の数を設定します。 "1"～"7"の範囲で設定してください。 開発中にこの値を変更する場合、ファームウェア書き込み後、R_BLE_GAP_DeleteBondInfo()、または”gap auth del remote all” コマンドにより、ボンディング情報を削除してください。

抽象 API はセキュリティデータ管理を使っているため、抽象 API が有効な場合は以下の実装を行う必要はありません。

4.4.1　初期化処理

セキュリティデータ管理は R_BLE_SECD_Init により、初期化処理を行います。初期化処理では Data Flash に保存したリモートデバイスの鍵を取り出して、Bluetooth LE Protocol Stack に再設定しています。

4.4.2　ローカルデバイスの鍵の取得

Data Flash に保存したローカルデバイスの鍵(IRK, CSRK)を R_BLE_SECD_ReadLocInfo で取得します。取得した鍵を表 9.10 の API で Bluetooth LE Protocol Stack に再設定します。LTK はリモートデバイスの鍵と共通となります。

4.4.3　ローカルデバイスの鍵の保存

ローカルデバイスが使用する、IRK、CSRK は R_BLE_SECD_WriteLocInfo()で保存します。IRK、CSRK 生成後に、この API により保存してください。

4.4.4　リモートデバイスの鍵の保存

ペアリング時にリモートデバイスから受信した鍵と鍵の情報は、表 4.10 の API 関数を GAP コールバック内でコールします。

表 4.10　リモートデバイスの鍵保存時に使用する API 関数

セキュリティデータ管理 API 関数	説明
R_BLE_SECD_RecvRemKeys	リモートデバイスから受信した鍵を保存する
R_BLE_SECD_WriteRemKeys	リモートデバイスから受信した鍵の情報を保存する

以下にリモートデバイスから受信した鍵と鍵の情報を保存するサンプルを示します。

```
/* GAP Callback */
void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_event_data)
{
    switch(event_type)
    {
        /* 省略  */
        case BLE_GAP_EVENT_PAIRING_COMP :
            {
                if(BLE_SUCCESS == event_result)
                {
                    st_ble_gap_pairing_info_evt_t * p_param;
                    p_param = (st_ble_gap_pairing_info_evt_t *)p_event_data->p_param;
                    R_BLE_SECD_WriteRemKeys(&p_param->bd_addr, &p_param->auth_info);
                }
            }
        break;

        case BLE_GAP_EVENT_PEER_KEY_INFO :
            {
                st_ble_gap_peer_key_info_evt_t * p_param;
                p_param = (st_ble_gap_peer_key_info_evt_t *)p_event_data->p_param;
                R_BLE_SECD_RecvRemKeys(&p_param->bd_addr, &p_param->key_ex_param);
            }
        break;
        /* 省略  */
    }
}
```

コード 4-18　鍵と鍵の情報を保存する処理の例

4.5　ボード LED & Switch

アプリケーションはボード上の LED と Switch の制御を簡単に行うために、LED and Switch 制御機能を使うことが可能です。表 4.11 のコンフィグレーションオプションをボードに合わせた設定にすることでボード上の LED, プッシュスイッチの制御が可能です。

表 4.11　ボードの LED, スイッチを設定するンコンフィグレーションオプション

コンフィグレーションオプション	設定値
BLE_CFG_BOARD_LED_SW_EN	1
BLE_CFG_BOARD_TYPE	0 (お客様独自のボード) 1 (Target Board) 2 (RSSK)

LED and Switch 制御用に、以下のヘッダファイルをインクルードします。

```
/* LED and Switch 制御用ヘッダファイルのインクルード */
#include "board/r_ble_board.h"
```

コード 4-19　LED and Switch 制御用ヘッダファイルのインクルード

4.5.1　お客様独自のボード用の設定

お客様独自のボードを使用する場合は、表 4.11 のコンフィグレーションオプションの設定と app_lib/board/r_ble_board.c の以下の点を変更してください。

(1)　LED, プッシュスイッチ(SW)のマクロ定義

お客様独自のボード用の

BLE_BOARD_SW1_IRQ

BLE_BOARD_SW2_IRQ

BLE_BOARD_LED1_PIN

BLE_BOARD_LED2_PIN

の定義をお客様独自のボードの設定に合わせて変更します。

```
#if (BLE_CFG_BOARD_TYPE == 1)   /* for RX23W Target Board(TB) */
#define BLE_BOARD_SW1_IRQ   (IRQ_NUM_5)
#define BLE_BOARD_SW2_IRQ   (IRQ_NUM_5)
#define BLE_BOARD_LED1_PIN  (GPIO_PORT_C_PIN_0)
#define BLE_BOARD_LED2_PIN  (GPIO_PORT_B_PIN_0)
#elif (BLE_CFG_BOARD_TYPE == 2) /* for RX23W RSSK board */
#define BLE_BOARD_SW1_IRQ   (IRQ_NUM_1)
#define BLE_BOARD_SW2_IRQ   (IRQ_NUM_0)
#define BLE_BOARD_LED1_PIN  (GPIO_PORT_4_PIN_2)
#define BLE_BOARD_LED2_PIN  (GPIO_PORT_4_PIN_3)
#else /* BLE_CFG_BOARD_TYPE */  /* for Custom board */
#define BLE_BOARD_SW1_IRQ   (IRQ_NUM_7)
#define BLE_BOARD_SW2_IRQ   (IRQ_NUM_5)
#define BLE_BOARD_LED1_PIN  (GPIO_PORT_C_PIN_5)
#define BLE_BOARD_LED2_PIN  (GPIO_PORT_C_PIN_6)
#endif /* BLE_CFG_BOARD_TYPE */
```

この部分を
お客様独自のボードの設定に
合わせて変更。

コード 4-20　SW, LED のマクロ定義の変更箇所

コード 4-20 の例では、SW1 に IRQ7, SW2 に IRQ5 を割り当て、LED1 は PC5 端子, LED2 は PC6 端子を設定しています。

(2)　irq_pin_set()内のレジスタの設定

お客様独自のボードのプッシュスイッチ用の IRQ 端子を使用するために、irq_pin_set()関数内のレジスタを設定している部分をお客様独自のボードの環境に合わせて変更します。

```
#if   (BLE_CFG_BOARD_TYPE == 1)
    /*Set IRQ5 pin */
    PORT1.PMR.BIT.B5 = 0U;
    PORT1.PDR.BIT.B5 = 0U;
    MPC.P15PFS.BYTE  = 0x40U;
#elif (BLE_CFG_BOARD_TYPE == 2)
    /*Set IRQ0 pin */
    PORT3.PMR.BIT.B0 = 0U;
    PORT3.PDR.BIT.B0 = 0U;
    MPC.P30PFS.BYTE  = 0x40U;
    /*Set IRQ1 pin */
    PORT3.PMR.BIT.B1 = 0U;
    PORT3.PDR.BIT.B1 = 0U;
    MPC.P31PFS.BYTE  = 0x40U;
#else /* (BLE_CFG_BOARD_TYPE == x) */
    /*Set IRQ5 pin */
    PORT1.PMR.BIT.B5 = 0U;
    PORT1.PDR.BIT.B5 = 0U;
    MPC.P15PFS.BYTE  = 0x40U;
    /*Set IRQ7 pin */
    PORT1.PMR.BIT.B7 = 0U;
    PORT1.PDR.BIT.B7 = 0U;
    MPC.P17PFS.BYTE  = 0x40U;
#endif /* (BLE_CFG_BOARD_TYPE == x) */
```

この部分を
お客様独自のボードの
環境に合わせて変更。

コード 4-21　irq_pin_set()の変更箇所

コード 4-21 の例では、SW1 用の IRQ7 に P17 端子を、SW2 用の IRQ5 に P15 端子を設定しています。

4.5.2　ボード LED & Switch 初期化

ボード上の LED と Switch の制御を行うために、アプリケーションの初期化時に

R_BLE_BOARD_Init

をコールします。

```
void app_main(void)
{
    /* Initialize BLE */
    R_BLE_Open();

    /* Configure the board */
    R_BLE_BOARD_Init();
    /* 省略  */

}
```

コード 4-22　ボード LED & Switch の初期化

4.5.3　ボード LED の ON/OFF

ボード上の LED の ON/OFF は
R_BLE_BOARD_SetLEDState
R_BLE_BOARD_ToggleLEDState
により行います。

R_BLE_BOARD_SetLEDState は ON/OFF 状態を指定します。
R_BLE_BOARD_ToggleLEDState は現在の状態と逆の状態になります。

4.5.4　スイッチ押下に割り当てるコールバック

ボード上の Switch 押下時に処理を行う関数の登録は、
R_BLE_BOARD_RegisterSwitchCb
により行います。

以下のサンプルでは、初期化時に SW2 を押下時に sw_cb により、LED1 を点灯／消灯します。

```
static void sw_cb(void)
{
    R_BLE_BOARD_ToggleLEDState(BLE_BOARD_LED1);
}

/* 省略  */

void app_main(void)
{
    /* Initialize BLE */
    R_BLE_Open();

    /* Configure the board */
    R_BLE_BOARD_Init();
    R_BLE_BOARD_RegisterSwitchCb(BLE_BOARD_SW2, sw_cb);
    /* 省略  */
}
```

コード 4-23　スイッチ押下に割り当てるコールバックの例

4.6　抽象 API

抽象 API は Bluetooth LE Protocol Stack でよく使う機能をより簡単に使うための API です。抽象 API は GAP、 GATT サーバ、GATT クライアント、Vendor Specific API を内部で使用し、各機能を実現しています。抽象 API でコールしている API と結果として通知されるイベントは表 4.12 の通りです。各抽象 API の詳細な仕様については R_BLE API ドキュメント(r_ble_api_spec.chm)を参照してください。
抽象 API のコードは変更しないでください。

表 4.12　抽象 API の使用 API と発生イベント

抽象 API	説明	使用 API	発生イベント
R_BLE_ABS_Init	初期化処理として以下を行います。 1. ホストスタックの初期化 2. GAP、GATTS、GATTC、VS イベントを通知するためのコールバックの登録 3. ペアリングパラメータ設定	R_BLE_GAP_Init R_BLE_GAP_SetPairingParams R_BLE_VS_Init R_BLE_GATTS_SetDbInst R_BLE_GATTS_Init R_BLE_GATTS_RegisterCb R_BLE_GATTC_Init R_BLE_GATTC_RegisterCb R_BLE_GAP_GetVerInfo R_BLE_SECD_Init R_BLE_SECD_ReadLocInfo R_BLE_GAP_SetLocIdInfo	BLE_GAP_EVENT_STACK_ON BLE_GAP_EVENT_LOC_VER_INFO
R_BLE_ABS_Reset	Bluetooth LE Protocol Stack の リセットを行います。	R_BLE_Close R_BLE_GAP_Terminate R_BLE_Open R_BLE_SetEvent	
R_BLE_ABS_StartLegacyAdv	Legacy Advertising 用にパラメータ設定、Advertising Data の設定を行い、Advertising を開始します。	R_BLE_GAP_SetAdvParam R_BLE_GAP_SetAdvSresData R_BLE_GAP_StartAdv	BLE_GAP_EVENT_ADV_PARAM_SET_COMP BLE_GAP_EVENT_ADV_DATA_UPD_COMP BLE_GAP_EVENT_ADV_ON BLE_GAP_EVENT_ADV_OFF
R_BLE_ABS_StartExtAdv	Extended Advertising 用にパラメータ設定、Advertising Data の設定を行い、Advertising を開始します。	R_BLE_GAP_SetAdvParam R_BLE_GAP_SetAdvSresData R_BLE_GAP_StartAdv	BLE_GAP_EVENT_ADV_PARAM_SET_COMP BLE_GAP_EVENT_ADV_DATA_UPD_COMP BLE_GAP_EVENT_ADV_ON BLE_GAP_EVENT_ADV_OFF
R_BLE_ABS_StartNonConnAdv	Non-Connectable Advertising 用にパラメータ設定、Advertising Data の設定を行い、Advertising を開始します。	R_BLE_GAP_SetAdvParam R_BLE_GAP_SetAdvSresData R_BLE_GAP_StartAdv	BLE_GAP_EVENT_ADV_PARAM_SET_COMP BLE_GAP_EVENT_ADV_DATA_UPD_COMP BLE_GAP_EVENT_ADV_ON BLE_GAP_EVENT_ADV_OFF
R_BLE_ABS_StartPerdAdv	Periodic Advertising 用にパラメータ設定、Periodic Advertising Data の設定を行い、Advertising を開始します。	R_BLE_GAP_SetAdvParam R_BLE_GAP_SetAdvSresData R_BLE_GAP_SetPerdAdvParam R_BLE_GAP_StartPerdAdv R_BLE_GAP_StartAdv	BLE_GAP_EVENT_ADV_PARAM_SET_COMP BLE_GAP_EVENT_ADV_DATA_UPD_COMP BLE_GAP_EVENT_PERD_ADV_PARAM_SET_COMP BLE_GAP_EVENT_PERD_ADV_ON BLE_GAP_EVENT_ADV_ON BLE_GAP_EVENT_ADV_OFF

抽象 API	説明	使用 API	発生イベント
R_BLE_ABS_StartScan	スキャンの設定を行い、開始します。	R_BLE_GAP_StartScan	BLE_GAP_EVENT_SCAN_ON BLE_GAP_EVENT_SCAN_OFF BLE_GAP_EVENT_SCAN_TO BLE_GAP_EVENT_ADV_REPT_IND
R_BLE_ABS_CreateConn	コネクションリクエストを行います。	R_BLE_TIMER_Create R_BLE_GAP_CreateConn R_BLE_TIMER_Start R_BLE_GAP_CancelCreateConn R_BLE_TIMER_Delete R_BLE_TIMER_Stop R_BLE_TIMER_Delete	BLE_GAP_EVENT_CREATE_CONN_COMP BLE_GAP_EVENT_CONN_CANCEL_COMP BLE_GAP_EVENT_CONN_IND
R_BLE_ABS_SetLocPrivacy	ローカルデバイスのプライバシを設定します。	R_BLE_GAP_EnableRpa R_BLE_VS_GetRand R_BLE_GAP_SetLocIdInfo R_BLE_GAP_ConfRslvList R_BLE_GAP_SetPrivMode	BLE_GAP_EVENT_RPA_EN_COMP BLE_VS_EVENT_GET_RAND BLE_GAP_EVENT_RSLV_LIST_CONF_COMP BLE_GAP_EVENT_PRIV_MODE_SET_COMP
R_BLE_ABS_StartAuth	ペアリングを開始します。既にペアリング済みの場合は暗号化を開始します。	R_BLE_GAP_GetDevSecInfo R_BLE_GAP_StartPairing R_BLE_GAP_ReplyPasskeyEntry R_BLE_GAP_ReplyNumComp R_BLE_GAP_ReplyExKeyInfoReq R_BLE_GAP_StartEnc R_BLE_GAP_ReplyLtkReq	BLE_GAP_EVENT_PAIRING_REQ BLE_GAP_EVENT_PASSKEY_ENTRY_REQ BLE_GAP_EVENT_PASSKEY_DISPLAY_REQ BLE_GAP_EVENT_NUM_COMP_REQ BLE_GAP_EVENT_KEY_PRESS_NTF BLE_GAP_EVENT_PEER_KEY_INFO BLE_GAP_EVENT_EX_KEY_REQ BLE_GAP_EVENT_PAIRING_COMP BLE_GAP_EVENT_LTK_REQ BLE_GAP_EVENT_LTK_RSP_COMP BLE_GAP_EVENT_ENC_CHG

5. Advertising

Bluetooth Low Energy のデバイスはスキャンしている周辺デバイスに対して Advertising によりデータを送信します。

5.1 手順

アプリケーション上での Advertising の手順を図 5-1 に示します。各ステップの詳細について以降の章で説明します。抽象 API を使用する場合、5.2 から 5.2.3 までの手順は 1 つの抽象 API コールで実施します。使い方については 5.5 を参照してください。

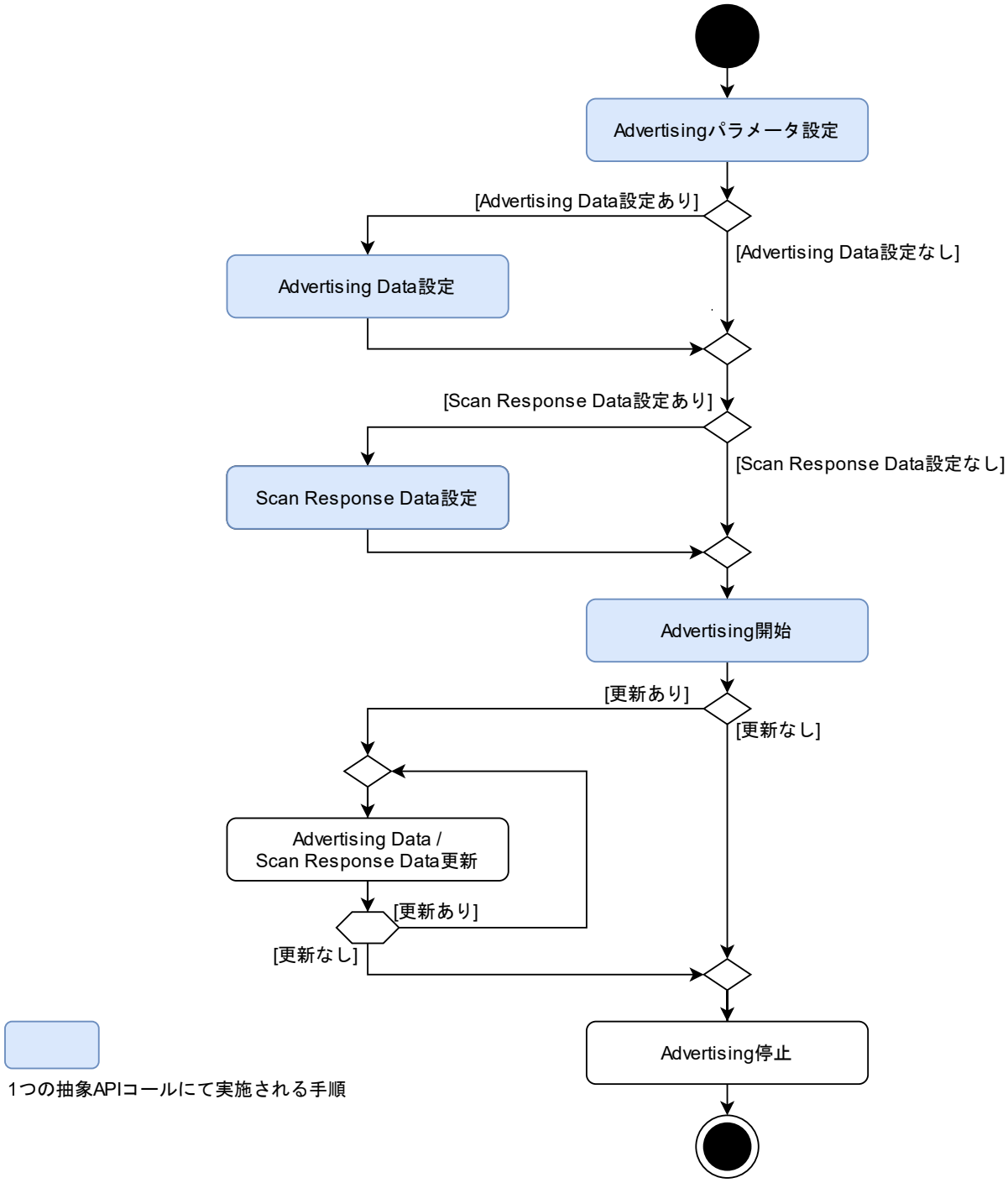


図 5-1　Advertising の手順

5.2　GAP API による Advertising

5.2.1　Advertising パラメータ設定

Advertising を開始するにあたって、R_BLE_GAP_SetAdvParam による、Advertising パラメータの設定が必要となります。Advertising 中はパラメータ変更できません。抽象 API を使用する場合、この手順は不要です。以降はユースケースによって異なるパラメータの設定について説明します。

5.2.1.1　Advertising のタイプの指定

下記項目からアプリケーションがどのタイプの Advertising を使うかを選択し、st_ble_gap_adv_param_t 構造体の adv_prop_type フィールドに表 5.1 の値を設定します。

- リモートデバイスからのコネクションリクエストへの対応(Connectable or Non-Connectable)
- スキャンリクエストへの対応(Scannable or Non-Scannable)
- 送信先を指定するかどうか(Direct or Undirect)
- リモートデバイスが対応している Advertising のタイプ(Legacy advertising or Extended advertising)
- Advertising Data の最大サイズ

表 5.1　Advertising のタイプと adv_prop_type フィールド

Advertising Type	Advertising PDU	adv_prop_type フィールドに設定する値	legacy or extended	最大サイズ（バイト）
Connectable and Scannable Undirected *5	ADV_IND	BLE_GAP_LEGACY_PROP_ADV_IND	legacy	31
Connectable Undirected	ADV_EXT_IND AUX_ADV_IND	BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_UNDIRECT	extended	245*1*4
Connectable Directed	ADV_DIRECT_IND	BLE_GAP_LEGACY_PROP_ADV_DIRECT_IND or BLE_GAP_LEGACY_PROP_ADV_HDC_DIRECT_IND	legacy	0
	ADV_EXT_IND AUX_ADV_IND	BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_DIRECT or BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_HDC_DIRECT	extended	239*1*4
Non-Connectable and Non-Scannable Undirected	ADV_NONCONN_IND	BLE_GAP_LEGACY_PROP_ADV_NONCONN_IND	legacy	31
	ADV_EXT_IND AUX_ADV_IND AUX_CHAIN_IND*2	BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_UNDIRECT	extended	BLE_CFG_RF_ADV_DATA_MAX の設定値*4
Non-Connectable and Non-Scannable Directed	ADV_EXT_IND AUX_ADV_IND AUX_CHAIN_IND*3	BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_DIRECT or BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_HDC_DIRECT	extended	BLE_CFG_RF_ADV_DATA_MAX の設定値*4
Scannable Undirected *5	ADV_SCAN_IND	BLE_GAP_LEGACY_PROP_ADV_SCAN_IND	legacy	31
	ADV_EXT_IND AUX_ADV_IND	BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_UNDIRECT	extended	0
Scannable Directed *5	ADV_EXT_IND AUX_ADV_IND	BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_DIRECT or BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_HDC_DIRECT	extended	0

*1 : adv_prop_type に BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER を追加する場合、最大サイズ-1 バイトとなります。

*2 : Advertising Data のサイズが 245 バイト以下の場合(Periodic advertising を使用する場合はさらに-18 バイト、BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER を使用する場合はさらに-1 バイト)、Advertising Data を AUX_ADV_IND だけで送信できるため、AUX_CHAIN_IND は未使用となります。

*3 : Advertising Data のサイズが 239 バイト以下の場合(Periodic advertising を使用する場合はさらに-18 バイト、BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER を使用する場合はさらに-1 バイト)、Advertising Data を AUX_ADV_IND だけで送信できるため、AUX_CHAIN_IND は未使用となります。

*4 : Advertising Data のサイズが 230 バイト以上の場合、Advertising Data が受信側の Bluetooth の仕様に従って分割されるため、必要に応じて受信側で結合してください。

*5 : スキャンレスポンスデータの PDU とサイズについては図 5-3 に示します。

Bluetooth LE Protocol Stack ライブラリの種類によって、サポートする Advertising のタイプが変わります。All features は Legacy、Extended の両方の Advertising をサポートしています。Balance, Compact は Legacy advertising のみサポートとなります。スキャナ側が Legacy advertising のみサポートしている場合、Extended advertising を受信することはできません。

Advertising のタイプが Extended かつ、Non-scannable の場合、各 PDU は図 5-2 に示す順番で送信されます。advDelay は 0ms から 10ms のランダムな delay です。

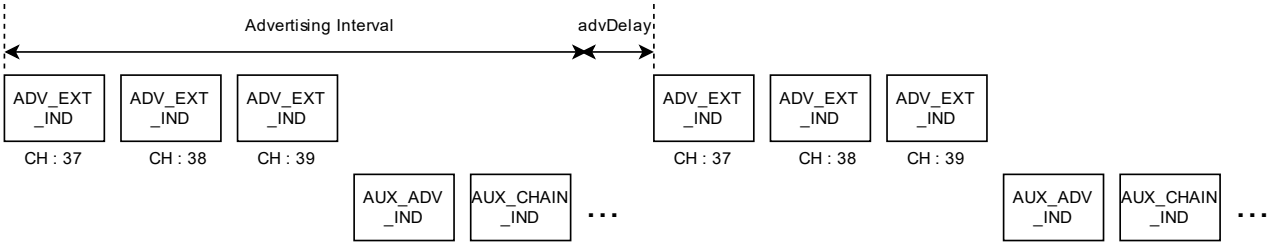


図 5-2　Extended Advertising の PDU

Scannable タイプを指定し、スキャンレスポンスデータを設定して Advertising を行った場合、リモートデバイスからのスキャンリクエストに対して、表 5.2 のスキャンレスポンスデータを図 5-3 のように送信します。

表 5.2　スキャンレスポンスデータ

adv_prop_type フィールドに設定した値	スキャンレスポンスデータ PDU	legacy or extended	Max サイズ（バイト）
BLE_GAP_LEGACY_PROP_ADV_IND BLE_GAP_LEGACY_PROP_ADV_SCAN_IND	SCAN_RSP	legacy	31
BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_UNDIRECT BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_DIRECT BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_HDC_DIRECT	AUX_SCAN_RSP AUX_CHAIN_IND[*1]	extended	BLE_CFG_RF_ADV_DATA_MAX の設定値[*2 *3]

[*1]：スキャンレスポンスデータのサイズが 253 バイト以下の場合(BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER を使用する場合はさらに-1 バイト)、スキャンレスポンスデータを AUX_SCAN_RSP だけで送信できるため、AUX_CHAIN_IND は未使用となります。

[*2]：adv_prop_type に BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER を追加する場合、最大サイズ-1 バイトとなります。

[*3]：スキャンレスポンスデータのサイズが 230 バイト以上の場合、スキャンレスポンスデータが受信側の Bluetooth の仕様に従って分割されるため、必要に応じて受信側で結合してください。

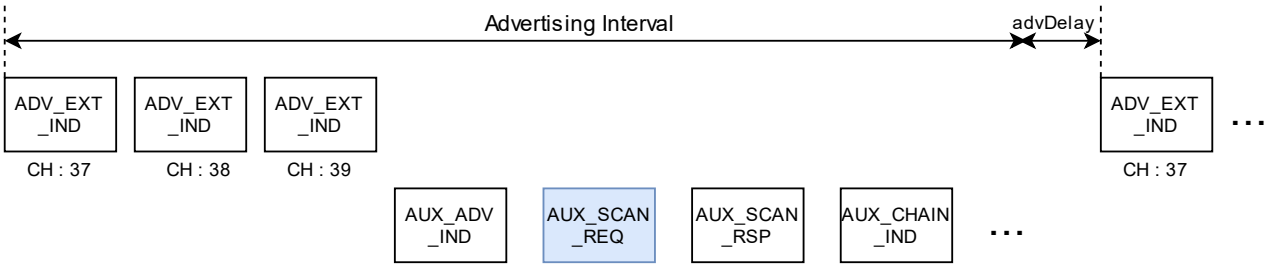


図 5-3　Scannable Advertising の PDU

青字の PDU はリモートデバイスの PDU です。

Direct タイプの場合、st_ble_gap_adv_param_t 構造体の p_addr_type、p_addr フィールドに Advertising を送信するリモートデバイスのアドレスを指定します。

Extended タイプの場合、st_ble_gap_adv_param_t 構造体の adv_phy、sec_adv_phy フィールドに Advertising を送信する PHY を指定します。adv_phy にはプライマリチャネル(CH:37/38/39)の PHY(1M PHY または Coded PHY)を指定します。sec_adv_phy にはセカンダリチャネル(CH:37/38/39 以外)の PHY(1M PHY または 2M PHY または Coded PHY)を指定します。

5.2.1.2　ホワイトリストの使用(既知のデバイスに応答する)

Connectable、または、Scannable タイプの Advertising を使う場合、ホワイトリストを使うことで要求を受け入れるリモートデバイスを限定することが可能です。

要求元デバイスの BD アドレスが判明している場合、下記 1, 2 を行います。

1. ホワイトリストに既知のデバイスの BD アドレスを登録
   R_BLE_GAP_ConfWhiteList により、既知のデバイスを登録します。

【注】ホワイトリストフィルタを有効にした動作(アドバタイズ、スキャン、コネクションリクエスト)を実行している場合はホワイトリストの追加・削除は行えません。

2. Advertising filter policy の設定
   st_ble_gap_adv_param_t 構造体の filter_policy フィールドに表 5.3 の値を設定します。

表 5.3　filter_policy フィールドの設定

filter_policy フィールドに設定する値	説明
BLE_GAP_SCAN_ALLOW_ADV_ALL(0x00)	すべてのデバイスからのスキャンリクエストとコネクションリクエストに応答します。
BLE_GAP_ADV_ALLOW_SCAN_WLST_CONN_ANY(0x01)	ホワイトリストに登録したデバイスからのスキャンリクエストに応答し、すべてのデバイスからのコネクションリクエストに応答します。
BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_WLST(0x02)	すべてのデバイスからのスキャンリクエストに応答し、ホワイトリストに登録したデバイスからのコネクションリクエストに応答します。
BLE_GAP_ADV_ALLOW_SCAN_WLST_CONN_WLST(0x03)	ホワイトリストに登録したデバイスからのスキャンリクエストとコネクションリクエストに応答します。

5.2.1.3　プライバシ

Advertising が他デバイスから追跡されないようにプライバシ機能が使用可能です。

「9.4.1 ローカルデバイスの RPA 生成」の手順でプライバシ用の鍵の準備を行った後、

st_ble_gap_adv_param_t 構造体の表 5.4 のフィールドを設定することで Advertising のアドレスが RPA になり、周期的（更新間隔のデフォルトは 900 秒です。R_BLE_GAP_SetRpaTo()で変更できます）に異なるアドレスに変わります。

表 5.4　プライバシ使用時に設定するパラメータ

フィールド	フィールドに設定する値	説明
o_addr_type	BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)	R_BLE_GAP_SetLocIdInfo()で登録した Identity Address が Public Address の場合に指定します。
	BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)	R_BLE_GAP_SetLocIdInfo()で登録した Identity Address が Static Address の場合に指定します。
o_addr	R_BLE_GAP_SetLocIdInfo()で登録した Static Address を指定します。	o_addr_type が BLE_GAP_ADDR_RPA_ID_RANDOM(0x03) の場合に指定します。
p_addr_type p_addr	R_BLE_GAP_ConfRslvList()で登録した リモートデバイスの Identity Address を指定します。	—

【BP】　デバイスを一意に識別するアドバタイズデータを含めると、プライベートアドレスを使用してデバイスを隠す目的が無効になる可能性があります。プライベートアドレスを使用する場合、アドバタイズメントペイロードの難読化が推奨されています。

【BP】　デバイスアドレスが周期的に異なるアドレスに変わる場合でもアドバタイズデータを使用してデバイスを追跡できます。アドレスとデータを同時に更新することが推奨されています。

5.2.1.4　同時実行

All features ライブラリを使用している場合、Advertising は同時に BLE_CFG_RF_ADV_SET_MAX の設定値まで行うことが可能です。Advertising の識別は st_ble_gap_ext_adv_param_t 構造体の adv_hdl で示される、Advertising handle で行います。図 5-1 の各手順は操作対象の Advertising を Advertising handle により選択します。

Balance, Compact ライブラリは同時に実行可能な Advertising は 1 つのみとなります。

抽象 API と GAP API による Advertising を併用する場合、Advertising 中の Advertising Handle は使用不可となるため、注意してください。

5.2.2　Advertising Data / スキャンレスポンスデータの設定 / 更新

Advertising Data / スキャンレスポンスデータの設定については、「5.4 Advertising Data / スキャンレスポンスデータ / Periodic Advertising Data の設定」をご参照ください。

Advertising Data / スキャンレスポンスデータの更新については、「5.4.2 Advertising 中の更新」をご参照ください。

5.2.3　Advertising 開始

Advertising を開始するときは、

```
ble_status_t R_BLE_GAP_StartAdv   ( uint8_t   adv_hdl,
                                    uint16_t  duration,
                                    uint8_t   max_extd_adv_evts)
```

をコールします。

All features ライブラリを使用する場合、Advertising を行う期間(duration パラメータ x10ms)、または、Advertising の送信数(max_extd_adv_evts パラメータ)で終了するタイミングを指定可能です。

5.2.4　Advertising 停止

Connectable タイプの Advertising 中にリモートデバイスとコネクション確立すると、Advertising が終了します。

API コールにより Advertising を停止するときは、

```
ble_status_t R_BLE_GAP_StopAdv   ( uint8_t   adv_hdl )
```

を使用します。Extended advertising で 252 バイト以上の Advertising data を更新する場合は、Bluetooth の仕様上、更新の前に Advertising を停止する必要があります。

5.3　GAP API による Periodic Advertising

Periodic Advertising は決まった間隔での送信が求められる場合に使います。All features ライブラリ使用時に Periodic Advertising が使用可能です。

アプリケーション上での Periodic Advertising の手順を図 5-4 に示します。Periodic Advertising 特有の各ステップの詳細について以降の章で説明します。

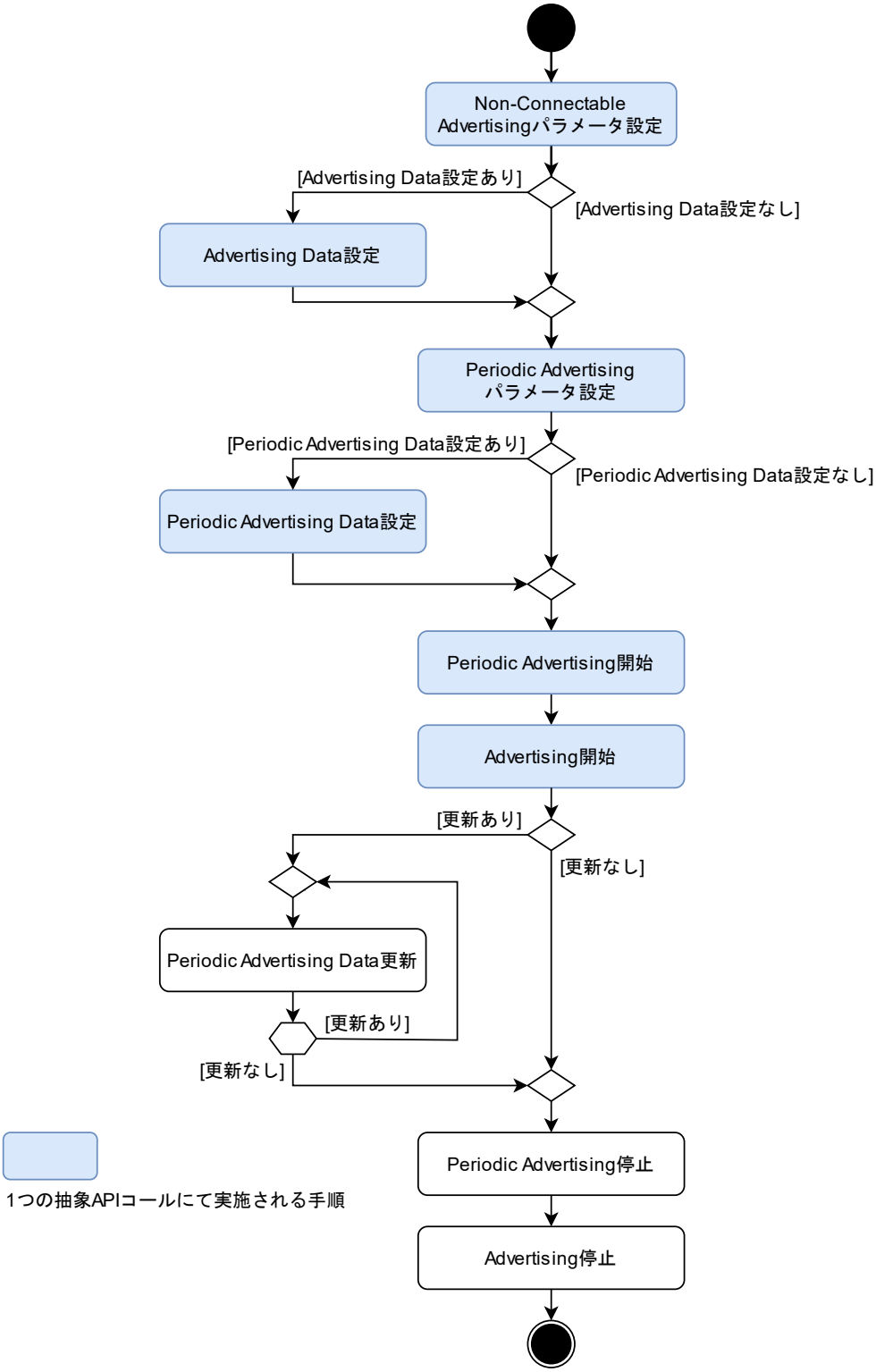


図 5-4　Periodic Advertising の手順

5.3.1　Non-Connectable Advertising パラメータ設定

Periodic Advertising を行うにあたり、R_BLE_GAP_SetAdvParam により Advertising パラメータ設定を行います。Advertising のタイプは表 5.1 の extended の non-connectable タイプである、以下を使います。

- BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_UNDIRECT
- BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_DIRECT
- BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_HDC_DIRECT

5.3.2　Periodic Advertising パラメータ設定

Periodic Advertising のパラメータ設定は

ble_status_t R_BLE_GAP_SetPerdAdvParam(st_ble_gap_perd_adv_param_t * p_perd_adv_param)

を使います。Periodic Advertising のパラメータ設定によって、表 5.5 の PDU(AUX_SYNC_IND と AUX_CHAIN_IND)が Non-connectable Advertising の PDU(ADV_EXT_INDs と AUX_ADV_IND)の後に続きます。R_BLE_GAP_SetAdvParam と R_BLE_GAP_SetPerdAdvParam で設定する interval の違いを図 5-5 に示します。図中の ADV_EXT_INDs は 37,38, 39ch の ADV_EXT_IND PDU をまとめたものです。

表 5.5　Periodic Advertising の PDU

Advertising Type	Periodic Advertising PDU	legacy or extended	Max サイズ（バイト）
Periodic Advertising	AUX_SYNC_IND	extended	BLE_CFG_RF_ADV_DATA_MAX の設定値*2 *3
	AUX_CHAIN_IND*1		

*1 : Periodic Advertising Data のサイズが 253 バイト以下の場合
(BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER を使用する場合はさらに-1 バイト)、Periodic Advertising Data を AUX_SYNC_IND だけで送信できるため、AUX_CHAIN_IND は未使用となります。

*2 : adv_prop_type に BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER を追加する場合、最大サイズ-1 バイトとなります。

*3 : Periodic Advertising Data のサイズが 248 バイト以上の場合、Periodic Advertising Data が受信側の Bluetooth の仕様に従って分割されるため、必要に応じて受信側で結合してください。

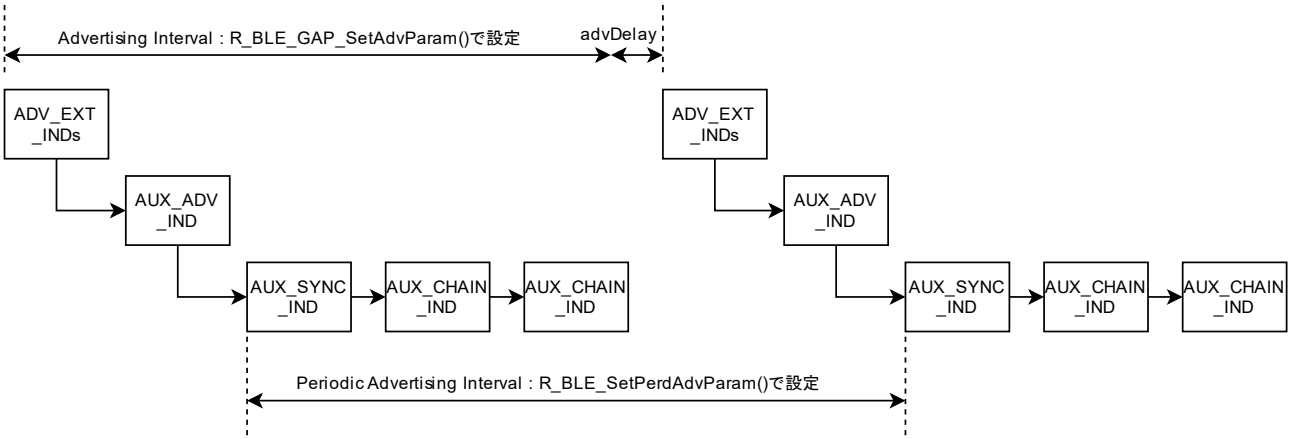


図 5-5　Periodic Advertising の PDU

5.3.3 Periodic Advertising Data 設定／更新

Periodic Advertising Data の設定については、「5.4 Advertising Data / スキャンレスポンスデータ / Periodic Advertising Data の設定」をご参照ください。

Periodic Advertising Data の更新については、「5.4.3 Periodic Advertising 中の更新」をご参照ください。

5.3.4 Periodic Advertising 開始

Periodic Advertising を開始するときは、

ble_status_t R_BLE_GAP_StartPerdAdv (uint8_t adv_hdl)

をコールします。Advertising が開始しておらず、non-connectable の Advertising PDU が送信されていない場合、この API をコールしても Periodic Advertising PDU は送信されません。

Periodic Advertising 開始までのサンプルを以下に示します。

```
/* Advertising data */
static uint8_t gs_adv_data[] =
{

    /* Flag (mandatory) */
    2,          /* Data Size */
    0x01,       /* Data Type: Flag */
    (BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE |
     BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED),    /* Data */

    /* Complete Local Name */
    9,          /* Data Size */
    0x09,       /* Data Type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /* Data */
};

/* Periodic Advertising Data */
static uint8_t gs_perd_adv_data[] =
{

    /* Complete Local Name */
    9,         /* Data Size */
    0xFF,      /* Data Flag: Manufacturer Specific data type  */
    0x36, 0x00,/* Company ID: Renesas Electronics Corporation */
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05,  /* Data */

};

/* 省略 */
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    ble_app_gapcb(type, result, p_data);
        st_ble_gap_adv_set_evt_t * p_adv_set_param;

        switch(type)
    {
                case BLE_GAP_EVENT_STACK_ON :
                {
                        st_ble_gap_adv_param_t adv_param =
                        {
                            .adv_hdl       = 0x02,
                            .adv_prop_type = BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_UNDIRECT,
                            .adv_intv_min  = 0x0200,
                            .adv_intv_max  = 0x0200,
                            .adv_ch_map    = BLE_GAP_ADV_CH_ALL,
                            .o_addr_type   = BLE_GAP_ADDR_PUBLIC,
                            .filter_policy = BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_ANY,
                            .adv_phy       = BLE_GAP_ADV_PHY_1M,
                            .sec_adv_phy   = BLE_GAP_ADV_PHY_1M,
                        };
                        /* Set Advertising parameter */
                        R_BLE_GAP_SetAdvParam(&adv_param);
                }
                break;

                case BLE_GAP_EVENT_ADV_PARAM_SET_COMP :
```

```
                {
                        p_adv_set_param = (st_ble_gap_adv_set_evt_t *)p_data->p_param;
                        st_ble_gap_adv_data_t adv_data_param = {
                                .adv_hdl        = 0x02,
                                .data_type      = BLE_GAP_ADV_DATA_MODE,
                                .data_length    = ARRAY_SIZE(gs_adv_data),
                                .p_data         = gs_adv_data  ,
                        };
                        /* Set Advertising Data */
                        R_BLE_GAP_SetAdvSresData(&adv_data_param);
                }
                break;

                case BLE_GAP_EVENT_PERD_ADV_PARAM_SET_COMP :
                {
                        /* Periodic Advertising Data parameter */
                        st_ble_gap_adv_data_t perd_adv_data_param = {
                                .adv_hdl        = 0x02,
                                .data_type      = BLE_GAP_PERD_ADV_DATA_MODE,
                                .data_length    = ARRAY_SIZE(gs_perd_adv_data),
                                .p_data         = gs_perd_adv_data  ,
                        };

                        /* Set Periodic Advertising Data */
                        R_BLE_GAP_SetAdvSresData(&perd_adv_data_param);
                }
                break;

                case BLE_GAP_EVENT_PERD_ADV_ON :
                {
                        p_adv_set_param = (st_ble_gap_adv_set_evt_t *)p_data->p_param;
                        /* Start Advertising */
                        R_BLE_GAP_StartAdv(0x02, 0, 0);
                }
                break;

                case BLE_GAP_EVENT_ADV_DATA_UPD_COMP :
                {
                    st_ble_gap_adv_data_evt_t * p_adv_data_set_param;
                    p_adv_data_set_param = (st_ble_gap_adv_data_evt_t *)p_data->p_param;
                        if(BLE_GAP_ADV_DATA_MODE == p_adv_data_set_param->data_type)
                        {
                                st_ble_gap_perd_adv_param_t perd_param =
                                {
                                        .adv_hdl       = 0x02,
                                        .prop_type     = 0x0000,
                                        .perd_intv_min = 0x0100,
                                        .perd_intv_max = 0x0100,
                                };
                                /* Set Periodic Advertising parameter */
                                R_BLE_GAP_SetPerdAdvParam(&perd_param);
                        }
                        else
                        {
                                if(BLE_GAP_PERD_ADV_DATA_MODE == p_adv_data_set_param->data_type)
                                {
                                        /* Start Periodic Advertising parameter */
                                R_BLE_GAP_StartPerdAdv(0x02);
                                }
                        }
                }
                break;

                default:
                        break;
    }
}
```

コード 5-1　Periodic Advertising 開始までの処理の例

5.3.5 Periodic Advertising 停止

Periodic Advertising を停止するときは、

ble_status_t R_BLE_GAP_StopPerdAdv(uint8_t adv_hdl)

をコールします。表 5.5 の PDU の送信のみ停止します。Periodic Advertising で 253 バイト以上の Periodic Advertising Data を更新する場合は、Bluetooth の仕様上、更新の前に Periodic Advertising を停止する必要があります。

5.4　Advertising Data / スキャンレスポンスデータ / Periodic Advertising Data の設定

Advertising Data / スキャンレスポンスデータ / Periodic Advertising Data の設定と Advertising 中の更新のどちらも R_BLE_GAP_SetAdvSresData を使用します。Advertising Data、スキャンレスポンスデータ、Periodic Advertising Data のフォーマットは同じです。st_ble_gap_adv_data_t パラメータ構造体の data_type フィールドが表 5.6 のように異なります。

表 5.6　data_type フィールド

データタイプ	data_type フィールドの設定値
Advertising Data	BLE_GAP_ADV_DATA_MODE(0x00)
スキャンレスポンスデータ	BLE_GAP_SCAN_RSP_DATA_MODE(0x01)
Periodic Advertising Data	BLE_GAP_PERD_ADV_DATA_MODE(0x02)

Advertising Data / スキャンレスポンスデータの設定を連続で行う場合は、Advertising Data 設定用に R_BLE_GAP_SetAdvSresData をコールした後、GAP のコールバック関数の中で Advertising Data の設定完了を確認してから、スキャンレスポンスデータ設定用に R_BLE_GAP_SetAdvSresData をコールしてください。

5.4.1　フォーマット

設定するデータのフォーマットを図 5-6 に示します。

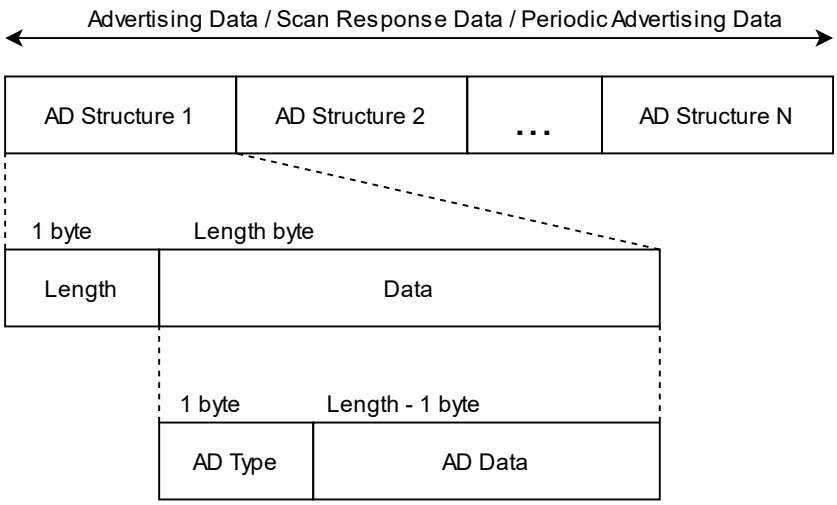


図 5-6　Advertising Data / スキャンレスポンスデータ / Periodic Advertising Data のフォーマット

Advertising Data / スキャンレスポンスデータ / Periodic Advertising Data は 1 つ以上の AD structure を含みます。各 AD structure は Length と AD Type と AD Data から構成されています。Length は AD Type のサイズ(1 バイト)と AD Data のサイズの和となります。Bluetooth SIG が定義している AD Type は Supplement to the Bluetooth Core Specification (CSS)に定義されています。よく使われる AD Type を表 5.7 に示します。

表 5.7　AD Type と AD Data

<table>
<tr><th colspan="2">Data type</th><th>AD Type</th><th>AD Data</th></tr>
<tr><td colspan="2">Flags</td><td>0x01</td><td>Connectable タイプの Advertising 時に使用します。
Bluetooth LE で使用する Flags の値について下記に示します。
<table><tr><th>Octet</th><th>Bit</th><th>説明</th></tr><tr><td>0</td><td>0</td><td>LE Limited Discoverable Mode</td></tr><tr><td>0</td><td>1</td><td>LE General Discoverable Mode</td></tr><tr><td>0</td><td>2</td><td>BR/EDR Not Supported.</td></tr></table>Discoverable Mode はスキャナ側でモードによってフィルタリングしたい場合に使用します。Discoverable Mode を追加する場合、Limited or General のどちらかを選択します。</td></tr>
<tr><td rowspan="6">Service UUID</td><td>Incomplete List of 16-bit Service UUIDs</td><td>0x02</td><td rowspan="6">UUID のリストを設定します。
サイズごとに AD Type は異なります。
すべての UUID を AD Data に含める場合は、Complete List、
一部の UUID のみ AD Data に含める場合は、Incomplete List を
指定します。</td></tr>
<tr><td>Complete List of 16-bit Service UUIDs</td><td>0x03</td></tr>
<tr><td>Incomplete List of 32-bit Service UUIDs</td><td>0x04</td></tr>
<tr><td>Complete List of 32-bit Service UUIDs</td><td>0x05</td></tr>
<tr><td>Incomplete List of 128-bit Service UUIDs</td><td>0x06</td></tr>
<tr><td>Complete List of 128-bit Service UUIDs</td><td>0x07</td></tr>
<tr><td rowspan="2">Local Name</td><td>Shortened Local Name</td><td>0x08</td><td>デバイス名の先頭から途中までの文字列を設定します。</td></tr>
<tr><td>Complete Local Name</td><td>0x09</td><td>デバイス名を設定します。</td></tr>
<tr><td colspan="2">Manufacturer Specific Data</td><td>0xFF</td><td>2 バイト以上のベンダー固有のデータです。
最初の 2 バイトは Company ID を設定します。
Company ID は Assigned Number
(https://www.bluetooth.com/specifications/assigned-numbers/)
をご参照ください。</td></tr>
</table>

以下は Flags と Complete Local Name を含む Advertising Data と Complete Local Name を含むスキャンレスポンスデータを設定するサンプルです。

```
/* Advertising Data */
uint8_t gs_adv_data[] =
{
    /* Flags */
    2,          /*  Data Size: 2byte */
    0x01,       /*  AD type: Flags */
    (BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE |
     BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED), /* Data */

    /* Complete Local Name */
    9,         /*  Data Size: 9byte */
    0x09,      /*  AD type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V',  /* Data */
};

/* Scan_Response Data */
uint8_t gs_sres_data[] =
{
    /* Complete Local Name */
    9,         /*  Data Size: 9byte */
    0x09,      /*  AD type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /* Data */

};
/* 省略  */

/* Advertising Data parameter */
st_ble_gap_adv_data_t adv_data_param = {
        .adv_hdl        = 0x00,
        .data_type      = BLE_GAP_ADV_DATA_MODE,
        .data_length    = ARRAY_SIZE(gs_adv_data),
        .p_data         = gs_adv_data  ,
};

/* Scan_Response Data parameter */
st_ble_gap_adv_data_t sres_data_param = {
```

```
        .adv_hdl        = 0x00,
        .data_type      = BLE_GAP_SCAN_RSP_DATA_MODE,
        .data_length    = ARRAY_SIZE(gs_sres_data),
        .p_data         = gs_sres_data,
};

/* 省略  */

/* Set Advertising Data */
R_BLE_GAP_SetAdvSresData(&adv_data_param);

/* 省略  */

/* GAP Callback  */
void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch(type)
    {
        /* 省略  */
        case BLE_GAP_EVENT_ADV_DATA_UPD_COMP :
            st_ble_gap_adv_data_evt_t * p_adv_data_set_param;
            p_adv_data_set_param = (st_ble_gap_adv_data_evt_t *)p_data->p_param;
            if((0x00 == p_adv_data_set_param->adv_hdl) &&
               (BLE_GAP_ADV_DATA_MODE == p_adv_data_set_param->data_type))
            {
                R_BLE_GAP_SetAdvSresData(&sres_data_param);
            }
        break;

        /* 省略  */

    }
}
```

コード 5-2　Advertising Data とスキャンレスポンスデータの設定の例

5.4.2　Advertising 中の更新

表 5.8 の条件を満たす場合、Advertising 中に Advertising Data / スキャンレスポンスデータを更新することが可能です。

表 5.8　Advertising Data / スキャンレスポンスデータの更新の条件

Advertising のタイプ	条件
Legacy advertising	条件なし
Extended advertising	data_length が 251 バイト以下

Advertising Data / スキャンレスポンスデータの更新は以下のようにパラメータを設定し、R_BLE_GAP_SetAdvSresData をコールします。

```
st_ble_gap_adv_data_t adv_data_param = {
    .adv_hdl        = “更新するAdvertisingのAdvertising handle,
    .data_type      = “BLE_GAP_ADV_DATA_MODEまたはBLE_GAP_SCAN_RSP_DATA_MODE”,
    .data_length    = “更新するデータのサイズ”,
    .p_data         = “更新するデータへのポインタ”,
};
```

コード 5-3　Advertising Data / スキャンレスポンスデータの更新用のパラメータ

Extended advertising で 252 バイト以上のデータを更新する場合、5.2.4 の手順で Advertising を停止してから R_BLE_GAP_SetAdvSresData により更新を行う必要があります。

5.4.3　Periodic Advertising 中の更新

表 5.9 の条件を満たす場合、Periodic Advertising 中に Periodic Advertising Data を更新することが可能です。

表 5.9　Periodic Advertising Data の更新の条件

Advertising のタイプ	条件
Periodic Advertising	data_length が 252 バイト以下

Periodic Advertising Data の更新は以下のようにパラメータを設定し、R_BLE_GAP_SetAdvSresData をコールします。

```
st_ble_gap_adv_data_t adv_data_param = {
    .adv_hdl        = “更新するPeriodic AdvertisingのAdvertising handle,
    .data_type      = BLE_GAP_PERD_ADV_DATA_MODE,
    .data_length    = “更新するデータのサイズ”,
    .p_data         = “更新するデータへのポインタ”,
};
```

コード 5-4　Periodic Advertising Data の更新用のパラメータ

Periodic Advertising で 253 バイト以上のデータを更新する場合、5.3.5 の手順で Periodic Advertising を停止してから R_BLE_GAP_SetAdvSresData により更新を行う必要があります。

5.4.4　バッファサイズ

Bluetooth LE Protocol Stack の Advertising Data / スキャンレスポンスデータ用のバッファは 4250 バイトです。表 5.1 に示すように Extended advertising は BLE_CFG_RF_ADV_DATA_MAX の設定値まで Advertising Data / スキャンレスポンスデータを設定することが可能です。ただし、同時実行中の Advertising が使っている Advertising Data / スキャンレスポンスデータの合計サイズはバッファサイズの 4250 バイト以下にする必要があります。

Periodic Advertising Data 用のバッファは 4306 バイトです。Periodic Advertising は BLE_CFG_RF_ADV_DATA_MAX の設定値まで設定することが可能です。同時実行中の Periodic Advertising が使っている Periodic Advertising Data の合計サイズはバッファサイズの 4306 バイト以下にする必要があります。

図 5-7、図 5-8 に同時実行中の Advertising Data の設定例を示します。ここでは、BLE_CFG_RF_ADV_DATA_MAX を 1650 に設定しています。Advertising Data / スキャンレスポンスデータバッファの空きサイズは R_BLE_GAP_GetRemainAdvBufSize()により知ることができます。

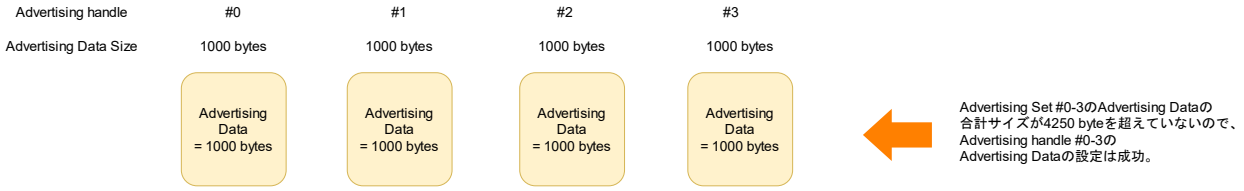


図 5-7　Advertising Data 設定の成功例

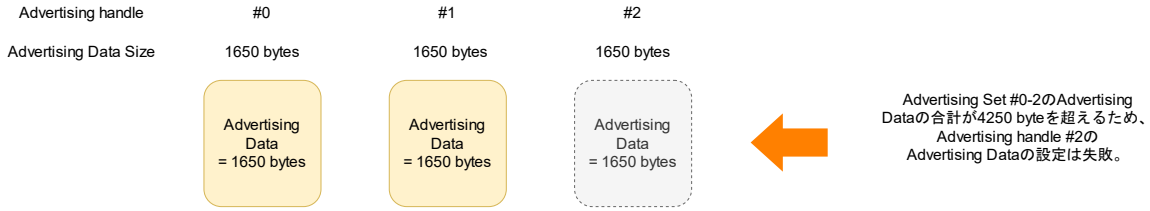


図 5-8　Advertising Data 設定の失敗例

5.5　抽象 API による Advertising

抽象 API が有効の場合、図 5-1 の Advertising パラメータ設定から Advertising 開始までを 1 つ API コールで行うことが可能です。抽象 API でサポートしている Advertising のタイプを表 5.10 に示します。

表 5.10　抽象 API がサポートする Advertising

抽象 API	Legacy or Extended	Advertising Type	Advertising PDU	Advertising handle	Advertising Data の Max サイズ (バイト)
R_BLE_ABS_StartLegacyAdv	Legacy	Connectable and Scannable Undirected	ADV_IND	0	31
R_BLE_ABS_StartExtAdv	Extended	Connectable Undirected	ADV_EXT_IND AUX_ADV_IND	1	245
		Connectable Directed	ADV_EXT_IND AUX_ADV_IND		239
R_BLE_ABS_StartNonConnAdv	Legacy	Non-Connectable and Non-Scannable Undirected	ADV_NONCONN_IND	2	31
	Extended		ADV_EXT_IND AUX_ADV_IND AUX_CHAIN_IND		BLE_CFG_RF_ADV_DATA_MAX の設定値
		Non-Connectable and Non-Scannable Directed	ADV_EXT_IND AUX_ADV_IND AUX_CHAIN_IND		BLE_CFG_RF_ADV_DATA_MAX の設定値
R_BLE_ABS_StartPerdAdv	Extended	Periodic	ADV_EXT_IND AUX_ADV_IND AUX_SYNC_IND AUX_CHAIN_IND	3	BLE_CFG_RF_ADV_DATA_MAX の設定値

抽象 API と GAP API による Advertising を併用する場合、Advertising 中の Advertising Handle は使用不可となるため、注意してください。

5.5.1　ホワイトリストの使用(既知のデバイスに応答する)

R_BLE_ABS_StartLegacyAdv と R_BLE_ABS_StartExtAdv はホワイトリストの使用が可能です。下記手順により、要求を受け入れるリモートデバイスを限定することが可能です。

1. ホワイトリストに既知のデバイスの BD アドレスを登録
   R_BLE_GAP_ConfWhiteList により、既知のデバイスを登録します。

【注】ホワイトリストフィルタを有効にした動作(アドバタイズ、スキャン、コネクションリクエスト)を実行している場合はホワイトリストの追加・削除は行えません。

2. Advertising filter policy の設定
   R_BLE_ABS_StartLegacyAdv の場合は st_ble_abs_legacy_adv_param_t 構造体の filter フィールドを R_BLE_ABS_StartExtAdv の場合は st_ble_abs_ext_adv_param_t 構造体の filter フィールドに表 5.3 の値を設定します。

5.5.2　プライバシ

R_BLE_ABS_StartLegacyAdv、R_BLE_ABS_StartExtAdv、R_BLE_ABS_StartNonConnAdv、R_BLE_ABS_StartPerdAdv はプライバシ機能が使用可能です。「9.4.1 ローカルデバイスの RPA 生成」の手順でプライバシ用の鍵の準備を行った後、st_ble_abs_legacy_adv_param_t、st_ble_abs_ext_adv_param_t、st_ble_abs_non_conn_adv_param_t 構造体の表 5.11 のフィールドを設定することで Advertising のアドレスが RPA になり、周期的（更新間隔のデフォルトは 900 秒です。R_BLE_GAP_SetRpaTo()で変更できます）に異なるアドレスに変わります。

表 5.11　プライバシ使用時に設定するパラメータ

フィールド	フィールドに設定する値	説明
o_addr_type	BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)	R_BLE_GAP_SetLocIdInfo()で登録した Identity Address が Public Address の場合に指定します。
	BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)	R_BLE_GAP_SetLocIdInfo()で登録した Identity Address が Static Address の場合に指定します。
o_addr	R_BLE_GAP_SetLocIdInfo()で登録した Static Address を指定します。	o_addr_type が BLE_GAP_ADDR_RPA_ID_RANDOM(0x03) の場合に指定します。
p_addr	R_BLE_GAP_ConfRslvList()で登録したリモートデバイスの Identity Address を指定します。	—

5.6　スマートフォンとコネクション

スマートフォンとコネクションするために、connectable の Legacy Advertising を行います。この Advertising は R_BLE_ABS_StartLegacyAdv を使用します。下記にスマートフォンとコネクションするための Advertising を送信するサンプルを示します。

```
/* Advertising Data */
static uint8_t gs_adv_data[] =
{
    /* Flag (mandatory) */
    2,          /**< Data Size */
    0x01,       /**< Data Flag: Flag */
    (BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE | BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED), /**< Data
Value */

    /* Complete Local Name */
    9,         /**< Data Size */
    0x09,       /**< Data Flag: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /**< Data Value */

};

/* Scan_Response Data */
static uint8_t gs_sres_data[] =
{
    /* Complete Local Name */
    9,         /**< Data Size */
    0x09,       /**< Data Flag: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /**< Data Value */

};

/* Advertising parameters */
static st_ble_abs_legacy_adv_param_t gs_adv_param =
{
    .slow_adv_intv   = 0x00A0,
    .slow_period     = 0,
    .p_adv_data      = gs_adv_data,
    .adv_data_length = ARRAY_SIZE(gs_adv_data),
    .p_sres_data     = gs_sres_data,
    .sres_data_length = ARRAY_SIZE(gs_sres_data),
    .adv_ch_map      = BLE_GAP_ADV_CH_ALL,
    .filter          = BLE_ABS_ADV_ALLOW_CONN_ANY,
    .o_addr_type     = BLE_GAP_ADDR_PUBLIC,
    .o_addr          = {0},
};

/* 省略  */

/* Advertisingの開始 */
R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
```

コード 5-5　スマートフォンとのコネクション用の Advertising の例

Advertising が始まると、BLE_GAP_EVENT_ADV_ON イベントを通知します。このイベント通知後、スマートフォンからスキャンしてコネクションが可能となります。

5.7　ビーコンとして使用

ビーコンとして non-connectable の Advertising を送信するユースケースにおいて、R_BLE_ABS_StartNonConnAdv を使う場合のサンプルを以下に示します。

```
/* Advertising Data */
static uint8_t gs_adv_data[] =
{
    /* Flag */
    2,          /**< Data Size */
    0x01,       /**< Data Flag: Flag */
    BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED,    /**< Data Value */

    /* Complete Local Name */
    9,         /* Data Size */
    0x09,      /* Data Flag: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /* Data */

};

/* Advertising parameters */
static st_ble_abs_non_conn_adv_param_t gs_non_conn_adv_param =
{
    .p_addr          = NULL,
    .p_adv_data      = gs_adv_data,
    .adv_intv        = 0x00A0,
    .duration        = 0,
    .adv_data_length = ARRAY_SIZE(gs_adv_data),
    .adv_ch_map      = BLE_GAP_ADV_CH_ALL,
    .o_addr_type     = BLE_GAP_ADDR_PUBLIC,
    .adv_phy         = BLE_GAP_ADV_PHY_1M,
    .sec_adv_phy     = BLE_GAP_ADV_PHY_1M,
    .o_addr          = {0},
};

/* 省略  */

/* Advertisingの開始 */
R_BLE_ABS_StartNonConnAdv (&gs_non_conn_adv_param);
```

コード 5-6　R_BLE_ABS_StartNonConnAdv の使用例

Advertising が始まると、BLE_GAP_EVENT_ADV_ON イベントを通知します。このイベント通知後、リモートデバイスからスキャンしてビーコンの検出が可能となります。

スマートフォンは Legacy advertising の Non-connectable advertising パケットのみ対応していることがあります。ビーコンの確認に使用するデバイスがスキャンできる Advertising を送信してください。

Apple 社の iBeacon、Google 社の Eddystone でビーコンを使う場合、Non-connectable の Advertising を使用します。詳細な仕様については以下をご参照ください。

iBeacon　: https://developer.apple.com/ibeacon/
Eddystone : https://github.com/google/eddystone

6. スキャン

スキャンは周辺のデバイスから Advertising を受信します。
スキャンを行う場合は、All features か Balance タイプの Bluetooth LE Protocol Stack ライブラリを使用します。All features は Extended advertising と Legacy advertising の Advertising パケットを受信します。Balance タイプは Legacy advertising の Advertising パケットのみ受信します。

6.1 スキャンの開始／停止

スキャンは以下の API をコールすることにより開始します。

スキャン開始 API :
- R_BLE_GAP_StartScan
- R_BLE_ABS_StartScan

上記 API のパラメータにスキャンを行う期間を指定した場合、その期間が経過すると、スキャンが停止します。それ以外の場合、以下の API をコールすることにより、スキャンが停止します。目的のデバイスを検出した場合やスキャンパラメータを変更したい場合などにスキャンを停止します。

スキャン停止 API :
- R_BLE_GAP_StopScan

6.2 スキャンパラメータ

スキャン開始 API のパラメータを表 6.1-表 6.5 に示します。
[R_BLE_GAP_StartScan] ※引数 1(st_ble_gap_scan_param_t *)、引数 2(st_ble_gap_scan_on_t *)

表 6.1　st_ble_gap_scan_param_t 構造体

タイプ	フィールド	説明
uint8_t	o_addr_type	アクティブスキャンのときにスキャンリクエストに含めるアドレスのタイプを指定します。
uint8_t	filter_policy	どのようなデバイスから受信するかを指定します。
st_ble_gap_scan_phy_param_t *	p_phy_param_1M	1MPHY のスキャンパラメータです。
st_ble_gap_scan_phy_param_t *	p_phy_param_coded	Coded PHY のスキャンパラメータです。

表 6.2　st_ble_gap_scan_phy_param_t 構造体

タイプ	フィールド	説明
uint8_t	scan_type	active or passive scan を指定します。 スキャンレスポンスデータを受信する場合はアクティブスキャンにします。
uint16_t	scan_intv	スキャンインターバルです。
uint16_t	scan_window	スキャンウインドウです。

表 6.3　st_ble_gap_scan_on_t 構造体

タイプ	フィールド	説明
uint8_t	proc_type	スキャンのプロシージャのタイプを指定します。
uint8_t	filter_dups	同じデバイスから重複する Advertising を受信するかどうかを指定します。
uint16_t	duration	スキャンを行う期間です。
uint16_t	period	スキャンを行う期間です。

[R_BLE_ABS_StartScan]

表 6.4　st_ble_abs_scan_param_t 構造体

タイプ	フィールド	説明
st_ble_abs_scan_phy_param_t *	p_phy_param_1M	1MPHY のスキャンパラメータです。
st_ble_abs_scan_phy_param_t *	p_phy_param_coded	Coded PHY のスキャンパラメータです。
uint8_t *	p_filter_data	フィルタリングするデータを指定します。
uint16_t	fast_period	fast scan を行う期間です。
uint16_t	slow_period	slow scan を行う期間です。
uint16_t	filter_data_length	フィルタリングするデータのサイズです。
uint8_t	dev_filter	どのようなデバイスから受信するかを指定します。
uint8_t	filter_dups	同じデバイスから重複する Advertising を受信するかどうかを指定します。
uint8_t	filter_ad_type	フィルタリングするデータの AD_TYPE を指定します。

表 6.5　st_ble_abs_scan_phy_param_t 構造体

タイプ	フィールド	説明
uint16_t	fast_intv	fast scan 用のスキャンインターバルです。
uint16_t	slow_intv	fast scan 用のスキャンウインドウです。
uint16_t	fast_window	slow scan 用のスキャンインターバルです。
uint16_t	slow_window	slow scan 用のスキャンウインドウです。
uint8_t	scan_type	active or passive scan を指定します。 スキャンレスポンスデータを受信する場合はアクティブスキャンにします。

スキャンを行う PHY は p_phy_param_1M、p_phy_param_coded により指定します。プライマリチャネル(CH:37/38/39)が 1M PHY の Advertising を受信するには、p_phy_param_1M の設定が必要です。プライマリチャネルが Coded PHY の Advertising を受信するには、p_phy_param_coded の設定が必要です。

スキャンを行う間隔、期間はスキャンインターバル、スキャンウインドウ、duration、period により指定します。
これらのパラメータの関係を図 6-1 に示します。

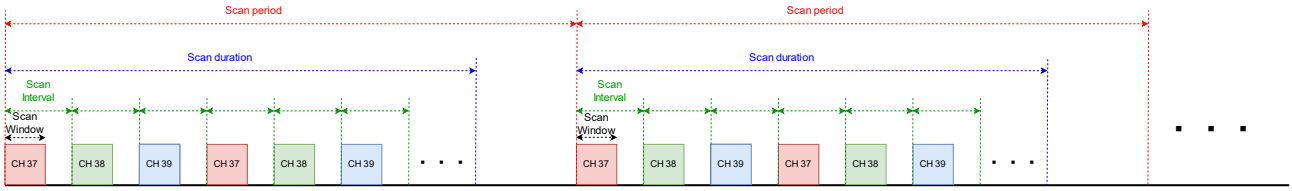


図 6-1　スキャンインターバル、ウインドウ、duration、period の関係

R_BLE_ABS_StartScan のパラメータである、"fast_xxx", "slow_xxx "はスキャンの頻度を途中で変更する場合に設定します。fast で Advertising 検出確率を上げ、slow で頻度を下げるユースケースで使用します。図 6-2 に fast と slow の関係、表 6.6 に fast, slow scan のイベントを示します。

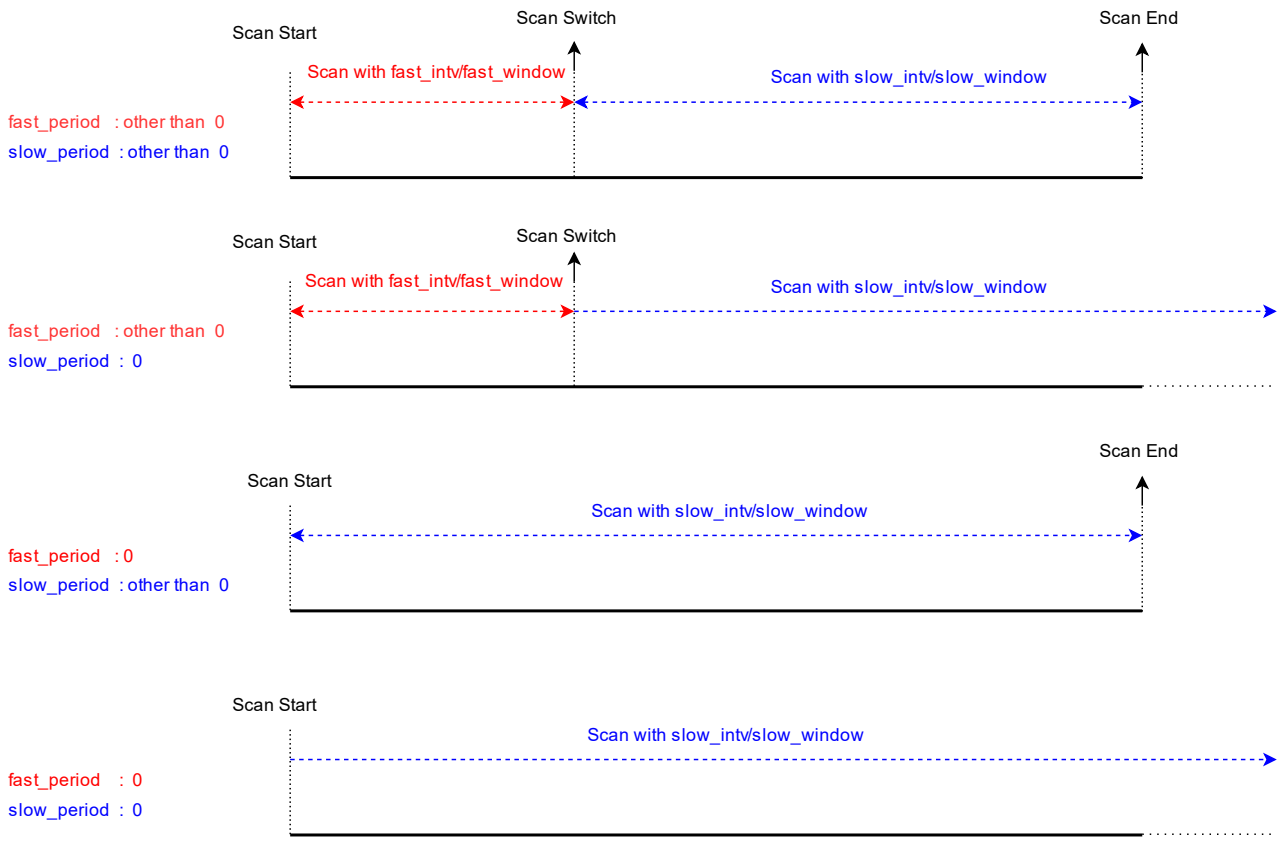


図 6-2　fast, slow パラメータの関係

表 6.6　fast, slow のイベントのタイプ

ライブラリ	Scan Start	Scan Switch	Scan End
All features	BLE_GAP_EVENT_SCAN_ON	BLE_GAP_EVENT_SCAN_TO BLE_GAP_EVENT_SCAN_ON	BLE_GAP_EVENT_SCAN_TO
Balance	BLE_GAP_EVENT_SCAN_ON	BLE_GAP_EVENT_SCAN_OFF BLE_GAP_EVENT_SCAN_ON	BLE_GAP_EVENT_SCAN_OFF

6.2.1　プライバシ

プライバシ機能により、スキャンリクエストに含まれるアドレスを RPA にすることが可能です。「9.4.1 ローカルデバイスの RPA 生成」に従って、事前準備を行います。R_BLE_GAP_StartScan にて RPA を使う場合は、プライバシを有効にするために st_ble_gap_scan_param_t 構造体(表 6.1)の表 6.7 のフィールドを設定します。

表 6.7　プライバシ使用時に設定するパラメータ(R_BLE_GAP_StartScan)

フィールド	フィールドに設定する値	説明
o_addr_type	BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)	R_BLE_GAP_SetLocIdInfo()で登録した Identity Address が Public Address の場合に指定します。
	BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)	R_BLE_GAP_SetLocIdInfo()で登録した Identity Address が Static Address の場合に指定します。

R_BLE_ABS_StartScan にて RPA を使う場合は、プライバシを有効にするために st_ble_abs_scan_param_t 構造体(表 6.4)の表 6.8 のフィールドを設定します。このフィールドの詳細な定義については API ドキュメントをご参照ください。

表 6.8　プライバシ使用時に設定するパラメータ(R_BLE_ABS_StartScan)

フィールド	フィールドに設定する値	説明
dev_filter	BLE_ABS_SCAN_ALL_RPA_PUBLIC (BLE_GAP_SCAN_ALLOW_ADV_ALL | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))	R_BLE_GAP_SetLocIdInfo()で登録した Identity Address が Public Address の場合に指定します。
	BLE_ABS_SCAN_WLST_RPA_PUBLIC (BLE_GAP_SCAN_ALLOW_ADV_WLST | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))	
	BLE_ABS_SCAN_EXC_DIR_RPA_PUBLIC (BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))	
	BLE_ABS_SCAN_EXC_DIR_WLST_RPA_PUBLIC (BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))	
	BLE_ABS_SCAN_ALL_RPA_STATIC (BLE_GAP_SCAN_ALLOW_ADV_ALL | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))	R_BLE_GAP_SetLocIdInfo()で登録した Identity Address が Static Address の場合に指定します。
	BLE_ABS_SCAN_WLST_RPA_STATIC (BLE_GAP_SCAN_ALLOW_ADV_WLST | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))	
	BLE_ABS_SCAN_EXC_DIR_RPA_STATIC (BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))	
	BLE_ABS_SCAN_EXC_DIR_WLST_RPA_STATIC (BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))	

なお、ローカルデバイスが RPA 生成・解決機能が有効になっている状態において、Non-resolvable private address を使用するリモートデバイスについてはアクティブスキャンが行えません。

【BP】 プライバシをサポートする場合、デバイスを追跡されるリスクを回避するためにスキャナブルアドバタイズメントやアクティブスキャンの使用を制限することが推奨されています。

6.3　スキャンで受信する情報

スキャン開始 API をコールすると、Bluetooth LE Protocol Stack は周辺デバイスからの Advertising Data の受信を BLE_GAP_EVENT_ADV_REPT_IND イベントにて通知します。送信側が AUX_CHAIN_IND を使用する場合、Advertising Data は分割して通知されます。さらに、Bluetooth の仕様に従って上位層へ通知できる Advertising Data のサイズは 229 バイト以下のため、230 バイト以上の Advertising Data は分割して通知されます。必要に応じて受信側で結合してください。

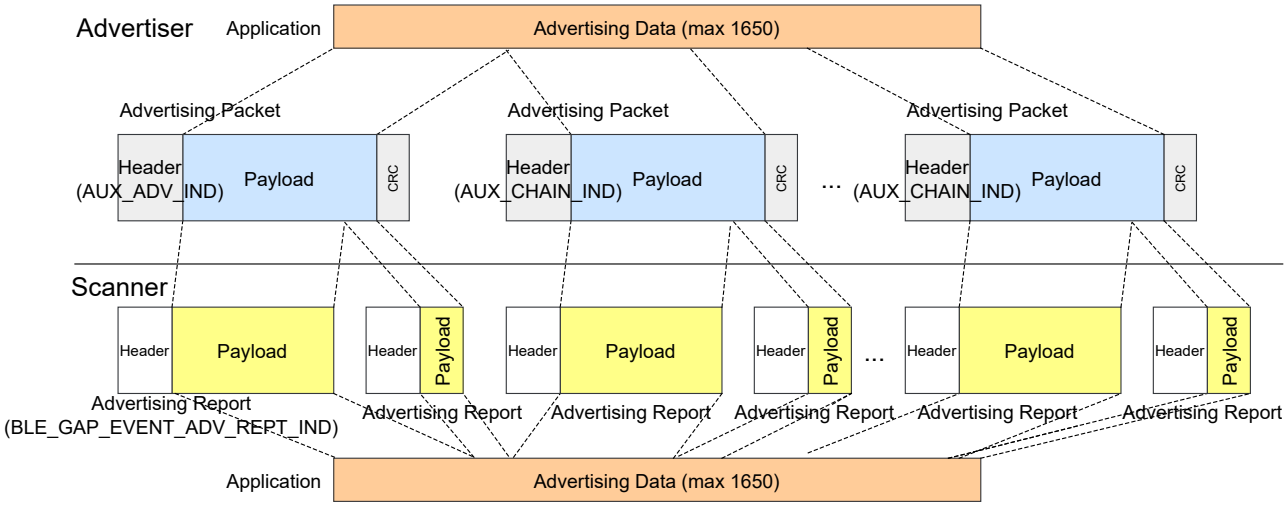


図 6-3　Advertising Data の分割と結合

受信した Advertising は st_ble_gap_adv_rept_evt_t 型に格納されます。表 6.9 に st_ble_gap_adv_rept_evt_t 構造体を示します。

表 6.9　st_ble_gap_adv_rept_evt_t 構造体

フィールドの型	フィールド	説明
uint8_t union {	adv_rpt_type	Advertising のタイプを示します。
st_ble_gap_adv_rept_t *	p_adv_rpt	Balance ライブラリ を使っている場合、このフィールドで Advertising を通知します。
st_ble_gap_ext_adv_rept_t *	p_ext_adv_rpt	All features ライブラリを使っている場合、このフィールドで Advertising を通知します。
st_ble_gap_perd_adv_rept_t *	p_per_adv_rpt	Periodic advertising を通知します。 All features ライブラリのみ使用可能です。
} param;		

使用している Bluetooth LE Protocol Stack ライブラリの種類によって、Advertising を示すフィールドが異なります。各フィールドの型について、表 6.10、表 6.11 に示します。

表 6.10　st_ble_gap_adv_rept_t 構造体

フィールドの型	フィールド	説明
uint8_t	num	受信した Advertising の数。常に 1 となります。
uint8_t	adv_type	Advertising パケットのタイプ
uint8_t	addr_type	受信した Advertising パケットに含まれるアドレスのタイプ
uint8_t *	p_addr	受信した Advertising パケットに含まれるアドレス
uint8_t	len	受信した Advertising データのサイズ
int8_t	rssi	受信した Advertising の RSSI
uint8_t *	p_data	受信した Advertising データ

表 6.11　st_ble_gap_ext_adv_rept_t 構造体

フィールドの型	フィールド	説明
uint8_t	num	受信した Advertising の数。常に 1 となります。
uint8_t	adv_type	Advertising パケットのタイプ
uint8_t	addr_type	受信した Advertising パケットに含まれるアドレスのタイプ
uint8_t *	p_addr	受信した Advertising パケットに含まれるアドレス
uint8_t	adv_phy	Advertising の primary PHY
uint8_t	sec_adv_phy	Advertising の secondary PHY
uint8_t	adv_sid	Advertising SID
int8_t	tx_pwr	Tx power
int8_t	rssi	受信した Advertising の RSSI
uint16_t	perd_adv_intv	Periodic advertising interval
uint8_t	dir_addr_type	Direct Advertising に含まれているアドレスタイプ
uint8_t *	p_dir_addr	Direct Advertising に含まれているアドレス
uint8_t	len	受信した Advertising データのサイズ
uint8_t *	p_data	受信した Advertising データ

各構造体の詳細については API ドキュメントをご参照ください。

コールバック関数で受信した Advertising から RSSI を表示する例を以下に示します。

```
/* GAP callback function */
void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch (type)
    {
        /* 省略 */
        case BLE_GAP_EVENT_ADV_REPT_IND:
        {
            st_ble_gap_adv_rept_evt_t *adv_rept_evt_param =
                (st_ble_gap_adv_rept_evt_t *)data->p_param;

            switch (adv_rept_evt_param->adv_rpt_type)
            {
                /* receive legacy advertising PDU */
                case 0x00:
                {
                    st_ble_gap_adv_rept_t *adv_rept_param =
                        (st_ble_gap_adv_rept_t *)adv_rept_evt_param->param.p_adv_rpt;

                    printf("RSSI : %d ¥n", adv_rept_param->rssi);
                } break;

                /* receive extended advertising PDU */
                case 0x01:
                {
                    st_ble_gap_ext_adv_rept_t *ext_adv_rept_param =
                        (st_ble_gap_ext_adv_rept_t *)ext_adv_rept_param->
                                                    param.p_ext_adv_rpt;

                    printf("RSSI : %d ¥n", ext_adv_rept_param->rssi);
                } break;
                /* 省略 */
```

コード 6-1　受信した Advertising から RSSI を表示する例

6.4　スキャンのフィルタリング

スキャンで受信する Advertising はフィルタリングすることが可能です。アプリケーションへ必要な Advertising のみ通知したい場合に使用します。
API で実施可能なフィルタリングは以下の通りです。

- 既知のデバイスから受信する
- 重複する Advertising を受信しない
- Discoverable Mode によるフィルタリング
- Advertising Data によるフィルタリング(抽象 API のみ)

各フィルタリングについて以降に説明します。

6.4.1　ホワイトリストの使用(既知のデバイスから受信する)

受信したい Advertising に含まれる BD アドレスが判明している場合、この方法によりフィルタリングを行います。スキャン開始前に下記 1, 2 を行います。

1. ホワイトリストに既知のデバイスの BD アドレスを登録
   R_BLE_GAP_ConfWhiteList により、既知のデバイスを登録します。

【注】ホワイトリストフィルタを有効にした動作(アドバタイズ、スキャン、コネクションリクエスト)を実行している場合はホワイトリストの追加・削除は行えません。

2. スキャンパラメータの設定
   スキャン開始 API の以下の Scan Filter Policy パラメータをホワイトリストに登録したデバイスから受信する設定 BLE_GAP_SCAN_ALLOW_ADV_WLST(0x01)にします。
   R_BLE_GAP_StartScan st_ble_gap_scan_param_t 構造体の filter_policy フィールド
   R_BLE_ABS_StartScan st_ble_abs_scan_param_t 構造体の dev_filter フィールド

6.4.2　重複する Advertising を受信しない

同じデバイスから重複して Advertising を受信したくない場合は、duplicate filtering を行います。
スキャン開始 API の Duplicate Filter パラメータ
R_BLE_GAP_StartScan st_ble_gap_scan_on_t 構造体の filter_dups フィールド
R_BLE_ABS_StartScan st_ble_abs_scan_param_t 構造体の filter_dups フィールド
を duplicate filtering を有効にする設定
BLE_GAP_SCAN_FILT_DUPLIC_ENABLE(0x01)
にします。

duplicate filtering でフィルタリング可能なデバイスの台数は 8 台です。9 台以上 Advertising するデバイスが周辺にいる場合、9 台目以降のデバイスは duplicate filtering されることなく受信します。

6.4.3　Discoverable Mode によるフィルタリング

受信した Advertising データに含まれる AD_TYPE が Flag の値を見て、Discoverable Mode によるフィルタリングが可能です。抽象 API は本機能をサポートしていません。
R_BLE_GAP_StartScan st_ble_gap_scan_on_t 構造体の proc_type フィールド
に表 6.12 の設定値を指定します。

表 6.12　Discoverable Mode によるフィルタリングで設定する値

パラメータ設定値	説明
BLE_GAP_SC_PROC_OBS(0x00)	Discoverable Mode に関係なく受信
BLE_GAP_SC_PROC_LIM(0x01)	LE Limited Discoverable Mode を受信
BLE_GAP_SC_PROC_GEN(0x02)	LE General Discoverable Mode を受信

6.4.4　Advertising Data によるフィルタリング

抽象 API は Advertising data に含まれている内容でフィルタリングが可能です。
st_ble_abs_scan_param_t 構造体の以下のパラメータにフィルタリングの情報を指定します。

p_filter_data : フィルタリングするデータ
filter_data_length : フィルタリングデータのサイズ
filter_ad_type : フィルタリングするデータの AD_TYPE

```
/* Scan filter data */
static uint8_t gs_filter_data[] =
{
    /* Complete Local Name */
    9,          /**< Data Size */
    0x09,        /**< Data Type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /**< Data Value */
};

/* Scan parameters */
static st_ble_abs_scan_param_t gs_scan_param =
{
    .p_phy_param_1M         = &gs_scan_phy_param,
    .p_filter_data          = gs_filter_data,
    .slow_period            = 0,
    .filter_data_length     = ARRAY_SIZE(gs_filter_data),
    .dev_filter             = BLE_GAP_SCAN_ALLOW_ADV_ALL,
    .filter_dups            = BLE_GAP_SCAN_FILT_DUPLIC_ENABLE,
};
```

コード 6-2　Advertising によるフィルタリングの設定例

6.5　Periodic Advertising Synchronization

スキャナは図 5-5 の AUX_ADV_IND に含まれる情報から Periodic Advertising によりアドバタイザと同期することが可能です。アプリケーション上でスキャナが Periodic Advertising Synchronization (Sync)を確立する手順を図 6-4 に示します。→Periodic Advertising Sync 特有の各ステップの詳細について以降の章で説明します。

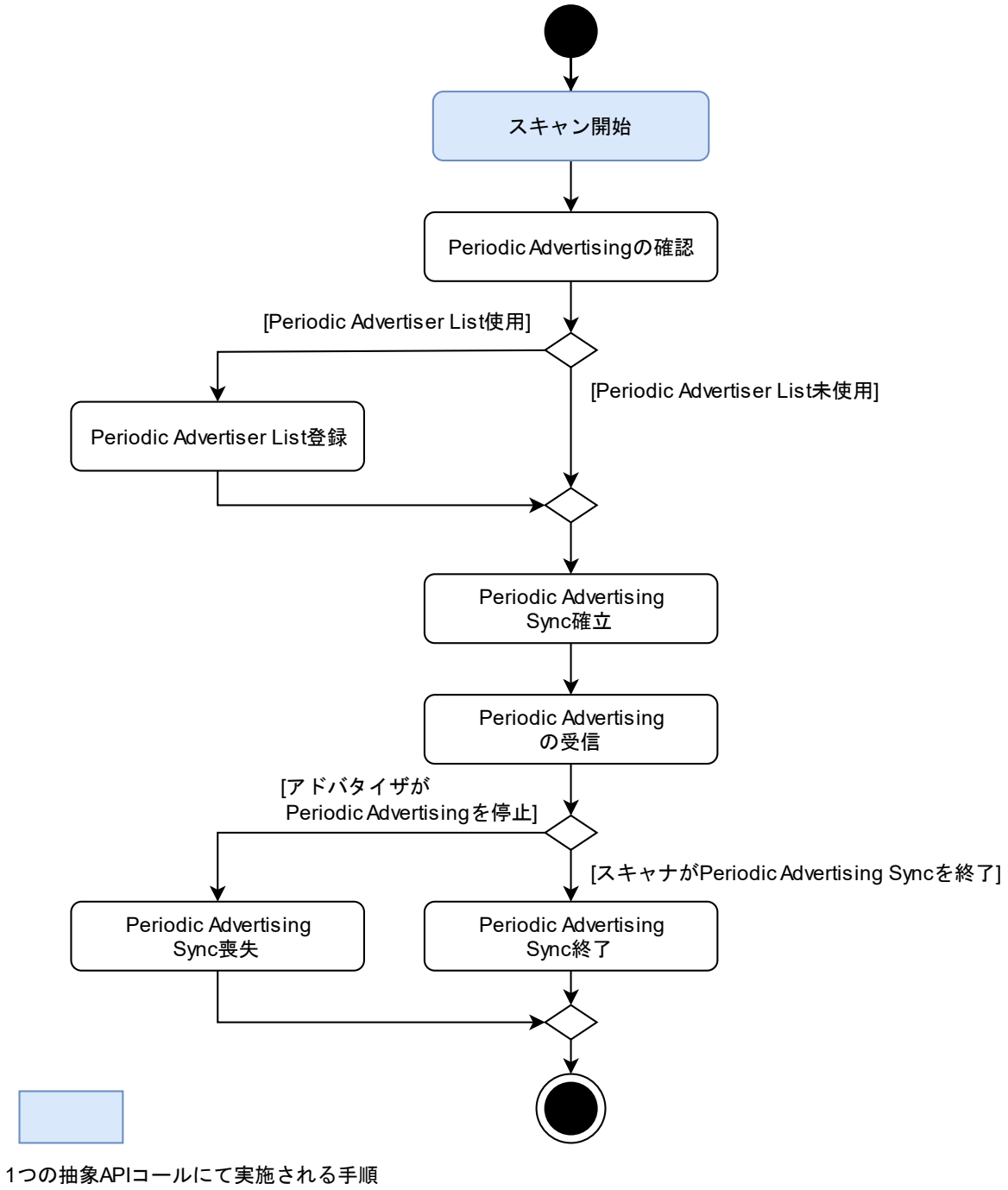


図 6-4　Periodic Advertising Sync の手順

6.5.1　スキャン開始

「6.1 スキャンの開始／停止」の手順に従って、スキャンを開始します。

6.5.2　Periodic Advertising の確認

受信した Advertising に含まれる、表 6.11 の perd_adv_intv が 0 でない場合、送信元のアドバタイザと Periodic Advertising Sync を確立することが可能です。「6.5.3 Periodic Advertiser List 登録」あるいは、「6.5.4 Periodic Advertising Sync 確立」にて、表 6.11 の addr_type、p_addr、adv_sid フィールドを使って、アドバタイザを指定します。

6.5.3　Periodic Advertiser List 登録

Periodic Advertiser Sync 確立時にアドバタイザを指定するときに Periodic Advertiser List を使うかアドレスを指定するか選択します。既知のアドバタイザの BD アドレスを、R_BLE_GAP_ConfPerdAdvList をコールして Periodic Advertiser List に登録することが可能です。

6.5.4　Periodic Advertising Sync 確立

Periodic Advertising Sync を確立するには、R_BLE_GAP_CreateSync をコールします。Periodic Advertising Sync が確立されると、BLE_GAP_EVENT_SYNC_EST イベントを通知します。R_BLE_GAP_CreateSync コール後、Periodic Advertising Sync 確立をキャンセルする場合、R_BLE_GAP_CancelCreateSync をコールします。キャンセルが完了すると、結果が BLE_ERR_NOT_YET_READY(0x0012)である、BLE_GAP_EVENT_SYNC_EST イベントが通知されます。

BLE_CFG_RF_SYNC_SET_MAX で設定した数の Periodic Advertising Sync を確立することが可能です。

スキャン開始から Periodic Advertising Sync 確立までのサンプルを以下に示します。

```
/* 省略 */

static st_ble_dev_addr_t gs_sync_advr;
static uint8_t gs_adv_sid;

static st_ble_abs_scan_phy_param_t gs_phy_param_1M =
{
    .fast_intv                = 0x0200,
    .slow_intv                = 0x0800,
    .fast_window              = 0x0100,
    .slow_window              = 0x0100,
    .scan_type                = BLE_GAP_SCAN_PASSIVE,
};

static st_ble_abs_scan_param_t gs_scan_param =
{
    .p_phy_param_1M           = &gs_phy_param_1M,
    .p_phy_param_coded        = NULL,
    .p_filter_data            = NULL,
    .fast_period              = 0x0100,
    .slow_period              = 0x0000,
    .filter_data_length       = 0,
    .dev_filter               = BLE_GAP_SCAN_ALLOW_ADV_ALL,
    .filter_dups              = BLE_GAP_SCAN_FILT_DUPLIC_DISABLE,
};


static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{

    /* 省略 */
    switch(type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            R_BLE_ABS_StartScan(&gs_scan_param);
```

```
        } break;

        case BLE_GAP_EVENT_ADV_REPT_IND:
        {
            st_ble_gap_adv_rept_evt_t * p_adv_rept_evt_param =
                (st_ble_gap_adv_rept_evt_t *)p_data->p_param;

            switch (p_adv_rept_evt_param->adv_rpt_type)
            {
                case 0x01:
                {
                    st_ble_gap_ext_adv_rept_t * p_ext_adv_rept_param =
                        (st_ble_gap_ext_adv_rept_t *)p_adv_rept_evt_param->param.p_ext_adv_rpt;

                    if(0x0000 != p_ext_adv_rept_param->perd_adv_intv)
                    {
                        /* found */
                        memcpy(gs_sync_advr.addr, p_ext_adv_rept_param->p_addr, BLE_BD_ADDR_LEN);
                        gs_sync_advr.type = p_ext_adv_rept_param->addr_type;
                        gs_adv_sid = p_ext_adv_rept_param->adv_sid;
                        R_BLE_GAP_ConfPerdAdvList(BLE_GAP_LIST_ADD_DEV,
                                                  &gs_sync_advr,
                                                  &gs_adv_sid,
                                                  1);
                    }

                } break;
                /* 省略 */
            }
        } break;

        case BLE_GAP_EVENT_PERD_LIST_CONF_COMP:
        {
            R_BLE_GAP_CreateSync(NULL, 0, 100, 100);
        } break;

        case BLE_GAP_EVENT_SYNC_EST:
        {
            if(BLE_SUCCESS == result)
            {
                R_BLE_CLI_Printf("sync established.¥n");
            }
        } break;

        /* 省略 */
    }
}

/* 省略 */
```

コード 6-3　Periodic Advertising Sync 確立までのサンプル

6.5.5　Periodic Advertising の受信

アドバタイザと Periodic Advertising Sync 確立後、BLE_GAP_EVENT_ADV_REPT_IND イベントにて Periodic Advertising 受信を通知します。受信した Periodic Advertising は表 6.9 の st_ble_gap_adv_rept_evt_t 型に格納されます。Periodic Advertising の場合の st_ble_gap_perd_adv_rept_t 型について、表 6.13 に示します。

表 6.13　st_ble_gap_perd_adv_rept_t 構造体

フィールドの型	フィールド	説明
uint16_t	sync_hdl	確立した Periodic Advertising Sync を識別する sync handle
int8_t	tx_pwr	tx power
int8_t	rssi	RSSI
uint8_t	rfu	Reserved for future use
uint8_t	data_status	Periodic Advertising Data の状態
uint8_t	len	Periodic Advertising Data のサイズ
uint8_t *	p_data	Periodic Advertising Data

6.5.6　Periodic Advertising Sync 喪失

アドバタイザが Periodic Advertising を停止した場合、BLE_GAP_EVENT_SYNC_LOST イベントにより、Periodic Advertising Sync の喪失が通知されます。

6.5.7　Periodic Advertising Sync 終了

スキャナが Periodic Advertising Sync を終了する場合、BLE_GAP_TerminateSync をコールします。Periodic Advertising Sync が終了すると、BLE_GAP_EVENT_SYNC_TERM イベントで通知されます。

7. コネクション

7.1 セントラルのコネクション手順

セントラルは以下の API をコールして、Connectable Advertising を実行しているペリフェラルに対してコネクションリクエストを送信します。コネクションリクエストを受信したペリフェラルは Connectable Advertising を終了してコネクションリクエストに応答します。

コネクションリクエスト API :
- R_BLE_GAP_CreateConn
- R_BLE_ABS_CreateConn

これらの API で指定するパラメータについては
API ドキュメント
R_BLE_GAP_CreateConn :
st_ble_gap_create_conn_param_t

R_BLE_ABS_CreateConn :
st_ble_abs_conn_param_t

をご参照ください。

Advertising のプライマリチャネル(CH:37/38/39)が 1M PHY のリモートデバイスとコネクションするには、下記の設定が必要です。
R_BLE_GAP_CreateConn st_ble_gap_create_conn_param_t 構造体の p_conn_param_1M フィールド
R_BLE_ABS_CreateConn st_ble_abs_conn_param_t 構造体の p_conn_1M フィールド

Advertising のプライマリチャネルが Coded PHY のリモートデバイスとコネクションするには、下記の設定が必要です。
R_BLE_GAP_CreateConn st_ble_gap_create_conn_param_t 構造体の p_conn_param_coded フィールド
R_BLE_ABS_CreateConn st_ble_abs_conn_param_t 構造体の p_conn_coded フィールド

コネクション確立後の PHY は Advertising のセカンダリチャネル(CH:37/38/39 以外)で指定された PHY になります。

コネクションが確立すると BLE_GAP_EVENT_CONN_IND イベントにてコネクションハンドルが通知されます。コネクションハンドルは下位 3bit が 0 から BLE_CFG_RF_CONN_MAX-1 の範囲で他のコネクションと重複しない番号が割り当てられるため、下記サンプルのようにマスクしてコネクションハンドルの管理配列の index として使用することが可能です。

```
#define BLE_APP_CONN_HDL_MASK              (0x0007)
uint16_t g_conn_hdl[BLE_CFG_RF_CONN_MAX];
void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
/* 省略 */
        case BLE_GAP_EVENT_CONN_IND:
        {
            if (BLE_SUCCESS == result)
            {
                /* Store connection handle */
                st_ble_gap_conn_evt_t *p_gap_conn_evt_param = (st_ble_gap_conn_evt_t *)p_data->p_param;
                uint16_t index = p_gap_conn_evt_param->conn_hdl & BLE_APP_CONN_HDL_MASK;
                g_conn_hdl[index] = p_gap_conn_evt_param->conn_hdl;
```

7.1.1　ホワイトリストの使用(既知のデバイスにコネクションする)

送信先デバイスをホワイトリストに登録してからコネクションリクエストを送信することが可能です。既知のコネクション済みのデバイスに対して、再コネクションを行う場合にホワイトリストを使用します。手順は以下の通りです。

1. ホワイトリストに再コネクションするリモートデバイスの BD アドレスを登録
   R_BLE_GAP_ConfWhiteList により、既知のデバイスを登録します。

【注】ホワイトリストフィルタを有効にした動作(アドバタイズ、スキャン、コネクションリクエスト)を実行している場合はホワイトリストの追加・削除は行えません。

2. コネクションパラメータの設定
   コネクションリクエスト API の Filter Policy パラメータ
   R_BLE_GAP_CreateConn st_ble_gap_create_conn_param_t 構造体の init_filter_policy フィールド
   R_BLE_ABS_CreateConn st_ble_abs_conn_param_t 構造体の filter フィールド
   をホワイトリストに登録したデバイスにコネクションリクエストを送信する設定
   BLE_GAP_INIT_FILT_USE_WLST(0x01)にします。

以下にホワイトリストでコネクションする場合のサンプルを示します。

```
/* remote device address */
dev.addr = {"リモートデバイスのアドレス" };
dev.type = BLE_GAP_ADDR_PUBLIC;

/* register remote device to white list */
R_BLE_GAP_ConfWhiteList(BLE_GAP_LIST_ADD_DEV, &dev, 1);

/* 省略  */

/* reconnect */
st_ble_gap_conn_param_t conn_1M = {
    .conn_intv_min  = 0x0100,
    .conn_intv_max  = 0x0100,
    .conn_latency   = 0x0000,
    .sup_to        = 0x03BB,
    .min_ce_length  = 0xFFFF,
    .max_ce_length  = 0xFFFF,
};

st_ble_gap_create_conn_param_t conn_param;
conn_param.init_filter_policy = BLE_GAP_INIT_FILT_USE_WLST;
conn_param.own_addr_type = BLE_GAP_ADDR_PUBLIC;

/*  set connection parameters for 1M */
st_ble_gap_conn_phy_param_t conn_phy_1M = {
    .scan_intv   = 0x0300,
    .scan_window = 0x0300,
    p_conn_param = &conn_1M,
};

conn_param.p_conn_param_1M = &conn_phy_1M;

R_BLE_GAP_CreateConn(&conn_param);

/* 省略  */
```

コード 7-1　ホワイトリストを使用したコネクションリクエストの例

7.1.2　プライバシ

プライバシ機能により、コネクションリクエストに含まれるアドレスを RPA にすることが可能です。「9.4.1 ローカルデバイスの RPA 生成」に従って、事前準備を行います。R_BLE_GAP_CreateConn にて RPA を使う場合は、プライバシを有効にするために st_ble_gap_create_conn_param_t 構造体の表 7.1 のフィールドを設定します。

表 7.1　プライバシ使用時に設定するパラメータ(R_BLE_GAP_CreateConn)

フィールド	フィールドに設定する値	説明
own_addr_type	BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)	R_BLE_GAP_SetLocIdInfo()で登録した Identity Address が Public Address の場合に指定します。
	BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)	R_BLE_GAP_SetLocIdInfo()で登録した Identity Address が Static Address の場合に指定します。
remote_bd_addr_type remote_bd_addr	R_BLE_GAP_ConfRslvList()で登録した リモートデバイスの Identity Address を指定します。	—

R_BLE_ABS_CreateConn にて RPA を使う場合は、プライバシを有効にするために st_ble_abs_conn_param_t 構造体の表 7.2 のフィールドを設定します。これらのフィールドの詳細な定義については API ドキュメントをご参照ください。

表 7.2　プライバシ使用時に設定するパラメータ(R_BLE_ABS_CreateConn)

フィールド	フィールドに設定する値	説明
filter	BLE_ABS_CONN_USE_ADDR_RPA_PUBLIC (BLE_GAP_INIT_FILT_USE_ADDR | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))	R_BLE_GAP_SetLocIdInfo()で登録した Identity Address が Public Address の場合に指定します。
	BLE_ABS_CONN_USE_WLST_RPA_PUBLIC (BLE_GAP_INIT_FILT_USE_WLST | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))	
	BLE_ABS_CONN_USE_ADDR_RPA_STATIC (BLE_GAP_INIT_FILT_USE_ADDR | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))	R_BLE_GAP_SetLocIdInfo()で登録した Identity Address が Static Address の場合に指定します。
	BLE_ABS_CONN_USE_WLST_RPA_STATIC (BLE_GAP_INIT_FILT_USE_WLST | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))	
remote_bd_addr_type remote_bd_addr	R_BLE_GAP_ConfRslvList()で登録した リモートデバイスの Identity Address を指定します。	—

7.2　コネクションキャンセル

コネクションリクエストはコネクションが確立されるか、または、キャンセルされるまで、次のコネクションリクエストを送信することができません。そのため、コネクションリクエスト送信後、別のデバイスに対してコネクションリクエストを送信したい場合、BLE_GAP_CancelCreateConn によりコネクションリクエストのキャンセルを行います。

キャンセル実施後、BLE_GAP_EVENT_CONN_IND イベントにて result が BLE_ERR_INVALID_HDL(0x000E)で通知されます。

7.3　複数台コネクション

本章では、同時に複数のデバイスとコネクションする方法とその際の注意点を説明します。Bluetooth LE Protocol Stack では、最大 7 台のデバイスと同時にコネクションできます。1 対 1 通信の場合と同様の手順で複数のデバイスとコネクションできます。アプリケーションはコネクション時に通知されるコネクションハンドルを使用してコネクションデバイスを指定します。コネクションハンドルはコネクションに対して割り振られるため、同一デバイスであっても再コネクション時に変化します。

GATT データベース内のキャラクタリスティックにアクセスするためのアトリビュートハンドルはデバイス毎に異なります。GATT クライアントとして複数のデバイスとコネクションする場合には、GATT サーバ毎にアトリビュートハンドルを保持する必要があります。app_lib の Profile Common 利用することで、コネクション順に最大 10 台までデバイス毎のアトリビュートハンドルを保持できます。

GATT サーバとして複数のデバイスからコネクションされる場合には、例えば、Client Configuration Characteristic Descriptor のように、デバイス毎に値を保持することが仕様で決められているものがあります。複数のクライアントからアクセスされる場合には、それぞれの値を保持するために GATT データベースのプロパティを設定します。

以降に、想定されるユースケース毎に複数台コネクションするアプリケーションコードの実装例を説明します。

7.3.1　複数のペリフェラルデバイスとコネクションする

自身をセントラルとして、複数のペリフェラルデバイスと通信します。例えば、複数のセンサデータを集計するアプリケーションが想定されます。ここでは、セントラルデバイスを GATT クライアントとします。

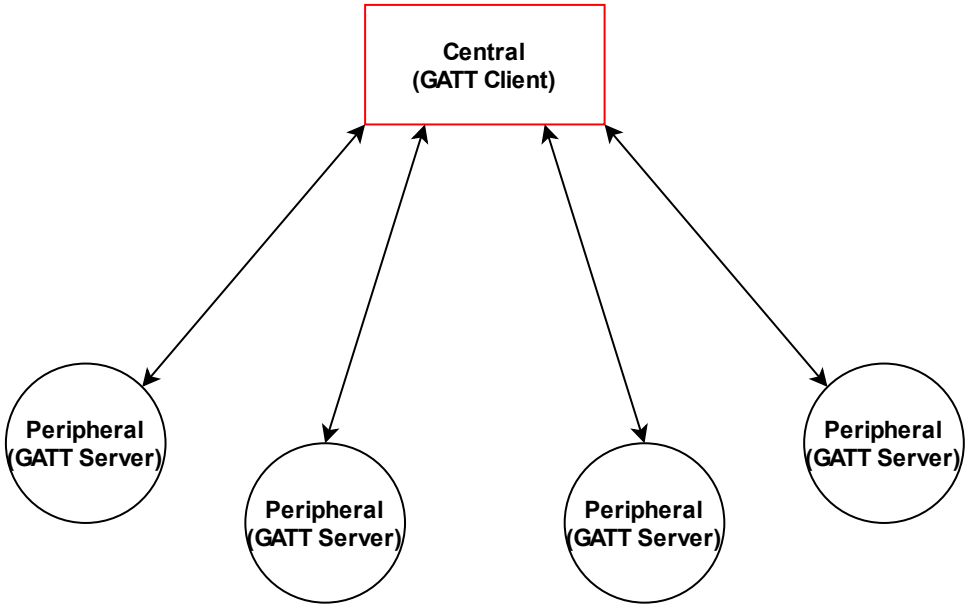


図 7-1　複数のペリフェラルデバイスとのコネクション

1 台ずつ確実にコネクションするためセントラルデバイスは、サービスディスカバリの完了を一区切りとして順にコネクションします。
以下に Bluetooth LE Protocol Stack の app_lib を利用してコネクションする場合のシーケンスチャートと実装例を示します。この手順を繰り返し、複数のペリフェラルデバイスとコネクションします。

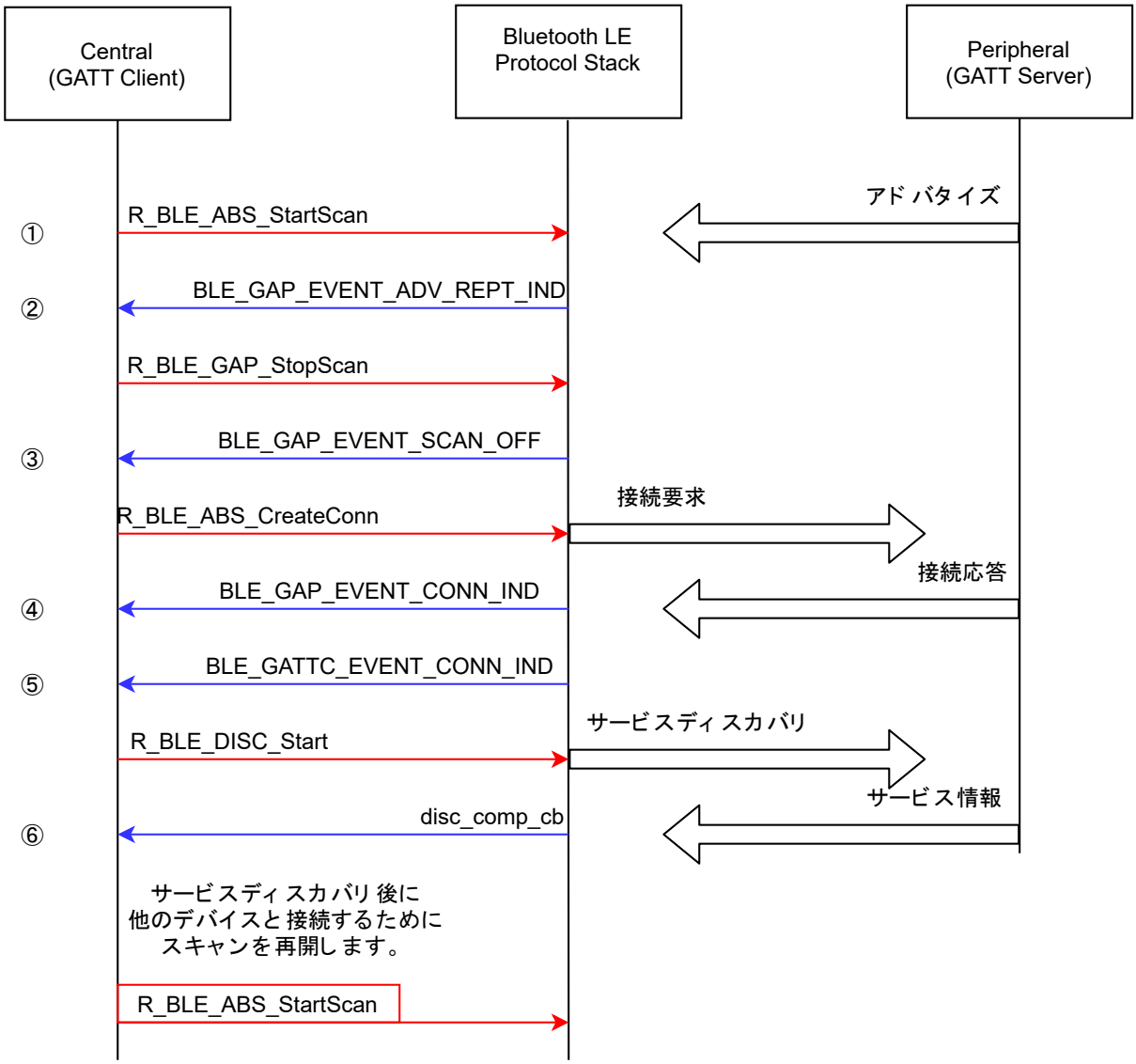


図 7-2　ペリフェラルデバイスへのコネクション時のシーケンスチャート(チャート中の丸数字は下記コード 7-3 内の番号に対応します。)

```
/* Scan phy parameters */
static st_ble_abs_scan_phy_param_t gs_scan_phy_param =
{
    /* TODO: Modify scan phy parameter. */
    .fast_intv   = 0x200,
    .fast_window = 0x100,
    .slow_intv   = 0x200,
    .slow_window = 0x100,
    .scan_type   = BLE_GAP_SCAN_PASSIVE,
};

/* Scan filter data */
static uint8_t gs_filter_data[] =
{
    /* TODO: Modify filter of advertise data. Value of Data Flag is defined in
https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile */

    /* Complete Local Name */
    9,         /**< Data Size */
    0x09,       /**< Data Type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /**< Data Value */
};

/* Scan parameters */
static st_ble_abs_scan_param_t gs_scan_param =
{
    /* TODO: Modify scan parameter. */
    .p_phy_param_1M     = &gs_scan_phy_param,
    .p_filter_data      = gs_filter_data,
    .slow_period        = 0,
    .filter_data_length = ARRAY_SIZE(gs_filter_data),
    .dev_filter         = BLE_GAP_SCAN_ALLOW_ADV_ALL,
    .filter_dups        = BLE_GAP_SCAN_FILT_DUPLIC_ENABLE,
};

/* Connection phy parameters */
static st_ble_abs_conn_phy_param_t gs_conn_phy_param =
{
    /* TODO: Modify connection phy parameter. */
    .conn_intv    = 0x0130,
    .conn_latency = 0x0000,
    .sup_to       = 0x03BB,
};

/*  Connection device address */
static st_ble_dev_addr_t gs_conn_bd_addr;

/* Connection parameters */
static st_ble_abs_conn_param_t gs_conn_param =
{
    .p_conn_1M = &gs_conn_phy_param,
    .p_addr    = &gs_conn_bd_addr,     /**< Set BD address of connecting device. */
    .filter    = BLE_GAP_INIT_FILT_USE_ADDR,
    .conn_to   = 5,
};
```

コード 7-2　スキャンパラメータとコネクションパラメータの初期値の設定

```
/* Connection handle */
uint16_t g_conn_hdl[BLE_CFG_RF_CONN_MAX];
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch (type)
    {
        case BLE_GAP_EVENT_STACK_ON: /* (1) */
        {
            R_BLE_ABS_StartScan(&gs_scan_param);
        } break;

        case BLE_GAP_EVENT_CONN_IND: /* (4) */
        {

            if (BLE_SUCCESS == result)
            {
                st_ble_gap_conn_evt_t *p_gap_conn_evt_param =
                    (st_ble_gap_conn_evt_t *)p_data->p_param;

                for(uint8_t i=0;i<BLE_CFG_RF_CONN_MAX;i++)
                {
                  if(g_conn_hdl[i] == BLE_GAP_INVALID_CONN_HDL)
                  {
                          g_conn_hdl[i] = p_gap_conn_evt_param->conn_hdl;
                  }
                }

            }
        } break;

        case BLE_GAP_EVENT_DISCONN_IND:
        {
         st_ble_gap_disconn_evt_t *p_gap_disconn_evt_param =
                        (st_ble_gap_disconn_evt_t*)p_data->p_param;


         for(uint8_t i=0;i<BLE_CFG_RF_CONN_MAX;i++)
         {
                  if(g_conn_hdl[i] == p_gap_disconn_evt_param->conn_hdl)
                  {
                          g_conn_hdl[i] = BLE_GAP_INVALID_CONN_HDL;
                  }
         }
        } break;

        case BLE_GAP_EVENT_ADV_REPT_IND: /* (2) */
        {
            st_ble_gap_adv_rept_evt_t *p_adv_rept_param = (st_ble_gap_adv_rept_evt_t *)p_data->p_param;
            st_ble_gap_ext_adv_rept_t *p_ext_adv_rept_param = (st_ble_gap_ext_adv_rept_t *)p_adv_rept_param-
>param.p_ext_adv_rpt;
            gs_conn_param.p_addr->type = p_ext_adv_rept_param->addr_type;
            memcpy(gs_conn_param.p_addr->addr, p_ext_adv_rept_param->p_addr, BLE_BD_ADDR_LEN)

            R_BLE_GAP_StopScan();
        } break;

        case BLE_GAP_EVENT_SCAN_OFF: /* (3) */
        {
            R_BLE_ABS_CreateConn(&gs_conn_param);
        }
        default:
        {
            /* Do nothing. */
        } break;
    }
}
```

コード 7-3　複数台コネクションする場合の GAP コールバック関数の実装例

```
/* XXX Service UUID */
static uint8_t XXXC_UUID[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00 };

/* Service discovery parameters */
static st_ble_disc_entry_t gs_disc_entries[] = {

    {
        .p_uuid     = XXXC_UUID,
        .uuid_type  = BLE_GATT_128_BIT_UUID_FORMAT,
         .serv_cb    = R_BLE_XXXC_ServDiscCb,
    },
};

static void disc_comp_cb(uint16_t conn_hdl)
{
    /* TODO: Add function after discovery completed */
    BLE_ABS_StartScan(&gs_scan_param); /* (6) */
    return;
}

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)
    {
        /* TODO: Set callback events of GATTC. Check BLE API reference for events. */

        case BLE_GATTC_EVENT_CONN_IND: /* (5) */
        {
            R_BLE_DISC_Start(p_data->conn_hdl, gs_disc_entries, ARRAY_SIZE(gs_disc_entries), disc_comp_cb);
        } break;

        default:
        {
            /* Do nothing. */
        } break;
    }
}
```

コード 7-4　Profile Common Library を利用したサービスディスカバリの実装例

app_lib の Profile Common の Discovery に、QE for BLE が生成するサービス API(r_ble_xxxc.c)の R_BLE_XXXC_ServDiscCb を登録（コード 7-4 内太枠）すると、Profile Common を介してサービス API にデバイス毎のアトリビュートハンドルが保持されます。

サービス API を使用することで、アプリケーションはデバイス毎のアトリビュートハンドルを管理することなく、コネクションハンドルを利用して各デバイスの GATT データベースにアクセスできます。

7.3.2　複数のセントラルデバイスとコネクションする

自身をペリフェラルとして、複数のセントラルデバイスと通信します。例えば家電機器を、複数のスマートフォンからコントロールする場合が想定されます。ここでは、ペリフェラルデバイスを GATT サーバとします。

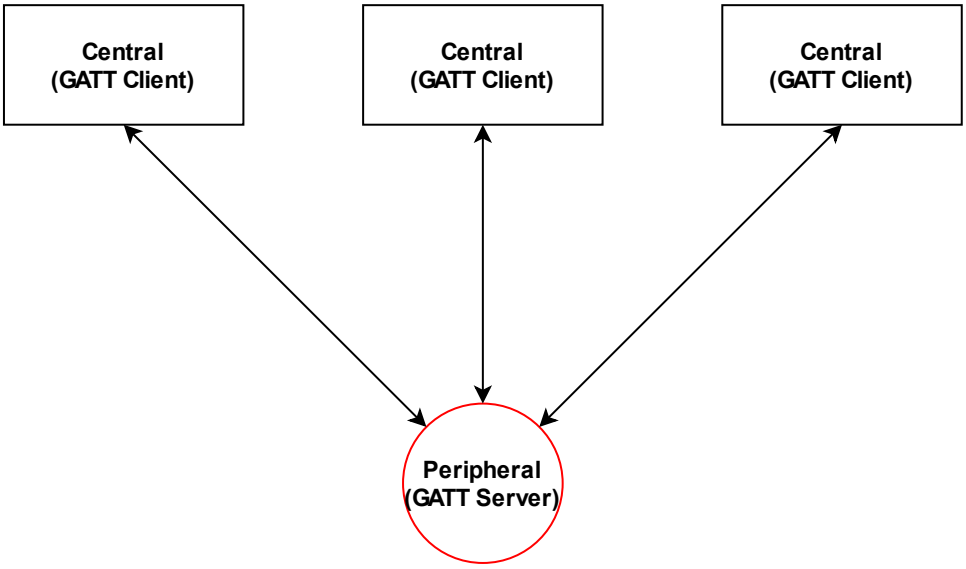


図 7-3　複数のセントラルデバイスとのコネクション

ペリフェラルとしてセントラルからコネクションされるとアドバタイズが停止します。コネクション後にアドバタイズを再開し別のデバイスからコネクションを受け入れます。
以下に Bluetooth LE Protocol Stack の app_lib を利用してコネクションする場合のシーケンスチャートと実装例を示します。この手順を繰り返し、複数のセントラルデバイスからのコネクションを受け付けます。

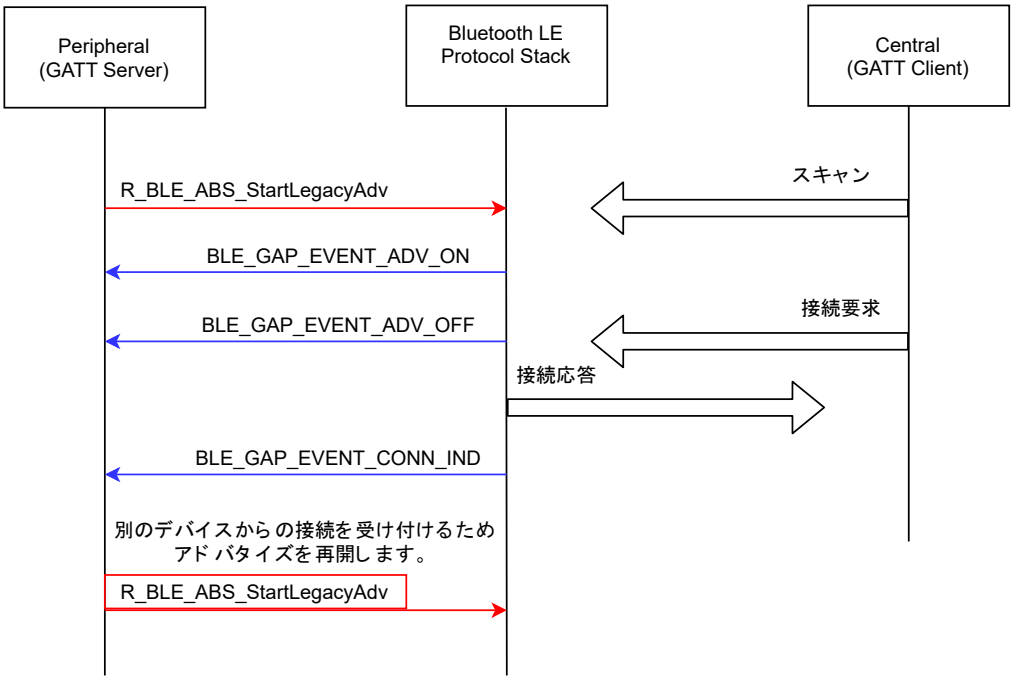


図 7-4　セントラルデバイスとコネクション時のシーケンスチャート

```
/* Advertising data */
static uint8_t gs_adv_data[] =
{
    /* TODO: Modify advertise data. Value of Data Flag is defined in
https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile */

    /* Flag (mandatory) */
    2,          /**< Data Size */
    0x01,       /**< Data Type: Flag */
    (BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE | BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED),    /**< Data Value */

    /* Complete Local Name */
    9,         /**< Data Size */
    0x09,       /**< Data Type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /**< Data Value */
};

/* Scan response Data */
static uint8_t gs_sres_data[] =
{
    /* TODO: Modify scan response data. Value of Data Flag is defined in
https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile */

    /* Complete Local Name */
    9,         /**< Data Size */
    0x09,       /**< Data Type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /**< Data Value */
};

/* Advertising parameters */
static st_ble_abs_legacy_adv_param_t gs_adv_param =
{
    /* TODO: Modify advertise parameters. */
    .slow_adv_intv   = 0x300,
    .slow_period     = 0,
    .p_adv_data      = gs_adv_data,
    .adv_data_length = ARRAY_SIZE(gs_adv_data),
    .p_sres_data     = gs_sres_data,
    .sres_data_length= ARRAY_SIZE(gs_sres_data),
    .adv_ch_map      = BLE_GAP_ADV_CH_ALL,
    .filter          = BLE_ABS_ADV_ALLOW_CONN_ANY,
    .o_addr_type     = BLE_GAP_ADDR_PUBLIC,
};
```

コード 7-5　アドバタイズパケットとパラメータの設定

```
uint16_t g_conn_hdl[BLE_CFG_RF_CONN_MAX];

static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch (type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
        } break;

        case BLE_GAP_EVENT_CONN_IND:
        {

            if (BLE_SUCCESS == result)
            {
                st_ble_gap_conn_evt_t *p_gap_conn_evt_param =
                    (st_ble_gap_conn_evt_t *)p_data->p_param;
                R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
                for(uint8_t i=0;i<BLE_CFG_RF_CONN_MAX;i++)
                {
                  if(g_conn_hdl[i] == BLE_GAP_INVALID_CONN_HDL)
                  {
                          g_conn_hdl[i] = p_gap_conn_evt_param->conn_hdl;
                  }

                }
            }
        } break;

        case BLE_GAP_EVENT_DISCONN_IND:
        {
         st_ble_gap_disconn_evt_t *p_gap_disconn_evt_param = (st_ble_gap_disconn_evt_t*)p_data->p_param;


         for(uint8_t i=0;i<BLE_CFG_RF_CONN_MAX;i++)
         {
                  if(g_conn_hdl[i] == p_gap_disconn_evt_param->conn_hdl)
                  {
                          g_conn_hdl[i] = BLE_GAP_INVALID_CONN_HDL;
                  }
         }
        } break;

        default:
        {
            /* Do nothing. */
        } break;
    }
}
```

コード 7-6　複数のセントラルデバイスからコネクションを受け入れる場合の、GAP コールバック関数の実装例

Bluetooth Low Energy では、通信のタイミングの制御はセントラル(セントラルデバイス)が行います。そのため、複数のセントラルデバイスとコネクションされている場合には、送信タイミングが意図せず衝突し早期に切断されることがあります。これを防ぐために、コネクションパラメータの更新を行いペリフェラルレイテンシやスーパービジョンタイムアウト時間に余裕を持たせることをお勧めします。コネクションパラメータの更新は、「8.3 コネクションパラメータの更新」をご参照ください。

GATT サーバは、コネクションされる全ての GATT クライアントに対して共通のキャラクタリスティックの値を公開する場合と、クライアント毎に異なる値を公開する場合があります。例えば、Client Configuration Characteristic Descriptor など、クライアント毎に異なる値を公開する場合は、QE for BLE のキャラクタリスティック画面で Aux Properties の Peer Specific をチェックします。これにより、Bluetooth LE Protocol Stack の GATT データベースに保持される値のテーブルとオプションが変更され、最大 7 つのクライアント毎に異なる値が保持されます。アクセスされたクライアント毎にデータベースの値が返されます。

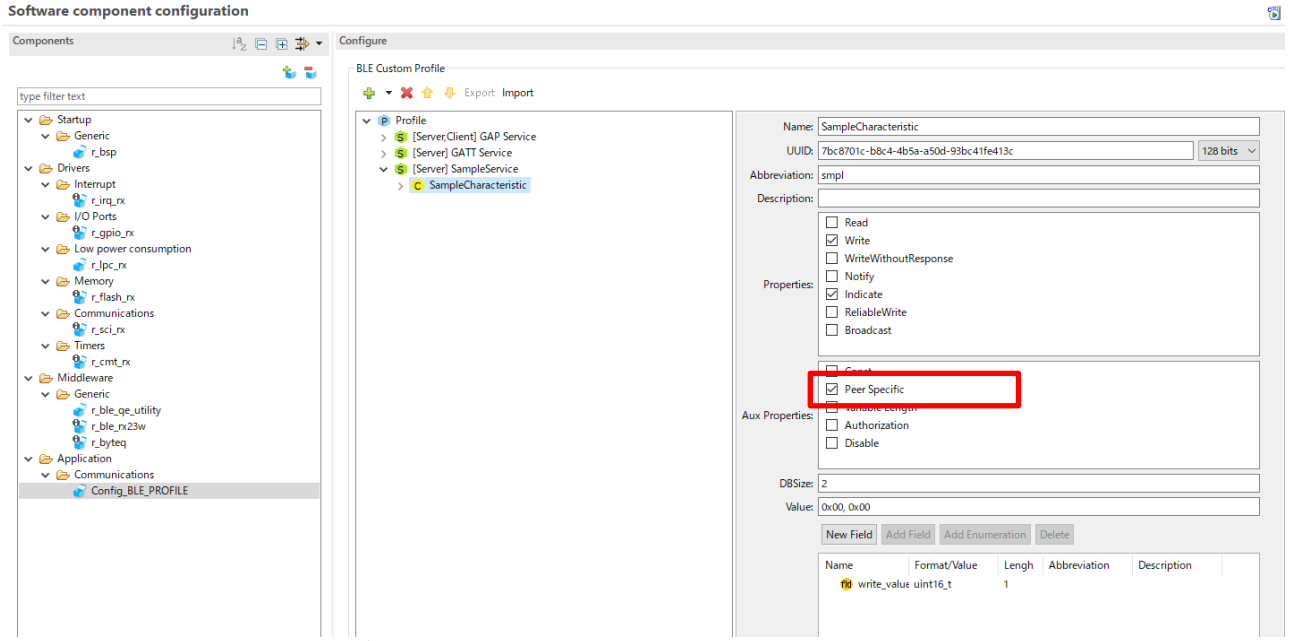


図 7-5　デバイス毎にキャラクタリスティックの値を保持する設定

7.3.3　マルチロール

Bluetooth Low Energy 通信では、同時にコネクションする複数のデバイスに対して異なる GAP ロールを実装できます。あるデバイスに対してはセントラル、別のデバイスに対してペリフェラルとして通信できます。ここでは、ローカルデバイスは、セントラルデバイスに対して GATT サーバ、ペリフェラルデバイスに対して GATT クライアントとして動作するとします。

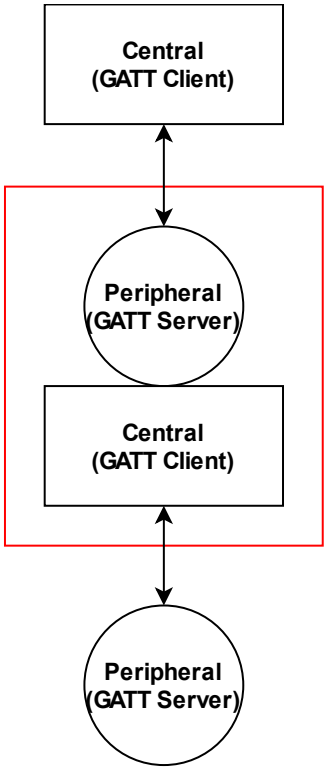


図 7-6　マルチロールのコネクション例

マルチロールコネクションでは、セントラルデバイス、ペリフェラルデバイスのどちらともコネクションするため、アドバタイズとスキャンの両方を行います。

マルチロールコネクションを行うアプリケーションでは、コネクションハンドルと GAP ロールを保持します。コネクションに対する自身の GAP ロールは、BLE_GAP_EVENT_CONN_IND イベントに通知されます。

以下に、セントラル、ペリフェラルとしてコネクションする場合の GAP コールバック関数の実装例を示します。GAP コールバック関数をロールごとに実装しています。

スキャンやアドバタイズの設定は上述のコード 7-2(スキャン)、コード 7-5(アドバタイズ)をご参照ください。

```
/* Connection handle */
uint16_t g_central_conn_hdl;

static void ble_central_gapcb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
        switch (type)
        {
                case BLE_GAP_EVENT_STACK_ON:
                {
                        R_BLE_ABS_StartScan(&gs_scan_param);
                } break;

                case BLE_GAP_EVENT_CONN_IND:
                {
                        if (BLE_SUCCESS == result)
                        {
                                st_ble_gap_conn_evt_t *p_gap_conn_evt_param =
                                        (st_ble_gap_conn_evt_t *)p_data->p_param;
                                if(0x00 == p_gap_conn_evt_param->role)
                                {
                                        g_central_conn_hdl = p_gap_conn_evt_param->conn_hdl;
                                }
                        }
                } break;

                case BLE_GAP_EVENT_DISCONN_IND:
                {
                        st_ble_gap_disconn_evt_t *p_gap_disconn_evt_param =
                                (st_ble_gap_disconn_evt_t *)p_data->p_param;
                        if(p_gap_disconn_evt_param->conn_hdl == g_central_conn_hdl)
                        {
                                g_central_conn_hdl = BLE_GAP_INVALID_CONN_HDL;
                        }
                } break;

                case BLE_GAP_EVENT_CONN_PARAM_UPD_REQ:
                {
                        st_ble_gap_conn_upd_req_evt_t *p_conn_upd_req_evt_param =
                                (st_ble_gap_conn_upd_req_evt_t *)p_data->p_param;
                        if(p_conn_upd_req_evt_param->conn_hdl == g_central_conn_hdl)
                        {
                                st_ble_gap_conn_param_t conn_updt_param = {
                                        .conn_intv_min = p_conn_upd_req_evt_param->conn_intv_min,
                                        .conn_intv_max = p_conn_upd_req_evt_param->conn_intv_max,
                                        .conn_latency  = p_conn_upd_req_evt_param->conn_latency,
                                        .sup_to        = p_conn_upd_req_evt_param->sup_to,
                                };

                                R_BLE_GAP_UpdConn(p_conn_upd_req_evt_param->conn_hdl,
                                                                BLE_GAP_CONN_UPD_MODE_RSP,
                                                                BLE_GAP_CONN_UPD_ACCEPT,
                                                                &conn_updt_param);
                        }
                } break;
                case BLE_GAP_EVENT_ADV_REPT_IND:
                {
                        st_ble_gap_adv_rept_evt_t *p_adv_rept_param =
                                (st_ble_gap_adv_rept_evt_t *)p_data->p_param;
                        st_ble_gap_ext_adv_rept_t *p_ext_adv_rept_param =
                                (st_ble_gap_ext_adv_rept_t *)p_adv_rept_param->param.p_ext_adv_rpt;

                        gs_conn_param.p_addr->type = p_ext_adv_rept_param->addr_type;
                        memcpy(gs_conn_param.p_addr->addr, p_ext_adv_rept_param->p_addr, BLE_BD_ADDR_LEN);

                        R_BLE_GAP_StopScan();
                } break;

                case BLE_GAP_EVENT_SCAN_OFF:
                {
                        R_BLE_ABS_CreateConn(&gs_conn_param);
                }break;

                default:
                {
                        /* Do nothing. */
```

```
                } break;
        }
}
```

コード 7-7　自身がセントラルロールとしてペリフェラルロールとコネクションする場合の gap_cb

```
/* Connection handle */
uint16_t g_peripheral_conn_hdl;

static void ble_peripheral_gapcb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch (type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
        } break;

        case BLE_GAP_EVENT_CONN_IND:
        {
            if (BLE_SUCCESS == result)
            {
                st_ble_gap_conn_evt_t *p_gap_conn_evt_param = (st_ble_gap_conn_evt_t *)p_data->p_param;
                if(0x01 == p_gap_conn_evt_param->role)
                {
                  g_peripheral_conn_hdl = p_gap_conn_evt_param->conn_hdl;
                }
            }
        } break;

        case BLE_GAP_EVENT_CONN_PARAM_UPD_REQ:
        {
            st_ble_gap_conn_upd_req_evt_t *p_conn_upd_req_evt_param =
                  (st_ble_gap_conn_upd_req_evt_t *)p_data->p_param;

            if(p_conn_upd_req_evt_param->conn_hdl == g_peripheral_conn_hdl)
            {
                st_ble_gap_conn_param_t conn_updt_param = {
                    .conn_intv_min = p_conn_upd_req_evt_param->conn_intv_min,
                    .conn_intv_max = p_conn_upd_req_evt_param->conn_intv_max,
                    .conn_latency  = p_conn_upd_req_evt_param->conn_latency,
                    .sup_to        = p_conn_upd_req_evt_param->sup_to,
                };

                R_BLE_GAP_UpdConn(p_conn_upd_req_evt_param->conn_hdl,
                                  BLE_GAP_CONN_UPD_MODE_RSP,
                                  BLE_GAP_CONN_UPD_ACCEPT,
                                  &conn_updt_param);
            }
        } break;

        case BLE_GAP_EVENT_DISCONN_IND:
        {
            st_ble_gap_disconn_evt_t *p_gap_disconn_evt_param =
                  (st_ble_gap_disconn_evt_t *)p_data->p_param;
            if(p_gap_disconn_evt_param->conn_hdl == g_peripheral_conn_hdl)
            {
                  g_peripheral_conn_hdl = BLE_GAP_INVALID_CONN_HDL;
            }
        } break;


        default:
        {
         /* Do Nothing */
        }break;
}
```

コード 7-8　ペリフェラルデバイスとしてコネクションされる場合の GAP コールバック関数の例

これら二つの GAP コールバック関数を gap_cb で呼び出します。

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
        ble_peripheral_gapcb(type, result, p_data);
        ble_central_gapcb(type, result, p_data);
}
```

コード 7-9　各ロールの GAP コールバック関数の呼び出し

マルチロールコネクションするアプリケーションは、GATT クライアントと GATT サーバの両方を実装することがあります。QE for BLE を利用し、GATT クライアント、GATT サーバ両方のサービス API を生成します。QE for BLE のサービス画面で、サーバとクライアントと両方をチェックし、コード生成を行います。

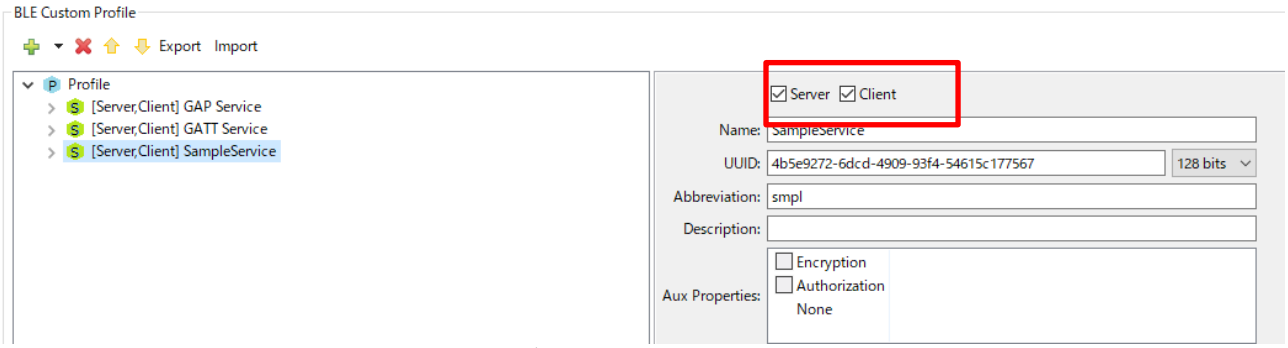


図 7-7　サービス画面での GATT ロールの選択

今回は自身がセントラルの場合に GATT クライアントとして動作させるため、ペリフェラルデバイスとコネクションした場合にサービスディスカバリを行います。

```
/* XXX Service UUID */
static uint8_t XXXC_UUID[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00 };

/* Service discovery parameters */
static st_ble_disc_entry_t gs_disc_entries[] = {

    {
        .p_uuid     = XXXC_UUID,
        .uuid_type  = BLE_GATT_128_BIT_UUID_FORMAT,
        .serv_cb    = R_BLE_XXXC_ServDiscCb,
    },
};
static void disc_comp_cb(uint16_t conn_hdl)
{
    /* TODO: Add function after discovery completed */
    return;
}

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)
    {
        /* TODO: Set callback events of GATTC. Check BLE API reference for events. */

        case BLE_GATTC_EVENT_CONN_IND:
        {
         if(g_central_conn_hdl == p_data->conn_hdl)
         {
                 R_BLE_DISC_Start(p_data->conn_hdl, gs_disc_entries, ARRAY_SIZE(gs_disc_entries),
disc_comp_cb);
         }
        } break;

        default:
        {
            /* Do nothing. */
        } break;
    }
}
```

コード 7-10　セントラルデバイスとしてのサービスディスカバリの実装例

app_lib の Profile Common の Discovery に、QE for BLE が生成するサービス API(r_ble_xxxc.c)の R_BLE_XXXC_ServDiscCb を登録する（コード 7-10 内太枠）と、サービス API にデバイス毎のアトリビュートハンドルが保持されます。サービス API を利用することで、アプリケーションはデバイス毎のアトリビュートハンドルを管理することなく、コネクションハンドルを利用して各デバイスの GATT データベースにアクセスできます。

7.4　切断

現在確立されているコネクションを切断する場合、以下の API をコールします。
ble_status_t R_BLE_GAP_Disconnect(uint16_t conn_hdl, uint8_t reason)

conn_hdl には、切断するコネクションのコネクションハンドルを、reason には、リモートデバイスに通知する切断理由を指定します。切断理由は通常、0x13(REMOTE USER TERMINATED CONNECTION)を指定します。切断の reason で指定するコードについては「Bluetooth Core Specification Vol.2 Part D ,"2 Error Code Descriptions"」を参照してください。この API はセントラル／ペリフェラルの両方からコールすることが可能です。

切断が発生すると、BLE_GAP_EVENT_DISCONN_IND イベントがアプリケーションに通知されます。切断理由は BLE_GAP_EVENT_DISCONN_IND イベントの st_ble_gap_disconn_evt_t イベント構造体の reason フィールドで通知されます。

ローカルデバイスから R_BLE_GAP_Disconnect により切断した場合、
reason: 0x16(Connection Terminated by Local Host)が通知されます。

リモートデバイスから切断した場合、ほとんどの場合、
reason: 0x13(Remote User Terminated Connection)が通知されます。それ以外の場合、リモートデバイスが指定した切断理由が通知されます。

コネクション開始してコネクションインターバル 6 回以内にパケットが受信されなかった場合、
(例として、アクティブスキャンをするデバイスが多い環境でペリフェラルデバイスがスキャンリクエストへの応答でビジーとなり、コネクションリクエストに応答できなかった場合)
reason: 0x3E(Connection Failed to be Established)が通知されます。
コネクションインターバルについては「8.3 コネクションパラメータの更新」を参照してください。

コネクション確立後、スーパービジョンタイムアウト時間内にパケットが受信されなかった場合、
reason: 0x08(Connection Timeout)が通知されます。
スーパービジョンタイムアウトについては「8.3 コネクションパラメータの更新」を参照してください。

ローカルデバイスとリモートデバイスの LTK が不一致の状態で暗号化を開始した場合、
reason: 0x3D(Connection Terminated due to MIC Failure)が通知されます。
そのリモートデバイスは信用できないため、ボンディング情報を削除してペアリングをやり直してください。
LTK については「9.1 ペアリング」を参照してください。

【注】切断されたリモートデバイスと再コネクションする場合には、ペリフェラル側が再度 Connectable Advertising を実行する必要があります。

8. 通信

Bluetooth Low Energy では、通信パラメータを変更することでアプリケーションに適した通信速度や消費電力に調整できます。本章では、Bluetooth LE Protocol Stack を用いた通信パラメータの設定方法を説明します。

表 8.1 に本章で説明する通信パラメータと機能をサポートするライブラリを示します。Optional 機能はリモートデバイスがサポートしていない場合があります。

表 8.1　通信パラメータの種類

通信パラメータ	機能名	機能詳細	ライブラリ	ロール
PHY	LE 2M PHY LE Coded PHY LE 1M PHY (1M 以外は optional)	コネクション時に Central のコネクションリクエストと Peripheral の Advertising のパラメータによって決定される。 コネクション後に変更可能。	All features / Balance All features / Balance 全ライブラリ	Central / Peripheral
最大送信パケット長	LE Data Length Extension (optional)	最大送信バイト数が 27 → 251 バイトに拡張可能。 初期値は BLE_CFG_RF_CONN_DATA_MAX で指定した値。 コネクション後に変更可能。	全ライブラリ	Central / Peripheral
コネクションパラメータ	-	コネクション時に Central のコネクションリクエストのパラメータによって決定される。 コネクション後に変更可能。	全ライブラリ	Central / Peripheral
MTU	-	初期値は 23 バイト。 コネクション中 1 度だけ変更可能。	全ライブラリ	Client

本章では、通信パラメータを変更するための API の使い方を説明します。API の詳細は、「RX23W グループ BLE Module Firmware Integration Technology アプリケーションノート(R01AN4860)」に同梱されている R_BLE API ドキュメント(r_ble_api_spec.chm)をご覧ください。

8.1 PHY の変更

PHY は物理層の変調方式とコーディングスキームを示すパラメータです。本パラメータを変更することで、スループットや電波の到達距離の向上が期待されます。以下に変調方式とコーディングスキームを示します。

- LE 1M PHY

  Bluetooth Low Energy の基本の変調方式です。全ての Bluetooth Low Energy デバイスが対応しています。不特定多数のデバイスとコネクションを行うアプリケーションの場合に設定します。
- LE 2M PHY

  LE 1M PHY からシンボルレートが 2 倍になり、パケット送信時間を短くした変調方式です。高スループット通信を行う場合に利用します。パケット送信時間が短くなるため、消費電力の減少も期待できます。
- LE Coded PHY

  パケットのヘッダ部及びペイロード部に 1/2 もしくは 1/8 の前方誤り訂正符号(コーディングスキーム)を付与した変調方式です。パケット到達率を向上させます。データ到達確実性が上がり従来に比べ通信距離を延ばすことを可能とします。

PHY の変更は、GAP API の R_BLE_GAP_SetPhy 関数を使用します。引数には、設定を変更するコネクションハンドルと送信用変調方式(tx_phys)と受信用変調方式(rx_phys)と送信用コーディングスキーム(phy_options)を指定します。受信用コーディングスキームは変更されません。

ローカルデバイスから PHY を変更する場合のシーケンスチャートを示します。どちらのロールからでも変更できます。リモートデバイスでは、変更を許可する PHY を R_BLE_GAP_SetDefPhy 関数で指定できます。指定しない場合、すべての PHY を受け入れます。

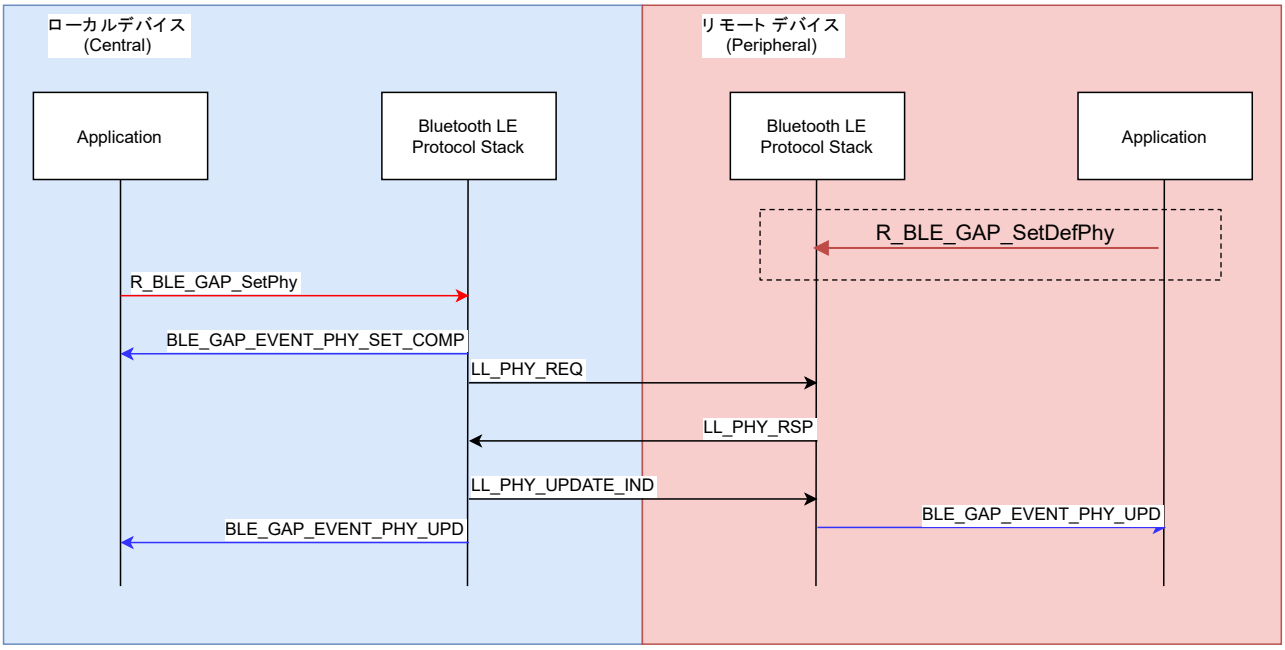


図 8-1　PHY の変更時のシーケンスチャート

以下に PHY を LE Coded PHY (S=8)に変更する場合のサンプルコードを示します。ビット和で複数の PHY を指定できます。使用中の PHY を含む複数の PHY を指定する場合、使用中の PHY は変更されません。使用中の PHY を含まず複数の PHY を指定する場合、最も高速な PHY に変更されます。指定した PHY は、リモートデバイスからの変更を許可する PHY にも適用されます。使用中の PHY は R_BLE_GAP_ReadPhy 関数で取得可能です。

```
st_ble_gap_set_phy_param_t set_phy = {
     .tx_phys = BLE_GAP_SET_PHYS_HOST_PREF_CD,
     .rx_phys = BLE_GAP_SET_PHYS_HOST_PREF_CD,
     .phy_options = BLE_GAP_SET_PHYS_OP_HOST_PREF_S_8
};

R_BLE_GAP_SetPhy(conn_hdl, &set_phy);
```

コード 8-1　PHY を LE Coded PHY (S=8)に変更するコード

PHY の変更によって、二つのイベントがアプリケーションに通知されます。これらのイベントは GAP のコールバック関数(gap_cb)に通知されます。

- BLE_GAP_EVENT_PHY_SET_COMP

  ローカルデバイスのコントローラ層が PHY の変更を受け入れた場合に通知されます。
- BLE_GAP_EVENT_PHY_UPD

  リモートデバイスが PHY の変更を受け入れた場合に通知されます。通知されるイベントデータの tx_phy と rx_phy は、それぞれローカルデバイスからリモートデバイスへの送信、リモートデバイスからローカルデバイスへの送信時に使用する実際の PHY を表します。

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch (type)
    {
      case BLE_GAP_EVENT_PHY_SET_COMP:
      {
        if(BLE_SUCCESS == result)
        {
          st_ble_gap_conn_hdl_evt_t *event_data =
            (st_ble_gap_conn_hdl_evt_t *)p_data->p_param;
          /*event_data->conn_hdl のPHYパラメータの変更がLink Layerに到達*/
        }
        else if(BLE_ERR_INVALID_HDL == result)
        {
          st_ble_gap_conn_hdl_evt_t *event_data =
              (st_ble_gap_conn_hdl_evt_t *)p_data->p_param;
          /*event_data->conn_hdl のコネクションが見つかりませんでした。*/
        }
        else
        {
          /* Do Nothing */
        }
      } break;

      case BLE_GAP_EVENT_PHY_UPD:
      {
        st_ble_gap_phy_upd_evt_t * event_data =
            (st_ble_gap_phy_upd_evt_t *)p_data->p_param;
      } break;
    }
}
```

コード 8-2　PHY 変更時に発生するイベント

PHY が変更されると、送信パケット長に対する送信時間が変化します。Bluetooth LE Protocol Stack は PHY が変更されると後述の最大送信パケット長も PHY に合わせて自動的に変更します。LE Coded PHY に変更されると、最大送信パケット長が 251 バイト、送信時間が 27 バイト分の 2704μsec に設定されます。最大送信パケット長を 28 バイト以上に変更する場合には、後述の「8.2 最大送信パケット長の変更」をご参照ください。

8.2　最大送信パケット長の変更

Link Layer におけるパケットの最大長を設定します。23 バイトを超えるアプリケーションデータを送受信する場合には、送信パケット長を拡張することで効率的な通信ができます。パケット長の拡張は、リモートデバイスが Bluetooth 4.2 で策定された LE Data Packet Length Extension 機能をサポートしている必要があります。

最大送信パケット長の変更は最大送信バイト数と最大送信時間を指定します。パケットの送信時間は前章の PHY の設定により変化します。以下に LE Data Packet Length Extension と LE Coded PHY のサポート有無によって設定可能な最大送信パケット長と最大送信時間を示します。

表 8.2　PHY と設定可能な最大送信パケット長、最大送信時間の関係

LE Data Packet Length Extension	LE Coded PHY feature supported	Parameters with names ending in “Octets”		Parameters with names ending in “Time"	
		Min	Max	Min	Max
No	No	27	27	328	328
Yes	No	27	251	328	2120
No	Yes	27	27	328	2704
Yes	Yes	27	251	328	17040

Bluetooth Core Specification V5.00 Vol 6, Part B

Bluetooth LE Protocol Stack は、リモートデバイスとコネクションされると、最大送信パケット長を BLE_CFG_RF_CONN_DATA_MAX で指定した値に変更する要求を出します。

最大送信パケット長の変更には、GAP API の R_BLE_GAP_SetDataLen 関数を使用します。引数には、設定を変更するコネクションハンドルと最大送信バイト数及び最大送信時間を指定します。最大送信時間は、マイクロ秒で入力します。Bluetooth LE Protocol Stack は、指定された最大送信バイト数と最大送信時間のうち小さいものを優先します。

図 8-2 に、最大送信パケット長の変更を行う場合のシーケンスチャートを示します。

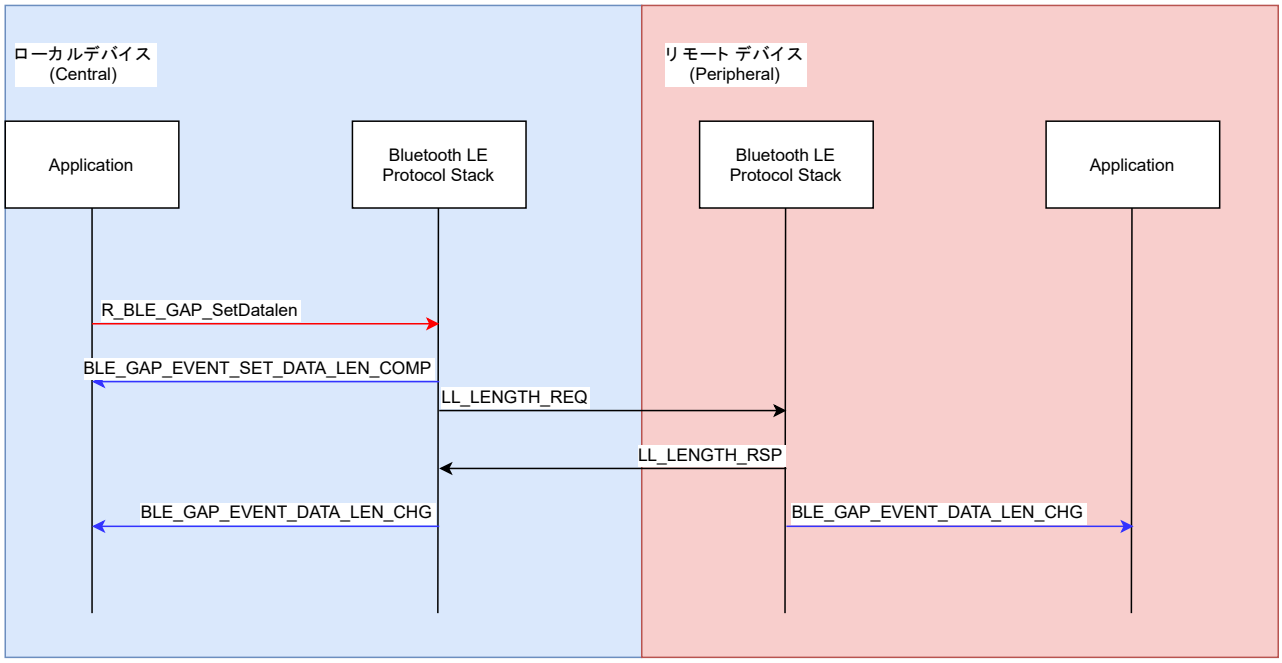


図 8-2　最大送信パケット長の変更時のシーケンスチャート

以下に LE 1M PHY を利用する場合に、パケット長を 251 バイトに拡張する場合の例を示します。

```
uint16_t tx_octets = 251;
uint16_t tx_time = 2120;

R_BLE_GAP_SetDataLen(conn_hdl, tx_octets, tx_time);
```

コード 8-3　送信パケット長の変更要求の例

送信パケット長の変更によって二つのイベントがアプリケーションに通知されます。これらのイベントは GAP のコールバック関数(gap_cb)に通知されます。

- BLE_GAP_EVENT_SET_DATA_LEN_COMP
  コントローラ層に送信パケット長の変更が受け入れられた場合に発生します。

- BLE_GAP_EVENT_DATA_LEN_CHG
  リモートデバイスとの間で送信パケット長が変更された場合に発生します。相手が LE Data Packet Length Extension をサポートしていない場合は発生しません。

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
        switch(type)
        {
                case BLE_GAP_EVENT_SET_DATA_LEN_COMP:
                {
                    st_ble_gap_conn_hdl_evt_t * event_data =
                    (st_ble_gap_conn_hdl_evt_t *)p_data->p_param;
                    /* Do Nothing */
                } break;
                case BLE_GAP_EVENT_DATA_LEN_CHG:
                {
                    st_ble_gap_data_len_chg_evt_t * event_data =
                    (st_ble_gap_data_len_chg_evt_t *)p_data->p_param;
                    /* Do Nothing */
                } break;
        }
}
```

コード 8-4　送信パケット長の変更イベント

8.3　コネクションパラメータの更新

コネクションパラメータは通信の頻度に関するパラメータです。アプリケーションの効率的な動作には、コネクションパラメータの設定が重要です。コネクションパラメータには以下の項目があります。

- コネクションインターバル

  パケット交換の間隔です。コネクションインターバルを短くすると、スループットが向上し消費電力が大きくなります。逆にコネクションインターバルを長くすると消費電力が小さくなります。

- ペリフェラルレイテンシ

  ペリフェラルがセントラルからのパケットを無視する回数です。ペリフェラルはセントラルからパケットを受け取ると応答を返します。ペリフェラルから送信するデータがない場合には、ペリフェラルレイテンシに設定した回数分だけ、セントラルからのパケットを無視できます。ペリフェラルはその回数分だけ応答を返す必要がなくなるため消費電力を下げることができます。

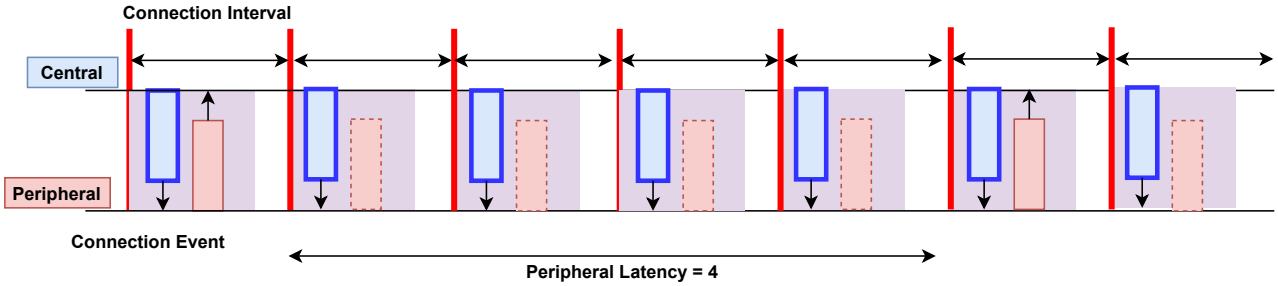


図 8-3　ペリフェラルレイテンシとコネクションイベントの模式図

- スーパービジョンタイムアウト

  パケット受信が途絶えてから切断までの時間です。最後のパケット受信からこの時間が経過するまでにパケットが届かない場合に切断と判断します。スーパービジョンタイムアウト時間内に 2 回以上のパケット交換を行うように設定します。

$$\text{スーパービジョンタイムアウト}(msec) > \left(1 + \text{ペリフェラルレイテンシ}(\text{回})\right) * \text{コネクションインターバル}(msec) * 2$$

- コネクションイベント時間

  コネクションインターバル毎に発生するコネクションイベント時間を指定します。0 を設定すると 1 コネクションイベントにつき 1 往復のみパケットが交換され、0xffff を指定すると次のコネクションイベントまでもしくは More Data bit がセットされなくなるまでパケットを交換し続けます。消費電力を削減する場合には 0 を、スループットを出す場合には 0xffff を指定します。

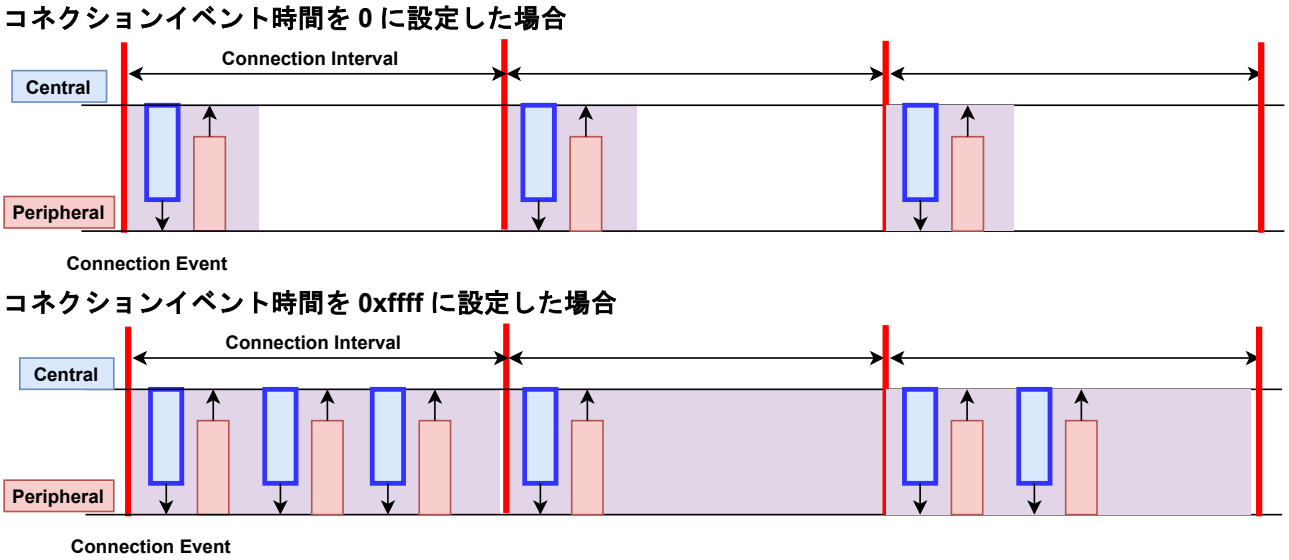


図 8-4　コネクションイベント時間とパケット交換の模式図

コネクションパラメータの決定と変更はセントラルが行いますが、ペリフェラルから変更を要求することも可能です。また、コネクションパラメータはコネクション中に何度でも更新できます。アプリケーションが柔軟にコネクションパラメータを更新することで効率的なデータ通信を実現します。例えば、以下のようなタイミングでコネクションインターバルを変更する事は有効です。

- コネクションインターバルを長くする場合
  送信したいデータがしばらく存在しない
  複数の通信相手と同時にデータ通信する

- コネクションインターバルを短くする場合
  サービスディスカバリを実行する
  小さなデータを一度に短時間で送信する

図 8-5 に、コネクションパラメータの更新を行う場合のシーケンスチャートを示します。ローカルデバイスをセントラル、リモートデバイスをペリフェラルとしています。コネクションパラメータの更新では、更新するデバイスのロールやプロシージャのサポート次第で Link Layer でやり取りする PDU が異なりますが、アプリケーションレベルでは、大きな違いはありません。他のロールについて Link Layer でやり取りされる PDU の詳細については「RX23W グループ BLE Module Firmware Integration Technology アプリケーションノート(R01AN4860)」に同梱されている R_BLE API ドキュメント(r_ble_api_spec.chm)をご覧ください。

【注】R_BLE_API ドキュメントで Connection Latency と記載されているパラメータは Peripheral Latency と同一です。本書では Peripheral Latency と読み替えてください。

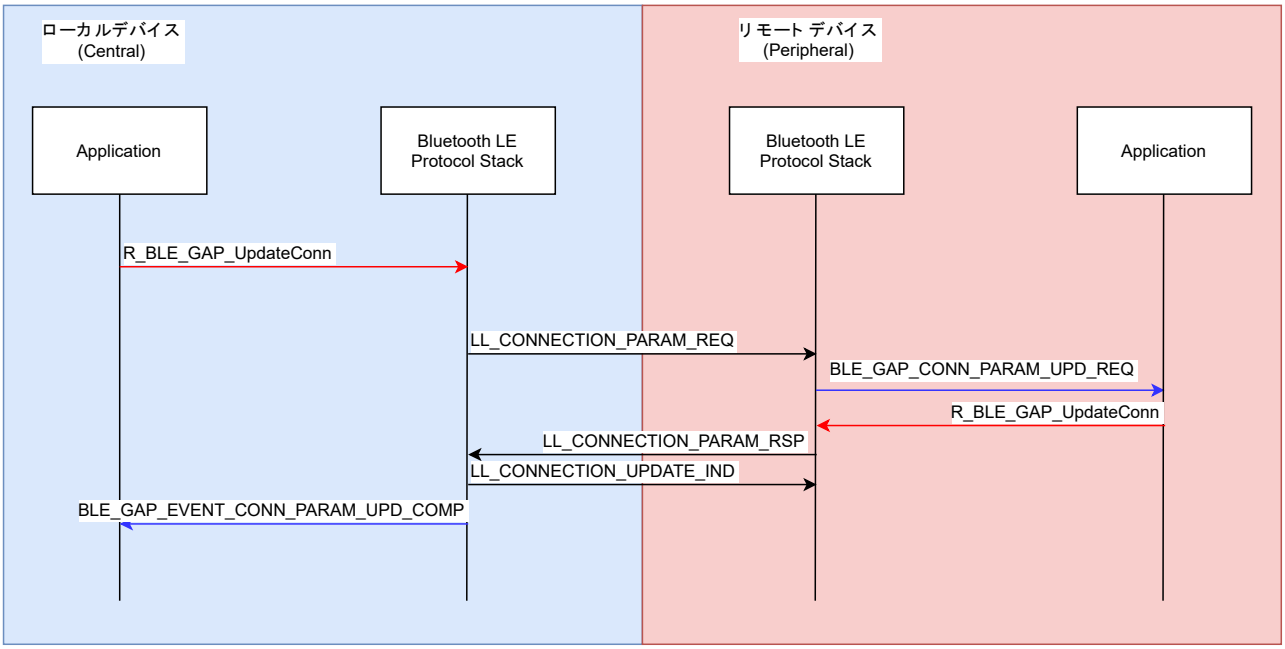


図 8-5　コネクションパラメータの更新時のシーケンスチャート

コネクションパラメータ更新の要求／応答には GAP_API の R_BLE_GAP_UpdConn 関数を使用します。ローカルデバイスから、コネクションパラメータの更新を要求する場合の例を示します。

```
st_ble_gap_conn_param_t conn_param = {
        .conn_intv_min = 0x0006,   //コネクションインターバル
        .conn_intv_max = 0x0006,
        .conn_latency  = 0x0000,   //ペリフェラルレイテンシ
        .sup_to        = 0x0C80,   //スーパービジョンタイムアウト
        .max_ce_length = 0xffff,   //コネクションイベント時間
        .min_ce_length = 0xffff
};

R_BLE_GAP_UpdConn(conn_hdl , BLE_GAP_CONN_UPD_MODE_REQ , 0 , &conn_param);
```

コード 8-5　コネクションパラメータの更新要求の実装例

コネクションパラメータの更新によって二つのイベントがアプリケーションに通知されます。これらのイベントは GAP のコールバック関数(gap_cb)に通知されます。

- BLE_GAP_EVENT_CONN_PARAM_UPD_REQ
  リモートデバイスからコネクションパラメータ更新の要求を受信した場合に通知されます。要求を受け入れるかどうかの処理を実装します。

- BLE_GAP_EVENT_CONN_PARAM_UPD_COMP
  コネクションパラメータの更新が完了すると通知されます。result 変数にコネクションパラメータ更新の要求が受け入れられたかどうかの情報が、イベント変数には、実際のコネクションで使用されるコネクションパラメータが格納されています。

以下に、リモートデバイスからのコネクションパラメータの更新要求に対する応答の実装例を示します。この例では、リモートデバイスからの要求を全て受け入れています。この処理は QE for BLE が出力する app_main.c に実装されています。

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch(type)
    {
      case BLE_GAP_EVENT_CONN_PARAM_UPD_REQ:
        {
            st_ble_gap_conn_upd_req_evt_t *p_conn_upd_req_evt_param =
                                (st_ble_gap_conn_upd_req_evt_t *)p_data->p_param;

            st_ble_gap_conn_param_t conn_updt_param = {
              .conn_intv_min = p_conn_upd_req_evt_param->conn_intv_min,
              .conn_intv_max = p_conn_upd_req_evt_param->conn_intv_max,
              .conn_latency  = p_conn_upd_req_evt_param->conn_latency,
              .sup_to    = p_conn_upd_req_evt_param->sup_to,
            };

            R_BLE_GAP_UpdConn(p_conn_upd_req_evt_param->conn_hdl,
                      BLE_GAP_CONN_UPD_MODE_RSP,
                      BLE_GAP_CONN_UPD_ACCEPT,
                      &conn_updt_param);
      } break;
    }
}
```

コード 8-6　コネクションパラメータの更新要求イベントに対する応答の実装例

スマートフォンとコネクションする場合には、OS によってコネクションパラメータの更新が受け付けられない場合があります。例えば iOS については、Apple デバイス用アクセサリのデザインガイドライン(https://developer.apple.com/jp/accessories/Accessory-Design-Guidelines-JP.pdf)のコネクションパラメータの章に記載されています。

リモートデバイスが拒否した場合、BLE_GAP_EVENT_CONN_PARAM_UPD_COMP イベント通知時の result 変数に BLE_ERR_INVALID_ARG(0x0003)が格納されます。以下に、リモートデバイスから拒否された後、パラメータを見直し、再度、更新要求を行う実装例を示します。

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch(type)
    {
      case BLE_GAP_EVENT_CONN_PARAM_UPD_COMP:
        {
          if(BLE_ERR_INVALID_ARG == result)
          {
               st_ble_gap_conn_param_t conn_param = {
                 .conn_intv_min = 0x0028,       /* コネクションインターバル */
                 .conn_intv_max = 0x0028,
                 .conn_latency  = 0x0000,       /* ペリフェラルレイテンシ */
                 .sup_to        = 0x0C80,       /* スーパービジョンタイムアウト */
                 .max_ce_length = 0xffff,       /* コネクションイベント時間 */
                 .min_ce_length = 0xffff
                };

               R_BLE_GAP_UpdConn(conn_hdl ,
                                 BLE_GAP_CONN_UPD_MODE_REQ ,
                                 0 ,
                                 &conn_param);

          }
        } break;
    }
}
```

コード 8-7　リモートデバイスから拒否された後のコネクションパラメータ更新要求

8.4　MTU の変更

MTU は、GATT における最大パケット長を表します。初期値は最小値である 23 バイトです。これをデフォルト MTU と呼びます。GATT の主なプロシージャである Read Characteristic Value や Write Characteristic Value, Write Without Response, Notification, Indication 動作によってデータ通信を行う場合の最大サイズは MTU に依存します。

デフォルト MTU が使用されているとき、クライアントが 22 より大きいデータを読み出す場合には、GATT の Read Long Characteristic Value、20 より大きいデータを書く場合には、Write Long Characteristic Value を使用します。これらのプロシージャは Read Characteristic Value, Write Characteristic Value よりも通信のオーバヘッドが大きくなります。また、デフォルト MTU の場合には、サーバからの Notification や Indication では 20 より大きいデータを送信できません。そのため、アプリケーションが 20 バイトよりも大きなキャラクタリスティックを扱う場合には MTU の変更が有効です。MTU は、GATT クライアントからコネクション中 1 度だけ変更できます。

オーバヘッドを最小にするために、MTU と最大送信パケット長との関係が

$$MTU(byte) = \textit{最大送信パケット長}(byte) - 4(byte)$$

となるように調整してください。

図 8-6 に、MTU の変更時のシーケンスチャートを示します。

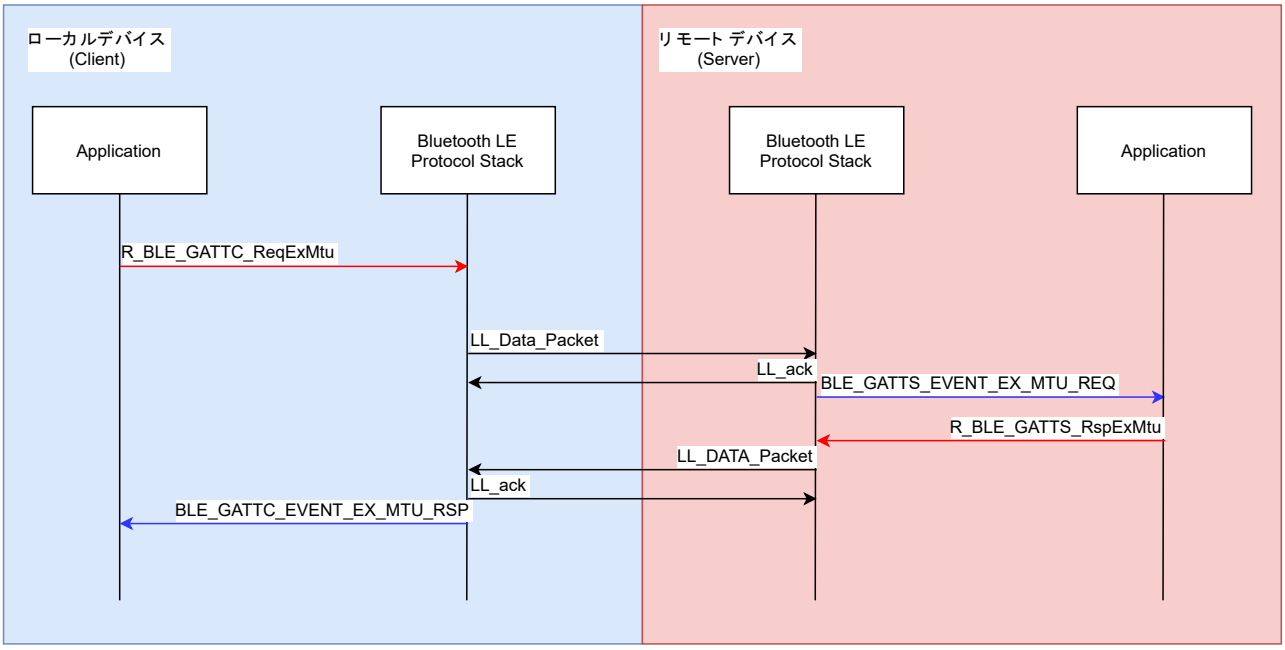


図 8-6　MTU 変更時のシーケンスチャート

MTU の変更は、GATT クライアント API の R_BLE_GATTC_ReqExMtu 関数を使用します。サポートする MTU を引数に指定します。

```
uint16_t mtu = 247
R_BLE_GATTC_ReqExMtu(conn_hdl, mtu);
```

コード 8-8　MTU の変更要求の例

MTU の変更によって二つのイベントがアプリケーションに通知されます。これらのイベントは GATT クライアント若しくは GATT サーバのコールバック関数(gattc_cb, gatts_cb)に通知されます。

- BLE_GATTS_EVENT_EX_MTU_REQ

  クライアントデバイスから MTU 変更の要求を受けた場合に、サーバに通知されます(gatts_cb)。サーバはこのイベント内で自身がサポートする MTU を返します。

- BLE_GATTC_EVENT_EX_MTU_RSP

  サーバデバイスから Exchange MTU Response を受けた場合にクライアントに通知されます(gattc_cb)。自身がサポートする MTU とレスポンスに含まれる MTU のうち小さい方が実際に使用される MTU になります。

コード 8-9 に、GATT サーバの Exchange MTU Request に対する応答の実装例を示します。応答には、GATT サーバ API の R_BLE_GATTS_RspExMtu 関数を使用します。引数に、ローカルデバイスがサポートする MTU を指定します。この処理は app_lib の Profile Common Server Library が提供する R_BLE_SERVS_GattsCb 関数に実装されています。GATT サーバが返す MTU のサイズは BLE_CFG_GATT_MTU_SIZE コンフィグレーションオプションに設定します。QE for BLE から GATT サーバのコードを生成する場合は、アプリケーションで MTU の応答を実装する必要はありません。

```
static void gatts_cb(uint16_t type, ble_status_t result, st_ble_gatts_evt_data_t
*p_data)
{
    switch (type)
    {
        case BLE_GATTS_EVENT_EX_MTU_REQ:
        {
            R_BLE_GATTS_RspExMtu(p_data->conn_hdl, BLE_CFG_GATT_MTU_SIZE);
        } break;
    }
}
```

コード 8-9　MTU 変更要求に対する応答の例

8.5　フロー制御

Bluetooth LE Protocol Stack は大きなアプリケーションデータを短時間で送信するために、フロー制御機能を実装しています。フロー制御機能を実現するために、Bluetooth LE Protocol Stack はアプリケーション通信用に送信バッファを 10 個用意しています。フロー制御機能を有効にすると送信バッファの空数に応じてイベントがアプリケーションに通知されます。

バッファの空数とイベント通知のタイミングを以下に示します。アプリケーションが送信関数を繰り返し呼び出し、バッファの空数が設定した下限値まで減少するとイベントが発生します。このイベントを受けてアプリケーションが送信関数の呼び出しを中止し、バッファのオーバフローを防ぎます。

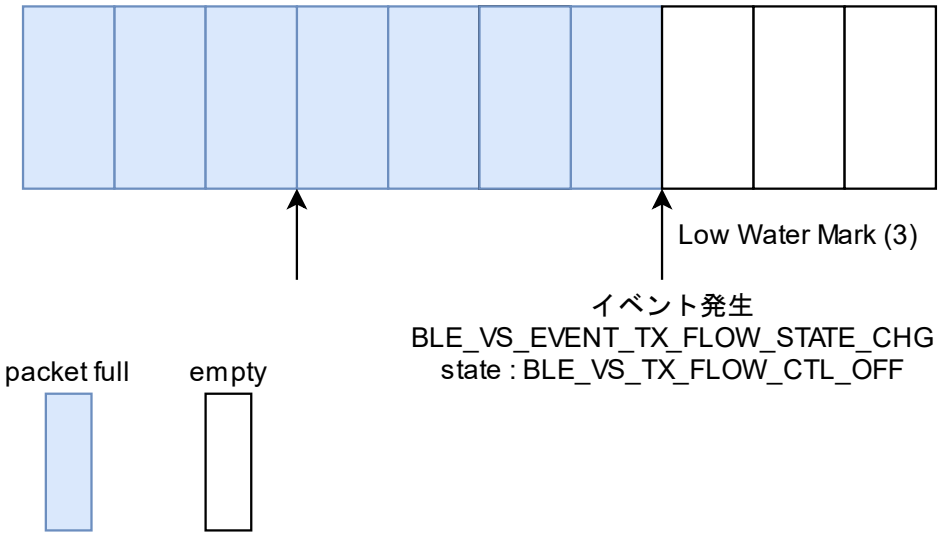


図 8-7　バッファの空数とイベント

リモートデバイスに対して、Bluetooth LE Protocol Stack が送信を行うと、バッファの空数は増加していきます。バッファの空数が設定した上限値まで回復するとイベントが発生します。このイベントを受けて、送信関数の呼び出しを再開します。これを繰り返すことで、大きなデータを効率的に送信できます。

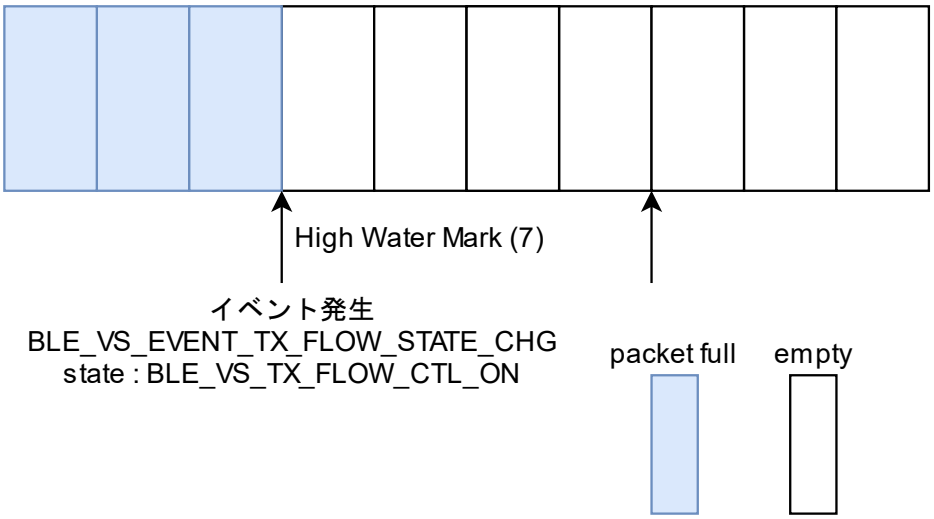


図 8-8　バッファの空数とイベント

Vendor Specific API の R_BLE_VS_SetTxLimit 関数と R_BLE_VS_StartTxFlowEvtNtf 関数によってフロー制御機能を有効にします。

R_BLE_VS_SetTxLimit 関数でイベントが発生するバッファの空数の下限と上限を設定します。R_BLE_VS_StartTxFlowEvtNtf 関数を実行しイベント通知を有効にします。

```
/* Enable Vender Specific Tx Flow Control */
#define LOW_WATER_MARK    (3)
#define HIGH_WATER_MARK   (7)

R_BLE_VS_SetTxLimit(LOW_WATER_MARK, HIGH_WATER_MARK);
R_BLE_VS_StartTxFlowEvtNtf();
```

コード 8-10　フロー制御機能の開始

フロー制御機能は BLE_VS_EVENT_TX_FLOW_STATE_CHG イベントをアプリケーションに通知します。このイベント変数の中に、現在のバッファの状態を示す情報が格納されています。以下にフロー制御機能の利用例を示します。この例では、バッファ内の空数が High Water Mark まで回復した場合に(10-Low Water Mark)回だけ送信関数を呼び出しバッファがオーバフローしないように連続送信しています。R_BLE_ServsCharNotification 関数はサンプルです。使用するサービスの関数に書き換えてください。

```
static void vs_cb(uint16_t type, ble_status_t result, st_ble_vs_evt_data_t *p_data)
{
      R_BLE_SERVS_VsCb(type, result, p_data);

      switch(type)
      {
          case BLE_VS_EVENT_TX_FLOW_STATE_CHG:
          {
              /* Apprize TxFlowState changed to txflow API */
            st_ble_vs_tx_flow_chg_evt_t * evt_data=
              (st_ble_vs_tx_flow_chg_evt_t *)p_data->p_param;
              if(BLE_VS_TX_FLOW_CTL_ON == evt_data->state)
              {
                for (int i=0; i<(10-LOW_WATER_MARK); i++)
                {
                    R_BLE_ServsCharNotification(conn_hdl, &app_data);
                }
              }
              else
              {
                /* Do Nothing */
              }
          } break;

}
```

コード 8-11　フロー制御機能イベントでの送信を行う実装例

8.6　高スループット通信

Bluetooth Low Energy を用いて高スループット通信を行う場合には、通信パラメータを最適値に設定することとフロー制御機能を利用して連続的に送信関数を呼び出すことが重要です。

アプリケーションノート「高速通信用サンプルプログラム(R01AN5437)」内で高スループット通信に関して詳しく説明していますのでご参照ください。

9. セキュリティ

本章では Bluetooth Low Energy が提供するセキュリティ機能について説明します。
最終製品に必要なセキュリティ機能を選択するためには、製品の使用事例を決定しセキュリティ要件を明確にすることが重要です。

9.1 ペアリング

Bluetooth のセキュリティ機能を使用する場合は、事前にペアリングを行います。下記のようなケースにおいて、ペアリングが必要になります。

- 変更可能な GATT キャラクタリスティックがある。
- プライベートアドレスのアドレス解決が必要。
- データの盗聴、改竄、デバイスの追跡など悪意のある攻撃からデバイスを保護したい。

ペアリングはリモートデバイスと鍵を交換します。交換する鍵は以下の通りです。リモートデバイスの鍵は BLE_GAP_EVENT_PEER_KEY_INFO イベントで通知されます。詳細については「9.1.7 鍵の交換」をご参照ください。

- LTK(Long Term Key) , EDIV, Rand
  Bluetooth 通信データを暗号化するときに使用します(鍵の交換は LE Secure Connections では実施されません)。

- IRK(Identity Resolving Key), Identity Address
  プライバシ機能で使用します。

- CSRK(Connection Signature Resolving Key)
  署名付きデータの送受信を行うときに使用します。

ペアリングの方式として、LE legacy pairing と LE Secure Connections があります。
LE Secure Connections は Bluetooth version 4.2 からサポートされた方式です。LE legacy pairing は LE Secure Connections をサポートしないデバイスのペアリング方式です。
Bluetooth LE Protocol Stack では、リモートデバイスが LE Secure Connections をサポートしている場合、LE Secure Connections でペアリングを行います。リモートデバイスが LE Secure Connections をサポートしていない場合、LE legacy pairing でペアリングを行います。

【BP】 LE Secure Connections は、LTK が無線区間で送られない最も安全なペアリングおよび暗号化メカニズムです。また LTK は 16 オクテットで十分なエントロピーを確保しており暗号化されたリンクをブルートフォース(総当たり)攻撃から保護します。これらのことよりセキュアな製品の設計には、リモートデバイス側の制約がない限り LE Secure Connections をサポートすることを推奨します。
また、セキュリティダウンメカニズムを狙う攻撃者からの攻撃を回避するために、LE Secure Connections でのみペアリングを行う場合は、BLE_GAP_SetPairingParams または、R_BLE_ABS_Init API のパラメータ構造体の sec_conn_only メンバに表 9.9 の BLE_GAP_SC_STRICT(0x01) を設定し LE legacy pairing を受け付けないようにすることを推奨します。

【BP】 署名付きデータは、暗号化のためのオーバヘッドを避けたいデバイスに対して提供され、データの完全性を保証することで攻撃者によるデータの改竄を回避することができます。しかし、暗号化による盗聴に対する保護をサポートしていないため攻撃者がリプレイ攻撃(通信を盗聴し得られたデータをそのまま利用することで利用者になりすまし不正にアクセス)を使用することが可能となるため、署名付きデータの使用を避け暗号化をサポートすることが推奨されています。

アプリケーションでのペアリングの手順を図 9-1 に示します。ペアリングの各ステップの詳細について以降の章で説明します。

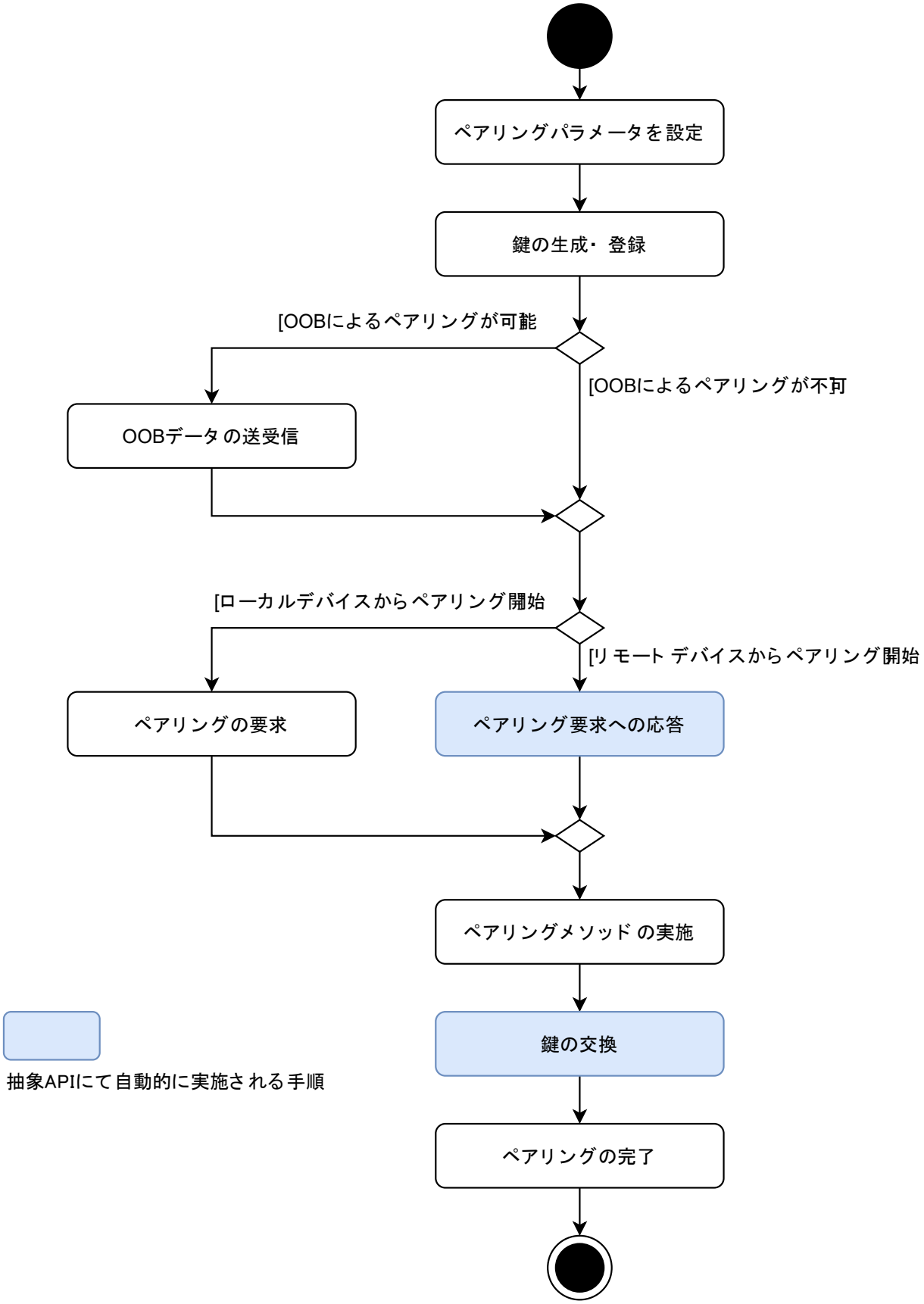


図 9-1　アプリケーションでのペアリングの手順

9.1.1　ペアリングパラメータ設定

ペアリングを開始する前にリモートデバイスと交換するパラメータを設定します。ペアリングパラメータの設定は以下の API により行います。ペアリングの開始前にペアリングパラメータ設定の API をコールしてください。

R_BLE_GAP_SetPairingParams
R_BLE_ABS_Init

設定するペアリングパラメータについて表 9.1 に示します。各パラメータの設定の内容について以降で説明します。

表 9.1　ペアリングパラメータ

	API	R_BLE_ABS_Init	R_BLE_GAP_SetPairing Params	設定範囲	QE for BLE のデフォルト R_BLE_ABS_Init
	パラメータ構造体	st_ble_abs_pairing_param_t	st_ble_gap_pairing_param_t		
1. 入出力機能		iocap	iocap	BLE_GAP_IOCAP_DISPLAY_ONLY(0x00)	BLE_GAP_IOCAP_NOINPUT_NOOUTPUT(0x03)
				BLE_GAP_IOCAP_DISPLAY_YESNO(0x01)	
				BLE_GAP_IOCAP_KEYBOARD_ONLY(0x02)	
				BLE_GAP_IOCAP_NOINPUT_NOOUTPUT(0x03)	
				BLE_GAP_IOCAP_KEYBOARD_DISPLAY(0x04)	
2. MITM プロテクションの要求		mitm	mitm	BLE_GAP_SEC_MITM_BEST_EFFORT(0x00)	BLE_GAP_SEC_MITM_BEST_EFFORT(0x00)
				BLE_GAP_SEC_MITM_STRICT(0x01)	
3. ボンディングの実施		パラメータなし BLE_GAP_BONDING(0x01)に固定	bonding	BLE_GAP_BONDING_NONE(0x00)	BLE_GAP_BONDING(0x01)
				BLE_GAP_BONDING(0x01)	
4. 暗号化鍵のサイズ	最大サイズ	パラメータなし 16 に固定	max_key_size	7～16	16
	最小サイズ	max_key_size	min_key_size		16
5. ペアリングで交換する鍵のタイプ	配布する鍵	loc_key_dist	loc_key_dist	0(鍵を配布しない)	BLE_GAP_KEY_DIST_ENCKEY(0x01)
				BLE_GAP_KEY_DIST_ENCKEY(0x01)	
	配布を希望する鍵	rem_key_dist	rem_key_dist	BLE_GAP_KEY_DIST_IDKEY(0x02)	0
				BLE_GAP_KEY_DIST_SIGNKEY(0x04)	
6. Key Press Notification のサポート		パラメータなし BLE_GAP_SC_KEY_PRESS_NTF_NOT_SPRT(0x00)に固定	key_notf	BLE_GAP_SC_KEY_PRESS_NTF_NOT_SPRT(0x00)	BLE_GAP_SC_KEY_PRESS_NTF_NOT_SPRT(0x00)
				BLE_GAP_SC_KEY_PRESS_NTF_SPRT(0x01)	
7. LE Secure Connections の要求		sec_conn_only	sec_conn_only	BLE_GAP_SC_BEST_EFFORT(0x00)	BLE_GAP_SC_BEST_EFFORT(0x00)
				BLE_GAP_SC_STRICT(0x01)	

1.　入出力機能

ローカルデバイスがサポートする入力機能(表 9.2)と出力機能(表 9.3)から入出力機能を表 9.4 のパラメータで指定します。

表 9.2　入力機能

入力機能	説明
No input	"Yes", "No"を示す入力手段がない
Yes / No	"Yes", "No"を示す入力手段をサポート
Keyboard	0 から 9 までの数値と"Yes", "No"を示す入力手段をサポート

表 9.3　出力機能

出力機能	説明
No output	6 桁の数値を表示する手段がない
Numeric output	6 桁の数値を表示する手段をサポート

表 9.4　入出力機能

		出力機能	
		No output	Numeric output
入力機能	No input	NoInputNoOutput BLE_GAP_IOCAP_NOINPUT_NOOUTPUT(0x03)	DisplayOnly BLE_GAP_IOCAP_DISPLAY_ONLY(0x00)
	Yes / No	NoInputNoOutput BLE_GAP_IOCAP_NOINPUT_NOOUTPUT(0x03)	DisplayYesNo BLE_GAP_IOCAP_DISPLAY_YESNO(0x01)
	Keyboard	KeyboardOnly BLE_GAP_IOCAP_KEYBOARD_ONLY(0x02)	KeyboardDisplay BLE_GAP_IOCAP_KEYBOARD_DISPLAY(0x04)

2.　MITM(man-in-the-middle)プロテクションの要求

MITM に対するプロテクションを要求するかどうかを表 9.5 のパラメータで指定します。

表 9.5　MITM プロテクション

MITM プロテクション	ペアリングパラメータに設定する値
相手に依存	BLE_GAP_SEC_MITM_BEST_EFFORT(0x00)
必要	BLE_GAP_SEC_MITM_STRICT(0x01)

MITM プロテクションが有効となるためには、「9.1.6 ペアリングメソッドの実施」の手順で Just Works 以外のペアリングメソッドでペアリングを完了する必要があります。MITM プロテクションを必要に設定し、両デバイスの入出力機能が表 9.13 の Just Works となるような組み合わせになった場合、ペアリングは失敗します。

【BP】 LE Secure Connections の場合、MITM がペアリング中に共有された秘密鍵を獲得する可能性を低下させるために、入力、出力、または OOB メカニズムを使用して認証されたペアリングをサポートするデバイスの設計を推奨します。

3.　ボンディングの実施

ローカルデバイスがボンディングを行うかどうかを表 9.6 のパラメータで指定します。
ボンディングについては「9.2 ボンディング」をご参照ください。

表 9.6　ボンディングの実施

ボンディングの実施	ペアリングパラメータに設定する値
なし	BLE_GAP_BONDING_NONE(0x00)
あり	BLE_GAP_BONDING(0x01)

R_BLE_ABS_Init の場合、ボンディングの実施は「あり」に固定されます。

4.　暗号化鍵のサイズ

サポートする暗号化鍵の最大、最小サイズを 7-16 バイト中から選択します。暗号化鍵のサイズが原因となり、リモートデバイスからアクセス拒否されないように、最大サイズは 16 バイトを指定することを推奨します。

R_BLE_ABS_Init の場合、サポートする暗号化鍵の最大は 16 バイトに固定されます。

【BP】 暗号化されたリンクをブルートフォース(総当り)攻撃から保護するために最大のエントロピー(16 オクテット)に設定することを推奨します。

5.　ペアリングで交換する鍵のタイプ

ペアリング時に配布する鍵と配布を希望する鍵のタイプを表 9.7 のパラメータで指定します。

表 9.7　鍵のタイプ

鍵のタイプ	ペアリングパラメータに設定する値
LTK, EDIV, Rand	BLE_GAP_KEY_DIST_ENCKEY(0x01)
IRK, Identity Address	BLE_GAP_KEY_DIST_IDKEY(0x02)
CSRK	BLE_GAP_KEY_DIST_SIGNKEY(0x04)

6.　Key Press Notification のサポート

Key Press Notification は「9.1.6 ペアリングメソッドの実施」の手順で Passkey Entry が選択された場合に使われる機能です。Key Press Notification をサポートする場合、デバイスが Passkey Entry でキー入力するときにリモートデバイスにイベント通知します。この機能のサポートを表 9.8 のパラメータで指定します。

表 9.8　Key Press Notification のサポート

Key Press Notification のサポート	ペアリングパラメータに設定する値
なし	BLE_GAP_SC_KEY_PRESS_NTF_NOT_SPRT(0x00)
あり	BLE_GAP_SC_KEY_PRESS_NTF_SPRT(0x01)

R_BLE_ABS_Init の場合、Key Press Notification のサポートは「なし」に固定されます。

7.　LE Secure Connections の要求

LE Secure Connections でのみペアリングを行うかどうかを表 9.9 のパラメータで指定します。

表 9.9　Secure Connections Only の要求

Secure Connections Only の要求	ペアリングパラメータに設定する値
相手に依存	BLE_GAP_SC_BEST_EFFORT(0x00)
あり	BLE_GAP_SC_STRICT(0x01)

BLE_GAP_SC_STRICT を指定して LE legacy pairing が開始した場合、BLE_ERR_SMP_LE_AUTH_REQ_NOT_MET(0x2003)が BLE_GAP_EVENT_PAIRING_COMP イベントの result で通知されます。

以下に R_BLE_GAP_SetPairingParams によるペアリングパラメータ設定のサンプルを示します。

```
st_ble_gap_pairing_param_t pairing_param = {
    .iocap        = BLE_GAP_IOCAP_NOINPUT_NOOUTPUT,
    .mitm         = BLE_GAP_SEC_MITM_BEST_EFFORT,
    .bonding      = BLE_GAP_BONDING,
    .max_key_size  = 16,
    .min_key_size  = 16,
    .loc_key_dist  = BLE_GAP_KEY_DIST_ENCKEY | BLE_GAP_KEY_DIST_IDKEY,
    .rem_key_dist  = BLE_GAP_KEY_DIST_ENCKEY | BLE_GAP_KEY_DIST_IDKEY,
    .key_notf     = BLE_GAP_SC_KEY_PRESS_NTF_NOT_SPRT,
    .sec_conn_only = BLE_GAP_SC_BEST_EFFORT,
};

R_BLE_GAP_SetPairingParams(&pairing_param);
```

コード 9-1　ペアリングパラメータ設定のサンプル

9.1.2　鍵の生成・登録

「9.1.7 鍵の交換」で配布する IRK と CSRK を生成します。R_BLE_VS_GetRand により生成した 16 バイトの乱数を IRK、CSRK として使用することが可能です。生成した鍵は表 9.10 の API により、Bluetooth LE Protocol Stack に登録します。

表 9.10　鍵の登録に使用する API

鍵	登録に使用する API
IRK, Identity Address	R_BLE_ABS_SetLocPrivacy[*1]　or R_BLE_GAP_SetLocIdInfo
CSRK	R_BLE_GAP_SetLocCsrk

[*1]: R_BLE_ABS_SetLocPrivacy は IRK 生成・登録を行い、
Identity Address は現在使用中の Public Address
または、Static Address を使用します。

鍵の生成・登録のサンプルを以下に示します。

```
/* 省略  */
/* IRK generation */
R_BLE_VS_GetRand(0x10);
/* 省略  */

/* Vendor Specific Callback function */
void vscb(uint16_t event_type, ble_status_t result,
          st_ble_vs_evt_data_t * p_event_data)
{
    /* 省略  */
    case BLE_VS_EVENT_GET_RAND
    {
        st_ble_vs_get_rand_comp_evt_t * p_rand_param;
        p_rand_param = (st_ble_vs_get_rand_comp_evt_t *)p_event_data->p_param;
        /* register local IRK and identity address */
        R_BLE_GAP_SetLocIdInfo(&loc_bd_addr, p_rand_param);
    } break;
    /* 省略  */
}
```

コード 9-2　鍵の生成・登録のサンプル

アプリケーションが RPA(Resolvable Privacy Address)を使わない場合、IRK の生成・登録は必要ありません。署名付きデータの送受信を行わない場合、CSRK の生成・登録は必要ありません。LTK(LE legacy pairing の場合は EDIV, Rand 含む)は必要に応じてプロトコルスタック側で生成されますのでアプリケーション側で生成・登録は必要ありません。

【BP】 LE legacy pairing でペアリングされたデバイス間で交換されるデータには EDIV が含まれています。EDIV は特定のデバイスペアに対して一意であるためプライベートアドレスを使用している場合 EDIV を使用してペアリングされたデバイスの追跡が可能になります。長期的な追跡を防ぐために定期的にデバイス間に新しいペアリング/ボンディングを確立して EDIV を更新することが推奨されています。

9.1.3　OOB(Out of band)データの送受信

ローカルデバイスとリモートデバイスが共通の Bluetooth 以外の通信手段(OOB)を持つ場合、ペアリング用データを OOB で送受信することが可能です。ペアリング用データは confirm value(16 バイト), random value(16 バイト)です。OOB でペアリングを行うためには表 9.11 の条件を満たす必要があります。可能な場合、ペアリング用データは開始前に送受信します。

表 9.11　OOB でペアリングを行うための条件

ペアリング方針	OOB でペアリングを行うための条件
LE Secure Connections	どちらかのデバイスが OOB で送信したデータをもう一方のデバイスが受信できる。
LE legacy pairing	両デバイスが OOB でデータを送受信できる。

リモートデバイスから OOB によりペアリング用データを受信した場合、R_BLE_GAP_SetRemOobData により、リモートデバイスのアドレスと受信データを登録します。これにより、ペアリングパラメータ交換時に OOB でデータを受信できたことをリモートデバイスに通知します。

ローカルデバイスが OOB によりデータを送信する場合、R_BLE_GAP_CreateScOobData をコールします。この API は SMP 仕様に従って confirm value(16 バイト), random value(16 バイト)を生成します。データ生成が完了すると、BLE_GAP_EVENT_SC_OOB_CREATE_COMP イベントが通知されます。生成したデータを OOB でリモートデバイスに送信します。

【BP】 OOB で交換される TK(Temporary Key)の最大エントロピーは 128 ビットで MITM 攻撃に対して最も耐性があります。LE legacy pairing をサポートする必要がある場合、OOB メカニズムのサポートが推奨されています。

9.1.4　ペアリングの要求

ローカルデバイスからペアリングを要求する場合は以下のいずれかの API により行います。
R_BLE_ABS_StartAuth
R_BLE_GAP_StartPairing

上記 API はセントラル、ペリフェラルのどちらでもコールすることが可能です。

9.1.5　ペアリング要求への応答

リモートデバイスからペアリング要求を受信した場合、
BLE_GAP_EVENT_PAIRING_REQ
イベントが通知されます。
このイベントに対して、以下の API で応答を返します。
R_BLE_GAP_ReplyPairing

ペアリング要求への応答のサンプルを以下に示します。

```
/* GAP Callback */
void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_event_data)
{
    /* 省略  */
    case BLE_GAP_EVENT_PAIRING_REQ :
    {
        st_ble_gap_pairing_info_evt_t * p_param;
        p_param = (st_ble_gap_pairing_info_evt_t *)p_event_data->p_param;
        R_BLE_GAP_ReplyPairing(p_param->conn_hdl, BLE_GAP_PAIRING_ACCEPT);
    }
    break;
    /* 省略  */
```

コード 9-3　ペアリング要求への応答のサンプル

抽象化 API が有効になっている場合、BLE_GAP_EVENT_PAIRING_REQ イベントにて、R_BLE_GAP_ReplyPairing をコールし、自動的にペアリング応答を返します。

9.1.6　ペアリングメソッドの実施

ペアリング開始、または、ペアリング要求への応答によって、ローカルデバイスとリモートデバイスはペアリングパラメータを交換します。パラメータ交換後、表 9.12 のいずれかのペアリングメソッドが選択され、両デバイスで実施されます。

表 9.12　ペアリングメソッド

ペアリングメソッド	説明	MITM プロテクション
OOB	事前に送受信した OOB データを使って Bluetooth LE Protocol Stack 内で処理が行われるため、アプリケーション上で処理を行うことはありません。	有効
Passkey Entry	一方のデバイスが、6 桁の数値を表示し、 もう一方のデバイスが、表示された数値を入力します。	有効
Numeric Comparison	両方のデバイスに 6 桁の数値を表示します。 表示された数値が同じであるかどうかを入力します。	有効
Just Works	自動的にペアリングが行わるため、アプリケーション上で処理を行うことはありません。	無効

下記の 1-3 に従って、ペアリングメソッドが決まります。

1. ペアリング開始前に OOB データの送受信が行われた場合、OOB が選択されます。
2. OOB 以外が選択され、両デバイスが MITM プロテクションを要求しない場合、Just Works が選択されます。
3. OOB 以外が選択され、どちらか、あるいは両デバイスが MITM プロテクションを要求する場合、表 9.13 に従って、ペアリングメソッドが選択されます。

表 9.13　ペアリングメソッドの選択

ペリフェラル	セントラル				
	DisplayOnly	DisplayYesNo	KeyboardOnly	NoInputNoOutput	KeyboardDisplay
DisplayOnly	Just Works	Just Works	Passkey Entry	Just Works	Passkey Entry
DisplayYesNo	Just Works	Just Works (LE legacy pairing)	Passkey Entry	Just Works	Passkey Entry (LE legacy pairing)
		Numeric Comparison (LE Secure Connections)			Numeric Comparison (LE Secure Connections)
KeyboardOnly	Passkey Entry	Passkey Entry	Passkey Entry	Just Works	Passkey Entry
NoInputNoOutput	Just Works	Just Works	Just Works	Just Works	Just Works
KeyboardDisplay	Passkey Entry	Passkey Entry (LE legacy pairing)	Passkey Entry	Just Works	Passkey Entry (LE legacy pairing)
		Numeric Comparison (LE Secure Connections)			Numeric Comparison (LE Secure Connections)

選択されたペアリングメソッドにより、イベントと応答に使用する API が異なります。

- JustWorks, OOB

通知されるイベント、API での応答は必要ありません。

- Passkey Entry

[入力側]

BLE_GAP_EVENT_PASSKEY_ENTRY_REQ イベントで数値の入力が要求されます。このイベントを受信し、リモートデバイスに数値が表示されたら、アプリケーションにて、R_BLE_GAP_ReplyPasskeyEntry に表示された数値をパラメータとして指定します。コマンドライン機能では、”gap auth passkey xxxxxx(6 桁の passkey)” を入力することで、R_BLE_GAP_ReplyPasskeyEntry で BLE_GAP_EVENT_PASSKEY_ENTRY_REQ イベントに応答することが可能です。

なお、Key Press Notification のサポートを「あり」(表 9.8)に設定した場合、数値入力時に R_BLE_GAP_NotifyKeyPress により、リモートデバイスにどのような入力を行ったのかを通知します。

[表示側]

BLE_GAP_EVENT_PASSKEY_DISPLAY_REQ イベントで数値の表示が要求されます。このイベントを受信したら、イベントで通知された数値を表示します。コマンドライン機能が有効な場合、ターミナルに 6 桁の 10 進数が表示されます。リモートデバイスが Key Press Notification のサポートがある場合、BLE_GAP_EVENT_KEY_PRESS_NTF イベントで入力されたキーの情報が送られてきます。リモートデバイスの数値入力が完了すると、次のステップに進みます。

- Numeric Comparison

BLE_GAP_EVENT_NUM_COMP_REQ イベントでローカルデバイスとリモートデバイスの数値の確認が要求されます。このイベントを受信したら、イベントで通知された数値を表示します。リモートデバイスが表示している数値を確認し、一致しているかどうかの結果を R_BLE_GAP_ReplyNumComp にて送信します。

【BP】 OOB 以外のペアリングメソッドを使用する場合、特定のセキュリティリスクまたはプライバシリスクが存在することを UX/UI がエンドユーザに通知することはセキュリティリスクまたはプライバシリスクを軽減する上で有益です。

9.1.7　鍵の交換

ペアリングメソッドの処理が完了すると、両デバイスが鍵の交換を行います。鍵交換の前にリモートデバイスとのコネクションが暗号化され、BLE_GAP_EVENT_ENC_CHG イベントで通知されます。

リモートデバイスから鍵が配布されると、BLE_GAP_EVENT_PEER_KEY_INFO イベントにより通知されます。このイベントで通知された鍵の保存については、「9.2.1 リモートデバイスの鍵の保存」をご参照ください。

ローカルデバイスの鍵の配布が要求されると、BLE_GAP_EVENT_EX_KEY_REQ イベントにより通知されます。このイベントに対して、R_BLE_GAP_ReplyExKeyInfoReq で応答します。下記に鍵の配布要求への応答のサンプルを示します。

```
/* GAP Callback */
void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_event_data)
{
    /* 省略  */
    case BLE_GAP_EVENT_EX_KEY_REQ :
        {
            st_ble_gap_conn_hdl_evt_t * p_param;
            p_param = (st_ble_gap_conn_hdl_evt_t *)p_event_data->p_param;
            R_BLE_GAP_ReplyExKeyInfoReq(p_param->conn_hdl);
        }
    break;
    /* 省略  */
```

コード 9-4　鍵の配布要求への応答のサンプル

抽象 API が有効な場合、BLE_GAP_EVENT_EX_KEY_REQ イベントにて、R_BLE_GAP_ReplyExKeyInfoReq をコールし、自動的に鍵の配布要求に対して、応答を返します。

9.1.8　ペアリングの完了

ペアリングが完了すると、BLE_GAP_EVENT_PAIRING_COMP イベントが通知されます。ペアリングが成功すると、このイベントの result は BLE_SUCCESS(0x0000)となります。それ以外の値はペアリングの失敗を示します。

30 秒以内にペアリングが完了しなかった場合は、
result: BLE_ERR_SMP_LE_TO(0x2011)
が通知されます。ペアリングをやり直してください。

リモートデバイスに関するボンディング情報が失われたが Resolving List に情報が残っている場合、
result: BLE_ERR_SMP_LE_DHKEY_CHECK_FAIL(0x200B)
が通知されます。R_BLE_GAP_ConfRslvList()により、Resolving List からもそのデバイスに関する情報を削除してください。

ボンディング情報が失われて暗号化要求できなかった場合、
result: BLE_ERR_SMP_LE_LOC_KEY_MISSING(0x2014)
が通知されます。「9.3.1 暗号化要求」を参照してください。

【BP】 ペアリング手順が失敗した場合、同じリモートアドレスとの間で次のペアリングを開始するまで待機間隔を経過する必要があります。待機間隔はペアリング手順の失敗を繰り返すごとに指数関数的に増加します(最大待機間隔は実装によって異なります)。待機間隔の導入は、攻撃(侵入)者が異なるキーを使用してペアリング手順を繰り返し試行する能力を低下させます。

9.2　ボンディング

ボンディングはペアリングにより交換した鍵をデバイスに保存します。ボンディングにより、ペアリング済みのデバイスと再コネクションするときにペアリングを行う必要がなくなります。ボンディングと Bluetooth LE Protocol Stack への鍵の再設定の手順を図 9-2 に示します。

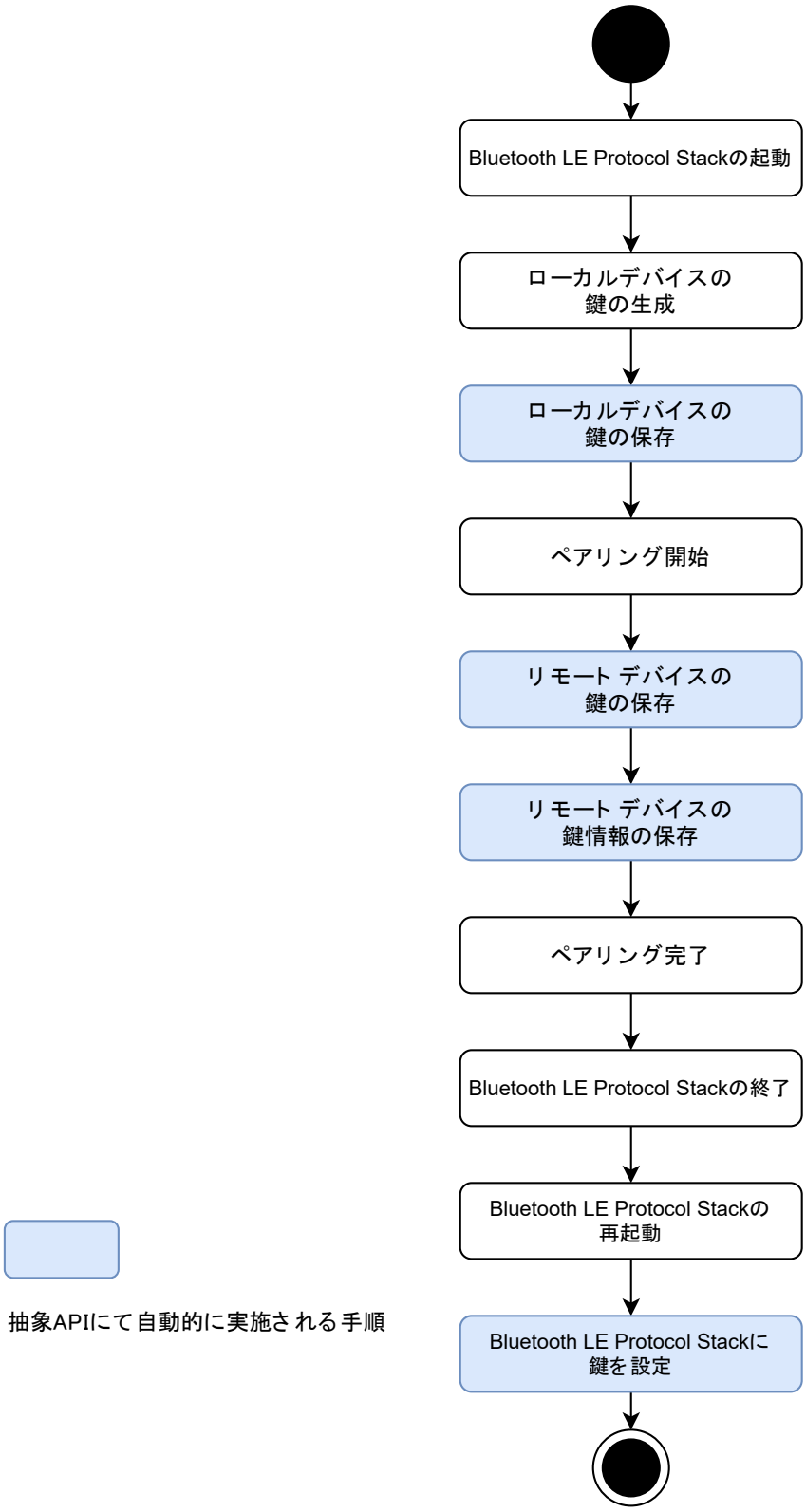


図 9-2　ボンディングの手順

9.2.1　リモートデバイスの鍵の保存

リモートデバイスの鍵として、
BLE_GAP_EVENT_PEER_KEY_INFO (鍵の通知)
BLE_GAP_EVENT_PAIRING_COMP　(鍵の情報の通知)
イベントで受信したデータを Data Flash に保存します。下記に鍵の保存のサンプルを示します。

```
case BLE_GAP_EVENT_PAIRING_COMP :
    {
        if(BLE_SUCCESS == event_result)
        {
            st_ble_gap_pairing_info_evt_t * p_param;
            p_param = (st_ble_gap_pairing_info_evt_t *)p_event_data->p_param;
            /* p_param->auth_infoをData Flashに保存するコードを追加 */
        }
    }
    break;

case BLE_GAP_EVENT_PEER_KEY_INFO :
    {
        st_ble_gap_peer_key_info_evt_t * p_param;
        p_param = (st_ble_gap_peer_key_info_evt_t *)p_event_data->p_param;
        /* p_param->key_ex_paramをData Flashに保存するコードを追加 */
    }
    break;
```

コード 9-5　鍵の保存のサンプル

ボンディング済みのリモートデバイスのアドレスを解決したい場合、Resolving List にもリモートデバイスの IRK と Identity Address を登録してください。

抽象 API とセキュリティデータ管理機能が有効になっている場合、BLE_GAP_EVENT_PEER_KEY_INFO イベントでのリモートデバイスの鍵の Data Flash への保存と BLE_GAP_EVENT_PAIRING_COMP イベントでのリモートデバイスの鍵情報の Data Flash への保存は自動的に行われます。
抽象 API または、セキュリティデータ管理が無効な場合、リモートデバイスの鍵と鍵情報は自動的に保存されません。
ボンディング情報(リモートデバイスの鍵)は RAM と DataFlash に保持されます。RAM のボンディング情報は BLE_CFG_RF_CONN_MAX の数まで保存することができます。DataFlash のボンディング情報は BLE_CFG_NUM_BOND の数まで保存することができます。ボンディングした順番に追加され、ボンディング数の上限を超えてボンディングを行った場合、以下の方針で削除し、上書きを行います。

- RAM 上のボンディング情報は現在コネクション中でない最も古いボンディング情報から自動的に削除されます。
- DataFlash 上のボンディング情報はコネクション状態に関係なく最も古いボンディング情報から自動的に削除されます。

ボンディング情報を自動で削除しないようにするには、希望するデバイス以外とのボンディングや希望するボンディング数を超えたボンディングをしないようにアプリケーションで管理する必要があります。例えば、ボンディングしたリモートデバイス以外と接続しない場合は「9.2.5 ボンディング後のリモートデバイスのフィルタリング」のようなホワイトリストを使った管理が有効です。また、BLE_CFG_RF_CONN_MAX と BLE_CFG_NUM_BOND は同じ数かつ希望するボンディング数+1 を設定することを推奨します。
希望するボンディング数が 2(BLE_CFG_RF_CONN_MAX=3, BLE_CFG_NUM_BOND=3)の場合の例を図 **9-3**, 図 **9-4** に示します。変更部分は太字で表示しています。

	接続情報(RAM)	ボンディング情報 (RAM)	ボンディング情報 (DataFlash)	
接続 (デバイスA)	**接続(デバイスA)**	**未ボンディング(デバイスA)**	解放	← old
	未接続	解放	解放	
	未接続	解放	解放	← new
ペアリング (デバイスA)	接続(デバイスA)	**ボンディング済み(デバイスA)**	**ボンディング済み(デバイスA)**	← old
	未接続	解放	解放	
	未接続	解放	解放	← new
接続 (デバイスB)	接続(デバイスA)	ボンディング済み(デバイスA)	ボンディング済み(デバイスA)	← old
	接続(デバイスB)	**未ボンディング(デバイスB)**	解放	
	未接続	解放	解放	← new
ペアリング (デバイスB)	接続(デバイスA)	ボンディング済み(デバイスA)	ボンディング済み(デバイスA)	← old
	接続(デバイスB)	**ボンディング済み(デバイスB)**	**ボンディング済み(デバイスB)**	
	未接続	解放	解放	← new
接続 (デバイスC)	接続(デバイスA)	ボンディング済み(デバイスA)	ボンディング済み(デバイスA)	← old
	接続(デバイスB)	ボンディング済み(デバイスB)	ボンディング済み(デバイスB)	
	接続(デバイスC)	**未ボンディング(デバイスC)**	解放	← new
ペアリング (デバイスC)	接続(デバイスA)	ボンディング済み(デバイスA)	ボンディング済み(デバイスA)	← old
	接続(デバイスB)	ボンディング済み(デバイスB)	ボンディング済み(デバイスB)	
	接続(デバイスC)	**ボンディング済み(デバイスC)**	**ボンディング済み(デバイスC)**	← new

※希望するボンディング数(=2)を超えたボンディングによって満杯となる

図 9-3 RAM, DataFlash 内のボンディング情報の管理(1)

	接続情報(RAM)	ボンディング情報 (RAM)	ボンディング情報 (DataFlash)	
切断 (デバイスB)	接続(デバイスA)	ボンディング済み(デバイスA)	ボンディング済み(デバイスA)	old
	未接続	ボンディング済み(デバイスB)	ボンディング済み(デバイスB)	
	接続(デバイスC)	ボンディング済み(デバイスC)	ボンディング済み(デバイスC)	new
接続 (デバイスD)	接続(デバイスA)	ボンディング済み(デバイスA)	ボンディング済み(デバイスA)	old
	接続(デバイスC)	**ボンディング済み(デバイスC)**	ボンディング済み(デバイスB)	
	接続(デバイスD)	**未ボンディング(デバイスD)**	ボンディング済み(デバイスC)	new
		※希望するデバイスのボンディング情報が自動で削除される		
切断 (デバイスD)	接続(デバイスA)	ボンディング済み(デバイスA)	ボンディング済み(デバイスA)	old
	接続(デバイスC)	ボンディング済み(デバイスC)	ボンディング済み(デバイスB)	
	未接続	**解放**	ボンディング済み(デバイスC)	new
接続 (デバイスD)	接続(デバイスA)	ボンディング済み(デバイスA)	ボンディング済み(デバイスA)	old
	接続(デバイスC)	ボンディング済み(デバイスC)	ボンディング済み(デバイスB)	
	接続(デバイスD)	**未ボンディング(デバイスD)**	ボンディング済み(デバイスC)	new
ペアリング (デバイスD)	接続(デバイスA)	ボンディング済み(デバイスA)	**ボンディング済み(デバイスB)**	old
	接続(デバイスC)	ボンディング済み(デバイスC)	**ボンディング済み(デバイスC)**	
	接続(デバイスD)	**ボンディング済み(デバイスD)**	**ボンディング済み(デバイスD)**	new
		※希望するボンディング数(=2)を超えたボンディングによって DataFlashの最も古いボンディング情報(デバイスA)が削除される		
再起動	**未接続**	**ボンディング済み(デバイスB)**	ボンディング済み(デバイスB)	old
	未接続	ボンディング済み(デバイスC)	ボンディング済み(デバイスC)	
	未接続	ボンディング済み(デバイスD)	ボンディング済み(デバイスD)	new

※再起動により、DataFlashのボンディング情報がRAMのボンディング情報にコピーされる

図 9-4 RAM, DataFlash 内のボンディング情報の管理(2)

9.2.1.1 RAM のボンディング情報

RAM のボンディング情報は接続情報と同じ領域で管理されるため、ペアリング・ボンディングしない場合でもコネクション中ではない、最も古いボンディング情報から自動で削除され、解放されます。

削除された RAM のボンディング情報を再設定する方法は、「9.2.3 保存した鍵の再設定」をご参照ください。

9.2.1.2 DataFlash のボンディング情報

BLE_CFG_NUM_BOND の数を超えて保存すると、コネクション状態に関係なく、最も古いボンディング情報を自動で削除して新しいボンディング情報を上書きします。図 **9-3**, 図 **9-4** をご参照ください。
DataFlash のボンディング情報を自動で削除しないようにするには、ボンディング情報を削除してからペアリング・ボンディングする等、BLE_CFG_NUM_BOND の数を超えないようにアプリケーションでボンディング数を管理する必要があります。ボンディング情報の削除方法は「9.2.4 保存した鍵の削除」をご参照ください。

9.2.2　ローカルデバイスの鍵の保存

ローカルデバイスがプライバシ機能を使う場合は、
　R_BLE_GAP_SetLocIdInfo
　R_BLE_ABS_SetLocPrivacy
で指定した IRK と Identity Address を保存する必要があります。
ローカルデバイスが署名付きデータの送受信を行う場合は、
　R_BLE_GAP_SetLocCsrk
で指定した CSRK を保存する必要があります。

セキュリティデータ管理が有効な場合は、「4.4.3 ローカルデバイスの鍵の保存」の R_BLE_SECD_WriteLocInfo を使うことで Data Flash に IRK と CSRK を保存することが可能です。

抽象 API とセキュリティデータ管理が有効な場合、R_BLE_ABS_SetLocPrivacy で生成した IRK は R_BLE_SECD_WriteLocInfo により、Data Flash へ自動的に保存されます。

9.2.3　保存した鍵の再設定

Bluetooth LE Protocol Stack を再起動する場合、デバイスに保存した鍵を
　R_BLE_GAP_SetBondInfo
により、Bluetooth LE Protocol Stack に再設定する必要があります。
抽象 API とセキュリティデータ管理が有効の場合、Bluetooth LE Protocol Stack の起動時に自動的に鍵を再設定します。
ボンディング済みのリモートデバイスのアドレスを解決したい場合、Resolving List にもリモートデバイスの IRK と Identity Address を再登録してください。

9.2.4　保存した鍵の削除

リモートデバイス側のボンディング情報が削除された場合は、R_BLE_GAP_DeleteBondInfo の第 2 引数の remote によって、必ず対応するローカルデバイス側のボンディング情報(リモートデバイスのセキュリティデータ)も削除してください。
同様にローカルデバイス側のボンディング情報(リモートデバイスのセキュリティデータ)を削除する場合も必ず対応するリモートデバイス側のボンディング情報を削除してください。
一方のみを削除すると、セキュリティ機能が使用できなくなり、下記のような現象が発生します。

- LTK が失われるためセキュリティ要件として暗号化が設定されている GATT サービスにアクセスできなくなります。
- IRK が失われるためリモートデバイスのアドレス解決ができなくなり、リモートデバイスの Identity Address を使用した接続ができなくなります。

RX23W デバイスとリモートデバイスのどちらか一方のボンディング情報を削除した場合に、削除が必要となるボンディング情報を図 **9-5** に示します。

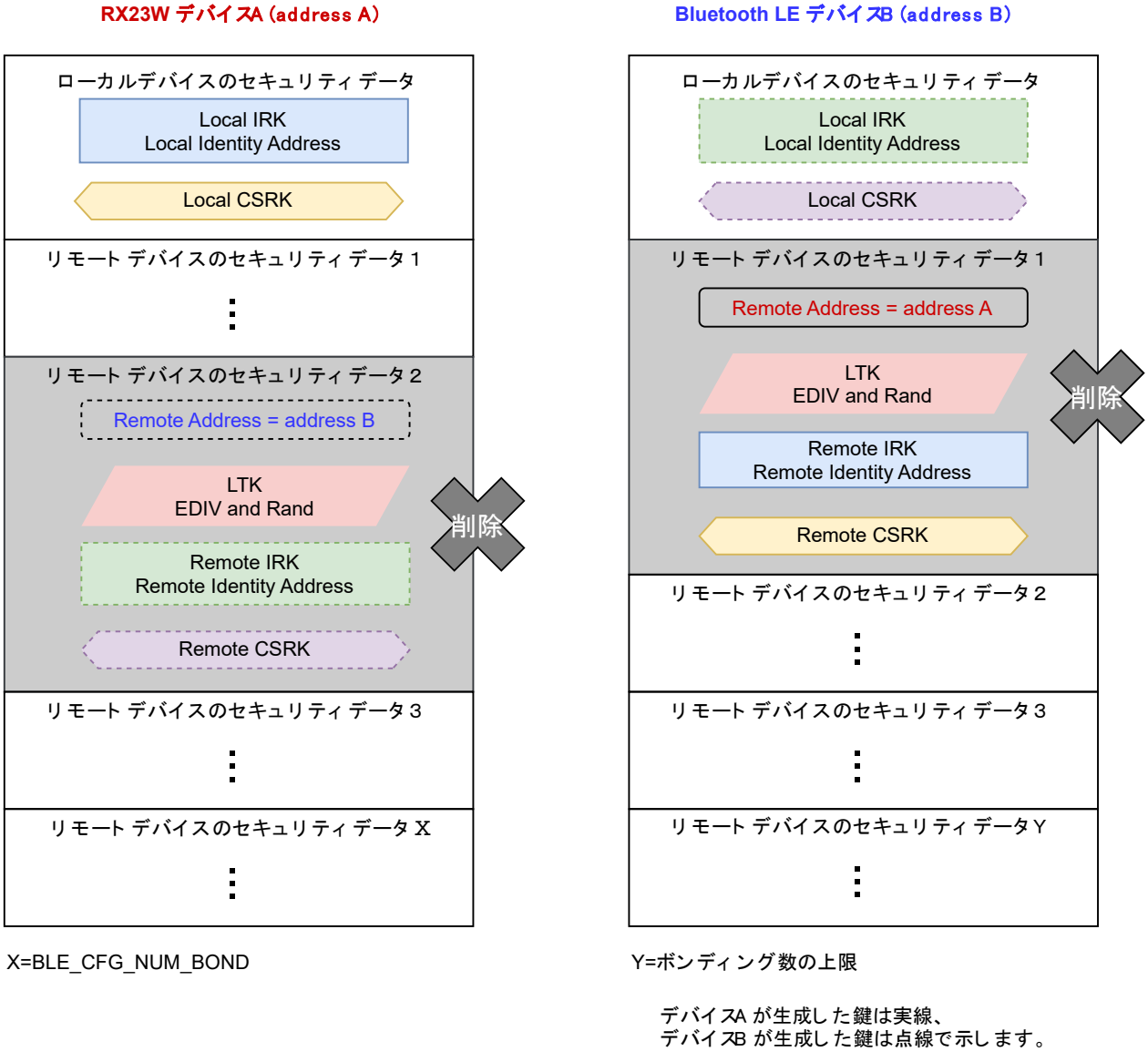


図 9-5 一方のデバイスが削除した場合に削除が必要なボンディング情報

ボンディング済みのリモートデバイスのアドレスを解決していた場合、Resolving List からもリモートデバイスの IRK と Identity Address を削除してください。

9.2.5 ボンディング後のリモートデバイスのフィルタリング

ボンディング済みのリモートデバイスとのみ接続・通信したい場合、ホワイトリストにリモートデバイスの Identity Address を登録してください。ホワイトリストには、All features ライブラリを使用する場合は 4 台まで、Balance, Compact ライブラリを使用する場合は 8 台までのアドレスを登録できます。ホワイトリストの使用方法については、ペリフェラルの場合は、5.2.1.2 と 5.5.1、セントラルの場合は、6.4.1 と 7.1.1 をご参照ください。

再起動時にはホワイトリストへのリモートデバイスのアドレスの再登録が必要となります。

保存した鍵を削除した場合、ホワイトリストからもリモートデバイスのアドレスを削除してください。

9.3　暗号化

Bluetooth では送信するパケットの暗号化によりセキュアな通信が可能となります。
ペアリング完了後にリモートデバイスと再コネクションするときに行われる暗号化はペアリングで交換した鍵を使用します。

【BP】 暗号化の手順が失敗した場合、その失敗を回避することや別の手段で接続するなどしないでください。利便性のために安全性の低いオプションに切り替えることは攻撃者にとって望ましい結果になります。どの段階でも失敗には攻撃者が存在し脆弱性や繰り返しの試行を通じてアクセスしようとしている可能性に注意が必要です。

9.3.1　暗号化要求

事前にペアリング、ボンディングを行った後、リモートデバイスに再コネクションするときにローカルデバイスからは下記のいずれかの API により暗号化の要求を行います。ペリフェラルは Security Request、セントラルは LL_ENC_REQ によって暗号化要求します。
R_BLE_ABS_StartAuth
R_BLE_GAP_StartEnc

暗号化の成功は、
BLE_GAP_EVENT_ENC_CHG
result: BLE_SUCCESS (0x0000)
イベントで通知します。

リモートデバイスの仕様によっては、ペリフェラルからの暗号化要求に対応していないものがあります。この場合、上記の API をコールした場合に、ペアリングが開始されることがあります。

暗号化要求のシーケンスについて以下に示します。

(1)　ローカルデバイス(セントラル)からの暗号化要求

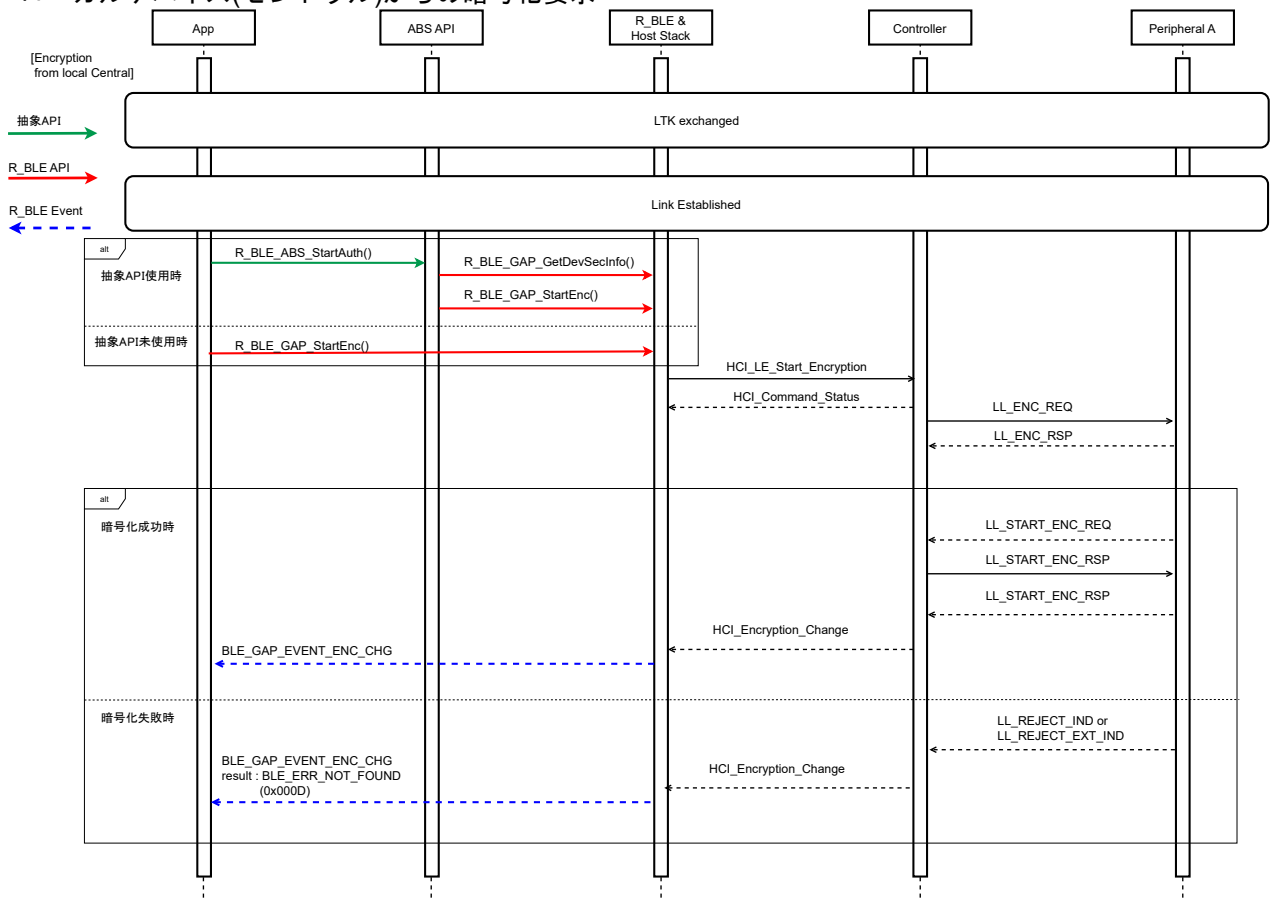


図 9-6　ローカルデバイス(セントラル)からの暗号化要求のシーケンス

ローカルデバイス(セントラル)から暗号化要求する前に、リモートデバイス(ペリフェラル)がペア設定(ボンディング情報)を失った場合は、
BLE_GAP_EVENT_ENC_CHG
result: BLE_ERR_NOT_FOUND(0x000D)
イベントが通知され、暗号化が失敗します。コネクションは維持されますが、暗号化を必要とするセキュリティ要件が設定されたサービスにアクセスできなくなります。
この場合、ローカルデバイス(セントラル)もボンディング情報を削除してペアリングからやり直す必要があります。

ローカルデバイス(セントラル)から暗号化要求する前に、ローカルデバイス(セントラル)がボンディング情報を失った場合は、ローカルデバイス(セントラル)がペアリング要求から開始して暗号化要求します。

(2)　ローカルデバイス(ペリフェラル)からの暗号化要求

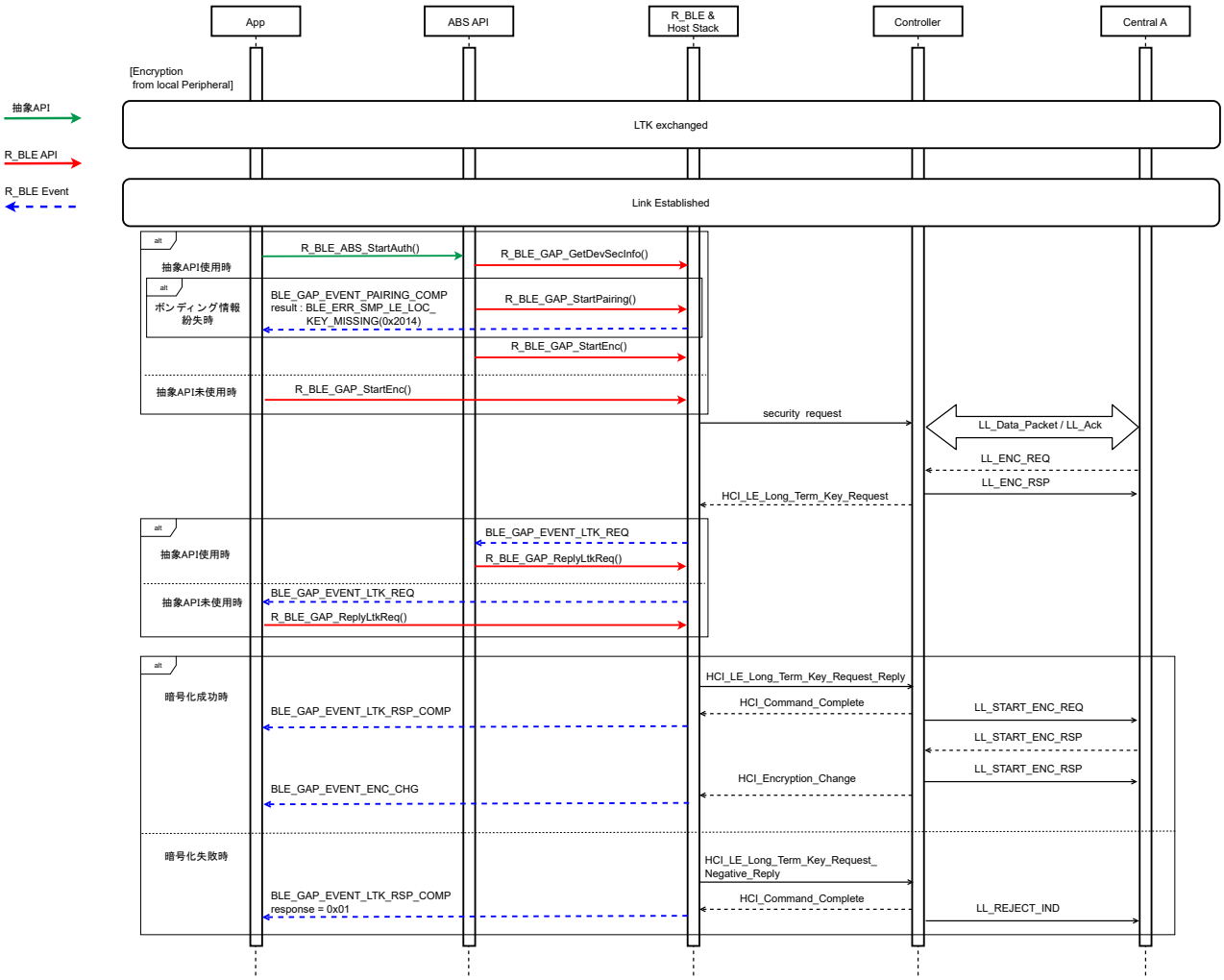


図 9-7　ローカルデバイス(ペリフェラル)からの暗号化要求のシーケンス

ローカルデバイス(ペリフェラル)が暗号化要求する前に、ローカルデバイス(ペリフェラル)がボンディング情報を失った場合は、
BLE_GAP_EVENT_PAIRING_COMP
result: BLE_ERR_SMP_LE_LOC_KEY_MISSING(0x2014)
イベントが通知され、暗号化が失敗します。コネクションは維持されますが、暗号化を必要とするセキュリティ要件が設定されたサービスにアクセスできなくなります。
この場合、リモートデバイス(セントラル)もボンディング情報を削除してペアリングからやり直す必要があります。

ローカルデバイス(ペリフェラル)が暗号化要求する前に、リモートデバイス(セントラル)がボンディング情報を失った場合は、暗号化要求するとペアリングからやり直されて暗号化が行われます。

9.3.2　暗号化要求への応答

暗号化要求への応答はロールによって異なります。
抽象化 API を使用している場合、自動的にリモートデバイスに応答を返します。
暗号化要求への応答のシーケンスについて以下に示します。

(1)　リモートデバイス(セントラル)からの暗号化要求への応答

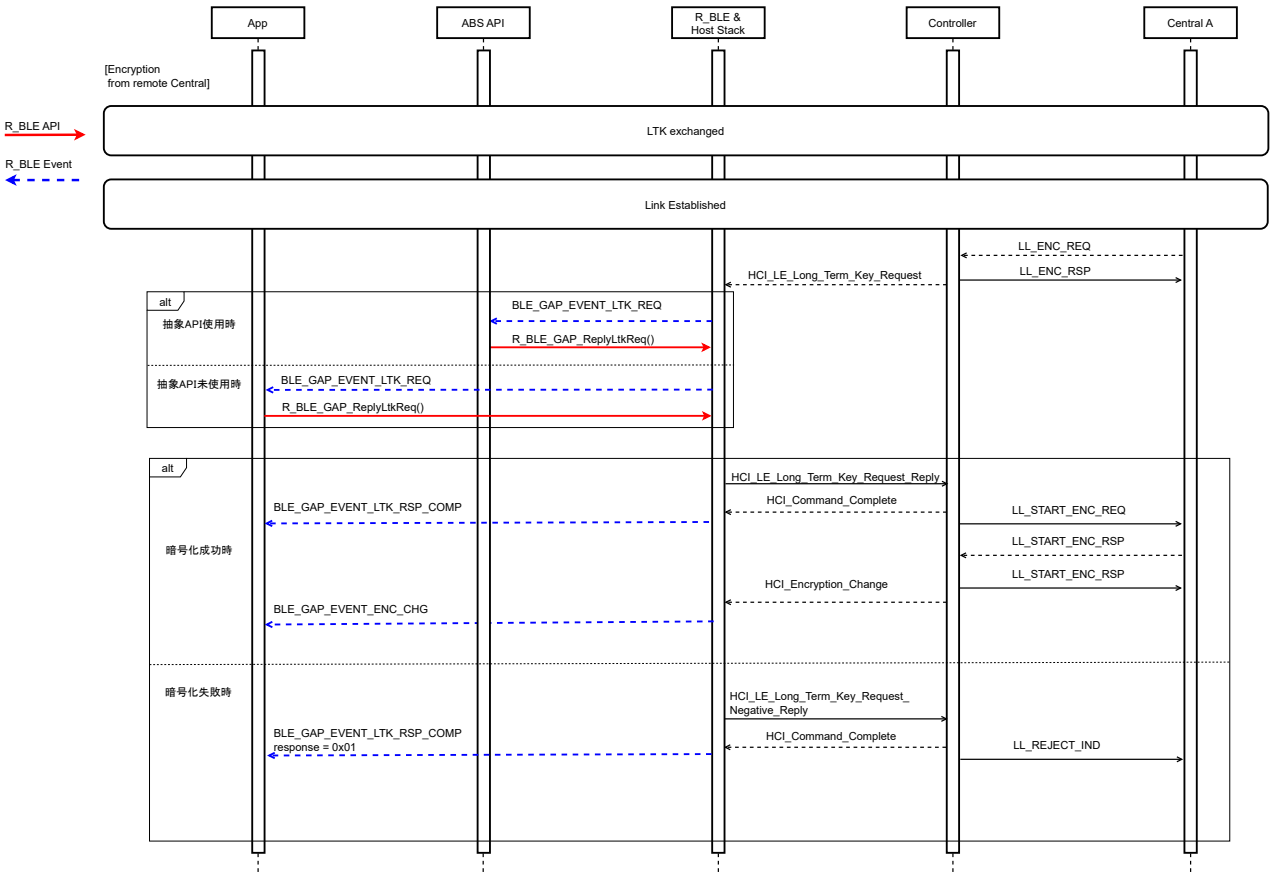


図 9-8　リモートデバイス(セントラル)からの暗号化要求への応答のシーケンス

リモートデバイス(セントラル)から暗号化要求を受信すると、
BLE_GAP_EVENT_LTK_REQ
イベントを通知します。このイベントで通知されるパラメータを指定して、
R_BLE_GAP_ReplyLtkReq
をコールして暗号化要求に応答します。成功した場合、
BLE_GAP_EVENT_LTK_RSP_COMP
イベントで通知します。
暗号化の成功は、
BLE_GAP_EVENT_ENC_CHG
result: BLE_SUCCESS (0x0000)
イベントで通知します。

リモートデバイス(セントラル)が暗号化要求する前に、ローカルデバイス(ペリフェラル)がボンディング情報を失った場合は、R_BLE_GAP_ReplyLtkReq
をコールしても

BLE_GAP_EVENT_LTK_RSP_COMP
イベントデータ response : 0x01
イベントが通知され、暗号化が失敗します。コネクションは維持されますが、暗号化を必要とするセキュリティ要件が設定されたサービスにアクセスできなくなります。
この場合、リモートデバイス(セントラル)もボンディング情報を削除してペアリングからやり直す必要があります。

リモートデバイス(セントラル)が暗号化要求する前に、リモートデバイス(セントラル)がボンディング情報を失った場合は、リモートデバイス(セントラル)がペアリング要求から開始して暗号化要求します。
スマートフォン(セントラル)との初回コネクション時はペアリング要求への応答が求められ、暗号化要求は求められません。ペアリング済みのスマートフォンとの再コネクション時は暗号化要求への応答が求められます。

リモートデバイス(セントラル)からの暗号化要求イベントと応答 API のサンプルを以下に示します。

```
/* GAP Callback */
void gap_cb(uint16_t event_type, ble_status_t event_result,
            st_ble_evt_data_t * p_event_data)
{
    /* 省略  */
    /* Receive encryption request from a remote device */
    case BLE_GAP_EVENT_LTK_REQ :
        {
            st_ble_gap_ltk_req_evt_t * p_param;
            p_param = (st_ble_gap_ltk_req_evt_t *)p_event_data->p_param;
            R_BLE_GAP_ReplyLtkReq(p_param->conn_hdl, p_param->ediv,
                                  p_param->p_peer_rand, BLE_GAP_LTK_REQ_ACCEPT);
        }
        break;
        /* 省略  */
```

コード 9-6　暗号化要求イベントと応答 API のサンプル

(2)　リモートデバイス(ペリフェラル)からの暗号化要求への応答

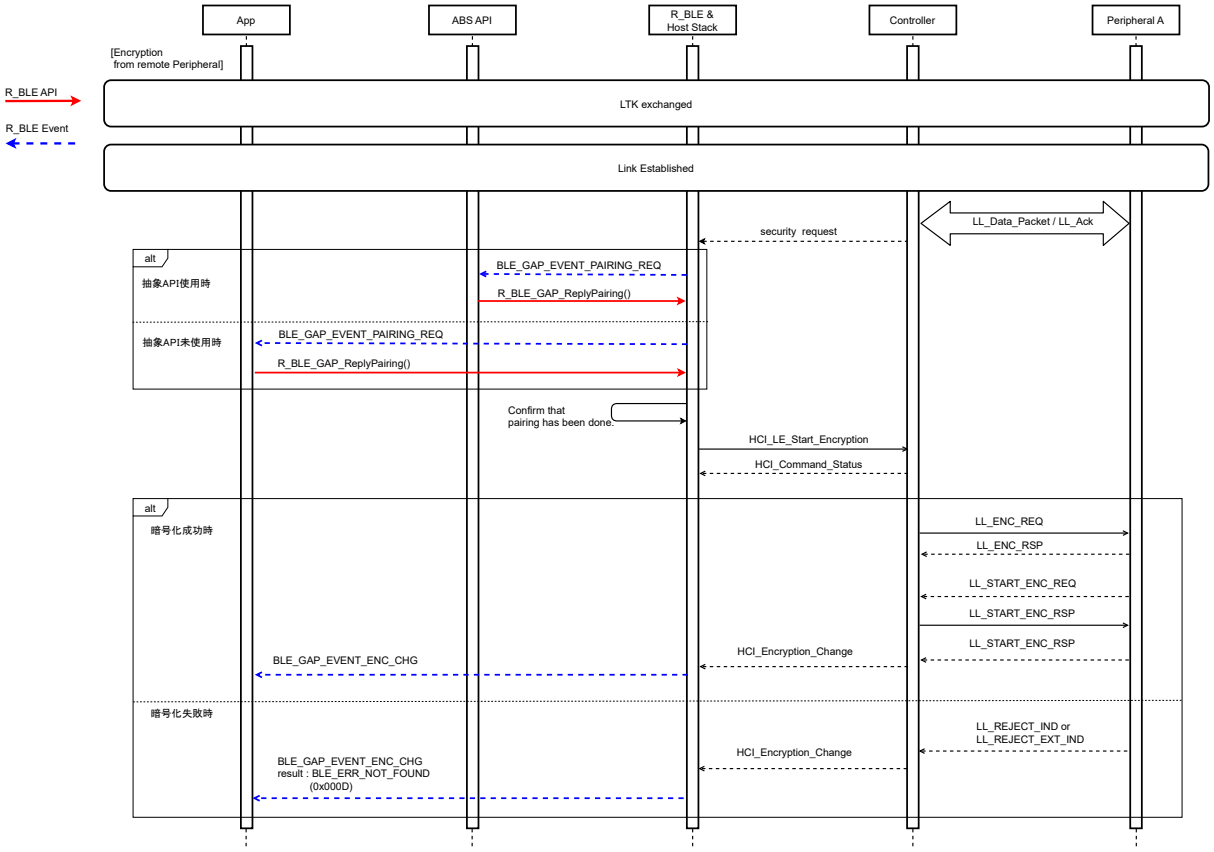


図 9-9　リモートデバイス(ペリフェラル)からの暗号化要求への応答のシーケンス

リモートデバイス(ペリフェラル)から暗号化要求を受信すると、
BLE_GAP_EVENT_PAIRING_REQ
イベントを通知します。
このイベントで通知されるパラメータを指定して、
R_BLE_GAP_ReplyPairing
をコールします。ボンディング済みの場合、この API で暗号化要求に応答します。
暗号化の成功は、
BLE_GAP_EVENT_ENC_CHG
result: BLE_SUCCESS (0x0000)
イベントで通知します。

リモートデバイス(ペリフェラル)が暗号化要求する前に、リモートデバイス(ペリフェラル)がボンディング情報を失った場合は、
BLE_GAP_EVENT_ENC_CHG
result: BLE_ERR_NOT_FOUND(0x000D)
イベントが通知され、暗号化が失敗します。コネクションは維持されますが、暗号化を必要とするセキュリティ要件が設定されたサービスにアクセスできなくなります。
この場合、ローカルデバイス(セントラル)もボンディング情報を削除してペアリングからやり直す必要があります。

リモートデバイス(ペリフェラル)が暗号化要求する前に、ローカルデバイス(セントラル)がボンディング情報を失った場合は、暗号化要求するとペアリングからやり直されて暗号化が行われます。

リモートデバイス(ペリフェラル)からの暗号化要求イベントと応答 API のサンプルはコード 9-3 と同じです。

9.4　プライバシ

プライバシは他のデバイスから追跡されないように周期的に異なるアドレスに変更して Advertising、スキャンリクエスト、コネクションリクエストする機能です。プライバシモードには、Device Privacy Mode と Network Privacy Mode があります。Device Privacy Mode ではローカルデバイスが RPA (Resolving Private Address)を使用する場合に、リモートデバイスのアドレスに関係なく、Advertising、スキャンリクエスト、コネクションリクエストを受け入れます。Network Privacy Mode ではローカルデバイスが RPA を使用する場合に、リモートデバイスの Identity Address を含む Advertising、スキャンリクエスト、コネクションリクエストを受け入れません。デフォルトは Network Privacy Mode に設定されており、RPA はローカルデバイス内の Resolving List を使用して生成・解決されます。

Resolving List にはリモートデバイスの IRK(Remote IRK)と Identity Address(ID)、ローカルデバイスの IRK(Local IRK)を 1 組として 8 組まで登録できます。

ローカルデバイスが RPA を生成して Advertising、スキャン、コネクションを開始する場合は、あらかじめ Resolving List に Local IRK とリモートデバイスの ID を登録し、指定されたリモートデバイスの ID で Resolving List を検索します。RPA 生成方法の詳細は 9.4.1 に記載します。

リモートデバイスからの Advertising、スキャンリクエスト、コネクションリクエストに含まれる RPA を解決する場合は、あらかじめリモートデバイスとのペアリングで入手した IRK と ID を Local IRK と共に Resolving List に登録しておくことで、受信した RPA と Resolving List 内の登録された IRK と ID から計算した RPA が一致する List を検索します。RPA 解決方法の詳細は 9.4.2 に記載します。

Advertising で Resolving List を利用して RPA を生成・解決する際のイメージを図 9-10 に示します。

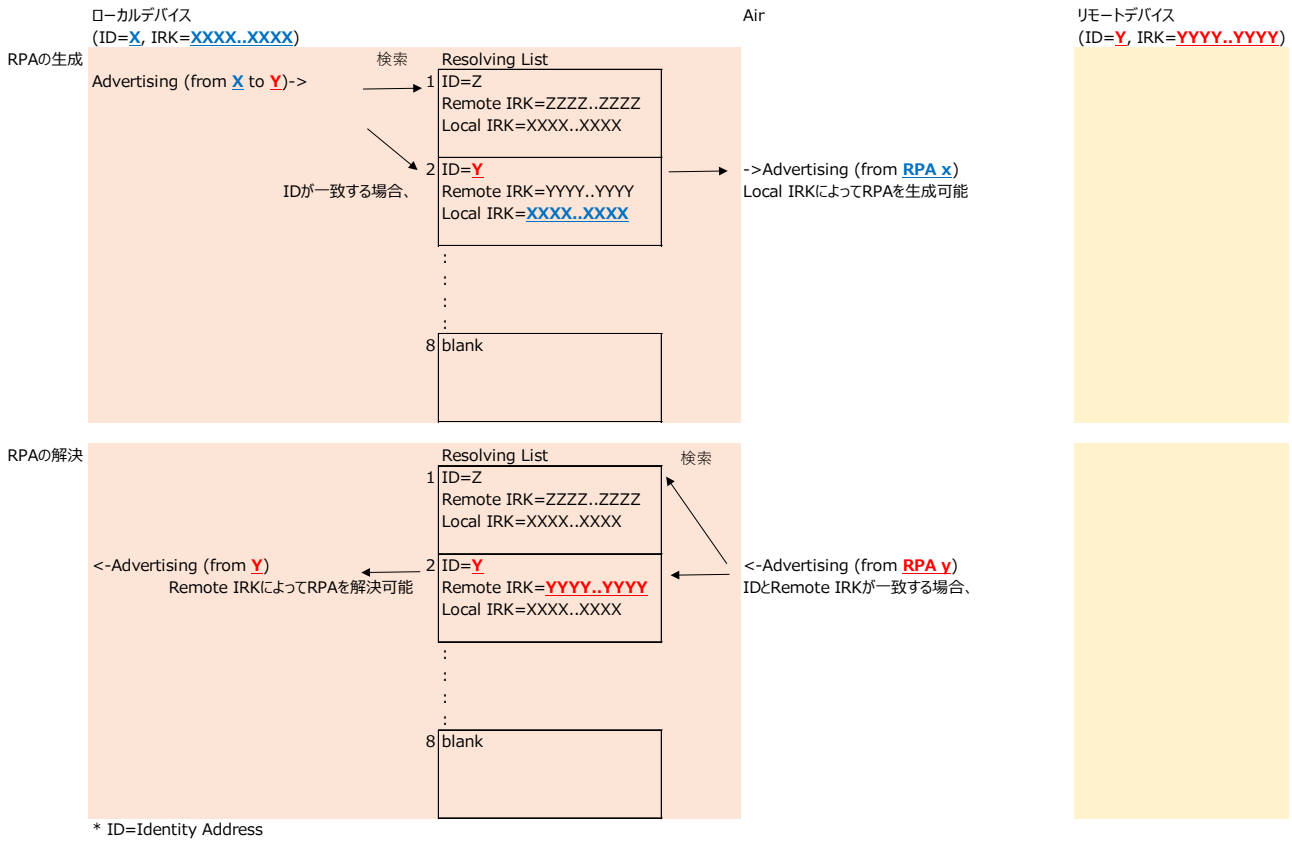


図 9-10　Resolving List のイメージ

アプリケーションでのプライバシの手順を図 9-11 に示します。ペアリングの各ステップの詳細について以降の章で説明します。

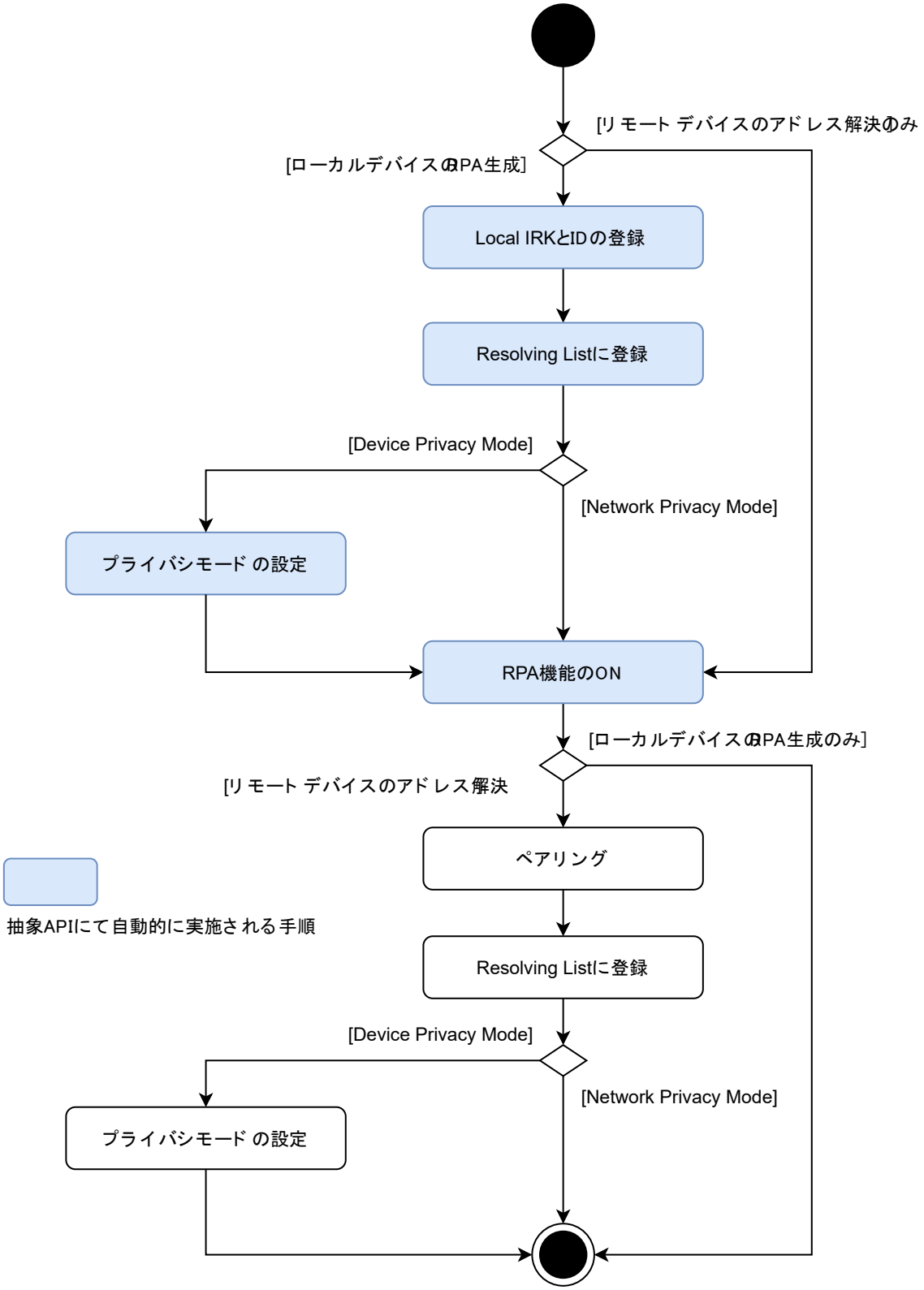


図 9-11　アプリケーションでのプライバシの手順

9.4.1　ローカルデバイスの RPA 生成

ローカルデバイスが RPA を使う前の事前準備として、以下の 1-4 の手順を行います。1-4 の API コールはまとめて R_BLE_ABS_SetLocPrivacy により代替することが可能です。

1. ローカルデバイスの鍵(IRK)、BD アドレスの登録
   R_BLE_VS_GetRand をコールし、BLE_VS_EVENT_GET_RAND で通知される、16 バイトの乱数を IRK として生成します。生成した IRK と Identity Address は R_BLE_GAP_SetLocIdInfo により、Bluetooth LE Protocol Stack に登録します。登録した IRK はペアリング時にリモートデバイスに配布されます。

2. Resolving List に IRK を登録
   1 で生成した IRK を R_BLE_GAP_ConfRslvList()により、Resolving List に登録します。このときに関連付けるリモートデバイスの Identity Address と IRK も登録が必要となります。関連付けるリモートデバイスの Identity Address と IRK にはすべて 0x00 の値を登録します。登録完了は BLE_GAP_EVENT_RSLV_LIST_CONF_COMP イベントにより通知されます。

3. プライバシモードの設定
   デフォルトの Network Privacy Mode を使う場合は、この手順は必要ありません。R_BLE_GAP_SetPrivMode()によりプライバシモードを設定します。プライバシモード設定完了は BLE_GAP_EVENT_PRIV_MODE_SET_COMP イベントにより通知されます。

4. RPA 機能の ON
   R_BLE_GAP_EnableRpa()により、RPA 生成・解決機能を有効にします。完了すると、BLE_GAP_EVENT_RPA_EN_COMP イベントを通知します。

以下に 1-4 までのサンプルを示します。ローカルデバイスが RPA を生成する場合、宛先アドレスを Resolving List の Identity Address と一致させる必要があります。

```
/* 省略  */
#include "sec_data/r_ble_sec_data.h"
/* 省略  */
st_ble_dev_addr_t gs_loc_bd_addr;
st_ble_dev_addr_t gs_rem_bd_addr;

/* Advertising parameters */
static st_ble_abs_legacy_adv_param_t gs_adv_param =
{
    /* TODO: Modify advertise parameters. */
    .p_addr         = &gs_rem_bd_addr,
    .o_addr_type     = BLE_GAP_ADDR_RPA_ID_PUBLIC,
    /* 省略 */
};
/* 省略  */

/* Vendor Specific callback function */
void vscb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_data)
{
    switch(event_type)
    {
         /* 省略  */
        case BLE_VS_EVENT_GET_RAND :
        {
            st_ble_vs_get_rand_comp_evt_t * p_rand_param;
            p_rand_param = (st_ble_vs_get_rand_comp_evt_t *)p_data->p_param;
            R_BLE_GAP_SetLocIdInfo(&gs_loc_bd_addr, p_rand_param->p_rand);

            /*  store local id info */
            R_BLE_SECD_WriteLocInfo(&gs_loc_bd_addr, p_rand_param->p_rand, NULL);
```

```
            /* Set all zero remote address & remote IRK */
            st_ble_gap_rslv_list_key_set_t peer_irk;

            memset(peer_irk.remote_irk, 0x00, BLE_GAP_IRK_SIZE);
            peer_irk.local_irk_type = BLE_GAP_RL_LOC_KEY_REGISTERED;
            memset(gs_rem_bd_addr.addr, 0x00, BLE_BD_ADDR_LEN);
            gs_rem_bd_addr.type = BLE_GAP_ADDR_PUBLIC;

            /* Add local IRK to resolving list */
            R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV, &gs_rem_bd_addr, &peer_irk, 1);
        }
        break;
        /* 省略  */
    }
}

/* GAP Callback */
void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_data)
{
    switch(event_type)
    {
         /* 省略  */
        case BLE_GAP_EVENT_RSLV_LIST_CONF_COMP :
            {
                st_ble_gap_rslv_list_conf_evt_t * p_rslv_list_conf;
                p_rslv_list_conf = (st_ble_gap_rslv_list_conf_evt_t *)p_data->p_param;
               if(BLE_GAP_LIST_ADD_DEV == p_rslv_list_conf->op_code)
               {
                   uint8_t priv_mode;
                   priv_mode = BLE_GAP_NET_PRIV_MODE ;

                   /* Set Network Privacy Mode. */
                   R_BLE_GAP_SetPrivMode(&gs_rem_bd_addr, &priv_mode, 1);
               }
            }
            break;

        case BLE_GAP_EVENT_PRIV_MODE_SET_COMP :
            {
               /* Enable RPA. */
               R_BLE_GAP_EnableRpa(BLE_GAP_RPA_ENABLED);
            }
            break;

        case BLE_GAP_EVENT_LOC_VER_INFO:
            {
               st_ble_gap_loc_dev_info_evt_t * ev_param;
               ev_param = (st_ble_gap_loc_dev_info_evt_t *)p_data->p_param;
               gs_loc_bd_addr = ev_param->l_dev_addr;
               /* Generate IRK */
               R_BLE_VS_GetRand(BLE_GAP_IRK_SIZE);
            } break;

        case BLE_GAP_EVENT_RPA_EN_COMP:
            {
                /* Start advertising */
                R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
            } break;
        /* 省略  */
    }
}
```

コード 9-7　ローカルデバイスが RPA を使う前の事前準備(1)

R_BLE_ABS_SetLocPrivacy を使った場合のサンプルを以下に示します。

```
/* 省略  */
st_ble_dev_addr_t gs_rem_bd_addr;

/* Advertising parameters */
static st_ble_abs_legacy_adv_param_t gs_adv_param =
{
    /* TODO: Modify advertise parameters. */
    .p_addr         = &gs_rem_bd_addr,
    .o_addr_type     = BLE_GAP_ADDR_RPA_ID_PUBLIC,
    /* 省略 */
};
/* 省略  */

/* GAP Callback */
void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_data)
{
    switch(event_type)
    {
        case BLE_GAP_EVENT_LOC_VER_INFO:
            {
                R_BLE_ABS_SetLocPrivacy(NULL, BLE_GAP_DEV_PRIV_MODE);
            } break;

        case BLE_GAP_EVENT_RPA_EN_COMP:
            {
                /* Start advertising */
                memset(gs_adv_param.p_addr->addr, 0x00, BLE_BD_ADDR_LEN);
                gs_adv_param.p_addr->type = BLE_GAP_ADDR_PUBLIC;
                R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
            } break;
        /* 省略  */
    }
}
```

コード 9-8　ローカルデバイスが RPA を使う前の事前準備(2)

Advertising、スキャンリクエスト、コネクションリクエストを行うときにローカルデバイスのアドレスタイプとして、RPA を指定することで、送信パケット内のアドレスが RPA となります。

[Advertising]
R_BLE_GAP_SetAdvParam()による Advertising parameter を設定時に、表 5.4 のパラメータを設定します。

[スキャン]
R_BLE_GAP_StartScan()による scan parameter 設定、ローカルデバイスのアドレスタイプとして RPA を指定することが可能です。「6.2.1 プライバシ」をご参照ください。

[コネクション]
R_BLE_GAP_CreateConn()による connect 用の API で ローカルデバイスのアドレスタイプとして RPA を指定することが可能です。「7.1.2 プライバシ」をご参照ください。

9.4.2　リモートデバイスのアドレス解決

リモートデバイスの RPA は以下の手順でアドレス解決を行います。

1. RPA 機能の ON
   R_BLE_GAP_EnableRpa()により、RPA 生成・解決機能を有効にします。完了すると、BLE_GAP_EVENT_RPA_EN_COMP イベントを通知します。R_BLE_ABS_SetLocPrivacy により代替することが可能です。ローカルデバイスが RPA を使わない場合、R_BLE_ABS_SetLocPrivacy の第 1 パラメータに all 0 の IRK を登録します。

2. ペアリング
   ペアリングにより、リモートデバイスの IRK と Identity Address を取得します。ペアリングの詳細については、「9.1 ペアリング」をご参照ください。

3. リモートデバイスの鍵(IRK)と BD アドレスの登録
   Resolving List にリモートデバイスの IRK と Identity Address を R_BLE_GAP_ConfRslvList()により、登録します。このとき、ローカルデバイスの IRK も同時に登録します。ローカルデバイスが RPA を使わない場合、第 3 パラメータの st_ble_gap_rslv_list_key_set_t 型配列の local_irk_type を BLE_GAP_RL_LOC_KEY_ALL_ZERO に設定することによって、all 0 の IRK を登録します。登録完了は BLE_GAP_EVENT_RSLV_LIST_CONF_COMP イベントにより通知されます。

4. プライバシモードの設定
   デフォルトの Network Privacy Mode を使う場合は、この手順は必要ありません。R_BLE_GAP_SetPrivMode()により、プライバシモードを変更します。変更が完了すると、BLE_GAP_EVENT_PRIV_MODE_SET_COMP イベントで通知されます。

5. RPA の解決
   1-3 の設定を行った後、Bluetooth LE Protocol Stack は受信したパケットに含まれるリモートデバイスの RPA を解決します。RPA の解決により、アプリケーションに通知するイベントに含まれる、リモートデバイスのアドレスは Identity Address となります。

以下に 1-5 までのサンプルを示します。ローカルデバイスがリモートデバイスの RPA を解決する場合、Resolving List の Identity Address および IRK がリモートデバイスの Identity Address および IRK と一致する必要があります。

```
/* 省略  */
static st_ble_abs_scan_phy_param_t gs_phy_param_1M =
{
    .fast_intv                = 0x0200,
    .slow_intv                = 0x0800,
    .fast_window              = 0x0100,
    .slow_window              = 0x0100,
    .scan_type                = BLE_GAP_SCAN_PASSIVE,
};
static st_ble_abs_scan_param_t gs_scan_param =
{
    .p_phy_param_1M           = &gs_phy_param_1M,
    .p_phy_param_coded        = NULL,
    .p_filter_data            = NULL,
    .fast_period              = 0x0100,
    .slow_period              = 0x0000,
    .filter_data_length       = 0,
    .dev_filter               = BLE_ABS_SCAN_ALL_STATIC,
    .filter_dups              = BLE_GAP_SCAN_FILT_DUPLIC_DISABLE,
};
static st_ble_abs_conn_phy_param_t gs_conn_phy_param =
{
    .conn_intv = 0x0130,
    .conn_latency = 0x0000,
```

```
    .sup_to = 0x03BB,
};
static st_ble_dev_addr_t gs_conn_bd_addr;
static st_ble_abs_conn_param_t gs_conn_param =
{
    .p_conn_1M = &gs_conn_phy_param,
    .p_addr = &gs_conn_bd_addr, /**< Set BD address of connecting device. */
    .filter = BLE_ABS_CONN_USE_ADDR_STATIC,
    .conn_to = 5,
};
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   = BLE_GAP_KEY_DIST_ENCKEY | BLE_GAP_KEY_DIST_IDKEY,
    .max_key_size   = 16,
};
st_ble_dev_addr_t r_id_addr;
/* 省略  */

void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_data)
{
    /* 省略  */
    switch(event_type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            R_BLE_VS_GetBdAddr(BLE_VS_ADDR_AREA_REG, BLE_GAP_ADDR_RAND);
        } break;
        case BLE_GAP_EVENT_RPA_EN_COMP:
        {
            R_BLE_ABS_StartScan(&gs_scan_param);
        } break;
        case BLE_GAP_EVENT_ADV_REPT_IND:
        {
            st_ble_gap_adv_rept_evt_t *p_adv_rept_param = (st_ble_gap_adv_rept_evt_t *)p_data->p_param;
            st_ble_gap_ext_adv_rept_t *p_ext_adv_rept_param = (st_ble_gap_ext_adv_rept_t *)p_adv_rept_param-
>param.p_ext_adv_rpt;
            gs_conn_param.p_addr->type = p_ext_adv_rept_param->addr_type;
            memcpy(gs_conn_param.p_addr->addr, p_ext_adv_rept_param->p_addr, BLE_BD_ADDR_LEN);
            R_BLE_GAP_StopScan();
        } break;
        case BLE_GAP_EVENT_SCAN_OFF:
        {
            R_BLE_ABS_CreateConn(&gs_conn_param);
        } break;
        case BLE_GAP_EVENT_CONN_IND:
        {
            st_ble_gap_conn_evt_t *p_gap_conn_evt_param =
                (st_ble_gap_conn_evt_t *)p_data->p_param;
            R_BLE_ABS_StartAuth(p_gap_conn_evt_param->conn_hdl);
        } break;
        case BLE_GAP_EVENT_PEER_KEY_INFO:
        {
            st_ble_gap_peer_key_info_evt_t *p_peer_key_info_evt_param =
                (st_ble_gap_peer_key_info_evt_t *)p_data->p_param;
            st_ble_gap_key_dist_t * key_info;
            st_ble_gap_rslv_list_key_set_t  key_set;
            key_info = p_peer_key_info_evt_param->key_ex_param.p_keys_info;
            R_BLE_CLI_Printf("keys : 0x%02x¥n", p_peer_key_info_evt_param->key_ex_param.keys);
            if(0 != (BLE_GAP_KEY_DIST_IDKEY & p_peer_key_info_evt_param->key_ex_param.keys))
            {
                /* Add remote address & irk to the resolving list. */
                memcpy(key_set.remote_irk, key_info->id_info, BLE_GAP_IRK_SIZE);
                key_set.local_irk_type = BLE_GAP_RL_LOC_KEY_REGISTERED;
                memcpy(r_id_addr.addr, &key_info->id_addr_info[1], BLE_BD_ADDR_LEN);
                r_id_addr.type = key_info->id_addr_info[0];
                R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV, &r_id_addr, &key_set, 1);
            }
        } break;
        case BLE_GAP_EVENT_RSLV_LIST_CONF_COMP :
        {
            st_ble_gap_rslv_list_conf_evt_t * p_rslv_list_conf;
```

```
            p_rslv_list_conf = (st_ble_gap_rslv_list_conf_evt_t *)p_data->p_param;
            if(BLE_GAP_LIST_ADD_DEV == p_rslv_list_conf->op_code)
            {
                uint8_t priv_mode;
                priv_mode = BLE_GAP_NET_PRIV_MODE ;
                /* Set Network Privacy Mode. */
                R_BLE_GAP_SetPrivMode(&r_id_addr, &priv_mode, 1);
            }
        }
        break;
        /* 省略  */
    }
}

/* Vendor Specific callback function */
void vs_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_data)
{
    switch(type)
    {
        case BLE_VS_EVENT_GET_ADDR_COMP:
        {
            /* Enable RPA. */
            R_BLE_GAP_EnableRpa(BLE_GAP_RPA_ENABLED);
        } break;
    }
}
/* 省略  */
```

コード 9-9　リモートデバイスのアドレス解決(1)

R_BLE_ABS_SetLocPrivacy を使った場合のサンプルを以下に示します。

```
/* 省略  */
#include "sec_data/r_ble_sec_data.h"
typedef struct
{
    /* identity address */
    st_ble_dev_addr_t idaddr[BLE_CFG_NUM_BOND + 1];
    /* local & remote IRK set */
    st_ble_gap_rslv_list_key_set_t key_set[BLE_CFG_NUM_BOND + 1];
    /* the number of identity info stored in Data Flash */
    uint8_t gs_bond_cnt;
}  st_ble_app_idinfo_t;
static st_ble_app_idinfo_t gs_idinfo;
static st_ble_gap_rslv_list_key_set_t g_ble_peer_dummy_irk;
static st_ble_abs_scan_phy_param_t gs_phy_param_1M =
{
    .fast_intv                = 0x0200,
    .slow_intv                = 0x0800,
    .fast_window              = 0x0100,
    .slow_window              = 0x0100,
    .scan_type                = BLE_GAP_SCAN_PASSIVE,
};
static st_ble_abs_scan_param_t gs_scan_param =
{
    .p_phy_param_1M           = &gs_phy_param_1M,
    .p_phy_param_coded        = NULL,
    .p_filter_data            = NULL,
    .fast_period              = 0x0100,
    .slow_period              = 0x0000,
    .filter_data_length       = 0,
    .dev_filter               = BLE_ABS_SCAN_ALL_STATIC,
    .filter_dups              = BLE_GAP_SCAN_FILT_DUPLIC_DISABLE,
};
static st_ble_abs_conn_phy_param_t gs_conn_phy_param =
{
    .conn_intv = 0x0130,
    .conn_latency = 0x0000,
    .sup_to = 0x03BB,
};
static st_ble_dev_addr_t gs_conn_bd_addr;
static st_ble_abs_conn_param_t gs_conn_param =
{
    .p_conn_1M = &gs_conn_phy_param,
```

```
    .p_addr = &gs_conn_bd_addr, /**< Set BD address of connecting device. */
    .filter = BLE_ABS_CONN_USE_ADDR_STATIC,
    .conn_to = 5,
};
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   = BLE_GAP_KEY_DIST_ENCKEY | BLE_GAP_KEY_DIST_IDKEY,
    .max_key_size   = 16,
};
static void ble_app_start_scan(void)
{
    R_BLE_ABS_StartScan(&gs_scan_param);
}
static void ble_app_conn_set_event(void)
{
    R_BLE_ABS_CreateConn(&gs_conn_param);
}
/* 省略  */

void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_data)
{
    /* 省略  */
    switch(event_type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            R_BLE_VS_GetBdAddr(BLE_VS_ADDR_AREA_REG, BLE_GAP_ADDR_RAND);
        } break;
        case BLE_GAP_EVENT_RPA_EN_COMP:
        {
            if((0 != gs_idinfo.gs_bond_cnt))
            {
                /* register remote address & irk */
                R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV,
                                       gs_idinfo.idaddr,
                                       gs_idinfo.key_set,
                                       gs_idinfo.gs_bond_cnt);
                R_BLE_SetEvent(ble_app_start_scan);
            }
            else
            {
                ble_app_start_scan();
            }
        } break;
        case BLE_GAP_EVENT_ADV_REPT_IND:
        {
            st_ble_gap_adv_rept_evt_t *p_adv_rept_param = (st_ble_gap_adv_rept_evt_t *)p_data->p_param;
            st_ble_gap_ext_adv_rept_t *p_ext_adv_rept_param = (st_ble_gap_ext_adv_rept_t *)p_adv_rept_param-
>param.p_ext_adv_rpt;
            gs_conn_param.p_addr->type = p_ext_adv_rept_param->addr_type;
            memcpy(gs_conn_param.p_addr->addr, p_ext_adv_rept_param->p_addr, BLE_BD_ADDR_LEN);
            R_BLE_GAP_StopScan();
        } break;
        case BLE_GAP_EVENT_SCAN_OFF:
        {
            st_ble_gap_rslv_list_key_set_t * p_key_set;
            uint8_t i;
            uint8_t peer_addr_type;
            peer_addr_type = gs_conn_param.p_addr->type;
            gs_conn_param.p_addr->type = gs_conn_param.p_addr->type % 2;
            if(BLE_GAP_ADDR_RAND < peer_addr_type)
            {
                /* Local device can resolve the remote device address. */
                R_BLE_SECD_GetIdInfo(gs_idinfo.idaddr, gs_idinfo.key_set, &gs_idinfo.gs_bond_cnt);
                R_BLE_CLI_Printf("Remote IRK count :0x%02x ¥n", gs_idinfo.gs_bond_cnt);
                p_key_set = NULL;
                if(0 != gs_idinfo.gs_bond_cnt)
                {
                    for(i=0; i<BLE_CFG_NUM_BOND + 1; i++)
                    {
```

```
                        if(0 == memcmp(gs_idinfo.idaddr[i].addr, gs_conn_param.p_addr,
sizeof(st_ble_dev_addr_t)))
                        {
                            p_key_set = &gs_idinfo.key_set[i];
                            break;
                        }
                    }
                }
                if(NULL == p_key_set)
                {
                    p_key_set = &g_ble_peer_dummy_irk;
                }
                p_key_set->local_irk_type = BLE_GAP_RL_LOC_KEY_ALL_ZERO;
                R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV, gs_conn_param.p_addr, p_key_set, 1);
                R_BLE_SetEvent(ble_app_conn_set_event);
            }
            else
            {
                /* Local device can't resolve the remote device address. */
                ble_app_conn_set_event();
            }
        } break;
        case BLE_GAP_EVENT_PEER_KEY_INFO:
        {
            st_ble_gap_peer_key_info_evt_t *p_peer_key_info_evt_param =
                (st_ble_gap_peer_key_info_evt_t *)p_data->p_param;
            st_ble_gap_key_dist_t * key_info;
            st_ble_gap_rslv_list_key_set_t  key_set;
            key_info = p_peer_key_info_evt_param->key_ex_param.p_keys_info;
            R_BLE_CLI_Printf("keys : 0x%02x¥n", p_peer_key_info_evt_param->key_ex_param.keys);
            if(0 != (BLE_GAP_KEY_DIST_IDKEY & p_peer_key_info_evt_param->key_ex_param.keys))
            {
                /* Add remote address & irk to the resolving list. */
                st_ble_dev_addr_t r_id_addr;
                memcpy(key_set.remote_irk, key_info->id_info, BLE_GAP_IRK_SIZE);
                key_set.local_irk_type = BLE_GAP_RL_LOC_KEY_REGISTERED;
                memcpy(r_id_addr.addr, &key_info->id_addr_info[1], BLE_BD_ADDR_LEN);
                r_id_addr.type = key_info->id_addr_info[0];
                R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV, &r_id_addr, &key_set, 1);
            }
        } break;
        /* 省略  */
    }
}
static void vs_cb(uint16_t type, ble_status_t result, st_ble_vs_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_VS_EVENT_GET_ADDR_COMP:
        {
            /* Get remote irk & identity address from Data Flash. */
            R_BLE_SECD_GetIdInfo(gs_idinfo.idaddr, gs_idinfo.key_set, &gs_idinfo.gs_bond_cnt);
            R_BLE_CLI_Printf("Remote IRK count :0x%02x ¥n", gs_idinfo.gs_bond_cnt);
            if((0 != gs_idinfo.gs_bond_cnt))
            {
                /* Already create local irk and have remote irk. */
                R_BLE_GAP_EnableRpa(BLE_GAP_RPA_ENABLED);
            }
            else
            {
                /* Initial state or remote device did not distribute irk. */
                uint8_t irk[BLE_GAP_IRK_SIZE];
                uint8_t irk_check[BLE_GAP_IRK_SIZE];
                uint8_t * p_irk;
                st_ble_dev_addr_t idaddr;
                ble_status_t retval;
                retval = R_BLE_SECD_ReadLocInfo(&idaddr, irk, NULL);
                memset(irk_check, 0x00, BLE_GAP_IRK_SIZE);
                p_irk = NULL;
                if((BLE_SUCCESS == retval) && (0 != memcmp(irk_check, irk, BLE_GAP_IRK_SIZE)))
                {
                    p_irk = irk;
                }
                R_BLE_ABS_SetLocPrivacy(p_irk, BLE_ABS_PRIV_NET_STATIC_IDADDR);
            }
```

```
        } break;
    }
}
static void disc_comp_cb(uint16_t conn_hdl)
{
    R_BLE_CLI_Printf("disc finished¥n");
    R_BLE_ABS_StartAuth(conn_hdl);
    return;
}
/* 省略 */
```

コード 9-10　リモートデバイスのアドレス解決(2)

RPA の解決後は、Identity Address を使って接続やホワイトリストの登録などをする必要があります。

Bleutooth LE Protocol Stack を再起動した場合、デバイスに保存した鍵を
R_BLE_GAP_ConfRslvList
により、Resolving List に再登録する必要があります。
「9.2.3 保存した鍵の再設定」をご参照ください。

9.4.2.1　ローカルデバイスが RPA を使わない場合

ローカルデバイスがリモートデバイスのアドレス解決を行い、自身は RPA を生成せず、パブリックアドレス、または、スタティックアドレスを使う場合、app_main.c を下記の方針に従って変更します。

1. g_ble_peer_dummy_irk の変更
コード 9-11 のように R_BLE_ABS_Init()をコールする前に抽象 API 内で定義されている g_ble_peer_dummy_irk の local_irk_type を BLE_GAP_RL_LOC_KEY_ALL_ZERO に設定します。

```
/******************************************************************************
 User global variables
******************************************************************************/
/* Start user code for global variables. Do not edit comment generated here */
extern st_ble_gap_rslv_list_key_set_t g_ble_peer_dummy_irk;
/* End user code. Do not edit comment generated here */
…

static ble_status_t ble_init(void)
{
    ble_status_t status;

/* Start user code for global value initialization. Do not edit comment generated here */
    g_ble_peer_dummy_irk.local_irk_type =  BLE_GAP_RL_LOC_KEY_ALL_ZERO;

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

    /* Initialize the Low Power Control function */
    R_BLE_LPC_Init();

    /* Initialize Timer Library */
    R_BLE_TIMER_Init();

    /* Initialize host stack */
    status = R_BLE_ABS_Init(&gs_abs_init_param);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }
```

コード 9-11　g_ble_peer_dummy_irk の変更のサンプル

2. R_BLE_SECD_GetIdInfo()で取得した Resolving List に設定する値の変更

R_BLE_SECD_GetIdInfo()により、Resolving List に設定する値のセットを取得する場合、
第 2 パラメータで取得した st_ble_gap_rslv_list_key_set_t 型配列の local_irk_type を
BLE_GAP_RL_LOC_KEY_ALL_ZERO に設定してから R_BLE_GAP_ConfRslvList()をコールします。

```
static void ble_app_set_resolving_list(void)
{
    st_ble_dev_addr_t idaddr[BLE_CFG_NUM_BOND + 1] = {0};
    st_ble_gap_rslv_list_key_set_t key_set[BLE_CFG_NUM_BOND + 1] = {0};
    uint8_t cnt = 0;
    uint8_t i;

    /* Get remote irk & identity address from Data Flash. */
    R_BLE_SECD_GetIdInfo(idaddr, key_set, &cnt);

    if(0 !=cnt)
    {
        for(i=0; i<cnt; i++)
        {
           key_set[i].local_irk_type = BLE_GAP_RL_LOC_KEY_ALL_ZERO;
        }

        R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV, idaddr, key_set, cnt);
```

コード 9-12　R_BLE_SECD_GetIdInfo()で取得した Resolving List に設定する値の変更のサンプル

10. プロファイル・サービス

Bluetooth LE 通信におけるプロファイルは、アプリケーションで共有するサービスと通信プロトコルを規定することによってデバイス間の相互コネクション性を確保するための仕組みです。プロファイルを用いたデータ通信は、デバイス間で GATT データベースと呼ばれる共通のデータ構造にアクセスすることで実現します。図 10-1 に示すように、GATT データベースは複数のサービスとそれらに含まれるキャラクタリスティックから構成されます。サービスはプロファイルの機能を実現する 1 つ以上のキャラクタリスティックで構成され、キャラクタリスティックにはデータ構造とアクセス手順が定義されています。キャラクタリスティックへアクセスするための手順を GATT プロシージャと呼び、この手順に従ってデータの送受信が行われます。

ユーザの使用するプロファイルは QE for BLE を使用して設計することが可能です。QE for BLE を使用したプロファイルの設計の方法は、「Bluetooth Low Energy プロファイル開発者ガイド (R01AN6459)」を参照してください。

ここでは Renesas が提供するプロファイルとサービスの紹介と各 GATT プロシージャの API の解説及びその利用方法について説明します。

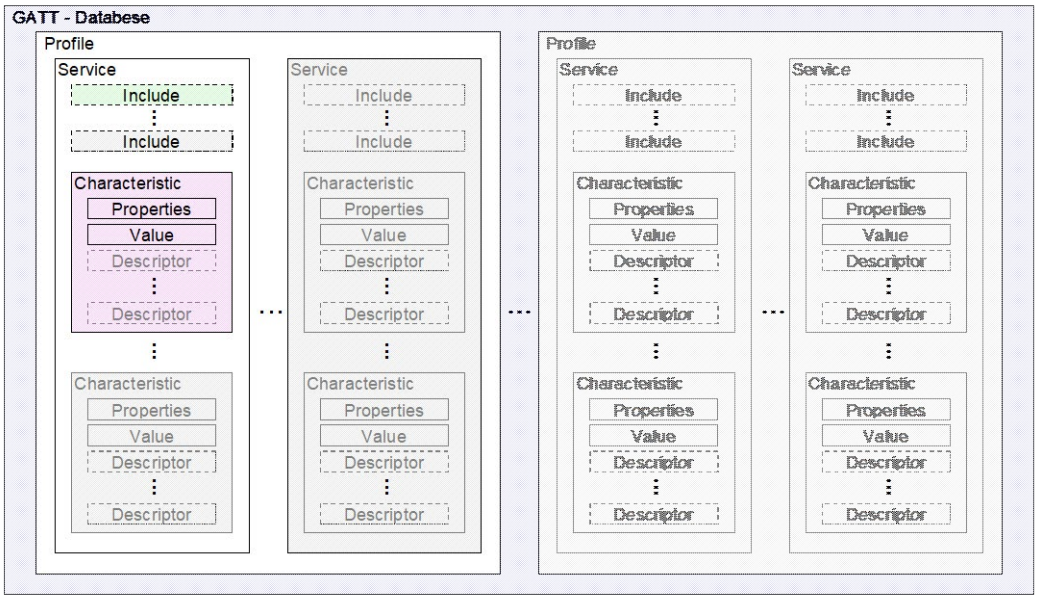


図 10-1 GATT データベースの構成

10.1 標準プロファイル及びサービス

QE for BLE を使用することで標準プロファイル及び標準サービスをアプリケーションで利用することができます。RX23W は表 10.1 に示す標準プロファイルと標準サービスをサポートします。表 10.1 では各標準サービスを構成するキャラクタリスティックの一覧を示します。

表 10.1 RX23W がサポートするプロファイル

用途・分野	プロファイル	サービス			
ヘルスケア向け プロファイル	Blood Pressure Profile	BLS	DIS		
	Health Thermometer Profile	HTS	DIS		
	Heart Rate Profile	HRS	DIS		
	Glucose Profile	GLS	DIS		
	Pulse Oximeter Profile	PLXS	DIS	BAS	CTS
		BMS			
	Continuous Glucose Monitoring Profile	CGMS	DIS	BMS	
	Reconnection Configuration Profile	RCS	BMS		
	Insulin Delivery Profile	IDS	DIS	BAS	CTS
		BMS	IAS		
スポーツ・ フィットネス向け プロファイル	Cycling Power Profile	CPS	DIS	BAS	
	Cycling Speed and Cadence Profile	CSCS	DIS		
	Running Speed and Cadence Profile	RSCS	DIS		
	Location and Navigation Profile	LNS	DIS	BAS	
	Weight Scale Profile	WSS	BCS	DIS	BAS
		CTS	UDS		
	Fitness Machine Profile	FTMS	DIS	UDS	
	Environmental Sensing Profile	ESS	DIS	BAS	
無線タグ向け プロファイル	Find Me Profile	IAS			
	Proximity Profile	IAS	LLS	TPS	
スマートフォン向け プロファイル	Alert Notification Profile	ANS			
	Phone Alert Status Profile	PASS			
	Time Profile	CTS	NDCS	RTUS	
HID(Human Interface Device)向け プロファイル	HID over GATT Profile	HIDS	DIS	BAS	
	Scan Parameters Profile	SCPS			
産業機器向け プロファイル	Automation IO Profile	AIOS			

表 10.2 標準サービスの構成一覧

サービス名	キャラクタリスティック名	GATT プロシージャ
Alert Notification Service **ANS**	Supported New Alert Category	Read
	New Alert	Notify
	Supported Unread Alert Category	Read
	Unread Alert Status	Notify
Automation IO Service **AIOS**	Digital 0	Read, Write, WriteWithoutResponse, Notify
	Digital 1	Read, Write, WriteWithoutResponse, Notify
	Analog 0	Read, Write, WriteWithoutResponse, Notify
	Analog 1	Read, Write, WriteWithoutResponse, Notify
	Aggregate	Read, Notify
Battery Service **BAS**	Battery Level	Read, Notify
Blood Pressure Service **BLS**	Blood Pressure Measurement	Indicate
	Intermediate Cuff Pressure	Notify
	Blood Pressure Feature	Read, Indicate
Body Composition Service **BCS**	Body Composition Feature	Read
	Body Composition Measurement	Indicate
Bond Management Service **BMS**	Bond Management Control Point	Write, ReliableWrite
	Bond Management Feature	Read, Indicate
Continuous Glucose Monitoring Service **CGMS**	CGM Measurement	Notify
	CGM Feature	Read, Indicate
	CGM Status	Read
	CGM Session Start Time	Read, Write
	CGM Session Run Time	Read
	Record Access Control Point	Write, Indicate
	CGM Specific Ops Control Point	Write, Indicate
Current Time Service **CTS**	Current Time	Read, Write, Notify
	Local Time Information	Read, Write
	Reference Time Information	Read
Cycling Power Service **CPS**	Cycling Power Measurement	Notify, Broadcast
	Cycling Power Feature	Read
	Sensor Location	Read
	Cycling Power Vector	Notify
	Cycling Power Control Point	Write, Indicate
Cycling Speed and Cadence Service **CSCS**	CSC Measurement	Notify
	CSC Feature	Read
	Sensor Location	Read
	SC Control Point	Write, Indicate

サービス名	キャラクタリスティック名	GATT プロシージャ
Device Information Service **DIS**	Manufacturer Name String	Read
	Model Number String	Read
	Serial Number String	Read
	Hardware Revision String	Read
	Firmware Revision String	Read
	Software Revision String	Read
	System ID	Read
	IEEE 11073-20601 Regulatory Certification Data List	Read
	PnP ID	Read
Environmental Sensing Service **ESS**	Descriptor Value Changed	Indicate
	Temperature 0	Read, Notify
	Temperature 1	Read, Notify
	Elevation 0	Read, Notify
	Elevation 1	Read, Notify
Fitness Machine Service **FTMS**	Fitness Machine Feature	Read
	Treadmill Data	Notify
	Cross Trainer Data	Notify
	Step Climber Data	Notify
	Stair Climber Data	Notify
	Rower Data	Notify
	Indoor Bike Data	Notify
	Training Status	Read, Notify
	Supported Speed Range	Read
	Supported Inclination Range	Read
	Supported Resistance Level Range	Read
	Supported Power Range	Read
	Supported Heart Rate Range	Read
	Fitness Machine Control Point	Write, Indicate
	Fitness Machine Status	Notify
GAP Service **GAP**	Device Name	Read, Write
	Appearance	Read
	Peripheral Preferred Connection Parameters	Read
	Central Address Resolution	Read
	Resolvable Private Address Only	Read
GATT Service **GATT**	Service Changed	Indicate
Glucose Service **GLS**	Glucose Measurement	Notify
	Glucose Measurement Context	Notify
	Glucose Feature	Read, Indicate
	Record Access Control Point	Write, Indicate

サービス名	キャラクタリスティック名	GATT プロシージャ
Health Thermometer Service **HTS**	Temperature Measurement	Indicate
	Temperature Type	Read
	Intermediate Temperature	Notify
	Measurement Interval	Read, Write, Indicate
Heart Rate Service **HRS**	Heart Rate Measurement	Notify
	Body Sensor Location	Read
	Heart Rate Control Point	Write
Human Interface Device Service **HIDS**	Protocol Mode	Read, WriteWithoutResponse
	Report	Read, Write, WriteWithoutResponse, Notify
	Report Map	Read
	Boot Keyboard Input Report	Read, Write, Notify
	Boot Keyboard Output Report	Read, Write, WriteWithoutResponse
	Boot Mouse Input Report	Read, Write, Notify
	HID Information	Read
	HID Control Point	WriteWithoutResponse
Immediate Alert Service **IAS**	Alert Level	WriteWithoutResponse
Insulin Delivery Service **IDS**	IDD Status Changed	Read, Indicate
	IDD Status	Read, Indicate
	IDD Annunciation Status	Read, Indicate
	IDD Features	Read, Indicate
	IDD Status Reader Control Point	Write, Indicate
	IDD Command Control Point	Write, Indicate
	IDD Command Data	InformativeText, Notify
	IDD Record Access Control Point	Write, Indicate
	IDD History Data	InformativeText, Notify
Link Loss Service **LLS**	Alert Level	Read, Write
Location and Navigation Service **LNS**	LN Feature	Read
	Location and Speed	Notify
	Position Quality	Read
	LN Control Point	Write, Indicate
	Navigation	Notify
Next DST Change Service **NDCS**	Time with DST	Read
Object Transfer Service **OTS**	OTS Feature	Read
	Object Name	Read, Write
	Object Type	Read
	Object Size	Read
	Object First-Created	Read, Write

サービス名	キャラクタリスティック名	GATT プロシージャ
	Object Last-Modified	Read, Write
	Object ID	Read
	Object Properties	Read, Write
	Object Action Control Point	Write, Indicate
	Object List Control Point	Write, Indicate
	Object List Filter 0	Read, Write
	Object List Filter 1	Read, Write
	Object List Filter 2	Read, Write
	Object Changed	Indicate
Phone Alert Status Service **PASS**	Alert Status	Read, Notify
	Ringer Setting	Read, Notify
	Ringer Control point	WriteWithoutResponse
Pulse Oximeter Service **PLXS**	PLX Spot-Check Measurement	Indicate
	PLX Continuous Measurement	Notify
	PLX Features	Read, Indicate
	Record Access Control Point	Write, Indicate
Reconnection Configuration Service **RCS**	RC Feature	Read, Indicate
	RC Settings	Read, Notify
	Reconnection Configuration Control Point	Write, Indicate
Reference Time Update Service **RTUS**	Time Update Control Point	WriteWithoutResponse
	Time Update State	Read
Running Speed and Cadence Service **RSCS**	RSC Measurement	Notify
	RSC Feature	Read
	Sensor Location	Read
	SC Control Point	Write, Indicate
Scan Parameters Service **SCPS**	Scan Interval Window	WriteWithoutResponse
	Scan Refresh	Notify
Tx Power Service **TPS**	Tx Power Level	Read
User Data Service **UDS**	First Name	Read, Write
	Last Name	Read, Write
	Email Address	Read, Write
	Age	Read, Write
	Date of Birth	Read, Write
	Gender	Read, Write
	Weight	Read, Write
	Height	Read, Write
	VO2 Max	Read, Write
	Heart Rate Max	Read, Write

サービス名	キャラクタリスティック名	GATT プロシージャ
	Resting Heart Rate	Read, Write
	Maximum Recommended Heart Rate	Read, Write
	Aerobic Threshold	Read, Write
	Anaerobic Threshold	Read, Write
	Sport Type for Aerobic and Anaerobic Thresholds	Read, Write
	Date of Threshold Assessment	Read, Write
	Waist Circumference	Read, Write
	Hip Circumference	Read, Write
	Fat Burn Heart Rate Lower Limit	Read, Write
	Fat Burn Heart Rate Upper Limit	Read, Write
	Aerobic Heart Rate Lower Limit	Read, Write
	Aerobic Heart Rate Upper Limit	Read, Write
	Anaerobic Heart Rate Lower Limit	Read, Write
	Anaerobic Heart Rate Upper Limit	Read, Write
	Five Zone Heart Rate Limits	Read, Write
	Three Zone Heart Rate Limits	Read, Write
	Two Zone Heart Rate Limit	Read, Write
	Database Change Increment	Read, Write, Notify
	User Index	Read
	User Control Point	Write, Indicate
	Language	Read, Write
	Registered User	Read, Write
	Preferred Units	Read, Write
	High Resolution Height	Read, Write
	Middle Name	Read, Write
	Stride Length	Read, Write
	Handedness	Read, Write
	Device Wearing Position	Read, Write
	Four Zone Heart Rate Limits	Read, Write
	High Intensity Exercise Threshold	Read, Write
	Activity Goal	Read, Write
	Sedentary Interval Notification	Read, Write
	Caloric Intake	Read, Write
Weight Scale Service **WSS**	Weight Scale Feature	Read
	Weight Measurement	Indicate

10.2 GATT プロシージャの API

QE for BLE ではキャラクタリスティックに設定される GATT プロシージャに応じて API が生成されます。ここでは QE for BLE から設定できる GATT プロシージャをそれぞれ実現するために API の使用方法について説明します。

以降の説明では QE for BLE でサービスの略称に「XXX」、キャラクタリスティックの略称に「YYY」を設定してコード生成を行った場合の関数名及びイベント名を使用して説明します。

10.2.1 Read 動作

Read 動作は GATT クライアントが GATT サーバの GATT データベースに設定されているデータを確認する動作です。GATT サーバ側の設定やステータスを確認する場合などで使用してください。

GATT サーバ：
GATT サーバのデバイスが「Read Request」を受信した場合、Bluetooth LE Protocol Stack が GATT データベースに設定されている値を「Read Response」で送信します。イベント「BLE_XXX_EVENT_YYY_READ_REQ」は「Read Request」を受信した後、「Read Response」で送信するデータを決定する前のイベントです。送信するデータを変更する場合は、関数「R_BLE_XXX_SetYYY()」を使用して GATT データベースを変更して下さい。また、関数「R_BLE_GATTS_SetErrRsp()」を使用することでエラーを送信することも可能です。

GATT クライアント：
アプリケーションで関数「R_BLE_XXX_ReadYYY()」を使用することで「Read Request」を送信します。「Read Response」で受信したデータはイベント「BLE_XXX_EVENT_YYY_READ_RSP」によってアプリケーションに通知されます。この時に通知される値は Bluetooth LE Protocol Stack によって decode 関数を使用してデコードされているため、QE for BLE の Field で作成した構造体の形式になります。また、イベント「BLE_XXX_EVENT_YYY_READ_RSP」が通知された時点で Read 動作は完了しているため、次の動作を行うための関数を呼び出すことができます。

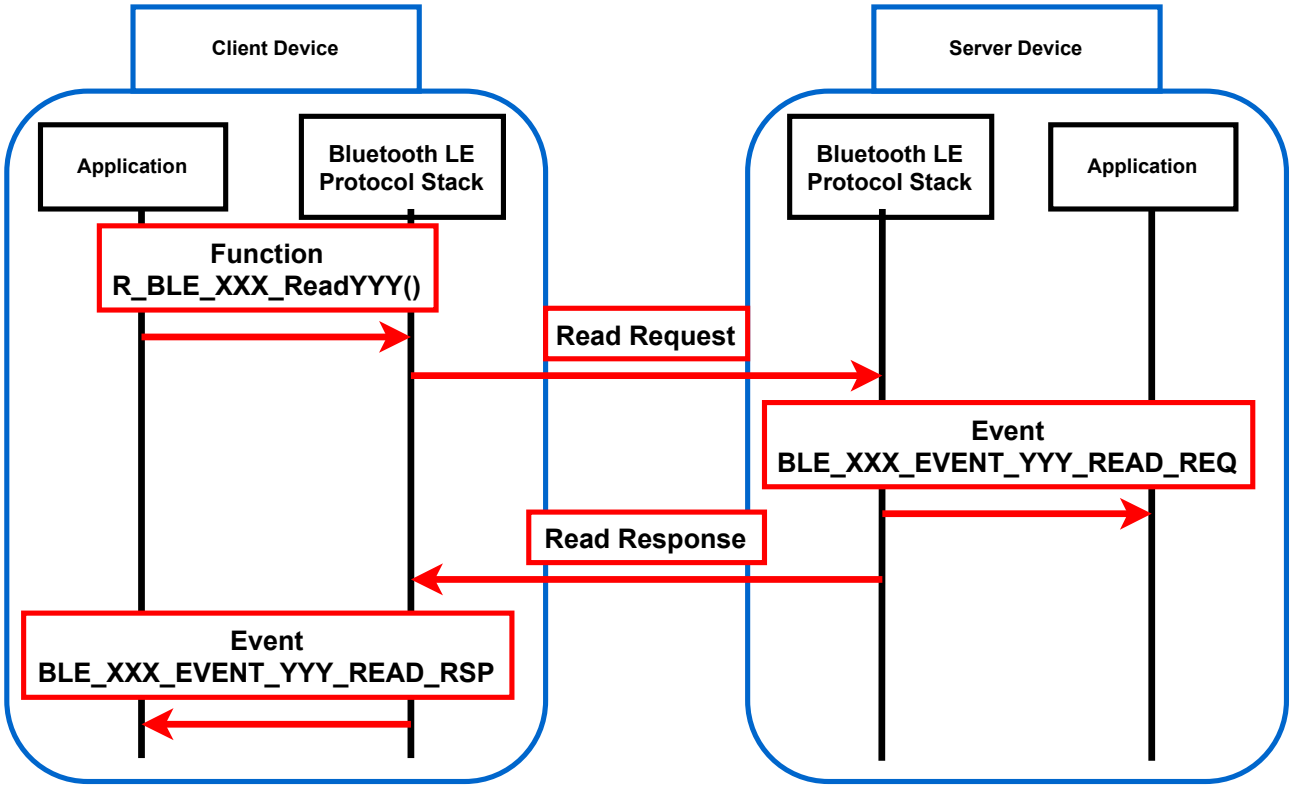


図 10-2　Read 動作の実行手順

10.2.2 Write 動作

Write 動作は GATT クライアントからデータを送信して GATT サーバの GATT データベースを変更する動作です。送信したデータが GATT データベースに反映されたかを GATT サーバからの応答で確認することができます。GATT サーバの設定を変更する場合などで使用してください。

GATT サーバ：
「Write Request」で受信したデータはイベント「BLE_XXX_EVENT_YYY_WRITE_REQ」及び「BLE_XXX_EVENT_YYY_WRITE_COMP」によってアプリケーションに通知されます。この時に通知される値は Bluetooth LE Protocol Stack によって decode 関数を使用してデコードされているため、QE for BLE の Field で作成した構造体の形式になります。イベント「BLE_XXX_EVENT_YYY_WRITE_REQ」は「Write Request」で受信したデータを GATT データベースに書き込まれる前に確認するイベントです。不正なデータを受信した場合、関数「R_BLE_GATTS_SetErrRsp()」を使用することでエラーを送信するとともに GATT データベースに反映されなくなります。エラーを送信しない場合、Bluetooth LE Protocol Stack が「Write Response」を送信するため、アプリケーションに応答するための処理を追加する必要はありません。イベント「BLE_XXX_EVENT_YYY_WRITE_COMP」は「Write Request」で受信したデータが GATT データベースに反映され、「Write Response」を送信した後のイベントです。GATT データベースを直接参照する処理や「Write Request」で受信したデータに対応する処理を追加してください。

GATT クライアント：
アプリケーションで関数「R_BLE_XXX_WriteYYY()」を使用することで「Write Request」を送信することができます。イベント「BLE_XXX_EVENT_YYY_WRITE_RSP」を確認することで Write 動作の結果を確認することができます。また、イベント「BLE_XXX_EVENT_YYY_WRITE_RSP」が通知された時点で Write 動作は完了しているため、次の動作を行うための関数を呼び出すことができます。

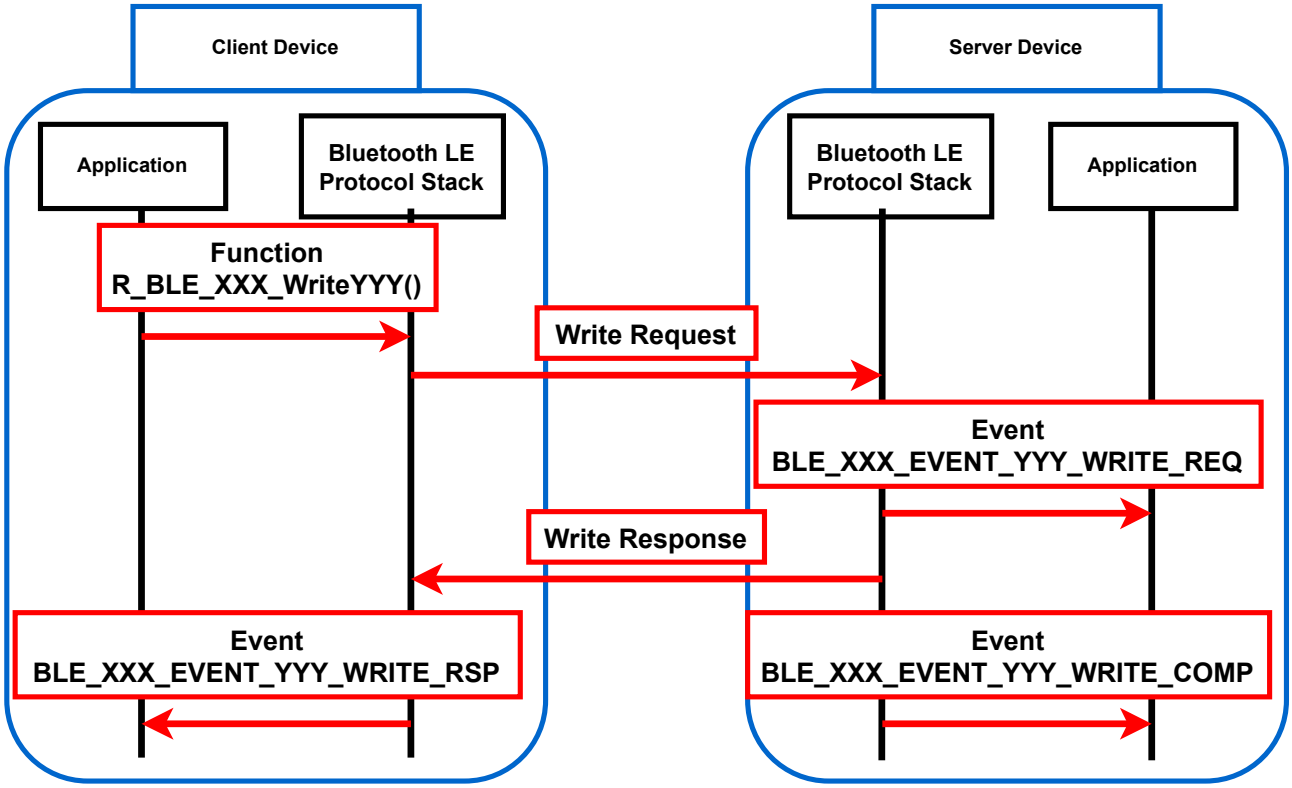


図 10-3　Write 動作の実行手順

10.2.3 WriteWithoutResponse 動作

WriteWithoutResponse 動作は GATT クライアントからデータを送信して GATT サーバの GATT データベースを変更する動作です。GATT サーバからの応答がないため、GATT クライアントからデータの連続送信や GATT サーバ側デバイスの低消費電力化が可能な一方、GATT クライアントが送信したデータが GATT データベースに反映されたことを確認することができません。デバイスの低消費電力化が必要な場合や、GATT クライアントから連続してデータを送信する必要がある場合に使用してください。

GATT サーバ：
「Write Command」で受信したデータはイベント「BLE_XXX_EVENT_YYY_WRITE_CMD」によってアプリケーションに通知されます。この時に通知される値は Bluetooth LE Protocol Stack によって decode 関数を使用してデコードされているため、QE for BLE の Field で作成した構造体の形式になります。イベント「BLE_XXX_EVENT_YYY_WRITE_CMD」が通知された時点では、GATT データベースへの変更は反映されていないため、GATT データベースを直接参照する処理を追加しないでください。

GATT クライアント：
アプリケーションで関数「R_BLE_XXX_WriteWithoutResponseYYY()」を使用することで「Write Command」を送信することができます。関数「R_BLE_XXX_WriteWithoutResponseYYY()」を使用した時点で GATT クライアント側の WriteWithoutResponse 動作は完了しているため、次の動作を行うための関数を呼び出すことができます。

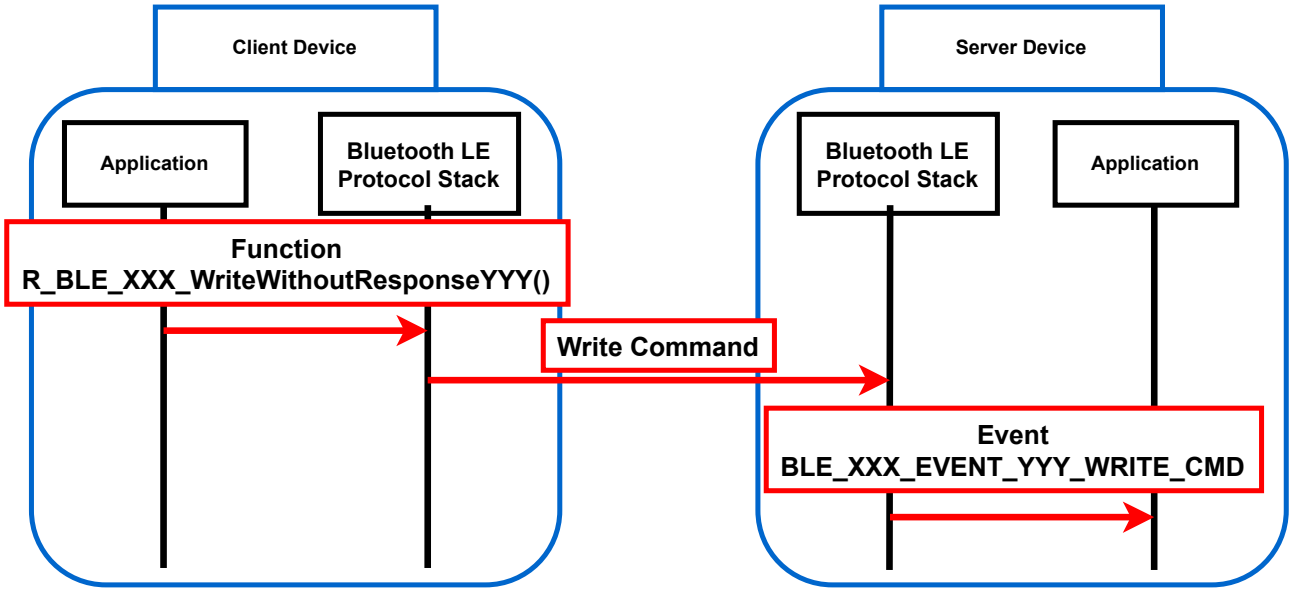


図 10-4　WriteWithoutResponse 動作の実行手順

10.2.4 Notification 動作

Notification 動作は GATT サーバから GATT クライアントにデータを送信する動作です。Notification 動作をする場合、CCCD がディスクリプタとして追加されている必要があります。また Notification 動作前に GATT クライアントは CCCD を適切な値に設定する必要があります。GATT クライアントからの応答がないため、GATT サーバからデータの連続送信が可能な一方、GATT サーバが送信したデータを GATT クライアントが受信したことを確認することができません。GATT サーバから連続してデータを送信する場合に使用してください。

GATT サーバ：
CCCD が適切な値に変更されたことを確認してから Notification 動作を行います。CCCD の書き込み完了時のイベント「BLE_XXX_EVENT_YYY_CLI_CNFG_WRITE_COMP」で「BLE_GATTS_CLI_CNFG_NOTIFICATION (0x0001)」が書き込まれていることを確認してください。その後、アプリケーションで関数「R_BLE_XXX_NotifyYYY()」を使用することで「Notification」が送信されます。CCCD の値が変更されていない場合、API「R_BLE_XXX_NotifyYYY()」を使用するとマクロ「BLE_ERR_INVALID_OPERATION」が返却され、GATT サーバから「Notification」は送信されません。関数「R_BLE_XXX_NotifyYYY()」を使用した時点で GATT サーバ側の Notification 動作は完了しているため、次の動作を行うための関数を呼び出すことができます。

GATT クライアント：
Notification 動作を行う前に、CCCD の値を適切な値に変更します。Notification 動作を行うキャラクタリスティックの CCCD に対して Write 動作を使用して「BLE_GATTS_CLI_CNFG_NOTIFICATION (0x0001)」を書き込んでください。「Notification」で受信したデータはイベント「BLE_XXX_EVENT_YYY_HDL_VAL_NTF」によってアプリケーションに通知されます。この時に通知される値は Bluetooth LE Protocol Stack によって decode 関数を使用してデコードされているため、QE for BLE の Field で作成した構造体の形式になります。

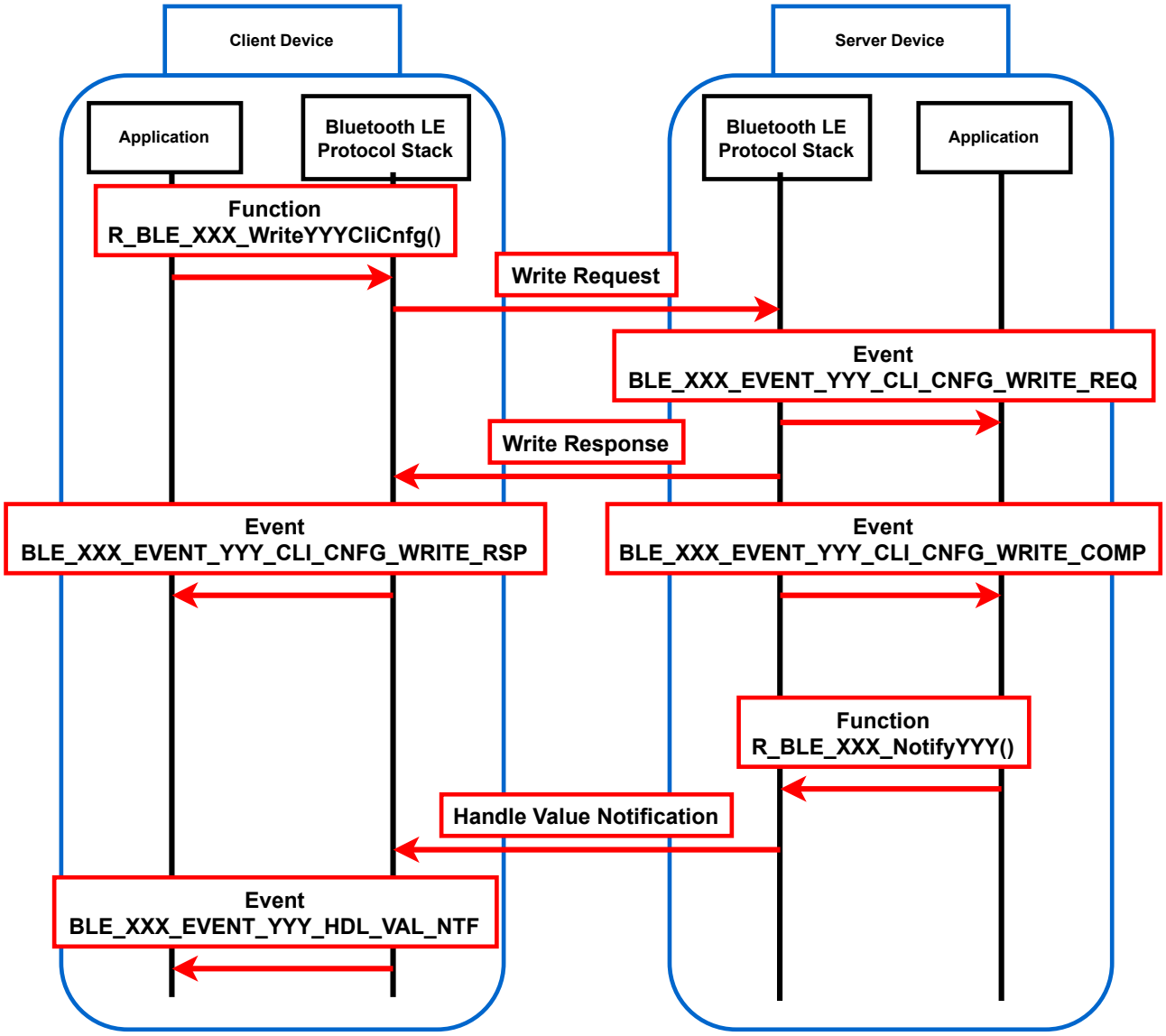


図 10-5　Notification 動作の実行手順

10.2.5 Indication 動作

Indication 動作は GATT サーバから GATT クライアントにデータを送信する動作です。Indication 動作をする場合、CCCD がディスクリプタとして追加されている必要があります。動作前に CCCD を適切な値に設定する必要があります。GATT サーバから送信したデータを GATT クライアントが受信したことを、GATT クライアントからの応答で確認することができます。

GATT サーバ：
CCCD が適切な値に変更されたことを確認してから Indication 動作を行います。CCCD の書き込み完了時のイベント「BLE_XXX_EVENT_YYY_CLI_CNFG_WRITE_COMP」で「BLE_GATTS_CLI_CNFG_INDICATION (0x0002)」が書き込まれていることを確認してください。その後、アプリケーションで関数「R_BLE_XXX_IndicateYYY()」を使用することで「Indication」が送信されます。CCCD の値が変更されていない場合、関数「R_BLE_XXX_IndicateYYY()」を使用するとマクロ「BLE_ERR_INVALID_OPERATION」が返却され、GATT サーバから「Indication」は送信されません。イベント「BLE_XXX_EVENT_YYY_HDL_VAL_CNF」が通知された時点で GATT サーバ側の Indication 動作は完了しているため、次の動作を行うための関数を呼び出すことができます。

GATT クライアント：
Indication 動作を行う前に、CCCD の値を適切な値に変更します。Indication 動作を行うキャラクタリスティックの CCCD に対して Write 動作を使用して「BLE_GATTS_CLI_CNFG_INDICATION (0x0002)」を書き込んでください。「Indication」で受信したデータはイベント「BLE_XXX_EVENT_YYY_HDL_VAL_IND」によってアプリケーションに通知されます。この時に通知される値は Bluetooth LE Protocol Stack によって decode 関数を使用してデコードされているため、QE for BLE の Field で作成した構造体の形式になります。またイベント「BLE_XXX_EVENT_YYY_HDL_VAL_IND」の後、Bluetooth LE Protocol Stack によって Confirmation は送信されるため、アプリケーションに応答するための処理を追加する必要はありません。

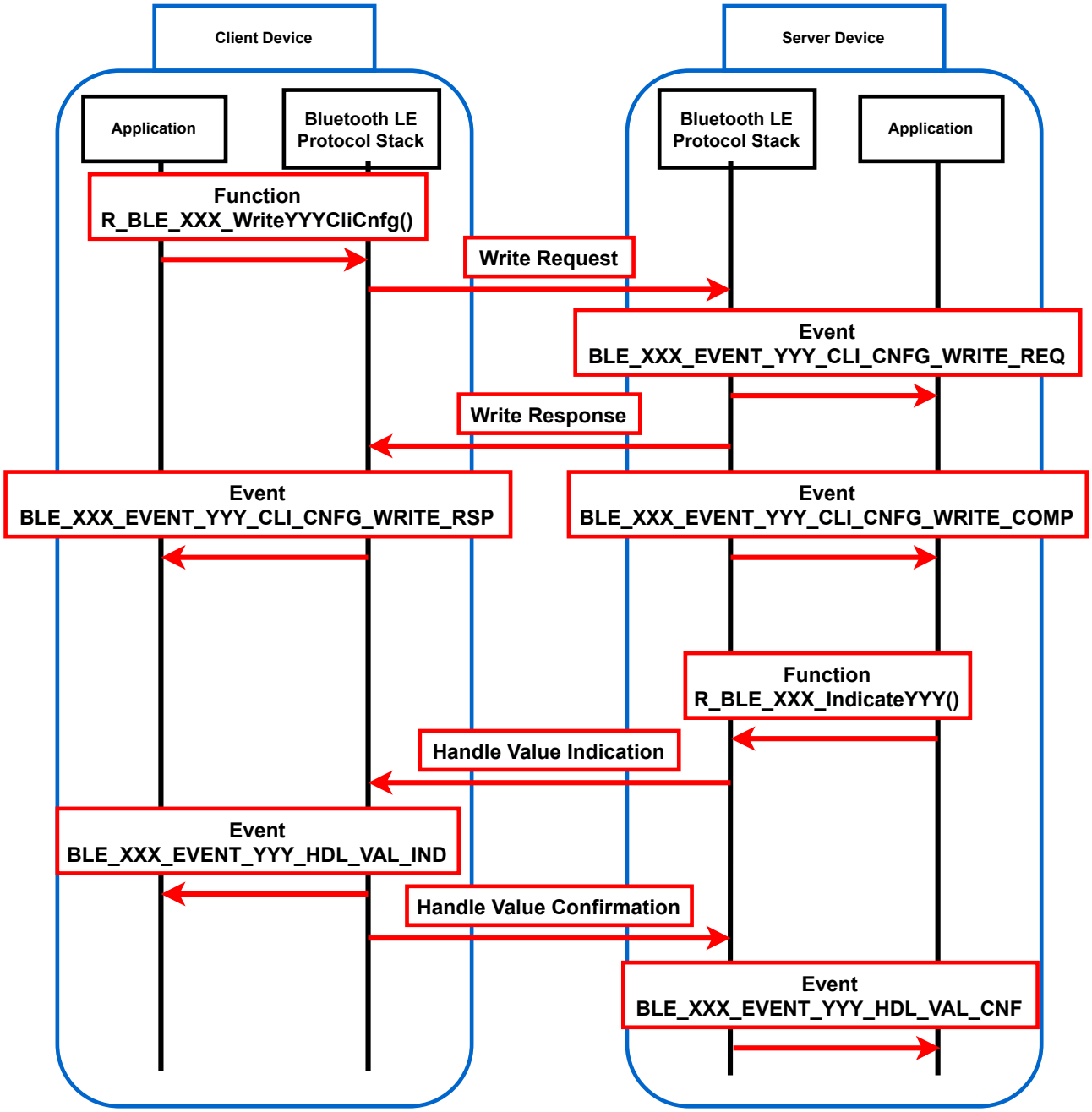


図 10-6　Indication 動作の実行手順

10.2.6 ReliableWrites 動作

ReliableWrites 動作は GATT クライアントから送信されたデータを GATT サーバで受信した後、正しい値が書き込まれていることを確認したうえで GATT データベースに反映する動作です。ReliableWrites 動作では動作手順が 2 段階あります。GATT クライアントから「Prepare Write Request」を使用してデータを送信し GATT サーバのキューで保持します。その後、GATT サーバが「Execute Write Request」を受信した時にキューに保持されたデータを GATT データベースに反映します。GATT クライアントは「Prepare Write Response」で正しいデータが書き込まれているか確認することができ、より信頼性の高いデータ転送が行えます。

ReliableWrites 動作の API は QE for BLE から生成されるサービスの API に含まれないため Bluetooth LE Protocol Stack の API を使用して実現する必要があります。また、ReliableWrites 動作を行うには、Characteristic Extended Properties Descriptor がディスクリプタとして追加されている必要があります。

GATT サーバ：
ReliableWrites 動作を行う前に API「R_BLE_GATTS_SetPrepareQueue()」を使用して受信するためのキューを確保します。確保するキューのサイズは ReliableWrites 動作が可能なキャラクタリスティックのサイズの合計より大きいサイズを指定してください（サイズの合計が 6 の場合、7 以上の値を指定）。
「Prepare Write Request」で受信したデータはイベント「BLE_XXX_EVENT_YYY_WRITE_REQ」でアプリケーションに通知されます。「Execute Write Request」を受信してキューに保持された値を GATT データベースに反映したことをイベント「BLE_XXX_EVENT_YYY_WRITE_COMP」でアプリケーションに通知します。

GATT クライアント：
アプリケーションから関数「R_BLE_GATTC_ReliableWrites()」を使用して「Prepare Write Request」を送信することができます。送信したデータごとに「Prepare Write Response」を受信し、イベント「BLE_GATTC_EVENT_RELIAVBLE_WRITE_TX_COMP」でデータを確認することができます。正しいデータを GATT サーバが受信していることを確認するした後、関数「R_BLE_GATTC_ExecWrite()」を使用して「Execute Write Request」を送信し、GATT データベースへの書き込みを行ってください。
確認したデータが間違っている場合は関数「R_BLE_GATTC_ExecWrite()」を使用して GATT サーバで保持しているデータを破棄するための「Execute Write Request」を送信してください。

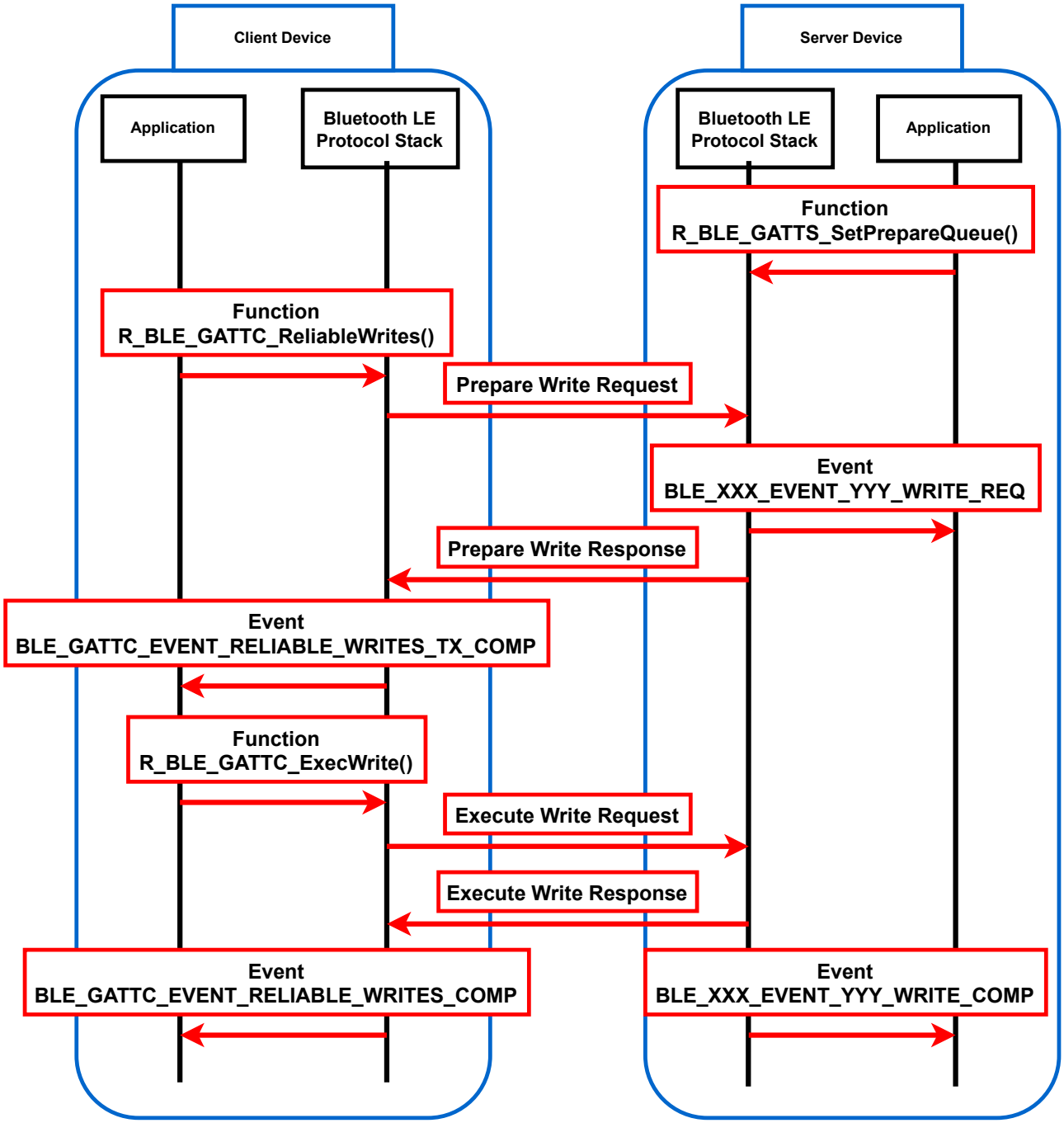


図 10-7　ReliableWrites 動作の実行手順

10.2.7 Broadcast 動作

Broadcast 動作は不特定多数のデバイスへコネクションを伴わないデータの送信を行うための動作です。送信側はブロードキャスタと呼ばれ、Advertising 動作を利用します。受信側はオブザーバと呼ばれ、スキャン動作を利用します。コネクションを伴わない通信のため、一度にデータ通信できるデバイスの数などの制限がない反面、受信側がデータを受信していることを保証できない通信になります。

Broadcast 動作の API は QE for BLE から生成されるサービスの API に含まれないため Bluetooth LE Protocol Stack の API を使用して実現する必要があります。また、Broadcast 動作を行うには Server Characteristic Configuration Descriptor がディスクリプタとして追加されている必要があります。

GATT サーバ (ブロードキャスタ) :
Advertising を使用してデータを送信します。Advertising の概要については「5.Advertising」を参照してください。
なお、Broadcast 動作としてアドバタイズを行う場合、以下の制限があります。

- Advertising のタイプの指定（5.2.1.1）について、表 5.1 の Advertising Type が「Non-Connectable and Non-Scannable Undirected」もしくは「Non-Connectable Non-Scannable Directed」で示されている値を adv_prop_type フィールドに設定してください。

- Advertising Data の設定（5.4）について、AD Type に「Service Data（16bit UUID の場合 0x16、128bit UUID の場合 0x21）」、AD Data にキャラクタリスティックが属するサービスの UUID とデータを入力した AD structure を設定することでサービスのデータを通信することができます。また AD Type が「Flags（0x01）」の AD structure を設定する場合、AD Data に「LE Limited Discoverable Mode」もしくは「LE General Discoverable Mode」を選択しないようにしてください。

GATT クライアント (オブザーバ) :
スキャン動作を使用してデータを受信します。スキャン動作の概要については「6.スキャン」を参照してください。スキャンの動作に関する制限はありませんが、ブロードキャスタが送信する Advertising Event を受信できるようにスキャンパラメータを設定してください。

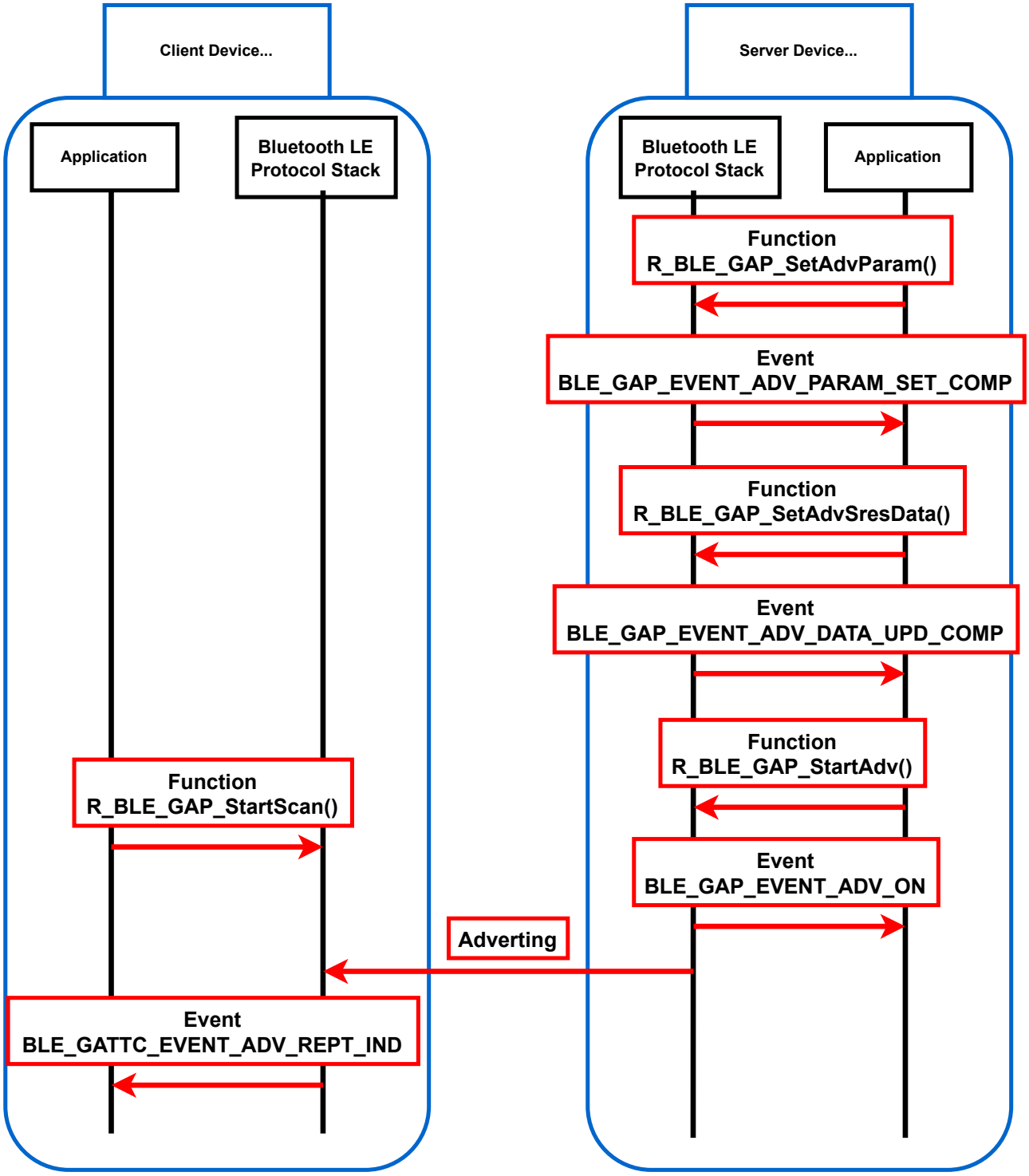


図 10-8　Broadcast 動作の実行手順

10.3 GATT プロシージャの実装方法

ここではデモアプリで使用される LED Switch Service を例にして、GATT プロシージャのアプリケーションへの実装方法についてユースケースを交えて説明します。表 10.3 に LED Switch Service の構成を示します。

表 10.3 LED Switch Service の構成

サービス名	キャラクタリスティック名	GATT プロシージャ
LED Switch Service **LSS**	LED Blink Rate	Read, Write
	Switch State	Notify

10.3.1 クライアント側からデータを送信する実装例

ユースケース：クライアント側のスイッチを押すとサーバ側の LED の点滅速度を変更する

クライアント側のボードにあるスイッチが押されたときに LSS の LED Blink Rate を利用してサーバ側の LED の点滅する速度を変えます。クライアント側はスイッチが押された後、Read 動作を使用して LED Blink Rate の値を確認した後、Write 動作を使用して変更後の値を送信します。サーバ側は受信した変更後の値を使用して LED の点滅間隔を変更します。

```
/* 省略 */

#include "timer/r_ble_timer.h"
static uint32_t gs_timer_hdl;
#include "board/r_ble_board.h"

/* 省略 */

static void timer_cb(uint32_t timer_hdl)
{
        R_BLE_BOARD_ToggleLEDState(BLE_BOARD_LED2);
}

/* 省略 */

static void lss_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t
*p_data)
{
    switch(type)
    {
        case BLE_LSS_EVENT_BLINK_RATE_WRITE_COMP:
        {
            uint8_t rate = *(uint8_t *)p_data->p_param;
            if (0 == rate)
            {
                R_BLE_TIMER_Stop(gs_timer_hdl);
                R_BLE_BOARD_SetLEDState(BLE_BOARD_LED2, false);
            }
            else
            {
                R_BLE_TIMER_UpdateTimeout(gs_timer_hdl, rate * 100);
            }
        } break;

        default:
            break;
    }
}

/* 省略 */
void app_main(void)
{
    /* Initialize BLE */
    R_BLE_Open();

    R_BLE_TIMER_Init();
    R_BLE_TIMER_Create(&gs_timer_hdl, 1, BLE_TIMER_PERIODIC, timer_cb);

    R_BLE_BOARD_Init();

/* 省略 */
}
```

タイマと LED を使用するための
ライブラリを追加する

タイマのコールバック毎に
LED を点滅させる

受信した Brink Rate の値を点滅
に使用しているタイマへ反映さ
せる

タイマ・LED の初期化

コード 10-1　サーバ側の app_main.c への実装

```
/* 省略 */

#include "board/r_ble_board.h"
#define LED_RATE_LOW  (0x01)
#define LED_RATE_HIGH (0xff)

/* 省略 */

static void sw_cb(void)
{
    R_BLE_LSC_ReadBlinkRate(g_conn_hdl);
}

/* 省略 */

static void lsc_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t
*p_data)
{
    switch(type)
    {
        case BLE_LSC_EVENT_BLINK_RATE_READ_RSP:
        {
            uint8_t read_rate = *(uint8_t *)p_data->p_param;
            uint8_t write_rate = 0;
            if (LED_RATE_LOW == read_rate)
            {
                write_rate = LED_RATE_HIGH;
            }
            else
            {
                write_rate = LED_RATE_LOW;
            }

            R_BLE_LSC_WriteBlinkRate(g_conn_hdl, &write_rate);
        } break;

        default:
            break;
    }
}

/* 省略 */

void app_main(void)
{
    /* Initialize BLE */
    R_BLE_Open();

    R_BLE_BOARD_Init();
    R_BLE_BOARD_RegisterSwitchCb(BLE_BOARD_SW2, sw_cb);

/* 省略 */
}
```

スイッチを使用するためのライブラリを追加する

スイッチ入力のコールバックで Read 動作を開始する

Read Response で受信した値に応じて Write 動作を開始する

スイッチの初期化

コード 10-2　クライアント側の app_main.c への実装

10.3.2 サーバ側からデータを送信する実装例

ユースケース：サーバ側のスイッチを押すとクライアント側の LED を点滅させる

サーバ側のボードにあるスイッチが押されるたびに LSS の Switch State を利用してクライアント側の LED を点滅させます。サーバ側はスイッチが押されるたびに Notify 動作を使用して押された回数を送信します。クライアント側は受信した値をもとに奇数の時に点灯させ、偶数の時に消灯させます。

```
/* 省略 */

#include "board/r_ble_board.h"

/* 省略 */

static uint8_t switch_count = 0;

/* 省略 */

static void sw_cb(void)
{
    switch_count++;
    R_BLE_LSS_NotifySwitchState(g_conn_hdl, &switch_count);
}

/* 省略 */

void app_main(void)
{
    /* Initialize BLE */
    R_BLE_Open();

    R_BLE_BOARD_Init();
    R_BLE_BOARD_RegisterSwitchCb(BLE_BOARD_SW2, sw_cb);

/* 省略 */
}
```

スイッチを使用するための
ライブラリを追加する

スイッチ入力のコールバックで
Notify 動作を開始する

スイッチの初期化

コード 10-3　サーバ側の app_main.c への実装

```
/* 省略 */
#include "board/r_ble_board.h"

/* 省略 */

static void lsc_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_LSC_EVENT_SWITCH_STATE_HDL_VAL_NTF:
        {
            uint8_t ntf_state = *(uint8_t *)p_data->p_param;
            if (ntf_state % 2 == 0)
            {
                R_BLE_BOARD_SetLEDState(BLE_BOARD_LED2, false);
            }
            else
            {
                R_BLE_BOARD_SetLEDState(BLE_BOARD_LED2, true);
            }

        } break;

        default:
            break;
    }
}

/* 省略 */

static void disc_comp_cb(uint16_t conn_hdl)
{
    /* TODO: Add function after discovery completed */
    static uint16_t s_cccd_req;
    s_cccd_req = BLE_GATTS_CLI_CNFG_NOTIFICATION;
    R_BLE_LSC_WriteSwitchStateCliCnfg(g_conn_hdl, &s_cccd_req);
    return;
}

/* 省略 */

void app_main(void)
{
    /* Initialize BLE */
    R_BLE_Open();

    R_BLE_BOARD_Init();

/* 省略 */

}
```

LED を使用するための
ライブラリを追加する

Notification で受信した値に
合わせて LED を点滅させる

ディスカバリの完了時に
CCCD に書き込んで Notify
動作を許可する

LED の初期化

コード 10-4　クライアント側の app_main.c への実装

11. デバッグ

GATT サーバアプリケーションでは Advertising、コネクション、GATT データベース、Indication、Notification、Read Response、Write Response を確認する必要があります。BTTS の Beacon Scanning や Data Comm Master、GATT Browser が利用可能です。

GATT クライアントアプリケーションではスキャン、コネクション、Service Discovery、Read Request、Write Request、Confirmation を確認する必要があります。BTTS の Beacon Advertising や Data Comm Slave が利用可能です。

【注】GATT Browser や BTTS ですべての機能を評価できるわけではありません。

アプリケーション調査のためにロガー機能が利用可能です。ロガー機能を使うと e²studio や IAR のデバッグ・コンソールにログ出力することができます。

GATT Browser については「GATTBrowser for Android スマートフォンアプリ取扱説明書 (R01AN3802)」または「GATTBrowser for iOS スマートフォンアプリ取扱説明書 (R21AN0017)」を参照してください。

BTTS については「Bluetooth Test Tool Suite 操作説明書 (R01AN4554)」を参照してください。ロガー機能の詳細については「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)」の「5.2 ロガー」を参照してください。

11.1 ロガー機能の使用

BLE_DEFAULT_LOG_LEVEL または BLE_LOG_LEVEL を r_ble_logger.h をインクルードする前に変更するとログレベルを変更することができます。両方を定義すると最後に定義した方が採用されます。どちらかを 0 にするとログ出力が無効になります。ログレベルは 1 で BLE_LOG_ERR、2 で BLE_LOG_ERR / BLE_LOG_WRN、3 で BLE_LOG_ERR / BLE_LOG_WRN / BLE_LOG_DBG のマクロ関数が有効になり、4 以上を指定して BLE_LOG マクロ関数を使うと拡張することもできます。

BLE_LOG_TAG を r_ble_logger.h をインクルードする前に変更するとログタグを変更することができます。

ログレベルを拡張し、R_BLE_ABS_StartLegacyAdv の引数を確認するコードの例を以下に示します。app_main.c と新規作成したソースファイル(appapp.c)でロガー機能を使用します。

```
[app_main.c]
#define BLE_DEFAULT_LOG_LEVEL (1)
#define BLE_LOG_LEVEL (4)
#define BLE_LOG_TAG "app_main"
#include "logger/r_ble_logger.h"
#define BLE_LOG_XXX(...) BLE_LOG(4, "XXX", __VA_ARGS__)
extern void appapp( void );

//static st_ble_abs_legacy_adv_param_t gs_adv_param =
st_ble_abs_legacy_adv_param_t gs_adv_param =
(省略)

    switch (type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {

            BLE_LOG_ERR("R_BLE_ABS_StartLegacyAdv");
            BLE_LOG_WRN("interval=%f", gs_adv_param.slow_adv_intv * 0.625 );
            for( int i=0; i<gs_adv_param.adv_data_length; i++ ){
                BLE_LOG_DBG("data[%02X]", gs_adv_param.p_adv_data[i] );
             }
             appapp();
             BLE_LOG_XXX("advlen=%d, sreslen=%d", gs_adv_param.adv_data_length,
gs_adv_param.sres_data_length );
            R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
(省略)
```

コード 11-1　R_BLE_ABS_StartLegacyAdv の引数を確認するコード例 (app_main.c)

```
[appapp.c]
#include "r_ble_rx23w_if.h"
#include "abs/r_ble_abs_api.h"
#define BLE_DEFAULT_LOG_LEVEL (1)
#define BLE_LOG_LEVEL (5)
#define BLE_LOG_TAG "appapp"
#include "logger/r_ble_logger.h"
#define BLE_LOG_YYY(...) BLE_LOG(5, "YYY", __VA_ARGS__)
extern st_ble_abs_legacy_adv_param_t gs_adv_param;

void appapp( void )
{
        for( int i=0; i<gs_adv_param.sres_data_length; i++ ){
                BLE_LOG_YYY("data[%02X]", gs_adv_param.p_sres_data[i] );
        }
}
```

コード 11-2　R_BLE_ABS_StartLegacyAdv の引数を確認するコード例 (appapp.c)

ロガー機能のログは e²studio 上の[Renesas Views] → [デバッグ] → [Renesas Debug Virtual Console]に表示されます。ロガー呼び出し 1 回につき 1 行表示されるので改行は不要です。デバッグ・コンソールに表示されたログは次回デバッグ実行時に自動的にクリアされません。右クリックから[クリア]でクリアしてください。

コンソール　タスク　問題　スマート・ブラウザー　Debugger Console　Renesas Debug Virtual Console　メモリー

```
app_main: [ERR] (ble_app_gapcb:181) R_BLE_ABS_StartLegacyAdv
app_main: [WRN] (ble_app_gapcb:182) interval=480.000000
app_main: [DBG] (ble_app_gapcb:184) data[02]
app_main: [DBG] (ble_app_gapcb:184) data[01]
app_main: [DBG] (ble_app_gapcb:184) data[06]
app_main: [DBG] (ble_app_gapcb:184) data[09]
app_main: [DBG] (ble_app_gapcb:184) data[09]
app_main: [DBG] (ble_app_gapcb:184) data[52]
app_main: [DBG] (ble_app_gapcb:184) data[42]
app_main: [DBG] (ble_app_gapcb:184) data[4C]
app_main: [DBG] (ble_app_gapcb:184) data[45]
app_main: [DBG] (ble_app_gapcb:184) data[2D]
app_main: [DBG] (ble_app_gapcb:184) data[44]
app_main: [DBG] (ble_app_gapcb:184) data[45]
app_main: [DBG] (ble_app_gapcb:184) data[56]
appapp: [YYY] (appapp:20) data[09]
appapp: [YYY] (appapp:20) data[09]
appapp: [YYY] (appapp:20) data[52]
appapp: [YYY] (appapp:20) data[42]
appapp: [YYY] (appapp:20) data[4C]
appapp: [YYY] (appapp:20) data[45]
appapp: [YYY] (appapp:20) data[2D]
appapp: [YYY] (appapp:20) data[44]
appapp: [YYY] (appapp:20) data[45]
appapp: [YYY] (appapp:20) data[56]
app_main: [XXX] (ble_app_gapcb:187) advlen=13, sreslen=10
```

図 11-1　ロガー機能によって表示されるログ

11.2 コマンドライン機能の使用

「1.6.1 主要機能」を参照して機能を有効にしてコード生成してください。ライブラリに組み込み済みの標準のコマンドライン機能を使用するためのコードを以下に示します。

```
[app_main.c]
#include "cli/r_ble_cli.h"
#include "cmd/r_ble_cmd_abs.h"
#include "cmd/r_ble_cmd_vs.h"
#include "cmd/r_ble_cmd_sys.h"
/* CommandLine parameters */
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
    &g_abs_cmd,
    &g_vs_cmd,
    &g_sys_cmd,
    &g_ble_cmd
};

(省略)

static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    R_BLE_CMD_AbsGapCb(type, result, p_data);
(省略)

static void vs_cb(uint16_t type, ble_status_t result, st_ble_vs_evt_data_t *p_data)
{
    R_BLE_CMD_VsCb(type, result, p_data);
(省略)

void app_main(void)
{
    (省略)
    /* Configure CommandLine */
    R_BLE_CLI_Init();
    R_BLE_CLI_RegisterCmds(gsp_cmds, ARRAY_SIZE(gsp_cmds));
    R_BLE_CMD_SetResetCb(ble_app_init);
    (省略)

    /* main loop */
    while (1)
    {
        /* Process Command Line */
        R_BLE_CLI_Process();
        (省略)
```

コード 11-3　コマンドライン機能の使用例

ターミナルからスキャン → コネクション → 切断する入力例を以下に示します。

```
$
$ gap scan 0x09 0x52
74:90:50:FF:FF:FF pub ff 0000
74:90:50:FF:FF:FF pub ff 0000
74:90:50:FF:FF:FF pub ff 0000

$ receive BLE_GAP_EVENT_SCAN_OFF result : 0x0000

$ gap conn 74:90:50:ff:ff:ff pub
receive BLE_GAP_EVENT_CONN_IND result : 0x0000
gap: connected conn_hdl:0x0020, addr:74:90:50:FF:FF:FF pub

$ receive BLE_GAP_EVENT_DATA_LEN_CHG result : 0x0000, conn_hdl : 0x0020
tx_octets : 0x00fb
tx_time   : 0x0848
rx_octets : 0x00fb
rx_time   : 0x0848

$ gap disconn 0x20
$ receive BLE_GAP_EVENT_DISCONN_IND result : 0x0000
gap: disconnected conn_hdl:0x0020, addr:74:90:50:FF:FF:FF pub, reason:0x16

$
$
```

Complete Local Name (0x09)が"R"(0x52)で始まるデータを含む Advertising のみスキャンされます。
【注】「gap scan 0x09 0x52,0x42」とすると、Complete Local Name (0x09)が" RB" から始まるデータを含む Advertising のみスキャンされます。
【注】gap scan コマンドは[Ctrl]+[c]または gap scan stop で停止されます。

BD アドレスとアドレスタイプを指定してコネクションします。

コネクションハンドルを指定して切断します。

11.3 RF 通信タイミング通知機能の使用

以下は RF 通信タイミングを確認するために"rf log on"コマンドでログ表示するサンプルです。このサンプルはコマンドライン機能と RF 通信タイミング通知機能を使用します。「1.6.1 主要機能」と「3.11 RF 通信タイミング通知」を参照して機能を有効にしてコード生成してください。src フォルダに r_ble_cmd_rf.h を新規作成してください。

```
[src¥r_ble_cmd_rf.h]
#include "r_ble_rx23w_if.h"
#include "cli/r_ble_cli.h"

#ifndef R_BLE_CMD_RF_H_
#define R_BLE_CMD_RF_H_

typedef struct
{
    uint32_t elapsed_time;
    uint16_t event_type;
    uint16_t event_data;
    uint8_t start_end;
} st_ble_rf_log_t;

#define BLE_RF_LOG_NUM_MAX 1000
extern st_ble_rf_log_t gs_rf_log[BLE_RF_LOG_NUM_MAX];
extern uint32_t gs_rf_log_idx;
extern uint32_t gs_timer_elapsed_time;
extern const st_ble_cli_cmd_t g_rf_cmd;
extern void save_rf_log( uint16_t event_type, uint16_t event_data, uint8_t start_end );

#endif /* R_BLE_CMD_RF_H_ */
```

コード 11-4　RF 通信タイミングのログ表示するサンプル(r_ble_cmd_rf.h)

src フォルダに r_ble_cmd_rf.c を新規作成してください。

```
[src¥r_ble_cmd_rf.c]
#include "r_ble_rx23w_if.h"
#include "cmd/r_ble_cmd.h"
#include "r_ble_cmd_rf.h"

#if (BLE_CFG_CMD_LINE_EN == 1) && (BLE_CFG_HCI_MODE_EN == 0)

#define pf R_BLE_CLI_Printf
st_ble_rf_log_t gs_rf_log[BLE_RF_LOG_NUM_MAX];
uint32_t gs_rf_log_idx = 0;
uint32_t gs_timer_elapsed_time;

void save_rf_log( uint16_t event_type, uint16_t event_data, uint8_t start_end )
{
    gs_rf_log[gs_rf_log_idx].elapsed_time = gs_timer_elapsed_time;
    gs_rf_log[gs_rf_log_idx].event_type = event_type;
    gs_rf_log[gs_rf_log_idx].event_data = event_data;
    gs_rf_log[gs_rf_log_idx].start_end = start_end;
    gs_rf_log_idx++;
    if( gs_rf_log_idx >= BLE_RF_LOG_NUM_MAX ){
        gs_rf_log_idx = 0;
    }
}

static void show_rf_log( uint32_t elapsed_time, uint16_t event_type, uint16_t event_data, uint8_t
start_end )
{
        switch( event_type )
        {
                case 0x0000:/*BLE_EVENT_TYPE_CONN*/
                {
                        if( start_end == 1 ){ pf("%010d,ConnS,%d¥n", elapsed_time, event_data ); }
                        if( start_end == 2 ){ pf("%010d,ConnE,%d¥n", elapsed_time, event_data ); }
                } break;
                case 0x0001:/*BLE_EVENT_TYPE_ADV*/
                {
                        if( start_end == 1 ){ pf("%010d,AdvS,%d¥n", elapsed_time, event_data ); }
```

```
                        if( start_end == 2 ){ pf("%010d,AdvE,%d¥n", elapsed_time, event_data ); }
                } break;
                case 0x0002:/*BLE_EVENT_TYPE_SCAN*/
                {
                        if( start_end == 1 ){ pf("%010d,ScanS,%d¥n", elapsed_time, event_data ); }
                        if( start_end == 2 ){ pf("%010d,ScanE,%d¥n", elapsed_time, event_data ); }
                } break;
                case 0x0003:/*BLE_EVENT_TYPE_INITIATOR*/
                {
                        if( start_end == 1 ){ pf("%010d,InitS,%d¥n", elapsed_time, event_data ); }
                        if( start_end == 2 ){ pf("%010d,InitE,%d¥n", elapsed_time, event_data ); }
                } break;
                case 0x0004:/*BLE_EVENT_TYPE_RF_DS_START*//*BLE_EVENT_TYPE_RF_DS_CLOSE*/
                {
                        if( start_end == 1 ){ pf("%010d,SleepS,%d¥n", elapsed_time, event_data ); }
                        if( start_end == 2 ){ pf("%010d,SleepE,%d¥n", elapsed_time, event_data ); }
                } break;
                default:
                {
                } break;
        }
}

static void exec_rf_log(int argc, char *argv[])
{
    ble_status_t status;
    if (strcmp(argv[1], "on") == 0)
    {
         R_BLE_CLI_Printf("time,type,data¥n");
         for(int i=0; i<BLE_RF_LOG_NUM_MAX; i++){
                show_rf_log( gs_rf_log[i].elapsed_time, gs_rf_log[i].event_type, gs_rf_log[i].event_data,
gs_rf_log[i].start_end );
         }
    }
    else
    {
        pf("rf %s: unrecognized operands¥n", argv[0]);
    }
}

static const st_ble_cli_cmd_t rf_log_cmd = {
    .p_name = "log",
    .exec   = exec_rf_log,
    .p_help = "Usage: rf log (on)¥n"
              "Show rf_event or not",
};

static const st_ble_cli_cmd_t * const rf_sub_cmds[] = {
    &rf_log_cmd,
};

const st_ble_cli_cmd_t g_rf_cmd = {
    .p_name      = "rf",
    .p_cmds      = rf_sub_cmds,
    .num_of_cmds = ARRAY_SIZE(rf_sub_cmds),
    .p_help      = "Sub Command: log¥n"
                   "Try 'rf sub-command help' for more information",
};

const st_ble_cli_cmd_t g_rf_cmd;

#endif /* (BLE_CFG_CMD_LINE_EN == 1) && (BLE_CFG_HCI_MODE_EN == 0) */
```

コード 11-5　RF 通信タイミングのログ表示するサンプル(r_ble_cmd_rf.c)

以下のコードで RF 通信タイミング通知をログ保存します。

```
[src¥smc_gen¥r_ble_rx23w¥src¥platform¥r_ble_pf_functions.c]
extern uint32_t gs_timer_elapsed_time;
#include "../../../../../src/r_ble_cmd_rf.h"

BLE_SECTION_P void r_ble_rf_notify_event_start(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( (uint16_t)(param>>16) )
    {
        case 0x0000:/*BLE_EVENT_TYPE_CONN*/
        {
            save_rf_log( BLE_EVENT_TYPE_CONN, 0x0000, 0x01 );
        } break;
        case 0x0001:/*BLE_EVENT_TYPE_ADV*/
        {
            save_rf_log( BLE_EVENT_TYPE_ADV, 0x0000, 0x01 );
        } break;
        case 0x0002:/*BLE_EVENT_TYPE_SCAN*/
        {
            save_rf_log( BLE_EVENT_TYPE_SCAN, 0x0000, 0x01 );
        } break;
        case 0x0003:/*BLE_EVENT_TYPE_INITIATOR*/
        {
            save_rf_log( BLE_EVENT_TYPE_INITIATOR, 0x0000, 0x01 );
        } break;
    }
}
BLE_SECTION_P void r_ble_rf_notify_event_close(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( (uint16_t)(param>>16) )
    {
        case 0x0000:/*BLE_EVENT_TYPE_CONN*/
        {
            save_rf_log( BLE_EVENT_TYPE_CONN, 0x0000, 0x02 );
        } break;
        case 0x0001:/*BLE_EVENT_TYPE_ADV*/
        {
            save_rf_log( BLE_EVENT_TYPE_ADV, 0x0000, 0x02 );
        } break;
        case 0x0002:/*BLE_EVENT_TYPE_SCAN*/
        {
            save_rf_log( BLE_EVENT_TYPE_SCAN, 0x0000, 0x02 );
        } break;
        case 0x0003:/*BLE_EVENT_TYPE_INITIATOR*/
        {
            save_rf_log( BLE_EVENT_TYPE_INITIATOR, 0x0000, 0x02 );
        } break;
    }
}

BLE_SECTION_P void r_ble_rf_notify_deep_sleep(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( param )
    {
        case BLE_EVENT_TYPE_RF_DS_START:
        {
            save_rf_log( 0x0004, 0x0000, 0x01 );
        } break;
        case BLE_EVENT_TYPE_RF_DS_CLOSE:
        {
            save_rf_log( 0x0004, 0x0000, 0x02 );
        } break;
    }
}
```

コード 11-6　RF 通信タイミングのログ表示するサンプル(ログの保存)

以下のコードでソフトウェアタイマを使用して 1ms 周期でタイマカウントをインクリメントします。

```
[app_main.c]
#include "timer/r_ble_timer.h"
/* timer handle */
static uint32_t gs_timer_hdl;

#include "cli/r_ble_cli.h"
#include "../../../src/r_ble_cmd_rf.h"
/* CommandLine parameters */
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
    &g_rf_cmd,
};

(省略)

static void timer_cb(uint32_t timer_hdl)
{
    gs_timer_elapsed_time++;
}

void app_main(void)
{
    (省略)
    /* Initialize timer */
    R_BLE_TIMER_Init();
    /* Create timer */
    gs_timer_hdl = BLE_TIMER_INVALID_HDL;
    gs_timer_elapsed_time = 0;
    R_BLE_TIMER_Create(&gs_timer_hdl, 1, BLE_TIMER_PERIODIC, timer_cb);
    R_BLE_TIMER_Start(gs_timer_hdl);
    /* Configure CommandLine */
    R_BLE_CLI_Init();
    R_BLE_CLI_RegisterCmds(gsp_cmds, ARRAY_SIZE(gsp_cmds));
    R_BLE_CMD_SetResetCb(ble_app_init);
    while (1)
    {
        /* Process Command Line */
        R_BLE_CLI_Process();
    (省略)
```

コード 11-7　RF 通信タイミングのログ表示するサンプル(タイマカウントのインクリメント)

“rf log on”コマンドを入力すると以下のようなログが出力されます。ただし、コネクション中に”rf log on”コマンドを入力するとログ出力のループ処理によって CPU が占有されて RF アイドル時間内で処理が終わらず、コネクションが維持できない場合があります。MCU/RF の動作概要については「1.5 Bluetooth LE Protocol Stack の動作概要」を参照してください。

```
[Advertising → コネクションのログ]
0000019851,AdvS,0
0000019854,AdvE,0
0000019854,SleepS,0
0000020286,SleepE,0
0000020289,AdvS,0
0000020292,AdvE,0
0000020293,SleepS,0
0000020728,SleepE,0
0000020731,AdvS,0
0000021069,ConnS,0
0000021070,ConnE,0
0000021392,ConnS,0
0000021394,ConnE,0
0000021715,ConnS,0
0000021715,ConnE,0
0000022038,ConnS,0
0000022038,ConnE,0
0000022360,ConnS,0
0000022361,ConnE,0
0000022683,ConnS,0
0000022684,ConnE,0
0000022686,SleepS,0
0000023025,SleepE,0
0000023028,ConnS,0
0000023029,ConnE,0
0000023029,SleepS,0
0000023370,SleepE,0
0000023373,ConnS,0
0000023374,ConnE,0
```

```
[Scan → コネクションのログ]
0000002629,ScanS,0
0000002776,ScanE,0
0000002776,SleepS,0
0000002918,SleepE,0
0000002920,ScanS,0
0000003067,ScanE,0
0000003067,SleepS,0
0000003209,SleepE,0
0000003211,ScanS,0
0000003234,InitS,0
0000003261,InitE,0
0000003287,InitS,0
0000003314,InitE,0
0000003341,InitS,0
0000003368,InitE,0
0000003395,InitS,0
0000003442,ConnS,0
0000003442,ConnE,0
0000003761,ConnS,0
0000003763,ConnE,0
0000004081,ConnS,0
0000004082,ConnE,0
0000004401,ConnS,0
0000004402,ConnE,0
0000004405,SleepS,0
0000004734,SleepE,0
0000004736,ConnS,0
0000004737,ConnE,0
0000004737,SleepS,0
0000005080,SleepE,0
```

11.4 サーバの動作確認

11.4.1 BTTS Beacon Scanning の使用

Beacon Scanning を使用するとペリフェラルからの Advertising の受信状況をログ出力できます。以下の例では Advertising Interval が 480ms の Advertising を受信しています。49 秒 172 から 49 秒 656 まで 484ms の間隔で受信したことがわかります。それぞれの Advertising の後にスキャンレスポンスデータを受信したこともわかります。

```
[17] 15:08:49:172 (result = 0x0000)
BLE_GAP_EVENT_ADV_REPT_IND
  adv_rpt_type = 0x01
  p_ext_adv_rpt:
    num = 0x01 adv_type = 0x0013
    addr_type = 0x00 p_addr = 0xFF,0xFF,0xFF,0x50,0x90,0x74
    adv_phy = 0x01 sec_adv_phy = 0x00
    adv_sid = 0xFF tx_pwr = 0x7F rssi = -37
    perd_adv_intv = 0x0000
    dir_addr_type = 0x00 p_dir_addr = 0x00,0x00,0x00,0x00,0x00,0x00
    len = 0x0D p_data = 0x02,0x01,0x06,0x09,0x09,0x52,0x42,0x4C,0x45,0x2D,0x44,0x45,0x56

[18] 15:08:49:174 (result = 0x0000)
BLE_GAP_EVENT_ADV_REPT_IND
  adv_rpt_type = 0x01
  p_ext_adv_rpt:
    num = 0x01 adv_type = 0x001B
    addr_type = 0x00 p_addr = 0xFF,0xFF,0xFF,0x50,0x90,0x74
    adv_phy = 0x01 sec_adv_phy = 0x00
    adv_sid = 0xFF tx_pwr = 0x7F rssi = -37
    perd_adv_intv = 0x0000
    dir_addr_type = 0x00 p_dir_addr = 0x00,0x00,0x00,0x00,0x00,0x00
    len = 0x0A p_data = 0x09,0x09,0x52,0x42,0x4C,0x45,0x2D,0x44,0x45,0x56

[19] 15:08:49:656 (result = 0x0000)
BLE_GAP_EVENT_ADV_REPT_IND
  adv_rpt_type = 0x01
  p_ext_adv_rpt:
    num = 0x01 adv_type = 0x0013
    addr_type = 0x00 p_addr = 0xFF,0xFF,0xFF,0x50,0x90,0x74
    adv_phy = 0x01 sec_adv_phy = 0x00
    adv_sid = 0xFF tx_pwr = 0x7F rssi = -37
    perd_adv_intv = 0x0000
    dir_addr_type = 0x00 p_dir_addr = 0x00,0x00,0x00,0x00,0x00,0x00
    len = 0x0D p_data = 0x02,0x01,0x06,0x09,0x09,0x52,0x42,0x4C,0x45,0x2D,0x44,0x45,0x56

[20] 15:08:49:658 (result = 0x0000)
BLE_GAP_EVENT_ADV_REPT_IND
  adv_rpt_type = 0x01
  p_ext_adv_rpt:
    num = 0x01 adv_type = 0x001B
    addr_type = 0x00 p_addr = 0xFF,0xFF,0xFF,0x50,0x90,0x74
    adv_phy = 0x01 sec_adv_phy = 0x00
    adv_sid = 0xFF tx_pwr = 0x7F rssi = -37
    perd_adv_intv = 0x0000
    dir_addr_type = 0x00 p_dir_addr = 0x00,0x00,0x00,0x00,0x00,0x00
    len = 0x0A p_data = 0x09,0x09,0x52,0x42,0x4C,0x45,0x2D,0x44,0x45,0x56
```

0x00 : Advertising Report.
0x01 : Extended Advertising Report.
0x02 : Periodic Advertising Report.

Connectable advertising &&
Scannable advertising &&
Legacy advertising PDU

Connectable advertising &&
Scannable advertising &&
Scan response &&
Legacy advertising PDU

11.4.2 BTTS Data Comm Master の使用

Data Comm Master を使用すると以下のスループットサービスを QE for BLE で追加したサーバアプリケーションに対して連続した Write Request を実行し、Write Response の確認ができます。

CUSTOM SERVICE
UUID: 9CEF3D10-7FAB-49DC-AB89-762C9079FE96
PRIMARY SERVICE

CUSTOM CHARACTERISTIC
UUID: 9CEF3D11-7FAB-49DC-AB89-762C9079FE96
Properties: Write / Write Without Response

CUSTOM CHARACTERISTIC
UUID: 9CEF3D12-7FAB-49DC-AB89-762C9079FE96
Properties: Indicate / Notify
Descriptors:
Client Characteristic Configuration
UUID: 0x2920

以下の例では Connection Interval が 1000ms の Write Request を送信しています。次のコネクションイベントで Write Response を受信し、その次のコネクションイベントで Write Request するので、16 秒 332 から 18 秒 349 まで約 2000ms の間隔で送信したことがわかります。

```
[61] 16:58:16:332 (result = 0x0000)
R_BLE_GATTC_WriteChar
       conn_hdl : 0x0020
    write_data ->
           attr_hdl : 0x0012
             value ->
                   value_len : 0x00F4
                       value : (長いので省略)

[62] 16:58:18:348 (result = 0x0000)
BLE_GATTC_EVENT_CHAR_WRITE_RSP
  value_hdl : 0x0012

[63] 16:58:18:349 (result = 0x0000)
R_BLE_GATTC_WriteChar
       conn_hdl : 0x0020
    write_data ->
           attr_hdl : 0x0012
             value ->
                   value_len : 0x00F4
                       value : (長いので省略)
[64] 16:58:20:365 (result = 0x0000)
BLE_GATTC_EVENT_CHAR_WRITE_RSP
  value_hdl : 0x0012
```

11.4.3 GATT Browser の使用

クライアントアプリケーションにコネクションして、GATT データベース、Indication、Notification、Read Response、Write Response の確認ができます。

11.5 クライアントの動作確認

11.5.1 BTTS Beacon Advertising の使用

Beacon Advertising を使用するとクライアントへ Advertising を送信することができます。クライアント側でコマンドライン機能を使用するとスキャンの確認ができます。

以下のコードを追加するとスキャンの開始と Advertising の受信を表示できます。

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_GAP_EVENT_SCAN_ON:
        {
            R_BLE_CLI_Printf("receive BLE_GAP_EVENT_SCAN_ON result : 0x%04x¥n", result );
        } break;
        case BLE_GAP_EVENT_ADV_REPT_IND:
        {
            R_BLE_CLI_Printf("receive BLE_GAP_EVENT_ADV_REPT_IND result : 0x%04x¥n", result );
        } break;
        (省略)
```

コード 11-8　クライアント側のスキャンの開始と Advertising の受信の表示例

以下が実行結果です。Beacon Advertising の Advertising はノンコネクタブルなのでコネクションは失敗します。

```
$ gap scan 0x09 0x52
receive BLE_GAP_EVENT_SCAN_ON result : 0x0000
74:90:50:FF:FF:FF pub ff 0000
receive BLE_GAP_EVENT_ADV_REPT_IND result : 0x0000
receive BLE_GAP_EVENT_SCAN_OFF result : 0x0000

$ receive BLE_GAP_EVENT_CONN_IND result : 0x000e

$
```

11.5.2 BTTS Data Comm Slave の使用

Data Comm Slave を使用すると以下のスループットサービスを QE for BLE で追加したクライアントアプリケーションに対して連続した Indication を実行し、Confirmation の確認ができます。コネクション、Service Discovery、Write Request の確認もできます。

CUSTOM SERVICE (サービスの略称は th にしてください)
UUID: 9CEF3D10-7FAB-49DC-AB89-762C9079FE96
PRIMARY SERVICE

CUSTOM CHARACTERISTIC
UUID: 9CEF3D11-7FAB-49DC-AB89-762C9079FE96
Properties: Write / Write Without Response

CUSTOM CHARACTERISTIC (キャラクタリスティックの略称は thin にしてください)
UUID: 9CEF3D12-7FAB-49DC-AB89-762C9079FE96
Properties: Indicate / Notify
Descriptors:
Client Characteristic Configuration
UUID: 0x2920

リモートデバイスからコネクションパラメータ更新リクエストが通知された場合、ローカルデバイスは応答を返す必要があります。app_main.c の gap のコールバック内に以下のコードを追加してください。

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_GAP_EVENT_CONN_PARAM_UPD_REQ:
        {
            st_ble_gap_conn_upd_req_evt_t *p_conn_upd_req_evt_param =
                                (st_ble_gap_conn_upd_req_evt_t *)p_data->p_param;

            st_ble_gap_conn_param_t conn_updt_param = {
                .conn_intv_min = p_conn_upd_req_evt_param->conn_intv_min,
                .conn_intv_max = p_conn_upd_req_evt_param->conn_intv_max,
                .conn_latency  = p_conn_upd_req_evt_param->conn_latency,
                .sup_to        = p_conn_upd_req_evt_param->sup_to,
            };

            R_BLE_GAP_UpdConn(p_conn_upd_req_evt_param->conn_hdl,
                              BLE_GAP_CONN_UPD_MODE_RSP,
                              BLE_GAP_CONN_UPD_ACCEPT,
                              &conn_updt_param);
        } break;
        (省略)
```

コード 11-9　コネクションパラメータ更新リクエストへの応答のサンプル

Data Comm Slave のスループットサービスのスループットキャラクタリスティックに対して Indication を有効化するための Write Request を実行する必要があります。app_main.c の disc のコールバック内に、以下のコードを追加してください。Service Discovery でサーバ側のスループットサービスが検出されると呼ばれます。

```
static void disc_comp_cb(uint16_t conn_hdl)
{
    /* TODO: Add function after discovery completed */
    {
        uint16_t s_cccd_req;
        s_cccd_req = BLE_GATTS_CLI_CNFG_NOTIFICATION | BLE_GATTS_CLI_CNFG_INDICATION;
        R_BLE_THC_WriteThinCliCnfg(g_conn_hdl, &s_cccd_req);
    }
    (省略)
```

コード 11-10　disc のコールバック内での Indication 有効化のサンプル

以下の例では Connection Interval が 1000ms の Indication を送信しています。次のコネクションイベントで Confirmation を受信し、その次のコネクションイベントで Indication するので、25 秒 266 から 27 秒 286 まで約 2000ms の間隔で送信したことがわかります。

```
[62] 19:03:25:266 (result = 0x0000)
R_BLE_GATTS_Indication
  conn_hdl : 0x0060
  ind_data ->
      attr_hdl : 0x0005
         value ->
             value_len : 0x0014
                 value : 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F
0x10 0x11 0x12 0x13

[63] 19:03:27:286 (result = 0x0000)
BLE_GATTS_EVENT_HDL_VAL_CNF
  attr_hdl : 0x0005

[64] 19:03:27:286 (result = 0x0000)
R_BLE_GATTS_Indication
  conn_hdl : 0x0060
  ind_data ->
      attr_hdl : 0x0005
         value ->
             value_len : 0x0014
                 value : 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F
0x10 0x11 0x12 0x13

[65] 19:03:29:207 (result = 0x0000)
BLE_GATTS_EVENT_HDL_VAL_CNF
  attr_hdl : 0x0005
```

11.6 その他

11.6.1 MCU パッケージ

e²studio で[Renesas Views] → [スマート・コンフィグレータ] → [MCU パッケージ]で RX23W のピン配置が表示されます。LED や Switch などの接続と設定が合っているか確認することができます。

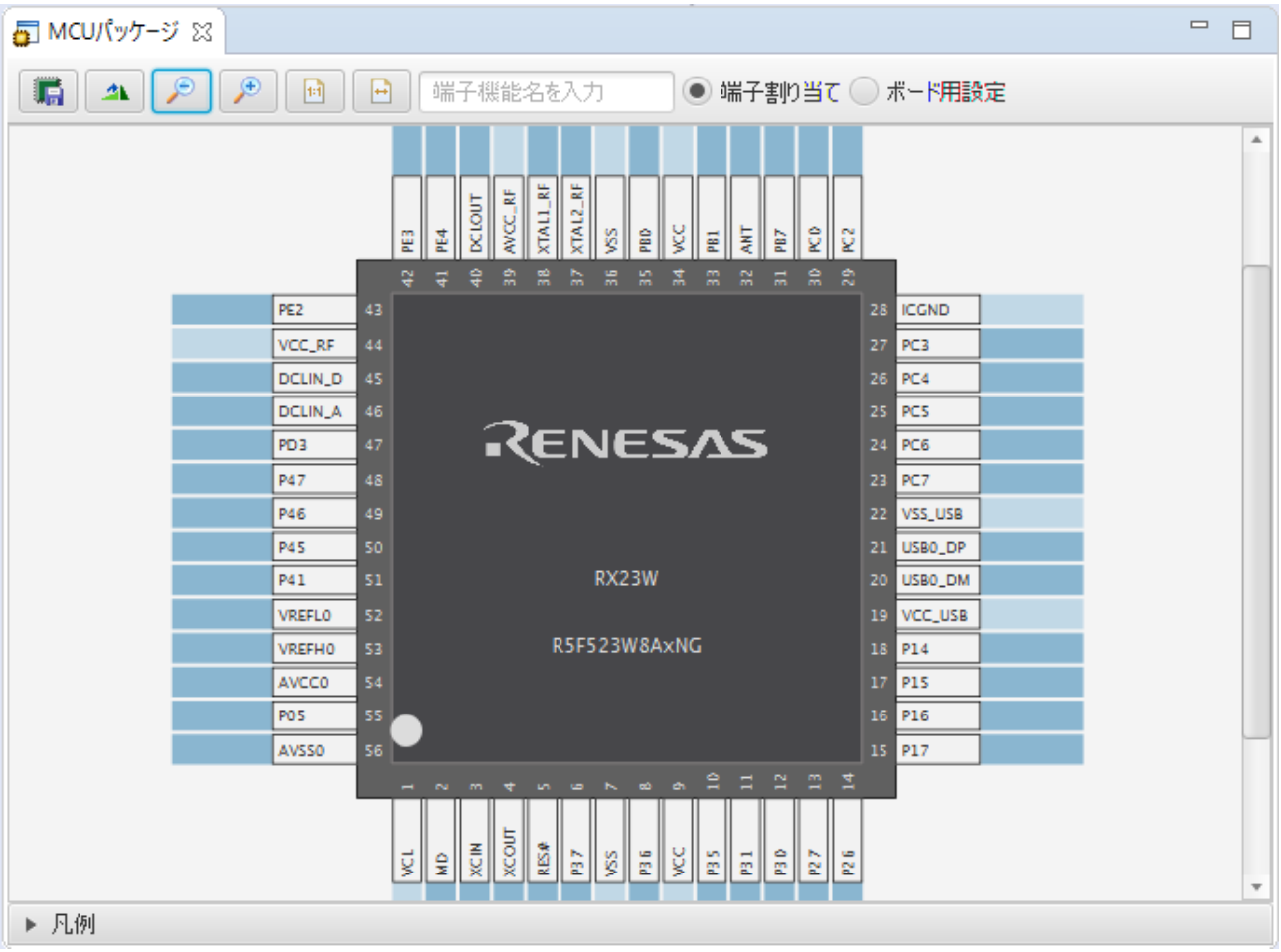


図 11-2　RX23W のピン配置

Target Board では以下のように接続されています。

表 11.1　Target Board のピン接続

ピン番号	ピン名	コメント
1	VCL	デジタル電源
2	MD	MCU ヘッダ CN3(任意に使用可能)
3	XCIN	
4	XCOUT	
5	RES#	MCU ヘッダ CN3(RES)
6	P37	MCU ヘッダ CN3(任意に使用可能)
7	VSS	MCU ヘッダ CN3(GND)
8	P36	MCU ヘッダ CN3(任意に使用可能)
9	VCC	MCU ヘッダ CN3(TGV)
10	P35	MCU ヘッダ CN3(任意に使用可能)
11	P31	MCU ヘッダ CN3(任意に使用可能)、Pmod™コネクタ
12	P30	MCU ヘッダ CN3(任意に使用可能)、Pmod™コネクタ
13	P27	MCU ヘッダ CN3(任意に使用可能)、Pmod™コネクタ
14	P26	MCU ヘッダ CN3(任意に使用可能)、Pmod™コネクタ

ピン番号	ピン名	コメント
15	P17	MCU ヘッダ CN3(任意に使用可能)
16	P16	MCU ヘッダ CN3(任意に使用可能)
17	P15	SW1(IRQ5 として使用)、MCU ヘッダ CN3(任意に使用可能)
18	P14	MCU ヘッダ CN3(任意に使用可能)
19	VCC_USB	
20	USB0_DM	MCU ヘッダ CN3(任意に使用可能)
21	USB0_DP	MCU ヘッダ CN3(任意に使用可能)
22	VSS_USB	
23	PC7	UART 通信(TXD8 として使用)、MCU ヘッダ CN3(任意に使用可能)
24	PC6	UART 通信(RXD8 として使用)、MCU ヘッダ CN3(任意に使用可能)
25	PC5	MCU ヘッダ CN3(任意に使用可能)
26	PC4	MCU ヘッダ CN3(任意に使用可能)
27	PC3	MCU ヘッダ CN3(任意に使用可能)
28	ICGND	GND、Bluetooth Low Energy ハードウェア
29	PC2	MCU ヘッダ CN4(任意に使用可能)
30	PC0	LED1、MCU ヘッダ CN4(任意に使用可能)
31	PB7	MCU ヘッダ CN4(任意に使用可能)、Pmod™コネクタ
32	ANT	Bluetooth Low Energy ハードウェア
33	PB1	MCU ヘッダ CN4(任意に使用可能)、Pmod™コネクタ、IRQ4
34	VCC	デジタル電源、MCU ヘッダ CN4(VCC)
35	PB0	LED2、MCU ヘッダ CN4(任意に使用可能)
36	VSS	デジタル電源、MCU ヘッダ CN4(GND)
37	XTAL2_RF	
38	XTAL1_RF	
39	AVCC_RF	Bluetooth Low Energy ハードウェア
40	DCLOUT	Bluetooth Low Energy ハードウェア
41	PE4	MCU ヘッダ CN4(任意に使用可能)
42	PE3	MCU ヘッダ CN4(任意に使用可能)
43	PE2	MCU ヘッダ CN4(任意に使用可能)
44	VCC_RF	Bluetooth Low Energy ハードウェア
45	DCLIN_D	Bluetooth Low Energy ハードウェア
46	DCLIN_A	Bluetooth Low Energy ハードウェア
47	PD3	Bluetooth Low Energy ハードウェア(CLKOUT_RF)、MCU ヘッダ CN4(任意に使用可能)、Pmod™コネクタ
48	P47	MCU ヘッダ CN4(任意に使用可能)
49	P46	MCU ヘッダ CN4(任意に使用可能)
50	P45	MCU ヘッダ CN4(任意に使用可能)
51	P41	MCU ヘッダ CN4(任意に使用可能)
52	VREFL0	アナログ電源、MCU ヘッダ CN4(VRL)
53	VREFH0	アナログ電源、MCU ヘッダ CN4(VRH)
54	AVCC0	アナログ電源、MCU ヘッダ CN4(AVC)
55	P05	MCU ヘッダ CN4(任意に使用可能)、Pmod™コネクタ
56	AVSS0	アナログ電源、MCU ヘッダ CN4(AVS)

11.6.2 MOT ファイルの生成

[プロジェクト] → [プロパティ] → [C/C++ ビルド] → [設定] → [ツール設定] → [Converter] → [出力] → [ロード・モジュール・コンバータを実行する]をチェック ON にして、[出力ファイル形式]を”モトローラ S 形式ファイルを出力する”に設定すると MOT ファイルが生成されます。

11.6.3 MAP ファイルの詳細出力

[プロジェクト] → [プロパティ] → [C/C++ ビルド] → [設定] → [ツール設定] → [Linker] → [リスト] → [リスト・ファイルを出力する]をチェック ON にして、[リスト・ファイルの内容]を”すべて指定する”に設定すると MAP ファイルの詳細が出力されます。

11.6.4 最適化

[プロジェクト] → [プロパティ] → [C/C++ ビルド] → [設定] → [ツール設定] → [Compiler] → [最適化] → [最適化レベル]を”レベル 0： 最適化を実施しない”に設定するとデバッグ中にメモリ内容を確認することができます。

12. 付録 A：サンプルアプリケーション

表 12.1 にアプリケーション開発者ガイドに付属する Target Board for RX23W 向けサンプルアプリケーションを示します。

表 12.1　サンプルアプリケーション

アプリケーション	プロジェクト	参照
Beacon サンプル	ble_demo_tbrx23w_beacon_sample	4.2.2　ユーザコマンドの作成手順 5.5　抽象 API による Advertising 5.7　ビーコンとして使用
Peripheral サンプル	ble_demo_tbrx23w_peripheral_sample	5.5　抽象 API による Advertising 5.6　スマートフォンとコネクション 7.3.2　複数のセントラルデバイスとコネクションする 8.4　MTU の変更 9.1.1　ペアリングパラメータ設定 9.4　プライバシ 10.2.4　Notification 動作 10.3.2　サーバ側からデータを送信する実装例
Central サンプル	ble_demo_tbrx23w_central_sample	6.1　スキャンの開始／停止 6.2　スキャンパラメータ 6.3　スキャンで受信する情報 6.4.4　Advertising Data によるフィルタリング 7.1　セントラルのコネクション手順 7.3.1　複数のペリフェラルデバイスとコネクションする 8.1　PHY の変更 8.4　MTU の変更 9.1.1　ペアリングパラメータ設定 9.1.4　ペアリングの要求 9.3.1　暗号化要求 9.4　プライバシ 10.2.2　Write 動作 10.3.1　クライアント側からデータを送信する実装例
Multi-role サンプル	ble_demo_tbrx23w_multirole_sample	5.5　抽象 API による Advertising 5.6　スマートフォンとコネクション 6.1　スキャンの開始／停止 6.2　スキャンパラメータ 6.3　スキャンで受信する情報 6.4.4　Advertising Data によるフィルタリング 7.1　セントラルのコネクション手順 7.3　複数台コネクション 8.1　PHY の変更 8.4　MTU の変更 9.1.1　ペアリングパラメータ設定 9.1.4　ペアリングの要求 9.3.1　暗号化要求 9.4　プライバシ 10.2.2　Write 動作 10.2.4　Notification 動作 10.3　GATT プロシージャの実装方法

サンプルアプリケーションの動作環境を表 12.2 に示します。BLE FIT, QE utility 以外は RX Driver Package v1.29 (https://www.renesas.com/software-tool/rx-driver-package)の FIT を使用しています。

表 12.2　サンプルアプリケーションの動作環境

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio V7.8.0 ルネサスエレクトロニクス製 e² studio 2021-04
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V2.08.00
ボード	Target Board for RX23W (以下、TB)
BLE FIT version	2.11
QE utility version	1.10
BSP version	5.63
LPC FIT version	2.01
CMT FIT version	4.70
IRQ FIT version	3.60
GPIO FIT version	3.70
SCI FIT version	3.70
BYTE_Q FIT version	1.82

e² studio にサンプルアプリケーションをインポートする方法を以下に説明します。

(1) スマート・ブラウザー上のアプリケーション開発者ガイド (タイトル：RX23W グループ Bluetooth Low Energy アプリケーション開発者ガイド，ドキュメント No：R01AN5504JJYYYY (YYYY はバージョン))を右クリックし、”サンプル・コード(プロジェクトのインポート)”を選択します。

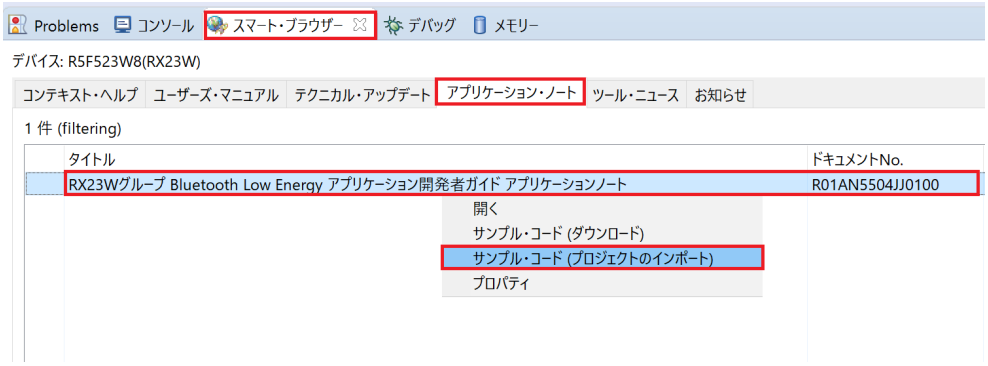


図 12-1：サンプルアプリケーションのプロジェクトのインポート

(2) r01an5504xxYYYY-rx23w-ble-adev.zip(YYYY はバージョン)をダウンロードする場所を指定し、ダウンロードします。ダウンロード完了後、”インポートするパッケージの選択”ウインドウが表示されます。ここでインポートするプロジェクトを選択します。

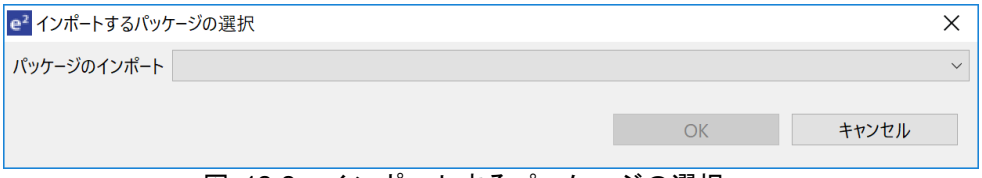


図 12-2：インポートするパッケージの選択

(3) “インポート”ウインドウ上で終了ボタンを押すと、サンプルアプリケーションのプロジェクトがインポートされます。

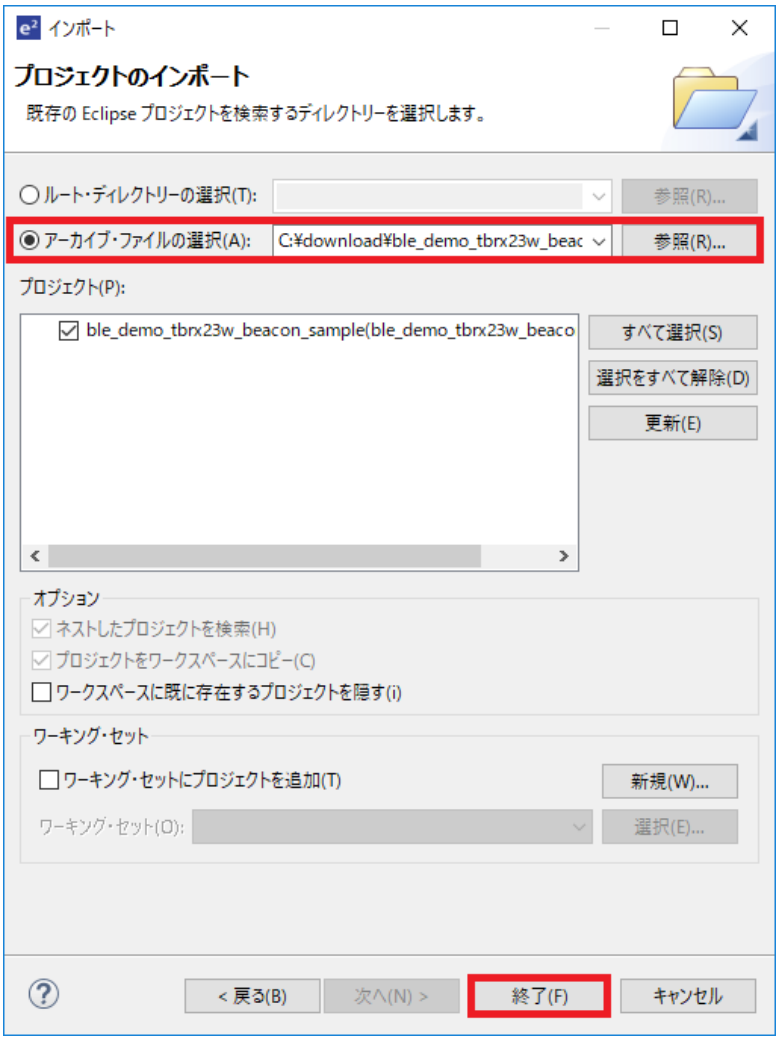


図 12-3 : プロジェクトのインポート

12.1 Beacon サンプル

12.1.1 対向デバイス

Beacon サンプルの Advertising の受信は以下のデバイスのスキャンで確認しています。

- iOS デバイス
- Android デバイス

12.1.2 動作

Beacon サンプルは起動後、Non connectable undirected advertising (ADV_NONCONN_IND)を開始します。

12.1.3 Advertising Data

Advertising Data のタイプは app_main.c の BLE_APP_BEACON_TYPE で以下のいずれかを指定します。

0 : iBeacon (default)
1 : Eddystone
2 : broadcast mode

各 Advertising Data について以下に説明します。

- iBeacon
  iBeacon の仕様は https://developer.apple.com/ibeacon で公開されています。
  Advertising Data は以下の内容を app_main.c 内で指定しています。

```
/* Advertising data */
static uint8_t gs_adv_data[] =
{
    /* TODO: Modify advertise data. Value of Data Flag is defined in
    https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile */

    /* Flag (mandatory) */
    2,          /**< Data Size */
    0x01,       /**< Data Type: Flag */
    (BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE | BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED),    /**< Data Value */

    /* Manufacturer data */
    0x1A,       /**< Data Size */
    0xFF,       /**< Data Type: Manufacturer data */
    0x4C, 0x00, /**< Company ID: Apple */
    0x02, 0x15, /**< Beacon Type:  */
    0x32, 0x46, 0x6a, 0x3a, 0x75, 0x52, 0xdb, 0xb4,
    0x97, 0x49, 0x19, 0x70, 0xd8, 0x56, 0x98, 0xaa,  /**< UUID: 32466a3a-7552-dbb4-9749-1970d85698aa */
    0x00, 0x01, /**< Major: 1 */
    0x00, 0x00, /**< Minor: 0 */
    0x00, /**< Measured Power:  */
};
```

コード 12-1　iBeacon 用のデフォルトの Advertising Data

UUID は開発するアプリケーション固有のものを指定します。
Major, Minor バージョンは開発するアプリケーションのバージョンを指定します。
Measured Power は iBeacon 仕様で指定された方法で計測し、指定します。

- Eddystone
  Eddystone の仕様は https://github.com/google/eddystone で公開されています。
  Beacon サンプルでは Eddystone-URL タイプを用意しています。本サンプルでは Eddystone Configuration GATT Service をサポートしていません。Advertising Data は以下の内容を app_main.c 内で指定しています。

```
/* Advertising data */
static uint8_t gs_adv_data[] =
{
    /* TODO: Modify advertise data. Value of Data Flag is defined in
    https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile */

    /* Flag (mandatory) */
    2,          /**< Data Size */
    0x01,       /**< Data Type: Flag */
    (BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE | BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED),    /**< Data Value */

    /* Complete list of 16bit Service UUIDs */
    3,          /**< Data Size */
    0x03,       /**< Data Type: Complete list of 16bit Service UUIDs */
    0xAA, 0xFE, /**< 16bit Eddystone UUID */

    /* Service Data */
    14,         /**< Data Size */
    0x16,       /**< Data Type: Service Data */
    0xAA, 0xFE, /**< 16bit Eddystone UUID */
    0x10,       /**< Frame Type: URL */
    0x00,       /**< Tx power: 0 dBm */
    0x01,       /**< URL Scheme: https://www. */
    0x72, 0x65, 0x6E, 0x65, 0x73, 0x61, 0x73, 0x07 /**< Encoded URL: renesas.com */
};
```

コード 12-2　Eddystone 用のデフォルトの Advertising Data

Service Data の Data Size, URL Scheme, Encoded URL については開発するアプリケーションに合わせて変更してください。

- broadcast mode
  Bluetooth Core Specification で定義された Broadcast Mode の条件を満たすように、Flags に”LE General Discoverable Mode”と”LE Limited Discoverable Mode”を含めない内容となっています。

```
/* Advertising Data */
static uint8_t gs_adv_data[] =
{
    /* TODO: Modify advertise data. Value of Data Flag is defined in
https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile */

    /* Flag (mandatory) */
    2,          /**< Data Size */
    0x01,       /**< Data Flag: Flag */
    BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED,    /**< Data Value */

    /* Complete Local Name */
    9,         /**< Data Size */
    0x09,       /**< Data Flag: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'A', 'D', 'V', /**< Data Value */

};
```

コード 12-3　broadcast mode 用のデフォルトの Advertising Data

12.1.4 コンフィグレーションオプション

Beacon サンプル用にデフォルトから変更した BLE FIT のコンフィグレーションオプションを表 12.3 に示します。

表 12.3　変更したコンフィグレーションオプション

マクロ名 (SC 表示)	設定値
BLE_CFG_LIB_TYPE (Type of Bluetooth LE Protocol Stack library)	2 : Compact
BLE_CFG_RF_CONN_MAX (Maximum number of connections)	1
BLE_CFG_CMD_LINE_EN (Enabled/Disabled command line function)	1 : 有効
BLE_CFG_CMD_LINE_CH (SCI CH for command line function)	8
BLE_CFG_BOARD_LED_SW_EN (Enabled/Disabled board LED and Switch control support)	1 : 有効
BLE_CFG_BOARD_TYPE (Board Type)	1 : Target Board

12.1.5 変更可能なパラメータ

(1)　アドレス

アドレスは app_main.c の BLE_BEACON_ADDR_TYPE で以下のいずれかを指定します。

BLE_GAP_ADDR_RAND　　: Static Address (default)
BLE_GAP_ADDR_RPA_ID_RANDOM　: RPA(static) address

(2)　Advertising Data、Advertising パラメータ

「12.1.3 Advertising Data」で記載したそれぞれの仕様に準拠する範囲で app_main.c の gs_adv_data (Advertising Data), gs_adv_param (Advertising パラメータ)を変更することが可能です。

12.1.6 コマンド

アドレスに BLE_GAP_ADDR_RPA_ID_RANDOM を指定する場合、現在のアドレスを表示するために、以下のコマンドを用意しています。

```
Beacon lrpa
```

12.2 Peripheral サンプル

12.2.1 対向デバイス

Peripheral サンプルの対向デバイスは以下となります。

- Central サンプル
- Multi-role サンプル
- iOS デバイス
- Android デバイス

12.2.2 動作

Peripheral サンプルは以下のような動作を行います。

- 起動後、Connectable and scannable undirected advertising (ADV_IND)の Advertising を開始します。最初の 30 秒間 fast Advertising (interval : 30ms)を行い、その後、slow Advertising (interval : 1000ms)を行います。
- リモートデバイスから Scan を行うと、”RBLE-P-DEV”という名前で検出されます。

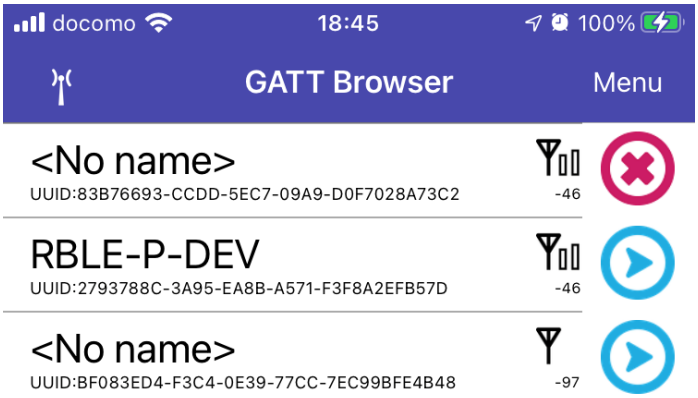


図 12-4 リモートデバイスの Scan 画面

- 接続すると、同時接続サポートなしの場合は Advertising が停止します。同時接続サポートありの場合は Advertising を継続します。同時接続サポートの変更方法は 12.2.4(3)をご参照ください。
- リモートデバイスから GATT サービス検索を行うと、以下が検出されます。

表 12.4　検出されるサービス、キャラクタリスティックと UUID

サービス，キャラクタリスティック	UUID
LED Switch サービス	58831926-5F05-4267-AB01-B4968E8EFCE0
Switch State キャラクタリスティック	58837F57-5F05-4267-AB01-B4968E8EFCE0
LED Blink Rate キャラクタリスティック	5883C32F-5F05-4267-AB01-B4968E8EFCE0

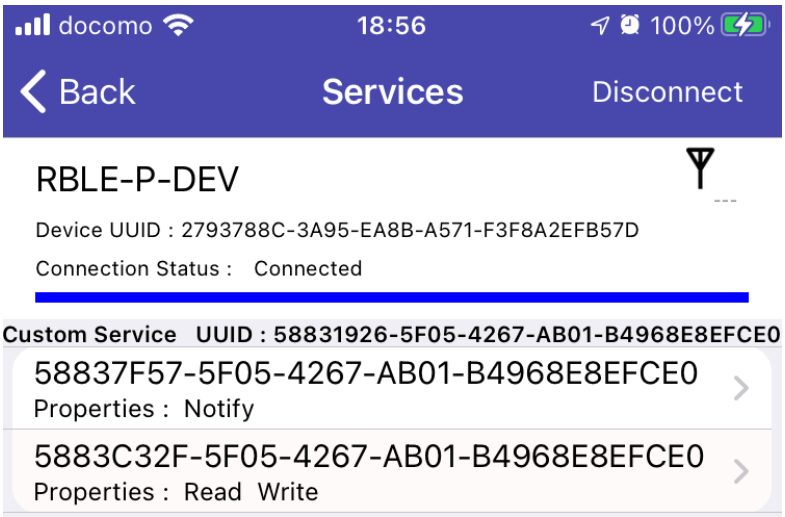


図 12-5 リモートデバイスのサービス検出画面

- gatt_db.c の gs_gatt_service 内の LED Switch サービスの設定にて、2 番目のパラメータを BLE_GATT_DB_SER_SECURITY_UNAUTH | BLE_GATT_DB_SER_SECURITY_ENC に設定しています。そのため、リモートデバイスから LED Switch サービスのキャラクタリスティックへアクセスする場合にペアリングが必要となります。

```
Static const st_ble_gatts_db_serv_cfg_t gs_gatt_service[] =
{
    /* 省略 */
    /* LED Switch Service */
    {
        /* Num of Services */
        {
            1,
        },
        /* Description */
        BLE_GATT_DB_SER_SECURITY_UNAUTH | BLE_GATT_DB_SER_SECURITY_ENC,
        /* Service Start Handle */
        0x0010,
        /* Service End Handle */
        0x0015,
        /* Characteristic Start Index */
        6,
        /* Characteristic End Index */
        7,
    },

};
```

コード 12-4　LED Switch サービスのセキュリティ設定

- リモートデバイスから Switch State キャラクタリスティックの Notification を有効にした後、ボード上の SW1 を押すと、Notification を送信します。
- リモートデバイスから LED Blink Rate キャラクタリスティックに値を書き込むと数値 x 100ms の間隔で LED の点滅が始まります。0 を書き込むと、LED が消灯します。
- 切断すると、Advertising を再開します。
- ボード上の SW1 を押下しながら、リセットボタンを押下すると、ボンディング情報を削除します。

12.2.3 コンフィグレーションオプション

Peripheral サンプル用にデフォルトから変更した BLE FIT のコンフィグレーションオプションを表 12.5 に示します。

表 12.5　変更したコンフィグレーションオプション

マクロ名 (SC 表示)	設定値
BLE_CFG_LIB_TYPE (Type of Bluetooth LE Protocol Stack library)	1 : Balance
BLE_CFG_RF_CONN_MAX (Maximum number of connections)	3
BLE_CFG_NUM_BOND (Number of remote device bonding information)	3
BLE_CFG_EN_SEC_DATA (Store Security Data in DataFlash)	1 : 有効
BLE_CFG_CMD_LINE_EN (Enabled/Disabled command line function)	1 : 有効
BLE_CFG_CMD_LINE_CH (SCI CH for command line function)	8
BLE_CFG_BOARD_LED_SW_EN (Enabled/Disabled board LED and Switch control support)	1 : 有効
BLE_CFG_BOARD_TYPE (Board Type)	1 : Target Board

12.2.4 変更可能なパラメータ

(1)　アドレス

アドレスは app_main.c の BLE_PERIPHERAL_ADDR_TYPE で以下のいずれかを指定します。

BLE_GAP_ADDR_RAND　　: Static Address (default)
BLE_GAP_ADDR_RPA_ID_RANDOM　: RPA(static) address

(2)　Advertising Data、スキャンレスポンスデータ、Advertising パラメータ

app_main.c の gs_adv_data (Advertising Data)、gs_sres_data (スキャンレスポンスデータ)、gs_adv_param (Advertising パラメータ) は変更可能です。gs_adv_data、gs_sres_data 内のデバイス名を変更し、Central サンプルとの接続を行う場合、Central サンプルのフィルタについても変更する必要があります。

(3)　同時複数台接続サポート

app_main.c の BLE_PERIPHERAL_MULTI_CONNS が有効になっている場合、複数の Central デバイスとの同時接続をサポートします。

12.3 Central サンプル

12.3.1 対向デバイス

Central サンプルの対向デバイスは以下となります。

- Peripheral サンプル

12.3.2 動作

Central サンプルは以下のような動作を行います。

- 起動後、Peripheral サンプルを検出するため、Scan を開始します。最初の 10 秒間 fast scan (interval : 60ms, window : 30ms)を行い、その後、10 秒間 slow scan (interval : 1200ms, window : 11.25ms)を行います。Scan 停止後、ボード上の SW1 を押下することで Scan を再開します。
- 検出した Peripheral サンプルに対して接続要求を送信します。Peripheral サンプルを検出すると、Scan を停止します。
- 接続完了後、パケット長更新を行います。
- パケット長更新後、MTU 変更要求をリモートデバイスに送信します。
  PHY を変更する場合、MTU 変更要求を行う前に PHY 変更要求をリモートデバイスに送信します。
- リモートデバイスから MTU 変更要求への応答を受信すると、表 12.4 の LED Switch サービスの検索を開始します。
- サービス検索完了後、Peripheral サンプルの Switch State キャラクタリスティックの CCCD に 1 を書き込みます。
- このときにペアリング済みでない場合、Peripheral サンプルからエラーが通知され、このエラーを受信後、ペアリングを開始します。ペアリング済みで暗号化済みでない場合にもエラーが通知され、このエラーを受信後、暗号化を行います。
- ペアリング、または暗号化が完了すると再度、Peripheral サンプルの Switch State キャラクタリスティックの CCCD に 1 を書き込みます。ペアリング、暗号化が完了している場合、書き込みに成功します。
- この状態で Peripheral サンプルのボード上の SW1 を押下すると Switch State が Notification で送信されます。
- ボード上の SW1 を押下しながら、リセットボタンを押下すると、ボンディング情報を削除します。

12.3.3 コンフィグレーションオプション

Central サンプル用にデフォルトから変更した BLE FIT のコンフィグレーションオプションを表 12.6 に示します。

表 12.6　変更したコンフィグレーションオプション

マクロ名 (SC 表示)	設定値
BLE_CFG_LIB_TYPE (Type of Bluetooth LE Protocol Stack library)	1 : Balance
BLE_CFG_RF_CONN_MAX (Maximum number of connections)	3
BLE_CFG_NUM_BOND (Number of remote device bonding information)	3
BLE_CFG_EN_SEC_DATA (Store Security Data in DataFlash)	1 : 有効
BLE_CFG_CMD_LINE_EN (Enabled/Disabled command line function)	1 : 有効

マクロ名 (SC 表示)	設定値
BLE_CFG_CMD_LINE_CH (SCI CH for command line function)	8
BLE_CFG_BOARD_LED_SW_EN (Enabled/Disabled board LED and Switch control support)	1 : 有効
BLE_CFG_BOARD_TYPE (Board Type)	1 : Target Board

12.3.4 変更可能なパラメータ

(1) アドレス

アドレスは app_main.c の BLE_CENTRAL_ADDR_TYPE で以下のいずれかを指定します。

BLE_GAP_ADDR_RAND　　: Static Address (default)
BLE_GAP_ADDR_RPA_ID_RANDOM　: RPA(static) address

(2) スキャンパラメータ，コネクションパラメータ

app_main.c の gs_scan_phy_param、gs_scan_param(スキャンパラメータ)と gs_conn_phy_param、gs_conn_param (コネクションパラメータ)は変更可能です。Peripheral サンプルの Advertising Data，スキャンレスポンスデータ内のデバイス名を変更する場合、スキャンフィルタである、gs_filter_data についても変更します。

(3) PHY

接続確立後に PHY を 1M から 2M に変更する場合、app_main.c の BLE_APP_CHANGE_PHY_2M に 1 を設定します。BLE_APP_CHANGE_PHY_2M の初期値は 0 です。

12.4 Multi-role サンプル

12.4.1 トポロジ

Multi-role サンプルは Central デバイス、Peripheral サンプルと接続し、Central デバイスから書き込まれたキャラクタリスティックのブリッジ、あるいは、Peripheral サンプルから通知されたキャラクタリスティックのブリッジを行います。図 12-6 に Multi-role サンプルのトポロジを示します。

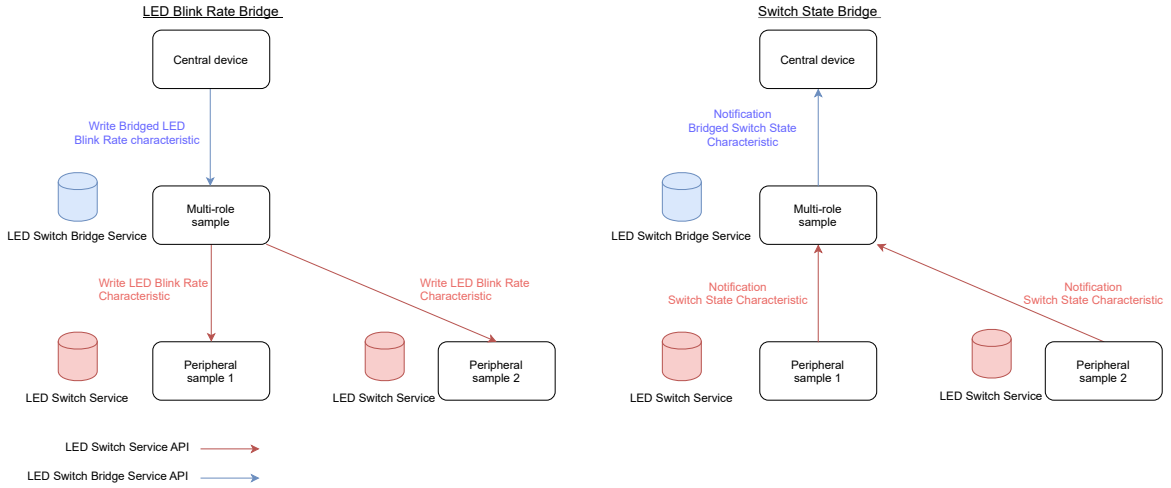


図 12-6 Multi-role サンプルの通信トポロジ

Central デバイスと Peripheral サンプル間のブリッジを実現するために、Multi-role サンプルは Bridged Switch State キャラクタリスティック(図 12-7)と Bridged LED Blink Rate キャラクタリスティック(図 12-8)を含む、LED Switch Bridge Service を追加しています。Bridged Switch State キャラクタリスティックではどの Peripheral サンプルから通知されたものかを示すためにアドレスを追加しています。Write 時は LED Switch Bridge Service のデータを LED Switch Service のデータに変換して Write します。Notification 時は LED Switch Service のデータを LED Switch Bridge Service のデータに変換して Notification します。

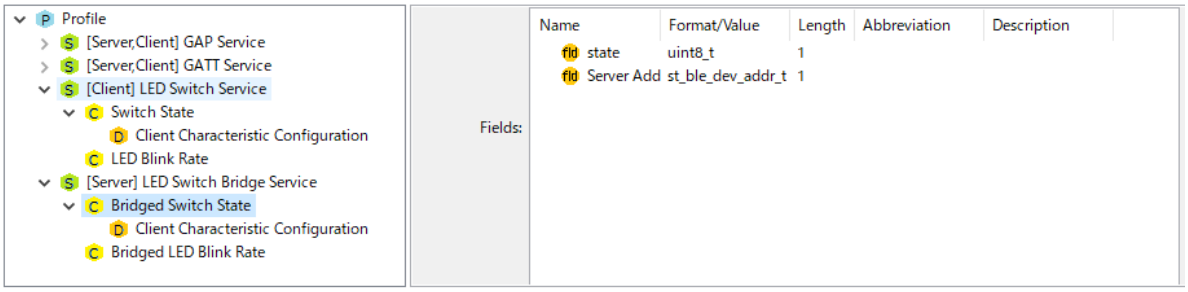


図 12-7 Bridged Switch State キャラクタリスティック

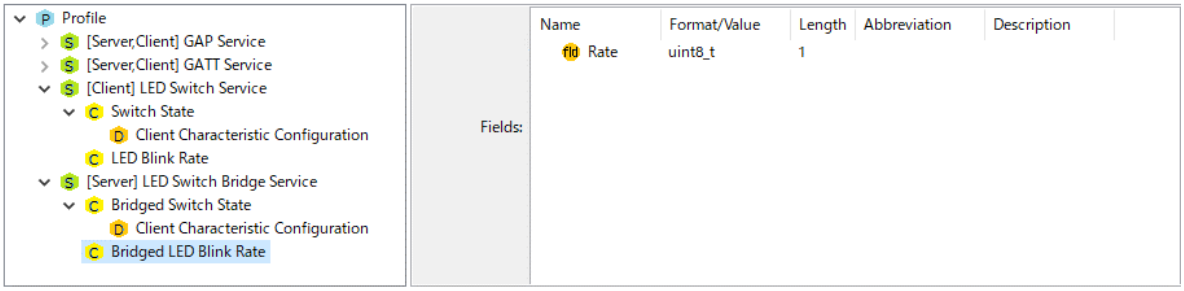


図 12-8 Bridged LED Blink Rate キャラクタリスティック

12.4.2 対向デバイス

Multi-role サンプルの対向デバイスは以下となります。

[Peripheral ロール] :
- iOS デバイス
- Android デバイス

[Central ロール] :
- Peripheral サンプル

12.4.3 動作

Multi-role サンプルは以下のような動作を行います。

[Central デバイスとの接続] :

- 起動後、Connectable and scannable undirected advertising (ADV_IND) の Advertising を開始します。最初の 30 秒間 fast Advertising (interval : 30ms)を行い、その後、slow Advertising (interval : 1000ms)を行います。
- リモートデバイスから Scan を行うと、”RBLE-MULTI-DEV”という名前で検出されます。

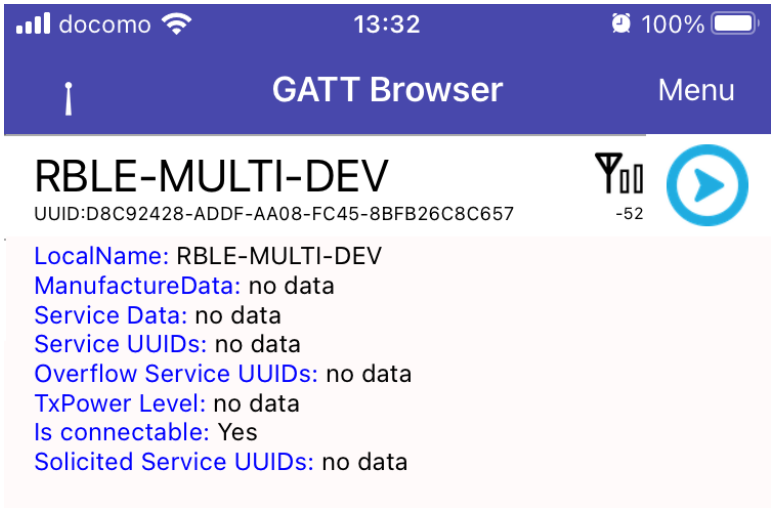


図 12-9 Central デバイスの Scan 画面

- 接続すると、Advertising が停止します。Multi-role サンプルでは、同時に接続可能な Central デバイスは 1 台のみとなります。
- リモートデバイスから GATT サービス検索を行うと、以下が検出されます。

表 12.7　検出されるサービス、キャラクタリスティックと UUID

サービス，キャラクタリスティック	UUID
LED Switch Bridge サービス	908DCB17-7F42-44AC-AB9D-C36F63DCEBD8
Bridged Switch State キャラクタリスティック	4CC8C6EC-3954-41D1-8CFF-3F2FE5EC0180
Bridged LED Blink Rate キャラクタリスティック	458B6862-6D2C-4356-8B2E-B88BCE7F0C84

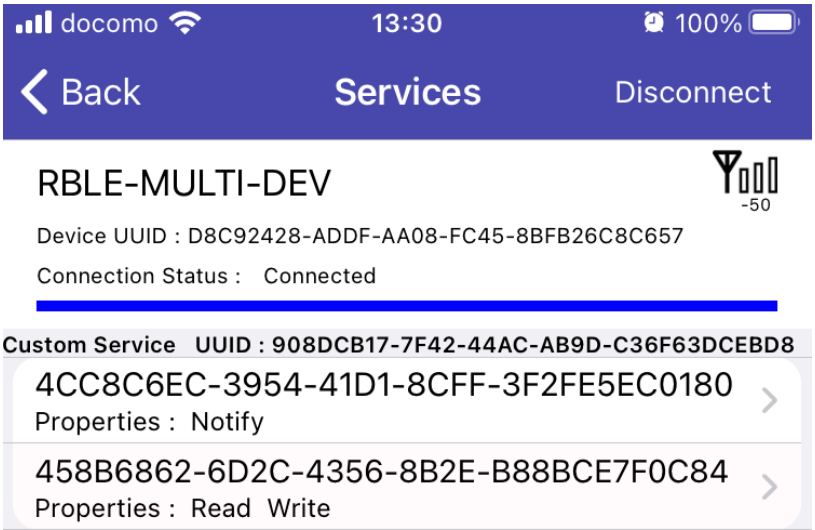


図 12-10 Central デバイスのサービス検出画面

- gatt_db.c の gs_gatt_service 内の LED Switch Bridge サービスの設定にて、2 番目のパラメータを BLE_GATT_DB_SER_SECURITY_UNAUTH | BLE_GATT_DB_SER_SECURITY_ENC に設定しています。そのため、Central デバイスから LED Switch Bridge サービスのキャラクタリスティックへアクセスする場合にペアリングが必要となります。

```
static const st_ble_gatts_db_serv_cfg_t gs_gatt_service[] =
{
    /* 省略 */
    /* LED Switch Bridge Service */
    {
        /* Num of Services */
        {
            1,
        },
        /* Description */
        BLE_GATT_DB_SER_SECURITY_UNAUTH | BLE_GATT_DB_SER_SECURITY_ENC,
        /* Service Start Handle */
        0x0010,
        /* Service End Handle */
        0x0015,
        /* Characteristic Start Index */
        6,
        /* Characteristic End Index */
        7,
    },
};
```

コード 12-5　LED Switch Bridge サービスのセキュリティ設定

- Central デバイスと切断すると、Advertising を再開します。

[Peripheral サンプルとの接続] :

- Peripheral サンプルを検出するため、ボード上の SW1 を押下すると、Scan を開始します。最初の 10 秒間 fast scan (interval : 60ms, window : 30ms)を行い、その後、10 秒間 slow scan (interval : 1200ms, window : 11.25ms)を行います。Scan 停止後、ボード上の SW1 を押下することで Scan を再開します。
- 検出した Peripheral サンプルに対して接続要求を送信します。Peripheral サンプルを検出すると、Scan を停止します。
- 接続完了後、パケット長更新を行います。
- パケット長更新後、MTU 変更要求をリモートデバイスに送信します。
  PHY を変更する場合、MTU 変更要求を行う前に PHY 変更要求をリモートデバイスに送信します。
- リモートデバイスから MTU 変更要求への応答を受信すると、LED Switch サービスの検索を開始します。

- サービス検索完了後、Peripheral サンプルの Switch State キャラクタリスティックの CCCD に 1 を書き込みます。
- このときにペアリング済みでない場合、Peripheral サンプルからエラーが通知され、このエラーを受信後、ペアリングを開始します。ペアリング済みで暗号化済みでない場合にもエラーが通知され、このエラーを受信後、暗号化を行います。
- ペアリング、または、暗号化が完了すると再度、Peripheral サンプルの Switch State キャラクタリスティックの CCCD に 1 を書き込みます。ペアリング、暗号化が完了している場合は書き込みに成功します。
- この状態で Peripheral サンプルのボード上の SW1 を押下すると Switch State が Notification で送信されます。

[ボンディング情報の削除] :

- ボード上の SW1 を押下しながら、リセットボタンを押下すると、ボンディング情報を削除します。

[Switch State キャラクタリスティックのブリッジ] :

下記 2 点の設定完了後、Peripheral サンプルのボード上の SW1 を押下すると、Multi-role サンプルが LED Switch Service の Switch State キャラクタリスティックを LED Switch Bridge Service の Bridged Switch State キャラクタリスティックに変換して動作するため、図 12-11 に示すように Central デバイスに Bridged Switch State キャラクタリスティックが通知されます。このキャラクタリスティックは図 12-7 に示すように Switch State と Peripheral サンプルのアドレスを含みます。

- Central デバイスと接続後、Central デバイスから Bridged Switch State キャラクタリスティックの CCCD に 1 を設定します。
- Peripheral サンプルと接続後、Multi-role サンプルが Switch State キャラクタリスティックの CCCD に 1 を自動設定します。

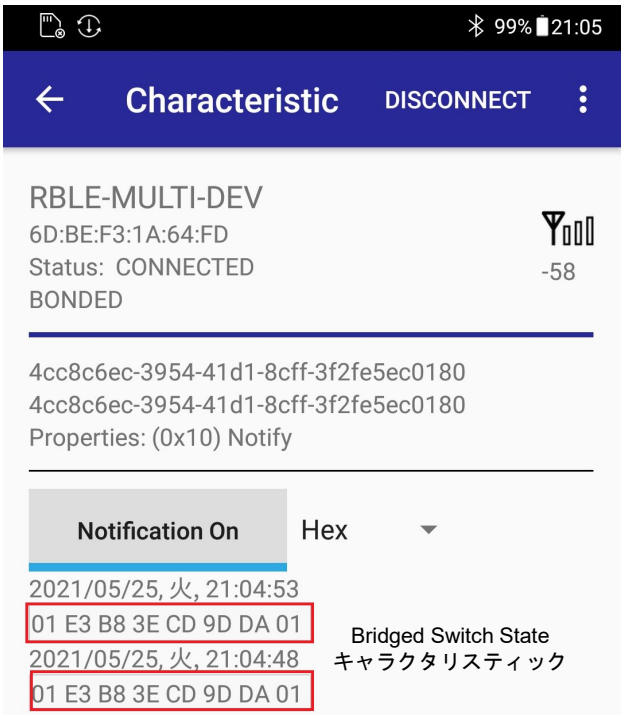


図 12-11 Central デバイスへの Bridged Switch State キャラクタリスティックの通知

[LED Blink Rate キャラクタリスティックのブリッジ] :

Central デバイスと Peripheral サンプルに接続後、図 12-12 に示すように Central デバイスから Bridged LED Blink Rate キャラクタリスティックに値を書き込むと、数値 x 100ms の間隔で Peripheral サンプルのボード上の LED の点滅が始まります。

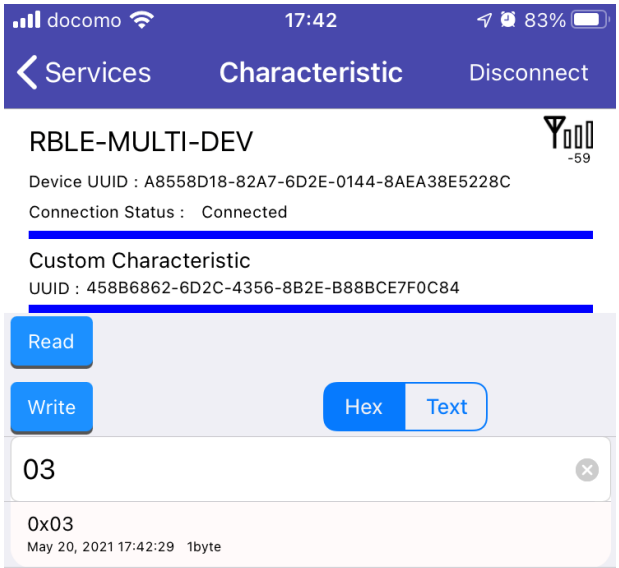


図 12-12 Central デバイスから Bridged LED Blink Rate キャラクタリスティックの書き込み

12.4.4 コンフィグレーションオプション

Multi-role サンプル用にデフォルトから変更した BLE FIT のコンフィグレーションオプションを表 12.8 に示します。

表 12.8　変更したコンフィグレーションオプション

マクロ名 (SC 表示)	設定値
BLE_CFG_LIB_TYPE (Type of Bluetooth LE Protocol Stack library)	1 : Balance
BLE_CFG_RF_CONN_MAX (Maximum number of connections)	3
BLE_CFG_NUM_BOND (Number of remote device bonding information)	3
BLE_CFG_EN_SEC_DATA (Store Security Data in DataFlash)	1 : 有効
BLE_CFG_CMD_LINE_EN (Enabled/Disabled command line function)	1 : 有効
BLE_CFG_CMD_LINE_CH (SCI CH for command line function)	8
BLE_CFG_BOARD_LED_SW_EN (Enabled/Disabled board LED and Switch control support)	1 : 有効
BLE_CFG_BOARD_TYPE (Board Type)	1 : Target Board

12.4.5 変更可能なパラメータ

(1) アドレス
アドレスは app_main.c の BLE_MULTIROLE_ADDR_TYPE で以下のいずれかを指定します。

BLE_GAP_ADDR_RAND　　: Static Address (default)
BLE_GAP_ADDR_RPA_ID_RANDOM　: RPA(static) address

(2) PHY
Central ロールとして接続確立後に PHY を 1M から 2M に変更する場合、app_main.c の BLE_APP_CHANGE_PHY_2M に 1 を設定します。BLE_APP_CHANGE_PHY_2M の初期値は 0 です。

(3) Advertising Data、スキャンレスポンスデータ、Advertising パラメータ
app_main.c の gs_adv_data (Advertising Data)、gs_sres_data (スキャンレスポンスデータ)、gs_adv_param (Advertising パラメータ) は変更可能です。

(4) スキャンパラメータ，コネクションパラメータ
app_main.c のスキャンパラメータである、gs_scan_phy_param、gs_scan_param とコネクションパラメータである、gs_conn_phy_param、gs_conn_param は変更可能です。Peripheral サンプルの Advertising Data、スキャンレスポンスデータ内のデバイス名を変更する場合、gs_filter_data についても変更します。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2020/07/28	–	初版発行
1.10	2021/05/31	84, 87, 89, 92, 96, 96, 98, 105	Advertising Data のサイズを修正
		149	表 9.5 の"あり"を"必要"に修正。 MITM プロテクション必要の場合に、両デバイスの入出力機能が Just Works となる組み合わせとなる場合、ペアリングに失敗することを追記。
		167	ローカルデバイスが RPA を使わず、リモートデバイスのアドレス解決を行う場合、all 0 の IRK を resolving list に登録することを追記。
		214	12. 付録 A：サンプルアプリケーションを追加。
		プログラム	サンプルアプリケーションを追加。
1.20	2022/06/30	21, 83	「抽象 API のコードは変更しないでください。」を追記。
		106-107	スキャンの抽象 API でのプライバシ有効時の設定を追記。
		116	コネクションハンドルの保持の方法を追記。
		118-118	コネクションの抽象 API でのプライバシ有効時の設定を追記。
		162-165	ボンディング数と同時最大接続数の関係の説明を追記。
		166-167	ボンディング情報の削除の説明を追記。
		167	ボンディングした相手とのみ接続する方法を追記。
		168-172	暗号化に失敗した場合のシーケンスについて追記。
		173-184	RPA の補足説明を追記。
1.30	2022/12/27	1, 7, 88, 107, 150, 153, 154, 156, 157, 159, 160, 168	実装者がセキュリティおよびプライバシに対してベストプラクティスな選択ができるように Bluetooth SIG が公開するガイド(Bluetooth® Security and Privacy Best Practices Guide)を基に推奨事項やリスクについて説明を追記。
		88, 101	RPA のデフォルト更新間隔と更新間隔を変更する API について追記。
		154	Secure Connection Only 指定において LE legacy pairing が開始した場合の通知内容について追記。
		175	ローカルデバイスが RPA を使う場合に関連付けるリモートデバイスの Identity Address と IRK の設定値を"ダミー"から"すべて 0x00"に表記を変更し、登録条件(未ペアリングの状態、あるいは、ローカルデバイスのみ RPA を使う場合)を削除。 関連するサンプルコードも"0xAA"と"0x55"から"0x00"に変更。
		8, 10, 25	QE for BLE V1.40 以降の QE Utility の提供形式を追記。
		16	「1.4.1 デバイスの識別」を追加。
		18	RF ハードウェアの状態の初期化について説明を追記。
		23, 29	ライブラリのタイプを選択する方法を追記。
		24	割り込み優先度について説明を追記。
		33	Public device address と Static address の注意書きを追記。
		37	R_BLE_Open とクロック周波数について説明を追記。
		48	R_BLE_Close と RF H/W の停止について説明を追記。
		49-56	result が BLE_SUCCESS 以外の場合の p_data について追記。
		86	Advertising 中のパラメータ変更不可を追記。
		87, 105, 116	Advertising/スキャン/コネクションの PHY について説明を追記。
		88, 100, 110, 117	ホワイトリスト使用中の設定不可を追記。
		88, 101	プライバシのための o_addr の説明を追記。
		135	切断理由について説明を追記。
		136	通信パラメータの変更ができるライブラリとロールを追記。

		137	変更を許可する PHY について説明を追記。
		160	ペアリング失敗時の result について説明を追記。
		174	プライバシの手順について図を追加。

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ないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。

4. 未使用端子の処理

   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。

5. クロックについて

   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

6. 入力端子の印加波形

   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。

7. リザーブアドレス（予約領域）のアクセス禁止

   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

8. 製品間の相違について

   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ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みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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

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

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

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。

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/

© 2022 Renesas Electronics Corporation. All rights reserved.