Bluetooth Low Energy スマートフォンサンプルアプリケーション
TryBT for Android

要旨

TryBT はルネサスエレクトロニクスの MCU である RX23W、RA4W1、または RE01B の評価ボードと Bluetooth® Low Energy 無線技術で通信できる Android サンプルアプリケーションです。本アプリはソースコードを含むサンプルプロジェクトとして配布され、ユーザはソースコードを変更して再利用できます。

本書は TryBT の開発環境の構築方法および TryBT の基本的なカスタマイズ方法を説明します。

対象デバイス

- Android 端末 (Android OS 6.0 以降)

関連ドキュメント

- RX23W グループ Target Board for RX23W クイックスタートガイド (R20QS0014)
- RX23W グループ Target Board for RX23W module クイックスタートガイド (R20QS0022)
- RA4W1 Group Evaluation Kit for RA4W1 EK-RA4W1 Quick Start Guide (R20QS0015)
- RE01B グループ Bluetooth Low Energy サンプルコード (using CMSIS Driver Package) (R01AN5606)
- Renesas Flash Programmer V3.08 フラッシュ書き込みソフトウェア ユーザーズマニュアル (R20UT4813)

目次

1. 概要

TryBT は Android 6.0 以降の Android 端末で動作します。評価ボードと Bluetooth Low Energy による接続を確立後、GATT サービスでデータ通信するサンプル画面を表示します。また本アプリのソースコードは改変可能な Android プロジェクトとして配布されます。

1.1 動作環境

TryBT の動作確認に必要なハードウェア:

- Android 端末 Android OS 6.0 以降
- Windows PC
- 下記のいずれかの評価ボード
  - Target Board for RX23W または Target Board for RX23W module 注 1
  - EK-RA4W1 注 2
  - EB-RE01B 注 3

注 1　Target Board for RX23W および Target Board for RX23W module には TryBT と通信可能なファームウェアが出荷時に書き込まれています。ファームウェアを再度書き込む場合は、以下のクイックスタートガイドの 5.1 節「出荷時ソフトウェアへの復元」を参照し、出荷時ファームウェアを書き込んでください。

RX23W グループ Target Board for RX23W クイックスタートガイド (R20QS0014)
→mot フォルダ内の ble_demo_tbrx23w_profile_server_preinstall_yyyymmdd.mot ファイル
RX23W グループ Target Board for RX23W module クイックスタートガイド(R20QS0022)
→mot フォルダ内の ble_demo_mtbrx23w_profile_server_preinstall_yyyymmdd.mot ファイル

注 2　EK-RA4W1 には TryBT と通信可能なファームウェアが出荷時に書き込まれています。ファームウェアを再度書き込む場合は、以下のクイックスタートガイドの 6 章「Restoring Factory Settings」を参照し、出荷時ファームウェアを書きこんでください。

RA4W1 Group Evaluation Kit for RA4W1 EK-RA4W1 Quick Start Guide (R20QS0015)
→bin.zip に含まれる Restore_Factory/r20qs0015.srec

注 3　EB-RE01B は出荷時にファームウェアが書き込まれていません。TryBT と通信可能なファームウェアを書き込む場合は、以下のドキュメントの 2.4 節「ファームウェア書き込み」を参照し、ファームウェアを書き込んでください。

RE01B グループ Bluetooth Low Energy サンプルコード (using CMSIS Driver Package) (R01AN5606)
→ROM_Files フォルダ内の ble_project_server.hex

以降、本書は評価ボードとして Target Board for RX23W を使用する場合の手順を記載しています。EK-RA4W1 または EB-RE01B を使用する場合も手順は同じです。

動作確認済みの Android 端末:

- Google Pixel 3a (Android OS 11)

TryBT の動作確認に必要なソフトウェア:

- Android Studio (入手方法は 2.1 節参照)

  注記: Java 11 以降の環境で一部ライブラリが廃止されたため、Android Gradle Plugin 4.1.0 以降を使用してください。

- Renesas Flash Programmer (入手方法は 5 章参照)

TryBT プロジェクト:

- 実装言語: Kotlin
- ビルド設定(build.gradle から抜粋)

```
android {
    compileSdkVersion 29
    buildToolsVersion "29.0.3"
    defaultConfig {
        applicationId "com.renesas.trybt"
        minSdkVersion 23
        targetSdkVersion 29
    }
}
dependencies {
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.core:core-ktx:1.3.2'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    implementation 'com.google.firebase:firebase-messaging:17.3.4'
    implementation 'com.jaredrummler:colorpicker:1.1.0'
    implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'
    implementation platform('com.google.firebase:firebase-bom:26.1.1')
    implementation 'com.google.firebase:firebase-analytics-ktx'
}
```

1.2　注意事項

- 本資料は 2021 年 3 月 3 日時点での動作確認結果に基づいて作成されました。本資料に記載された情報は、弊社または第三者が提供するソフトウェアおよびツールの全てのバージョンに対応することを保証するものではありません。
- 本資料に記載された情報およびソフトウェアの利用に起因して Windows PC、Android 端末に損害が発生した場合でも弊社は一切の責任を負いません。本資料末尾の「ご注意書き」もあわせてご確認ください。

2.　環境構築

Android アプリの開発環境である Android Studio をインストールします。本資料では Windows PC に Android Studio を初めてインストールする場合の手順を記載します。なお本資料内の画面は Android Studio 4.1.1 のものであり、異なるバージョンの Android Studio では画面が異なる場合があります。

2.1　Android Studio のダウンロード

1. 以下の URL にアクセスし、"DOWNLOAD ANDROID STUDIO"をクリックします。
   https://developer.android.com/studio?hl=ja

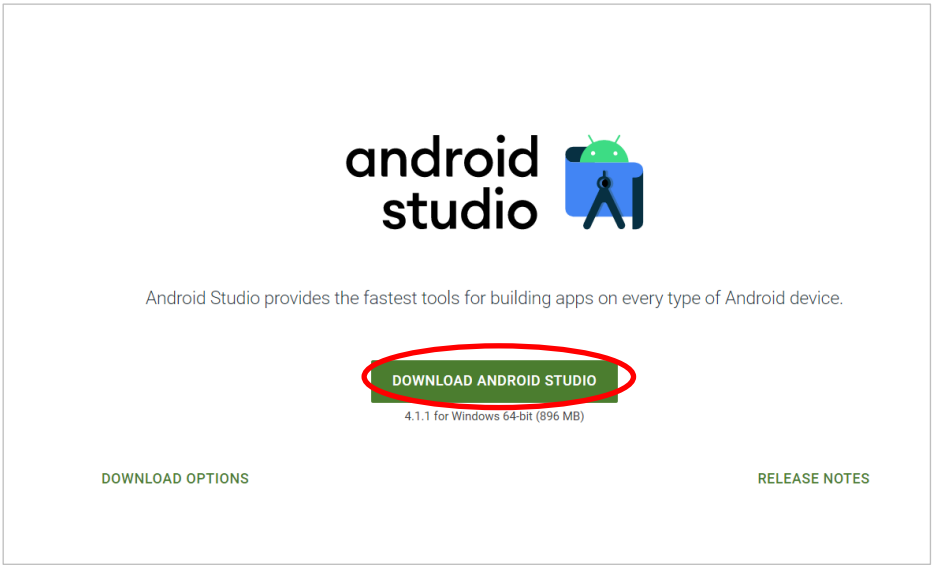

図 2-1　Android Studio のダウンロード(1)

2. 利用規約に同意して"DOWNLOAD"をクリックします。

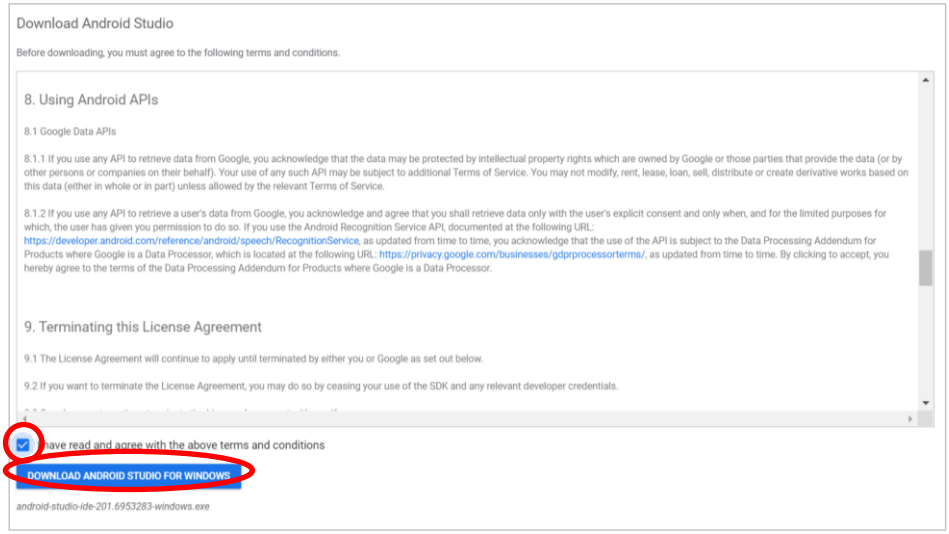

図 2-2　Android Studio のダウンロード(2)

2.2　Android Studio のインストール

1. ダウンロードした Android Studio のインストーラーを実行します。
2. "Next"をクリックします。

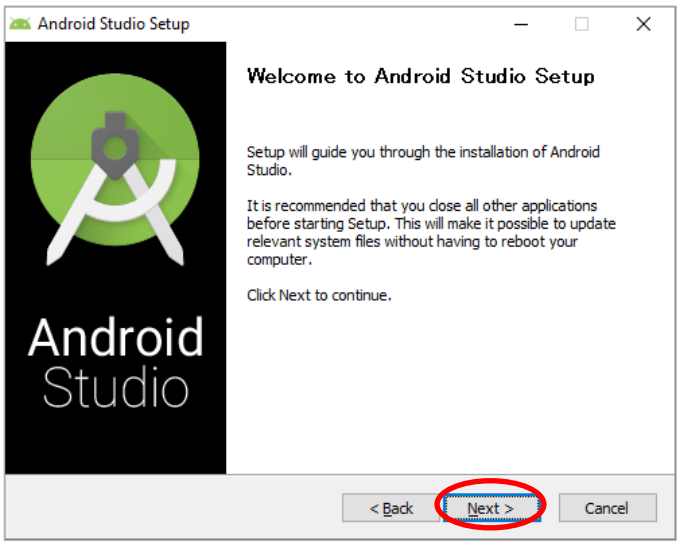

図 2-3　Android Studio のインストール(2)

3. "Android Virtual Device"のチェックを外して"Next"をクリックします。

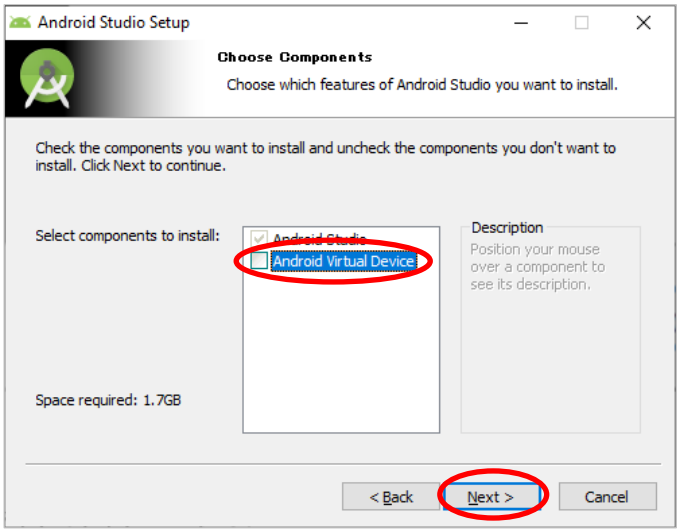

図 2-4　Android Studio のインストール(3)

4. インストール先を選択して"Next"をクリックします。

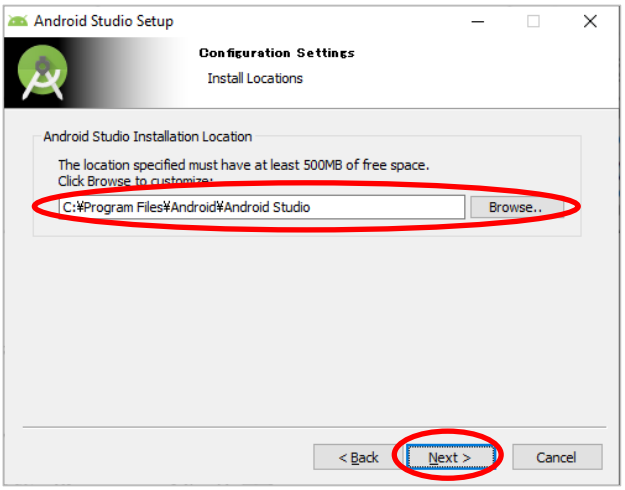

図 2-5　Android Studio のインストール(4)

5. "Install"をクリックします。

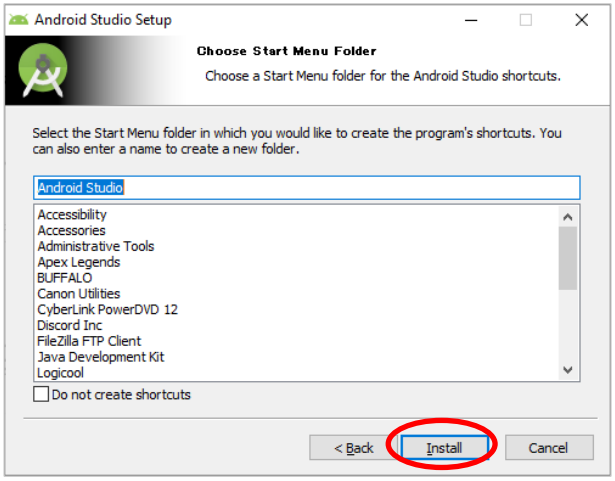

図 2-6　Android Studio のインストール(5)

6. インストールが完了したら"Next"をクリックします。

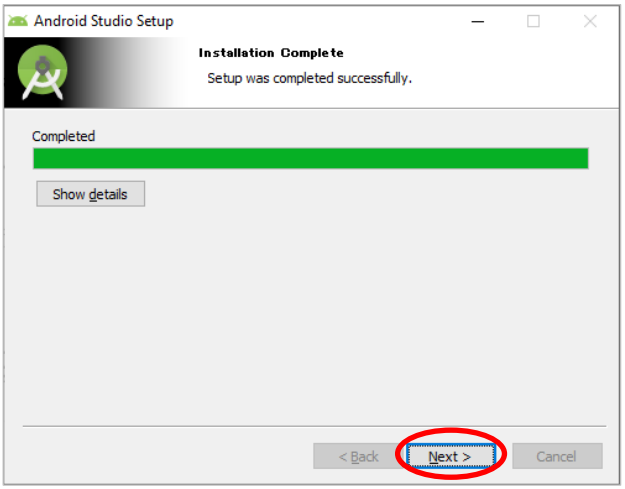

図 2-7　Android Studio のインストール(6)

7. "Finish"をクリックすると、Android Studio が起動します。

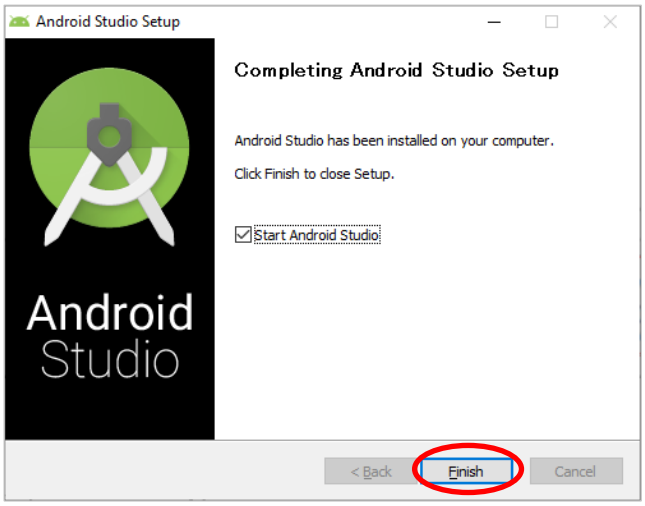

図 2-8　Android Studio のインストール(7)

8. "Do not import settings"を選択して"OK"をクリックします。

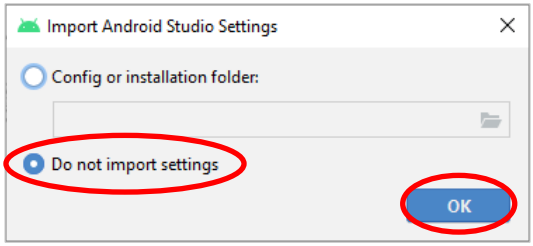

図 2-9　Android Studio のインストール(8)

9. "Next"をクリックします。

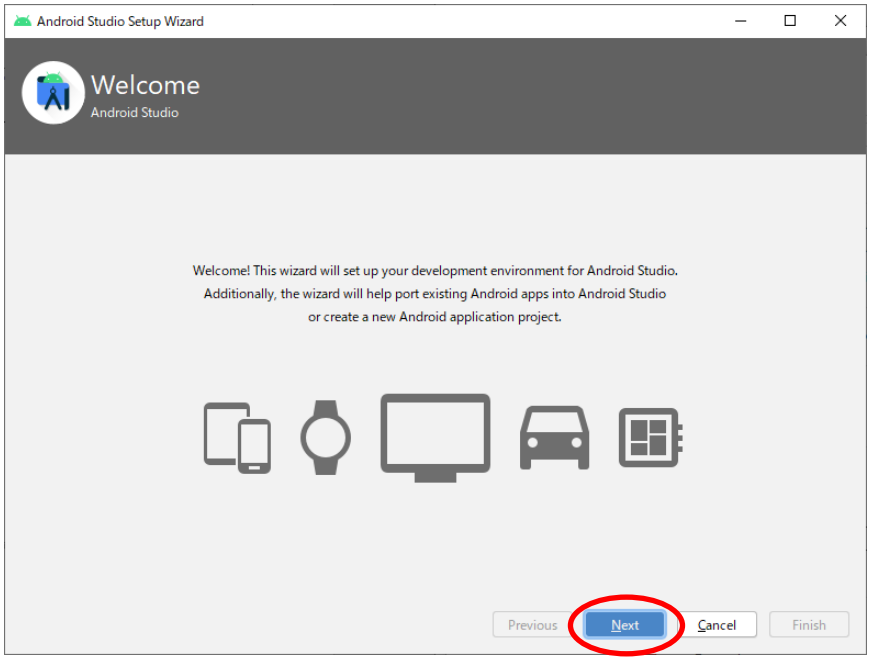

図 2-10　Android Studio のインストール(9)

10. "Next"をクリックします。

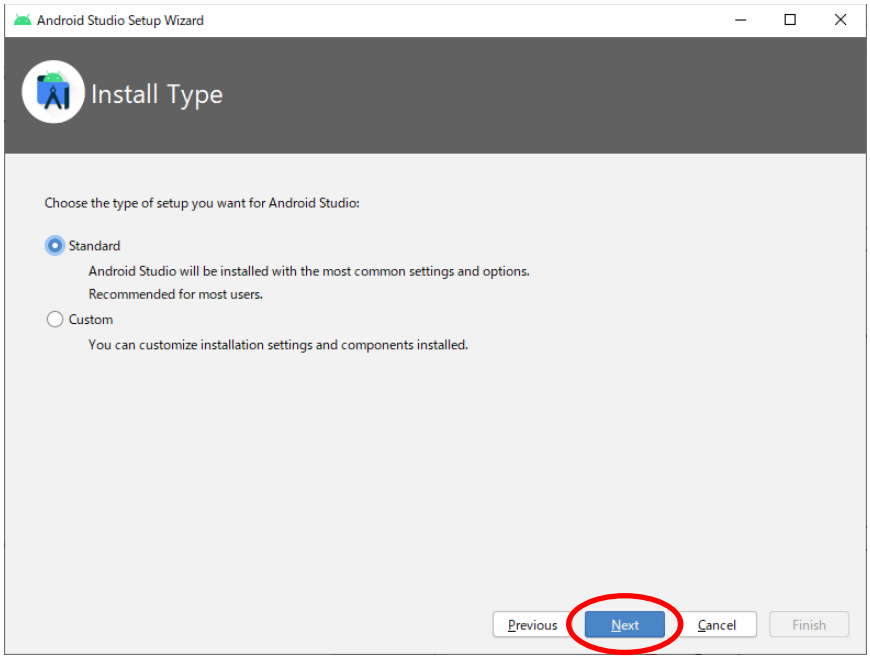

図 2-11　Android Studio のインストール(10)

11. "Next"をクリックします。

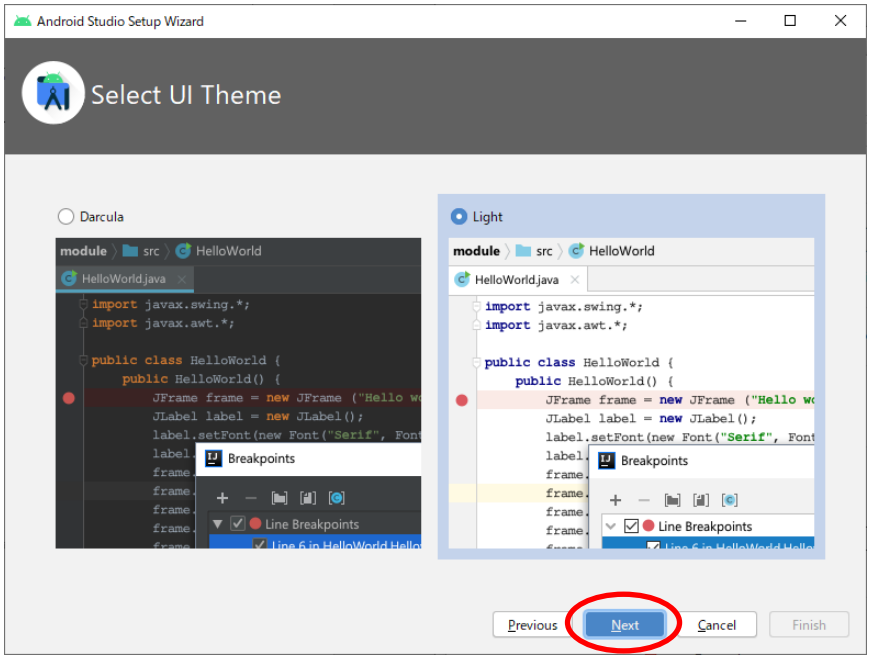

図 2-12　Android Studio のインストール(11)

12. コンポーネントのダウンロードが完了後、"Finish"をクリックします。

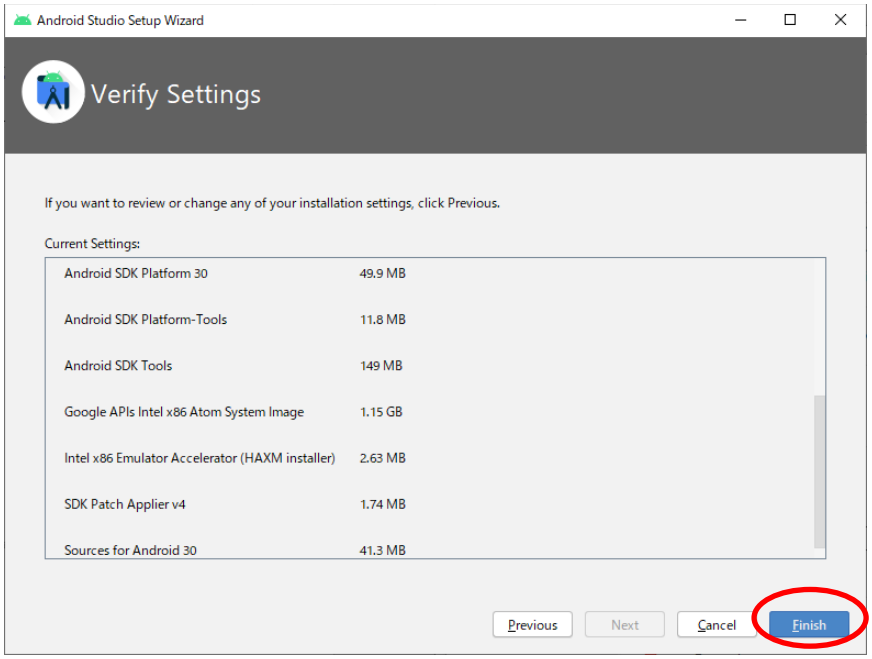

図 2-13　Android Studio のインストール(12)

2.3　TryBT プロジェクトのインポート

1. 本資料に添付された TryBT プロジェクトの zip ファイルを展開してください。
2. 展開したフォルダを任意の位置に移動してください。
3. Android Studio を起動します。
4. "Open an Existing Project"をクリックします。

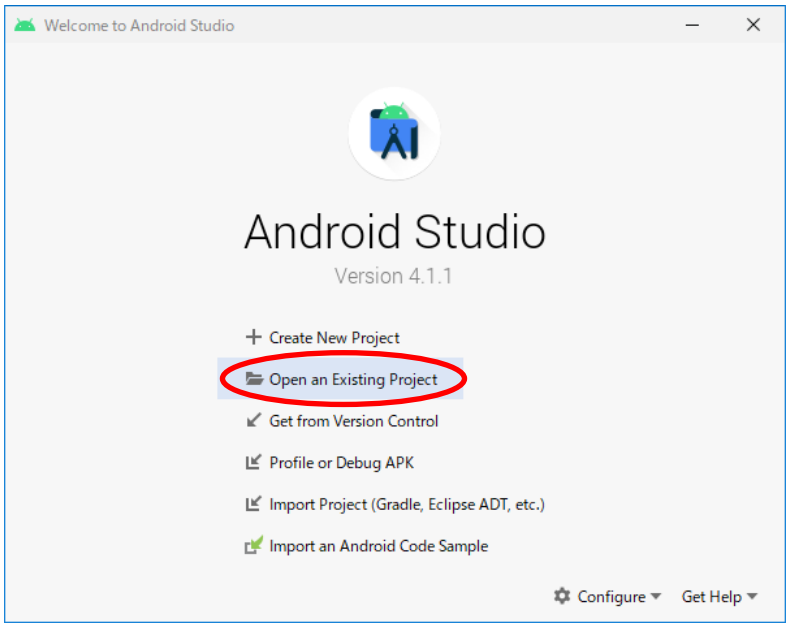

図 2-14　TryBT プロジェクトのインポート(1)

5. 手順 2 で展開した TryBT フォルダを指定してください。

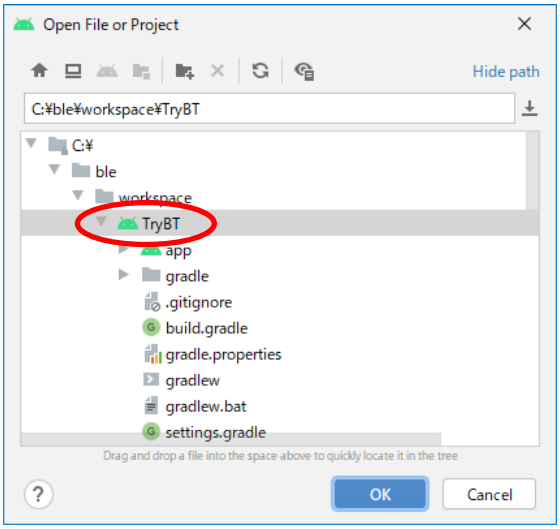

図 2-15　TryBT プロジェクトのインポート(2)

2.4　Android SDK Platform のインストール

1. Android Studio を起動します。
2. メニューの"SDK Manager"ボタンをクリックして、SDK Manager を起動します。

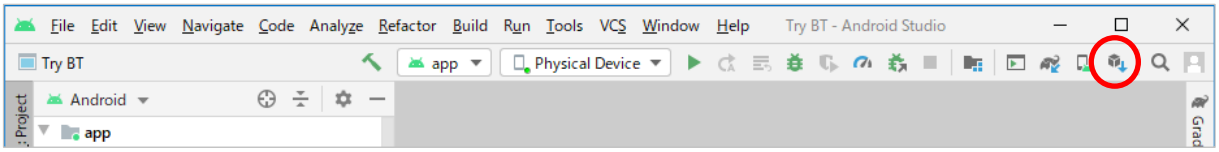

図 2-16　SDK Manager の起動

3. 旧バージョンの Android OS に対応するには、SDK Manger で旧バージョンの Android SDK をインストールする必要があります。例として Android OS 6.0 以降のバージョンに対応するには、SDK Manager で Android 6.0 (Marshmallow): API Level 23 をチェックして"Apply"をクリックします。

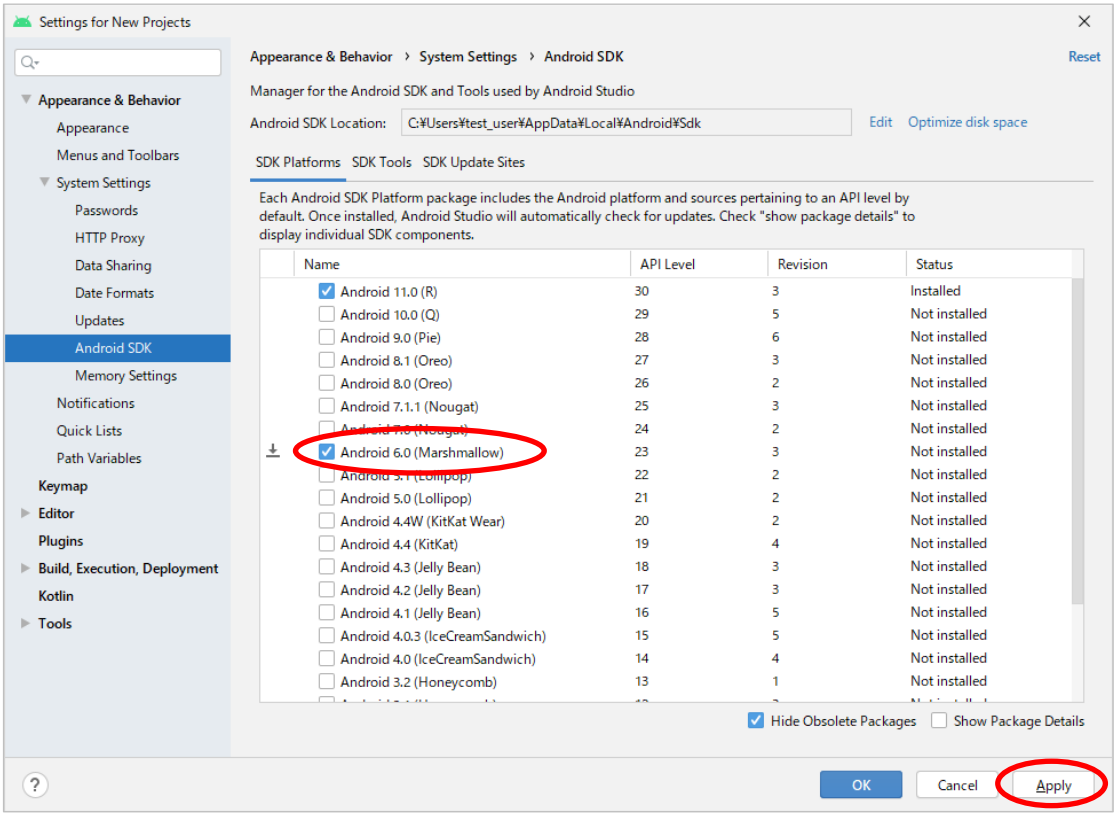

図 2-17　Android SDK Platform のインストール(1)

4. Confirm Change ダイアログで"OK"をクリックします。

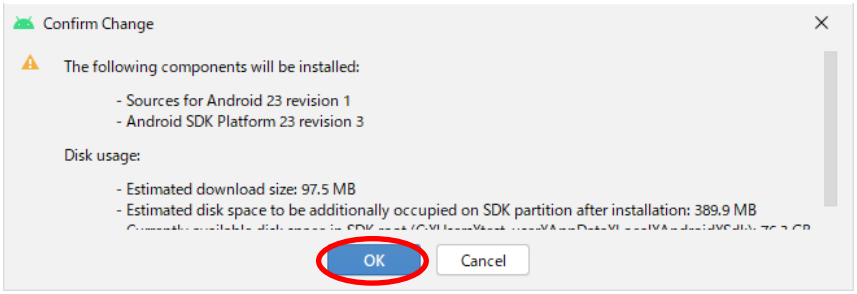

図 2-18　Android SDK Platform のインストール(2)

5. License Agreement ダイアログで"Accept"を選択し、"Next"をクリックします。

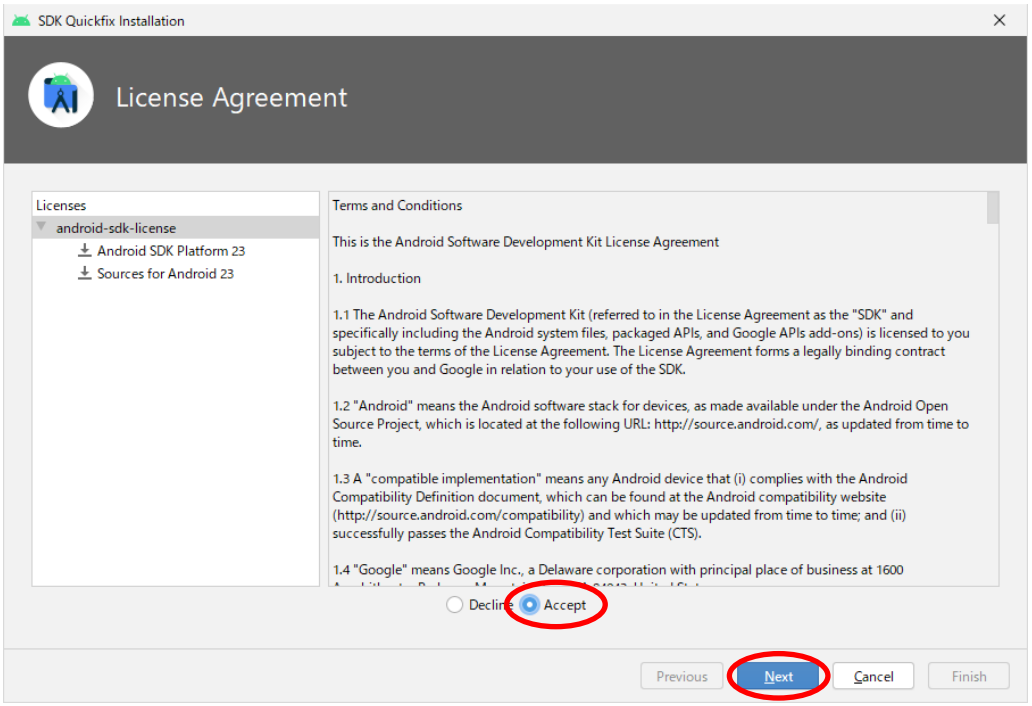

図 2-19　Android SDK Platform のインストール(3)

6. SDK のインストール完了後、"Finish"をクリックします。
7. SDK Manager の"OK"をクリックします。

3.　Android 端末を使用するための設定

TryBT は Android OS 6.0 以降の端末で動作します。本章では Google Pixel 3a を使用する場合の設定手順を例として記載します。

3.1　Android 端末用 USB ドライバのインストール

Android 端末の実機を使用してアプリを開発するために必要な USB ドライバを Windows PC にインストールします。

1. ブラウザで https://developer.android.com/studio/run/win-usb?hl=ja を開きます。
2. "こちらをクリックして Google USB ドライバの ZIP ファイルをダウンロード"をクリックし、利用規約に同意してダウンロードしてください。

図 3-1　USB ドライバのダウンロード

3. ダウンロードした USB ドライバの zip ファイルを展開します。
4. Android 端末を Windows PC に USB で接続します。
5. Device Manager を起動して Portable Devices に Pixel3 が表示されていることを確認します。

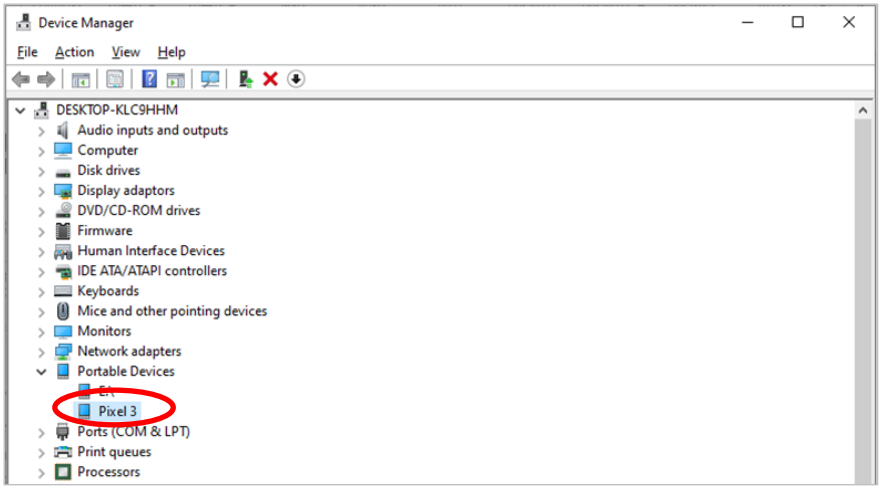

図 3-2　USB ドライバのインストール(1)

6. Pixel3 を右クリックして"Update drivers"を選択し、"Browse my computer for driver software"をクリックします。

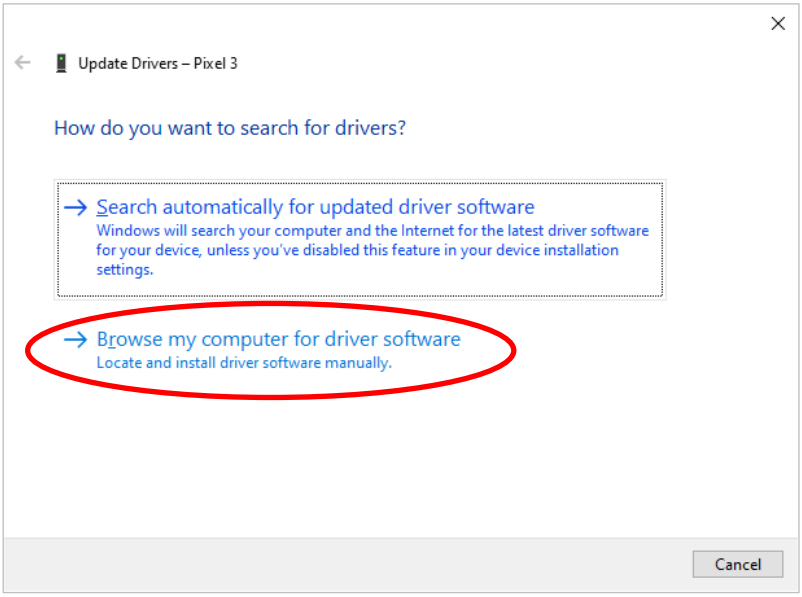

図 3-3　USB ドライバのインストール(2)

7. 手順 3 で展開したフォルダを指定して"Next"をクリックします。

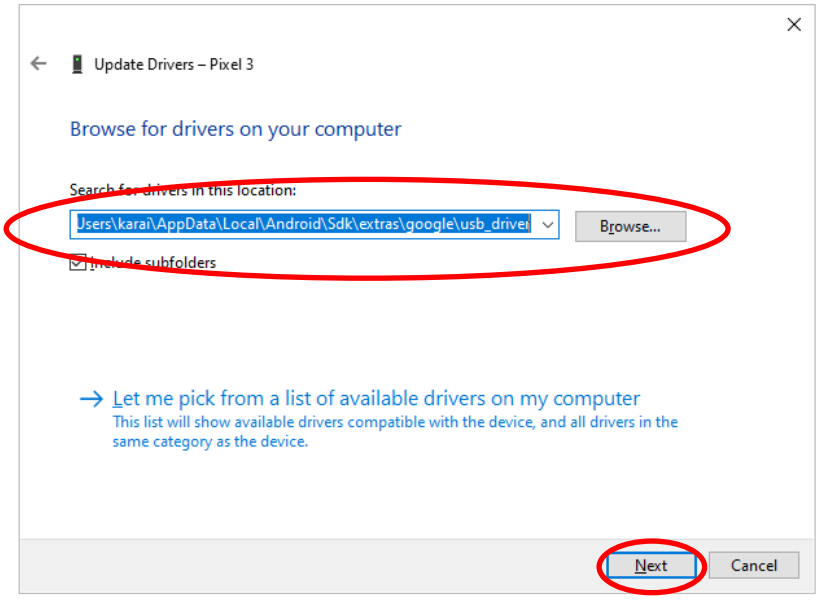

図 3-4　USB ドライバのインストール(3)

※USB ドライバの更新が必要ない場合は「このデバイスに最適なドライバが既にインストールされています」と表示されます。
※USB ドライバのインストールが正常終了しない場合は https://developer.android.com/studio/run/oem-usb?hl=ja も参照してください。

3.2　Android 端末の開発者モード設定

ビルドしたアプリをインストールするために、Android 端末を開発者モードに変更する必要があります。本資料では Google Pixel 3a の開発者モードへの変更手順を記載します。

1. 設定画面を開いて"デバイス情報"を選択します。

図 3-5　Android 端末の開発者モード設定(1)

2. "ビルド番号"を連続でタップし続けます。パスワードを入力すると開発者モードが解放されます。成功すると"これでデベロッパーになりました！"と表示されます。

図 3-6　Android 端末の開発者モード設定(2)

3. 設定画面に戻って"システム"→"開発者向けオプション"を選択します。

図 3-7　Android 端末の開発者モード設定(3)

4. USB デバッグをオンに設定してください。

図 3-8　Android 端末の開発者モード設定(4)

4.　TryBT のインストール

1. Android Studio を起動して、TryBT プロジェクトを開きます。
2. Android 端末を PC に USB 接続します。
3. Android 端末に表示されるダイアログで USB デバッグを許可します。
4. 実機を接続すると Android Studio の app の横に実機名が表示されます。
   ※初回接続時は実機名が表示されるまで長時間かかる場合があります。

図 4-1　TryBT のインストール(1)

5. 実機名の横の実行ボタンを選択すると、接続した実機に TryBT がインストールされます。
   ※初回実行時はインストール開始まで長時間かかる場合があります。

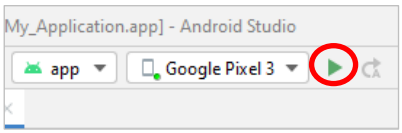

図 4-2　TryBT のインストール(2)

6. Android 端末で TryBT が起動すればインストールは完了です。

図 4-3　TryBT のスプラッシュ画面

5. 評価ボードへのファームウェア書き込み

評価ボードとして Target Board for RX23W を使用する例を説明します。その他のボードについては 1.1 節で紹介されているドキュメントを参照してください。

Target Board for RX23W には TryBT と通信可能なファームウェアが出荷時に書き込まれています。本章は Target Board for RX23W にファームウェアを再度書き込む場合の手順を示します。

1. 以下の URL にアクセスしてください。My Renesas アカウントでログイン後、免責事項に同意することで zip ファイルをダウンロードできます。
   https://www.renesas.com/document/scd/rx23w-group-target-board-rx23w-quick-start-guide-sample-code

2. 手順 1 でダウンロードした zip ファイルを展開してください。ビルド済みのファームウェアは下記の mot ファイルです。
   ./mot/ble_demo_tbrx23w_profile_server_preinstall_20191009.mot

   次ページでは、ビルド済みのファームウェアを Target Board for RX23W に書き込む手順を示します。また、ファームウェアの書き込みには以下のツールを使用します。
   Renesas Flash Programmer (Programming GUI)
   https://www.renesas.com/software-tool/renesas-flash-programmer-programming-gui

3. ファームウェアの書き込み時は Target Board for RX23W の ESW 1-2 を ON に変更して、PC と ECN1 コネクタを USB ケーブルで接続します。

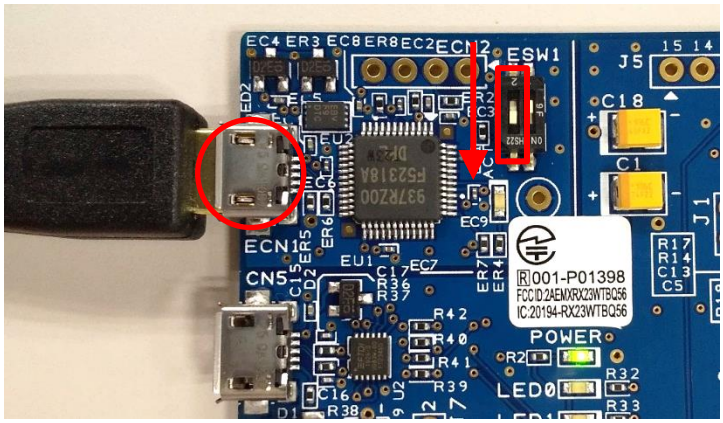

図 5-1　ファームウェア書き込み時の Target Board for RX23W の設定

4. Renesas Flash Programmer を起動して[ファイル]→[新しいプロジェクトを作成]を選択します。

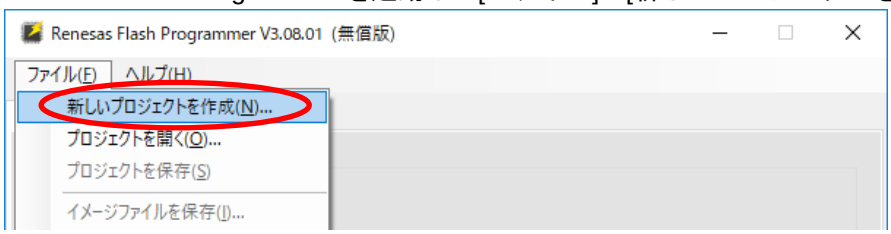

図 5-2　Target Board for RX23W へのファームウェア書き込み(1)

5. [新しいプロジェクトの作成]ダイアログで以下を設定して[接続]ボタンをクリックします。

マイクロコントローラ:　RX200
プロジェクト名:　任意のプロジェクト名
作成場所:　任意の場所
ツール:　E2 emulator Lite
インタフェース:　FINE
電源:　供給しない

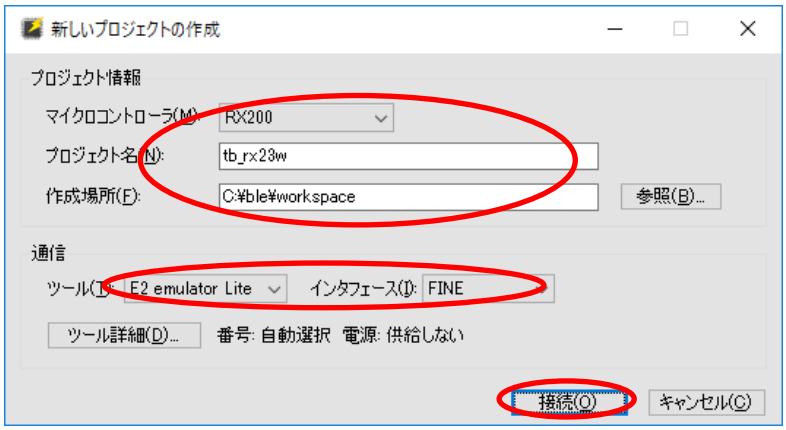

図 5-3　Target Board for RX23W へのファームウェア書き込み(2)

6. "操作が成功しました"と表示されれば、設定は完了です。
7. 手順 2 で展開したフォルダ内の下記ファームウェアを指定して[スタート]ボタンをクリックします。
   プログラムファイル:　mot/ble_demo_tbrx23w_profile_server_preinstall_20191009.mot

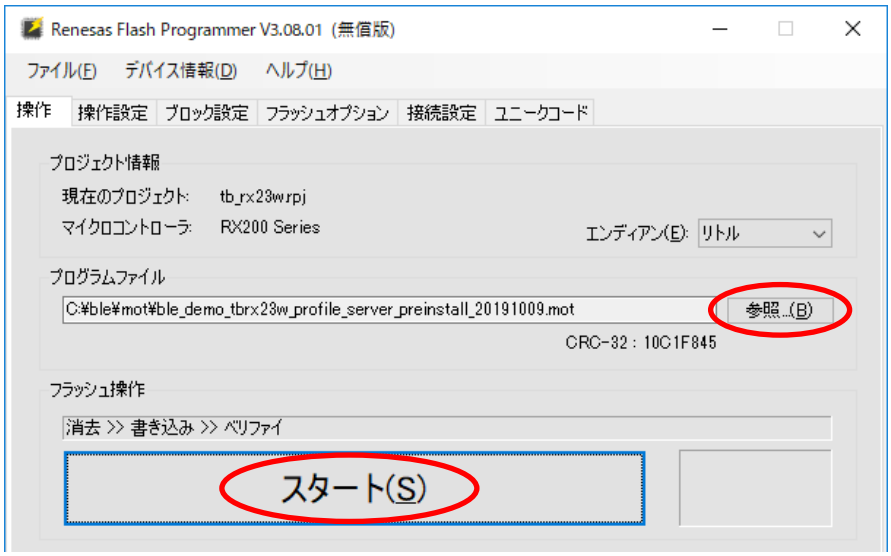

図 5-4　Target Board for RX23W へのファームウェア書き込み(3)

8. "操作が成功しました"と表示されれば、ファームウェアの書き込みは完了です。
9. PC から Target Board for RX23W を取り外します。
10. ファームウェアの実行時は Target Board for RX23W と ESW 1-2 を OFF に変更して、PC と CN5 コネクタを USB ケーブルで接続します。

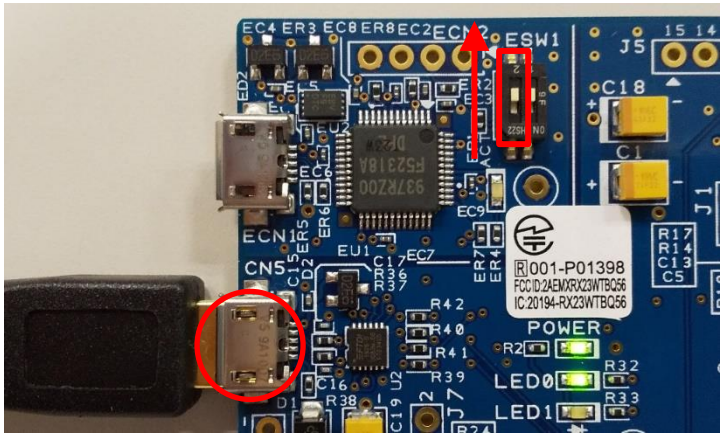

図 5-5　ファームウェア実行時の Target Board for RX23W の設定

6.　TryBT の基本操作

評価ボードとして Target Board for RX23W を使用する例を説明します。

6.1　デバイスー覧画面

アプリを起動するとデバイスー覧画面が表示されます。接続可能なデバイスと接続状況を表示します。
Target Board for RX23W はデバイスー覧画面で"RBLE-DEV"と表示されます。"RBLE-DEV"をタップすると、Target Board for RX23W との接続を確立し、接続デバイス詳細画面を表示します。

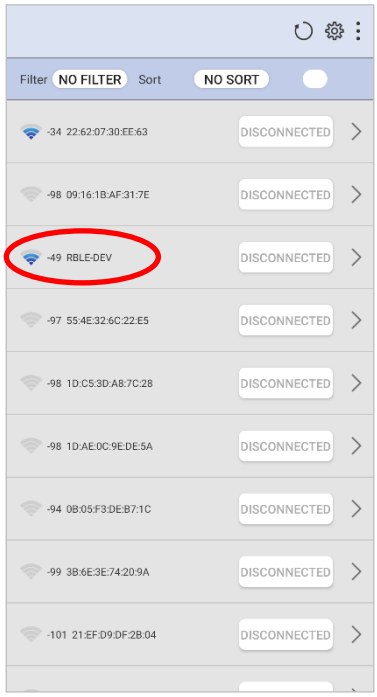

図 6-1　デバイスー覧画面(1)

"Filter type"を選択するとデバイスをフィルタリングできます。

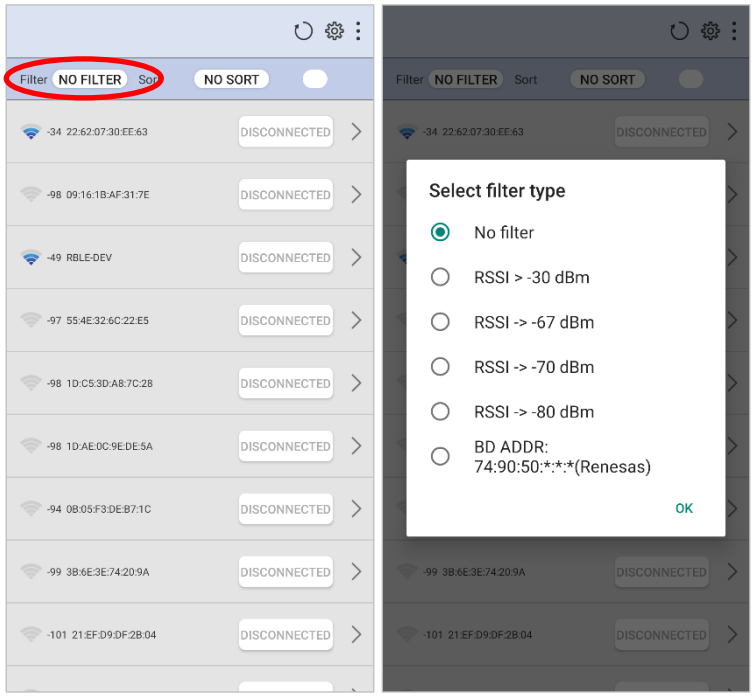

図 6-2　デバイスー覧画面(2)

"Sort order"を選択するとデバイスのソート順を指定できます。

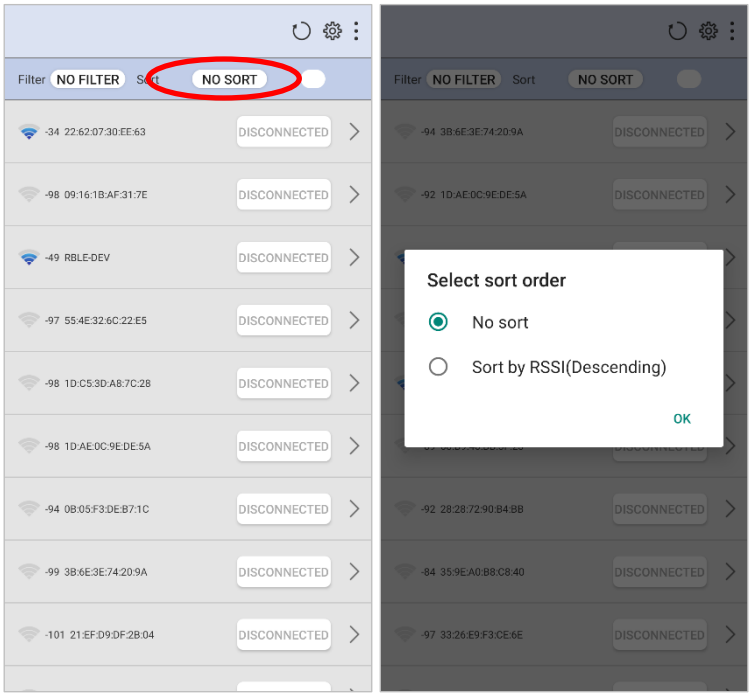

図 6-3　デバイス一覧画面(3)

リロードボタンをタップするとデバイス一覧を再読み込みできます。

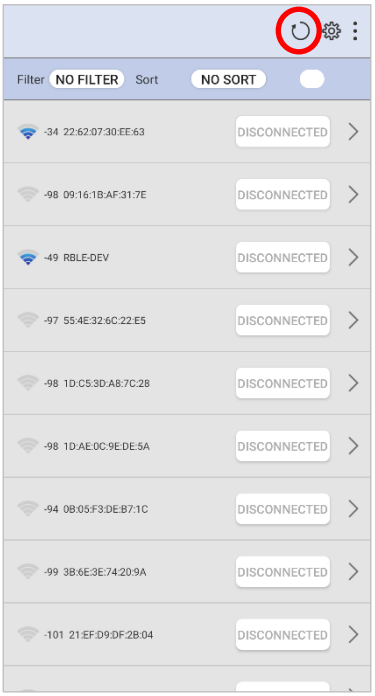

図 6-4　デバイス一覧画面(4)

設定ボタンをタップすると"Register UUID name"と"Bluetooth Settings"が表示されます。

"Register UUID name"を選択すると、GATT プロファイルのサービス名とその UUID を登録できます。設定されたサービス名は接続デバイス詳細画面の詳細情報で表示されます。なお本設定で登録できる UUID は１つのみです。表示例は図 6-11 を参照してください。

Target Board for RX23W に実装されているサービス名を表示する場合は以下のいずれかを登録します。

00001800-0000-1000-8000-00805f9b34fb: GAP Service
00001801-0000-1000-8000-00805f9b34fb: GATT Service
58831926-5f05-4267-ab01-b4968e8efce0: LED Switch Service

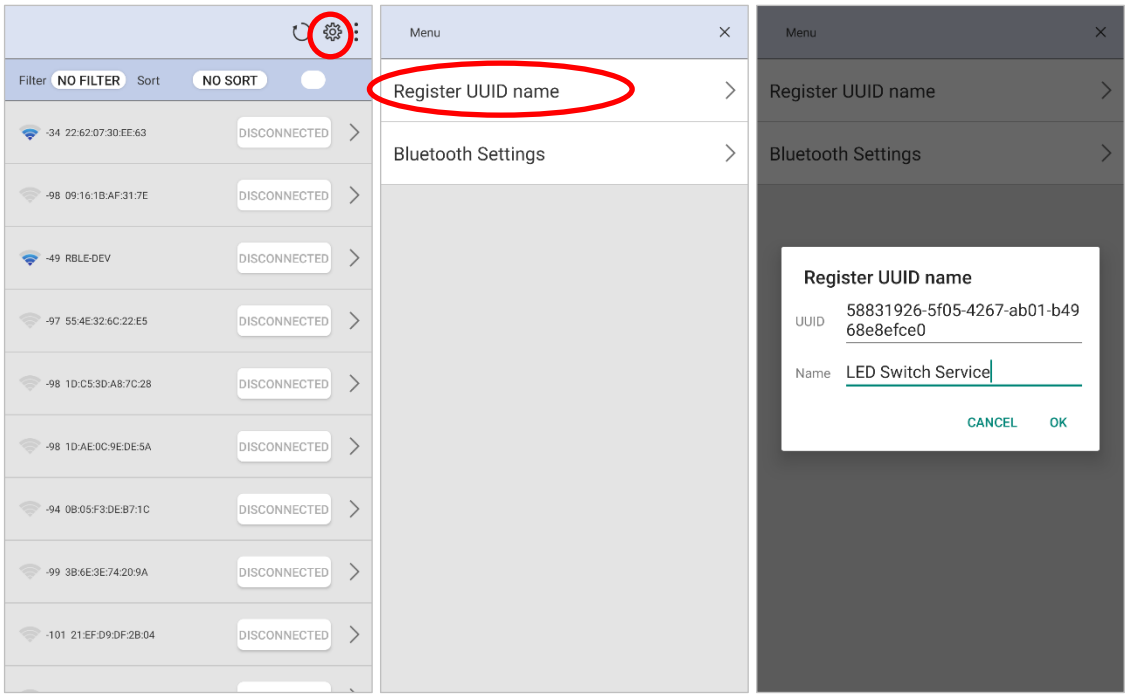

図 6-5　デバイス一覧画面(5)

"Bluetooth Settings"を選択すると OS の Bluetooth 設定画面を表示できます。

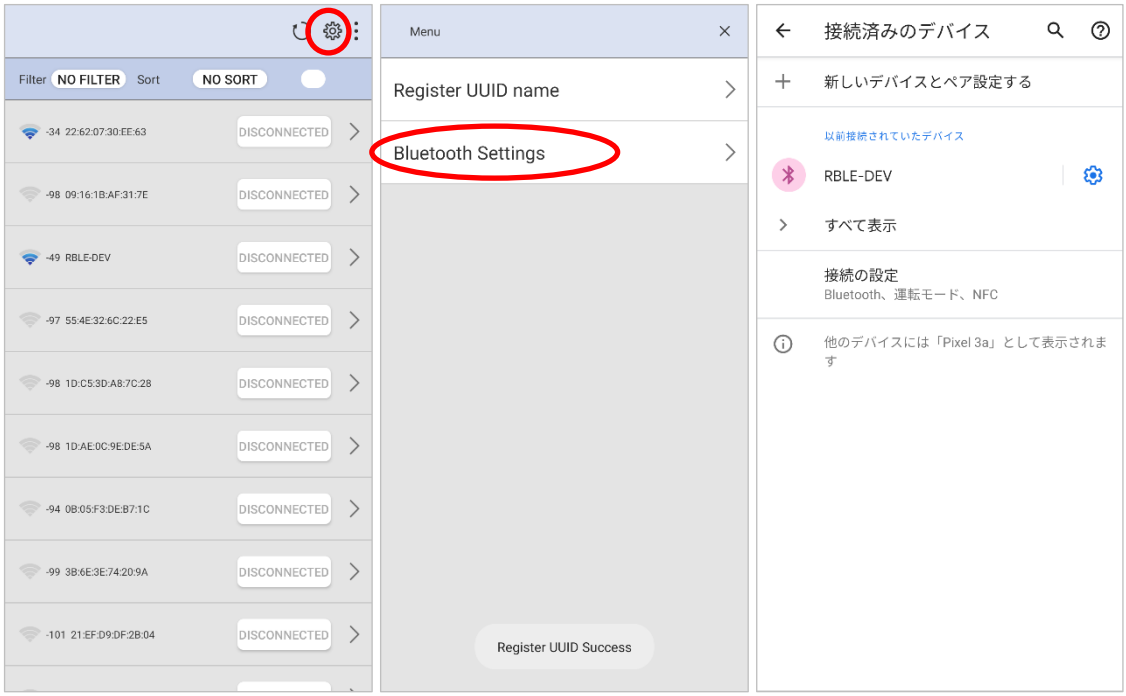

図 6-6　デバイス一覧画面(6)

6.2　接続デバイス詳細画面

接続デバイス詳細画面は Target Board for RX23W との接続状態を表示します。

"CONNECTED"は接続中であることを示します。"CONNECTED"をタップすると接続を切断します。同様に"DISCONNECTED"は切断状態であることを示します。"DISCONNECTED"をタップすると再接続します。

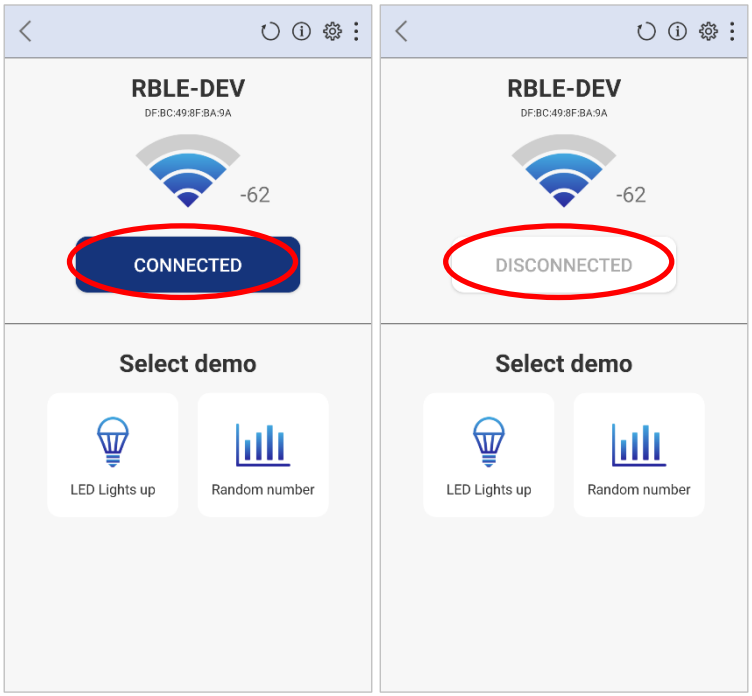

図 6-7　接続デバイス詳細画面(1)

"LED Lights up"ボタンをタップするとライトデモ画面を表示します。画面左上の戻るボタンで接続デバイス詳細画面に戻ります。

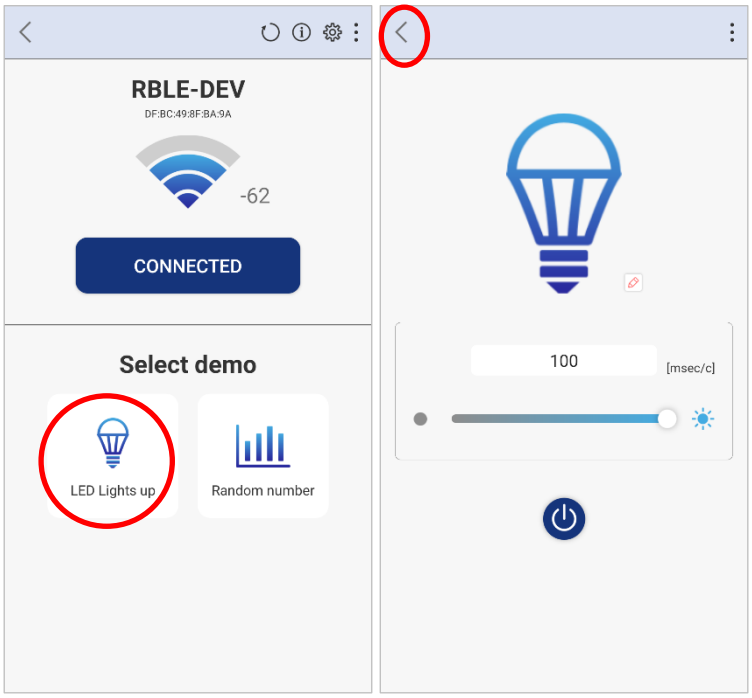

図 6-8　接続デバイス詳細画面(2)

"Random number"ボタンをタップするとデータデモ画面を表示します。画面左上の戻るボタンで接続デバイス詳細画面に戻ります。

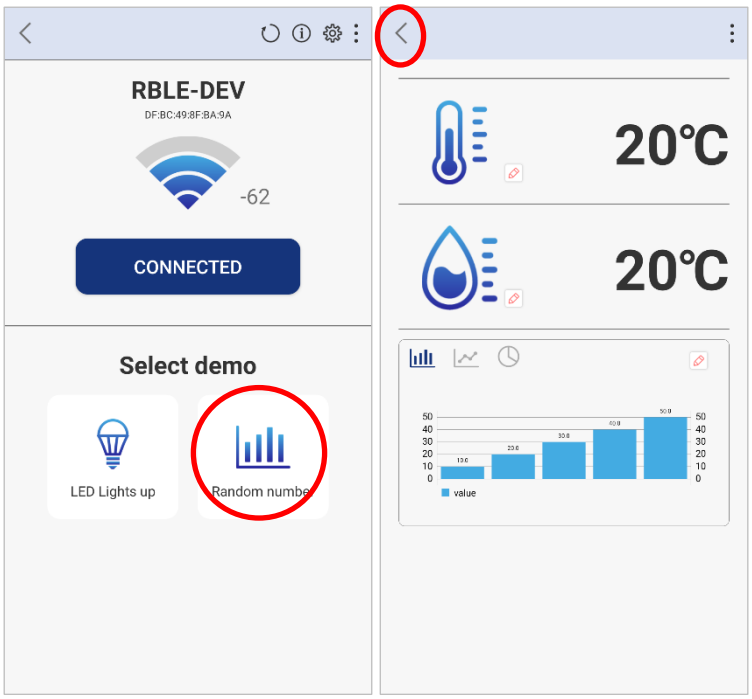

図 6-9　接続デバイス詳細画面(3)

リロードボタンをタップすると接続した Target Board for RX23W の情報を再読み込みできます。

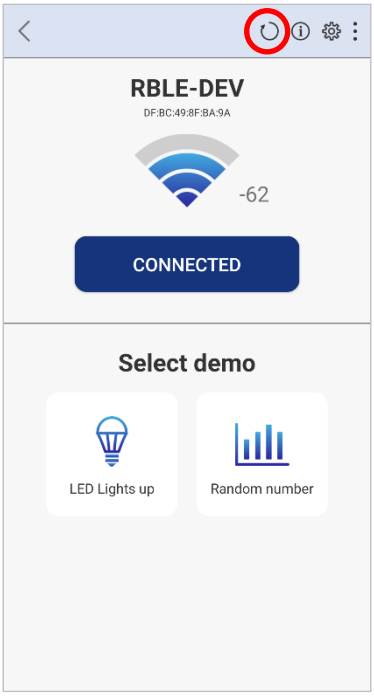

図 6-10　接続デバイス詳細画面(4)

Info ボタンを選択すると評価ボードの詳細情報を表示します。

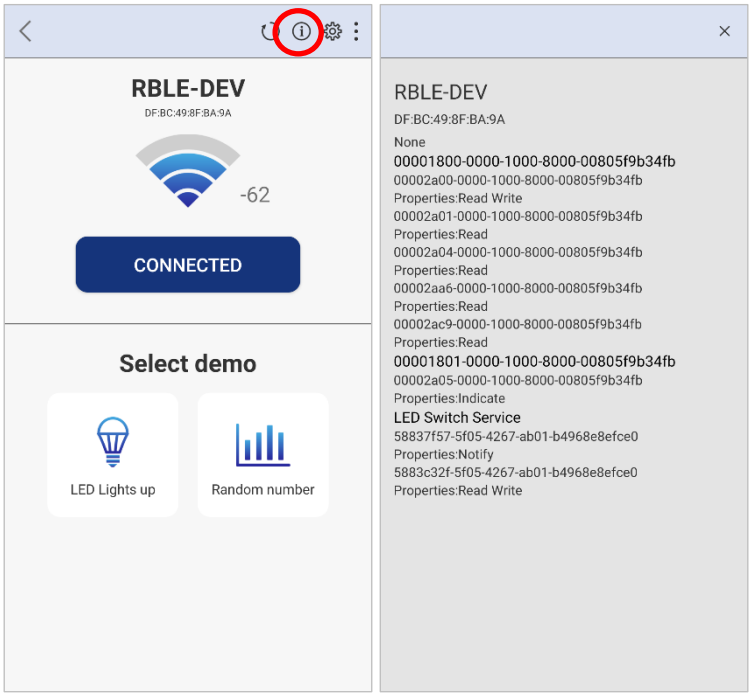

図 6-11　接続デバイス詳細画面(5)

設定ボタンをタップすると"Create bond"と"Bluetooth Settings"が表示されます。

"Bluetooth Settings"を選択すると Android OS の Bluetooth 設定画面を表示します。

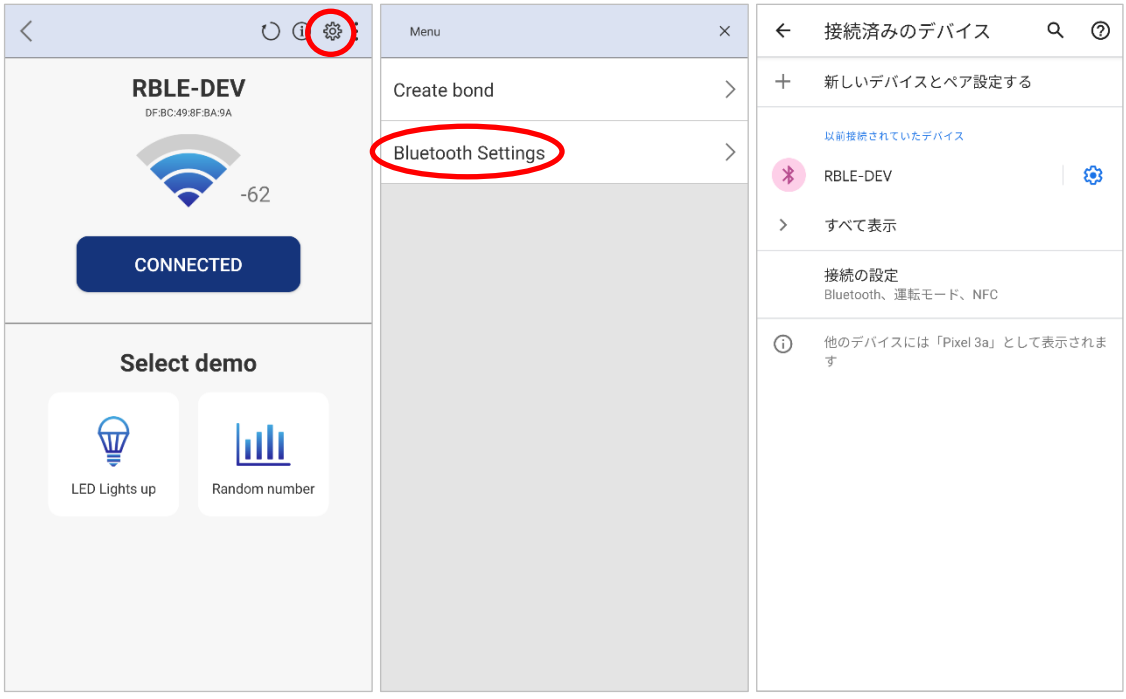

図 6-12　接続デバイス詳細画面(6)

"Create bond"を選択するとペアリングを実行します。

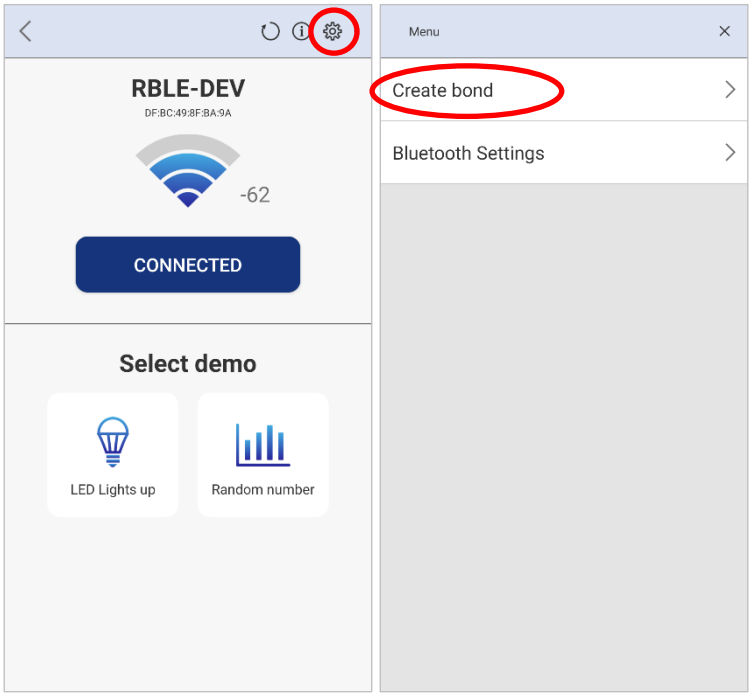

図 6-13　接続デバイス詳細画面(7)

※ペアリングの実行後、RX23W はボンディング情報をデータフラッシュに格納します。Target Board for RX23W のファームウェアを再度書き込むなどしてデータフラッシュ上のボンディング情報を消去した場合、TryBT と再接続できなくなる可能性があります。RX23W のボンディング情報を消去した場合は、Android 端末に格納されたボンディング情報も消去してください。

Android 端末のボンディング情報の消去は、Android OS の Bluetooth 設定画面で行えます。

図 6-14　Android OS の Bluetooth 設定画面

6.3　ライトデモ画面

ライトデモ画面では Target Board for RX23W の LED 点滅を操作するデモを実行できます。

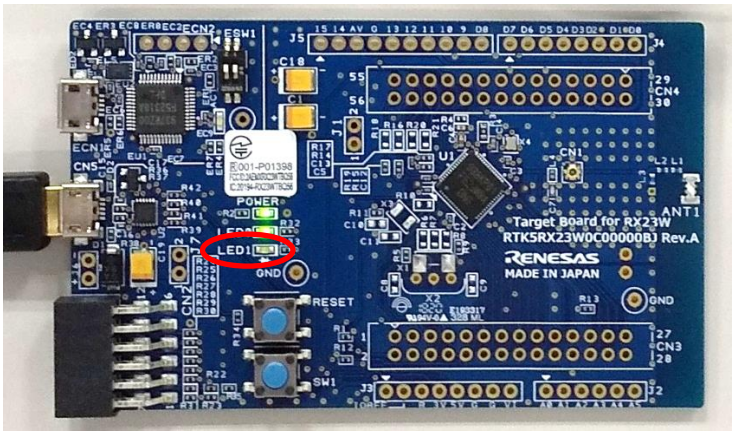

図 6-15　Target Board for RX23W のユーザ LED

スライダーまたはテキストエディットで LED の点滅間隔をミリ秒で指定します(100ms～10000ms)。

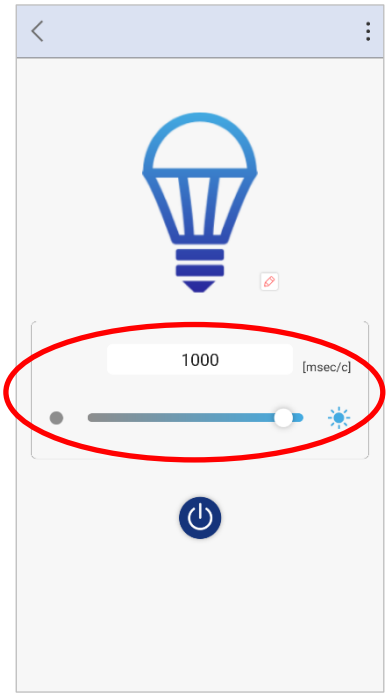

図 6-16　ライトデモ画面(1)

パワーボタンで LED 点滅の停止と再開を操作します。

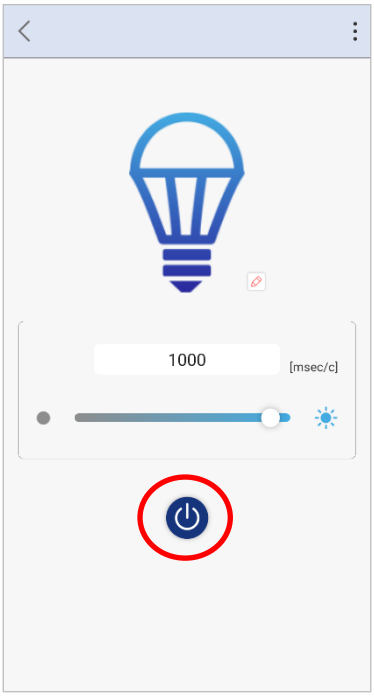

図 6-17　ライトデモ画面(2)

TryBT はアプリの実行中にグラフの色やアイコンを変更できるカスタマイズモードを持ちます。ランプアイコンの近くに表示されるペンマークをタップするとカスタマイズモードとなり、ランプアイコンを変更できます。

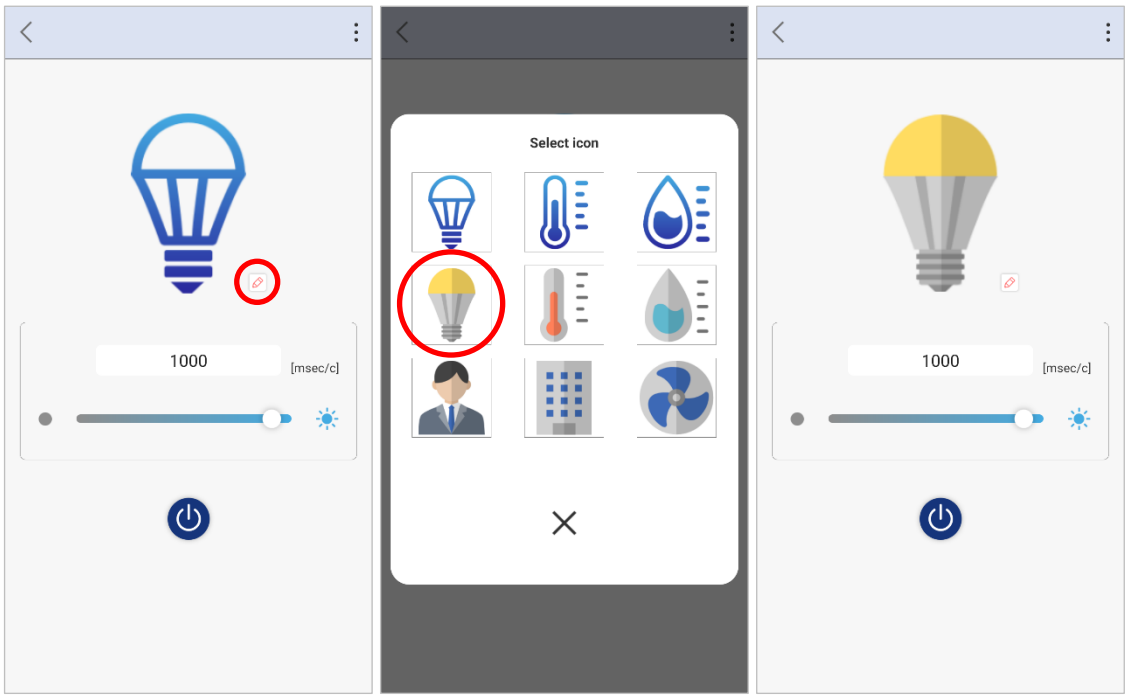

図 6-18　ライトデモ画面(3)

6.4　データデモ画面

データデモ画面では Target Board for RX23W 上のスイッチを押すたびに乱数を発生させて温度と湿度をグラフで表示します。グラフのデータは最大 10 点まで保持します。

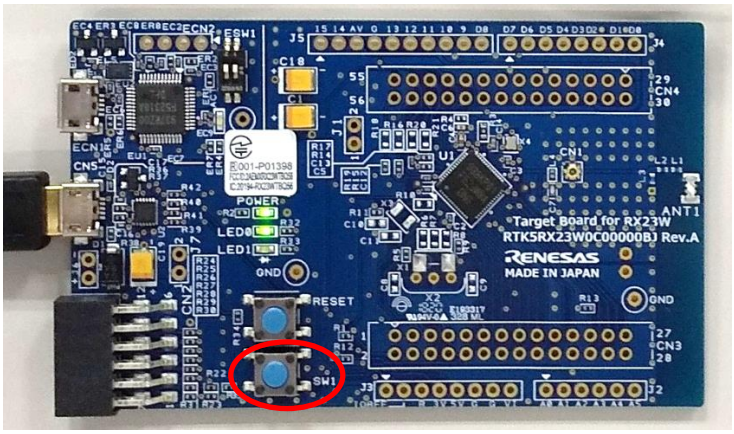

図 6-19　Target Board for RX23W のユーザスイッチ

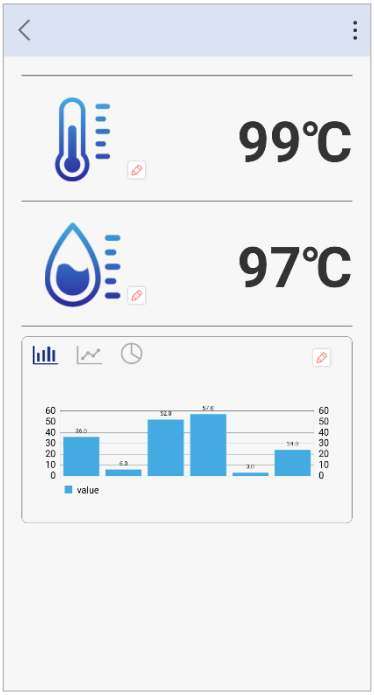

図 6-20　データデモ画面(1)

グラフ表示は棒グラフ、折れ線グラフ、円グラフを切り替えできます。

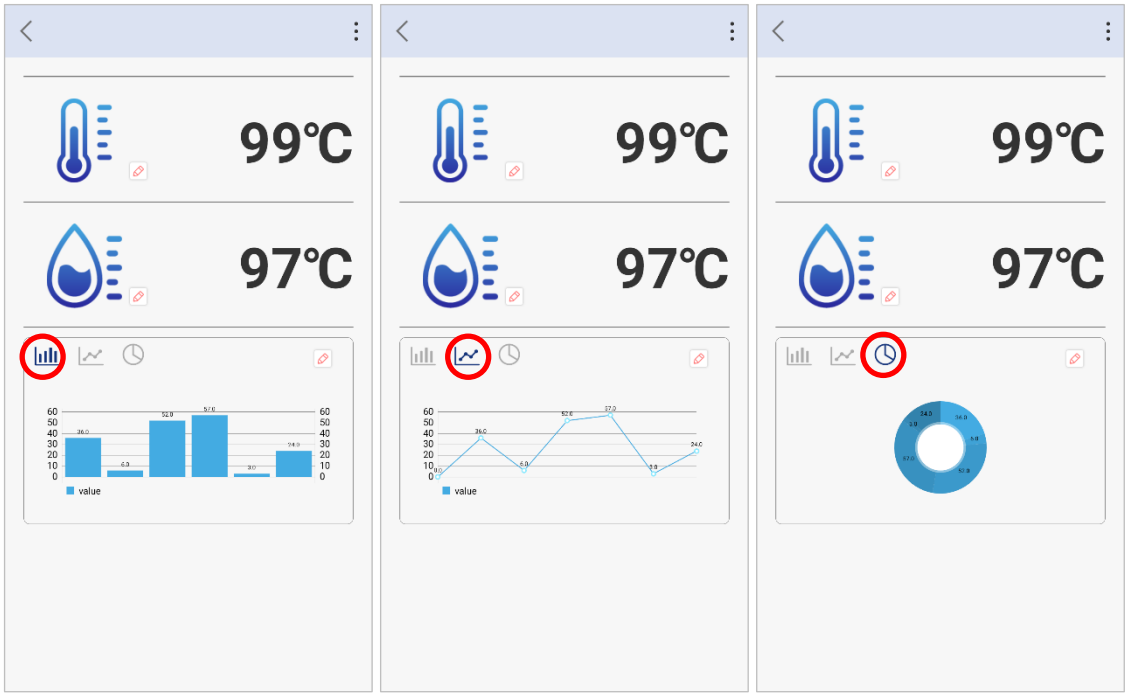

図 6-21　データデモ画面(2)

温度計と湿度計のアイコンの近くに表示されるペンマークをタップするとカスタマイズモードとなり、温度計と湿度計のアイコンを変更できます。

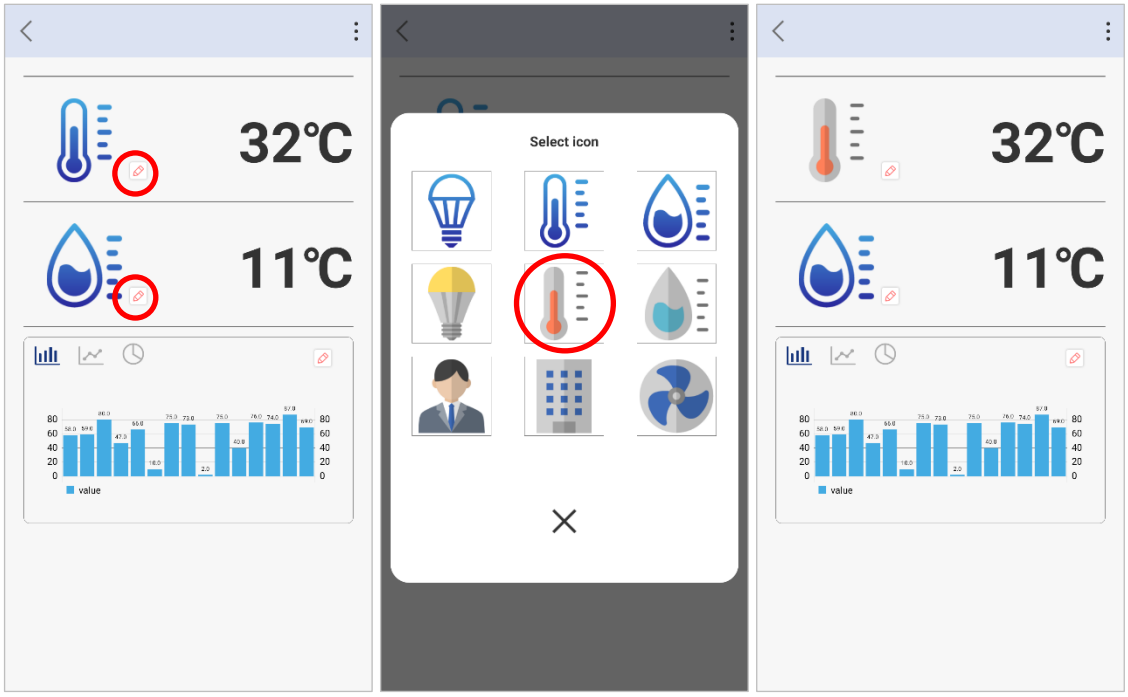

図 6-22　データデモ画面(3)

グラフの近くに表示されるペンマークをタップするとカスタマイズモードとなり、グラフの色を変更できます。

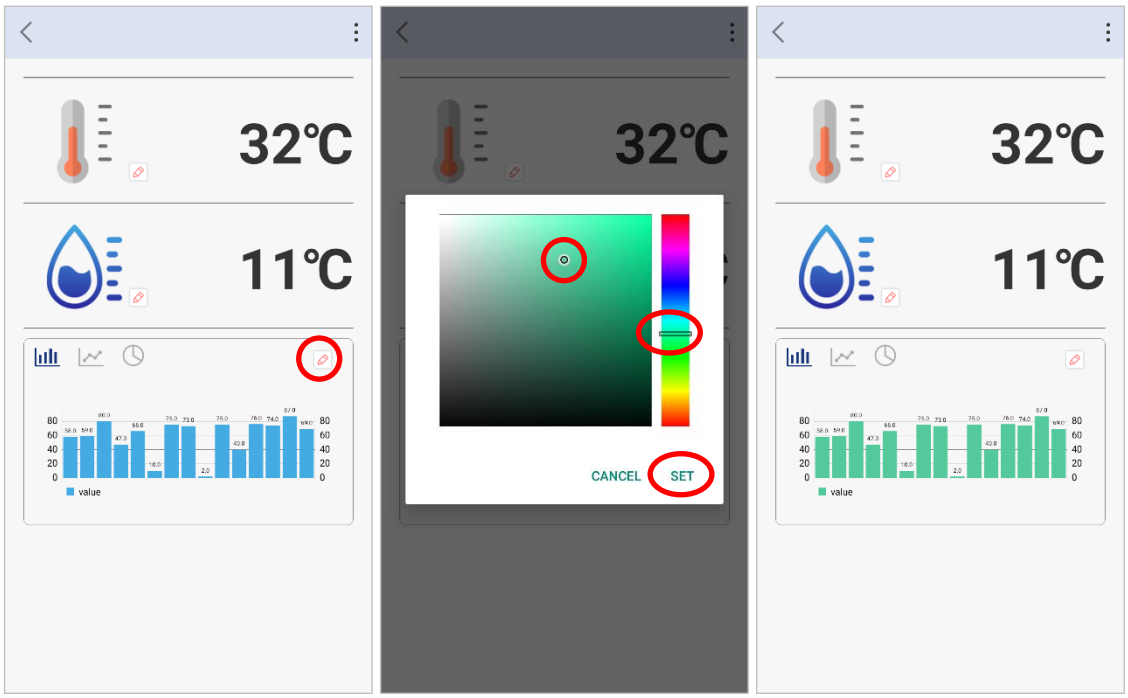

図 6-23　データデモ画面(4)

7.　TryBT のカスタマイズ

本章ではアプリタイトルやスプラッシュ画面、アイコンデータの変更方法とカスタマイズモードの有効・無効を設定する方法を示します。TryBT のソフトウェア実装をカスタマイズする際は、次章以降に記載された TryBT プロジェクトの解説もあわせて参照してください。

7.1　アプリタイトルのカスタマイズ

本アプリのタイトルを変更できます。

1. Android Studio で TryBT プロジェクトを開きます。画面左上のペインで"Android"を選択して"app→manifests→AndroidManifest.xml"をダブルクリックして開きます。

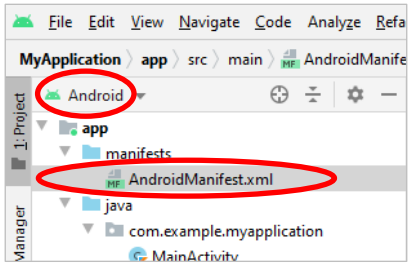

図 7-1　アプリタイトルのカスタマイズ(1)

2. AndroidManifest.xml で application タグの android:label 属性を変更すると、アプリタイトルが変更されます。

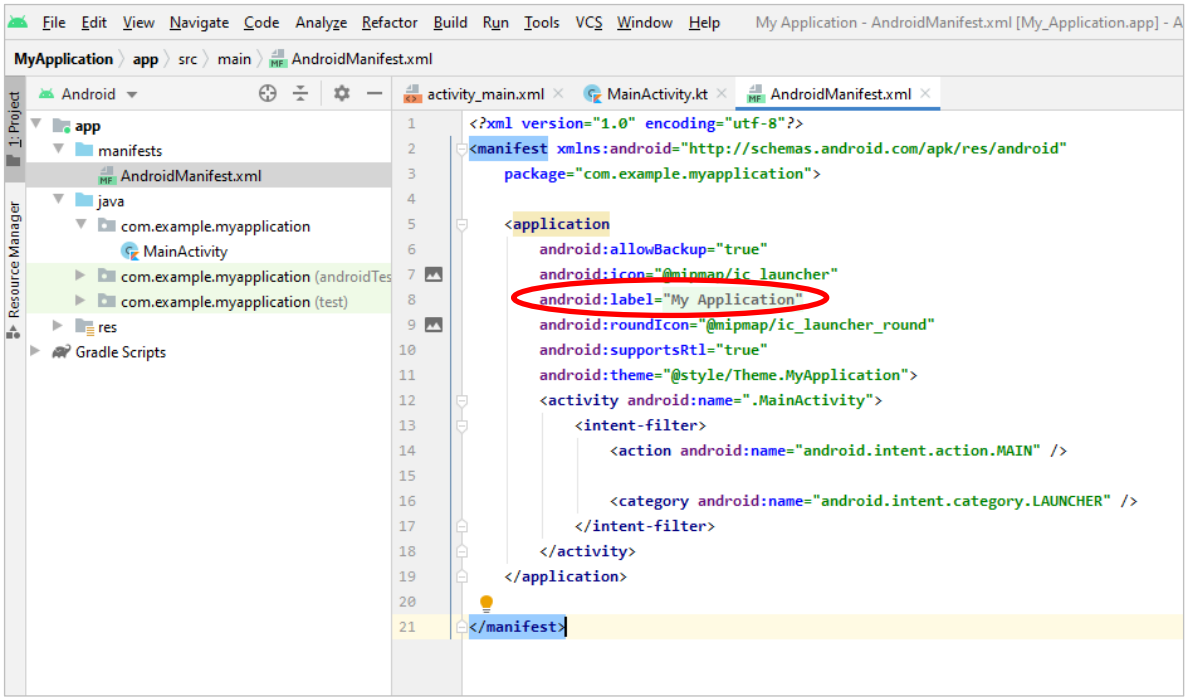

図 7-2　アプリタイトルのカスタマイズ(2)

3. 4 章に記載された手順でアプリを再インストールします。

7.2　スプラッシュ画面のカスタマイズ

アプリ起動時のスプラッシュ画面を変更できます。

図 7-3　スプラッシュ画面のカスタマイズ

1. Android Studio で TryBT プロジェクトを開きます。画面左上のペインで"Android"を選択して "res→drawable→splash_background.png"と"res→drawable→splash_logo.png"を変更します。

   ※splash_background.png の解像度は 1080x2160、splash_logo.png の解像度は 860x287 に設定してください。

2. 4 章に記載された手順でアプリを再インストールします。

7.3　アイコンデータのカスタマイズ

ライトデモ画面とデータデモ画面でカスタマイズ可能なアイコンを変更できます。

※アイコンは最大 9 個までです。
※アイコンは png フォーマットで 200x200 の解像度で用意してください。

1. Android Studio で TryBT プロジェクトを開きます。画面左上のペインで Android を選択して res→drawable を開きます。
2. 手順 1 で指定した位置の icon01.png～icon09.png を用意したアイコンと差し換えてください。
3. 4 章に記載された手順でアプリを再インストールしてください。

7.4　カスタマイズモードの有効化・無効化

本アプリは、実行中にアイコンなどを変更できるカスタマイズモードを持ちます。

1. Android Studio で TryBT プロジェクトを開きます。画面左上のペインで Android を選択して res→values→values.xml を開きます。
2. values.xml で is_customize_mode 属性の値を true または false に変更します。
   true: カスタマイズモードを有効化
   false: カスタマイズモードを無効化

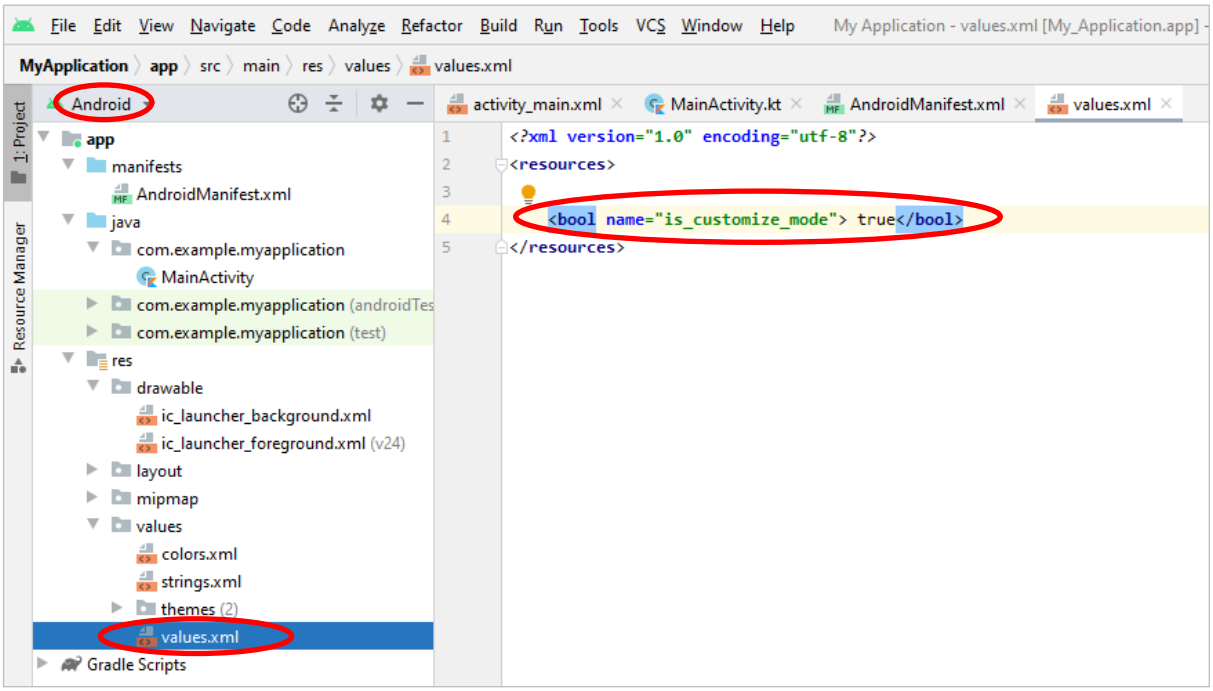

図 7-4　カスタマイズモードの有効化・無効化

3. 4 章に記載された手順でアプリを再インストールします。

8. TryBT のファイル構成

TryBT のフォルダ構成を以下に示します。本節では build.gradle、AndroidManifest.xml、kt ファイルについて解説します。

Android プロジェクトの概要については https://developer.android.com/studio/projects?hl=ja もあわせて参照してください。

```
TryBT/
  |   .gitignore
  |   build.gradle
  |   gradle.properties
  |   gradlew
  |   gradlew.bat
  |   settings.gradle
  |
  +---app/
  |   |   .gitignore
  |   |   build.gradle
  |   |   google-services.json
  |   |   proguard-rules.pro
  |   |   trybtkeystore
  |   |
  |   +---libs/
  |   +---src/
  |       +---androidTest/
  |       +---main/
  |       |   |   AndroidManifest.xml
  |       |   |
  |       |   +---java/
  |       |   |   +---com/
  |       |   |       +---renesas/
  |       |   |           +---trybt/
  |       |   |               |   MainApplication.kt
  |       |   |               |   MyFirebaseMessagingService.kt
  |       |   |               |   SplashActivity.kt
  |       |   |               |
  |       |   |               +---ui/
  |       |   |               |   |   AppSettingManager.kt
  |       |   |               |   |
  |       |   |               |   +---bluetoothDetail/
  |       |   |               |   |       BluetoothDetailActivity.kt
  |       |   |               |   |       BluetoothDeviceInfoActivity.kt
  |       |   |               |   |
  |       |   |               |   +---bluetoothList/
  |       |   |               |   |       BluetoothListActivity.kt
  |       |   |               |   |       BluetoothListAdapter.kt
  |       |   |               |   |       BluetoothListFilterDialogFragment.kt
  |       |   |               |   |       BluetoothListSortDialogFragment.kt
  |       |   |               |   |       BluetoothUUIDDialogFragment.kt
  |       |   |               |   |
  |       |   |               |   +---colorPicker/
  |       |   |               |   |       ColorPickerDialogFragment.kt
  |       |   |               |   |
  |       |   |               |   +---iconSelect/
  |       |   |               |   |       IconSelectDialogFragment.kt
  |       |   |               |   |       IconSelectPageAdapter.kt
  |       |   |               |   |
  |       |   |               |   +---lightsUpDemo/
  |       |   |               |   |       LightsUpDemoActivity.kt
  |       |   |               |   |
```

```
    |       |   |               |   +---menu/
    |       |   |               |   |       MenuInfo.kt
    |       |   |               |   |       MenuListActivity.kt
    |       |   |               |   |       MenuListAdapter.kt
    |       |   |               |   |
    |       |   |               |   +---randomNumberDemo/
    |       |   |               |   |       RandomNumberDemoActivity.kt
    |       |   |               |   |
    |       |   |               |   +---setting/
    |       |   |               |   |       BluetoothDetailSettingActivity.kt
    |       |   |               |   |       BluetoothListSettingActivity.kt
    |       |   |               |   |
    |       |   |               |   +---versionInfo/
    |       |   |               |           VersionInfoActivity.kt
    |       |   |               |
    |       |   |              +---util/
    |       |   |                      TryBTUtil.kt
    |       |   |
    |       |   +---res/
    |       +---test/
    +---gradle/
```

図 8-1　TryBT のフォルダ構成

8.1　build.gradle について

build.gradle は TryBT のビルド設定を行うためのファイルです。build.gradle にはプロジェクト用、モジュール用の 2 つが存在します。

8.1.1　./build.gradle(プロジェクト用)

プロジェクト全体のビルド設定を記載します。主にプロジェクト内に複数のモジュール（アプリ、ライブラリなど）がある場合に使用します。TryBT プロジェクトは 1 つのアプリのみ含まれるため、基本的にデフォルトのままです。

8.1.2　./app/build.gradle(モジュール用)

プロジェクト内のアプリ、ライブラリが複数ある場合、それぞれのビルド設定を記載することができます。TryBT では 1 つのアプリのビルド設定を記載します。基本的な設定項目を解説します。

表 8-1　build.gradle の属性

属性名	概要	
compileSdkVersion	本アプリのビルドに使用する Android SDK のバージョンを API レベル（数字）で指定します。 例：29 ※minSdkVersion で指定された API レベル以上のものを指定する。 ※AndroidSDK のバージョンと API レベルについては以下を参照。 https://developer.android.com/studio/releases/platforms?hl=ja	
buildToolsVersion	ビルドツールのバージョンを指定します。 例："29.0.3" ※バージョンの一覧は以下を参照。 https://developer.android.com/studio/releases/build-tools?hl=ja	
defaultConfig	applicationId	Android アプリの ID を指定します。通常、会社名のドメイン名の逆順+アプリ名です。（半角英数） 例："com.renesas.trybt"
	minSdkVersion	アプリケーションが対応する最小の API レベルを記載します。 例：23
	versionCode	1 から始まる通しのバージョンを指定します。こちらはユーザには表示されません。 例：2
	versionName	ユーザに見えるバージョン名を文字列で指定します。 例："1.0.0"
dependencies	implementation	プロジェクトで使用する外部ライブラリを指定します。 例：'com.google.firebase:firebase-analytics-ktx'

8.2　./app/src/main/AndroidManifest.xml について

AndroidManifest.xml は TryBT の画面構成や必要な権限について定義するためのファイルです。

表 8-2　AndroidManifest.xml の属性

属性名	概要	
users-permission	android:name	OS の許可が必要な権限（Bluetooth、インターネットアクセス等）を指定します。 例："android.permission.BLUETOOTH"
application	android:name	アプリケーション全体を制御する android.app.Application の継承クラスを指定します。 例：".MainApplication"
	android:icon	アプリのアイコンを指定します。アイコンは mipmap フォルダ、または drawable フォルダに配置します。 例："@mipmap/ic_launcher"
	android:label	アプリ名を定義します。string.xml に記載されたアプリ名が参照されます。 例："@string/app_name"
activity	android:name	アプリ内で使用する activity を宣言します。 例：".ui.menu.MenuListActivity" ※画面を追加した場合、画面に対応する activity を追記しないとアプリケーションがクラッシュします。

8.3　./app/src/main/java のフォルダ構成と.kt ファイルについて

TryBT のロジックは Kotlin 言語で実装されており、ファイルの拡張子は.kt です。通常、各 kt ファイルには 1 つのクラスが実装され、ファイル名はクラス名と同一です。

kt ファイルはパッケージという単位で管理されます。各パッケージは"./app/src/main/java"フォルダに格納されます。パッケージ名が com.renesas.trybt.ui.bluetoothDetail の場合、そのパッケージフォルダは com/renesas/trybt/ui/bluetoothDetail となります。

本節では TryBT を構成するパッケージと各ファイルに実装されたクラスの概要について記載します。

- com.renesas.trybt パッケージ

TryBT のライフサイクルを管理するクラスを格納するパッケージです。

表 8-3　com.renesas.trybt のクラス

クラス名	概要
MainApplication	アプリケーション全体のライフサイクルを管理します。TryBT では Bluetooth デバイスの状態変更イベントなども管理します。
MyFirebaseMessagingService	プッシュ通知を行う際の処理を記載します。
SplashActivity	スプラッシュ画面です。

- com.renesas.trybt.ui パッケージ

TryBT アプリの画面ごとのパッケージやクラスを格納するパッケージです。

表 8-4　com.renesas.trybt.ui のクラス

クラス名	概要
AppSettingManager	ソート設定などの各画面情報の保存・読み込みを行います。

- com.renesas.trybt.ui.bluetoothDetail パッケージ

接続デバイス詳細画面に関するクラスを格納するパッケージです。

表 8-5　com.renesas.trybt.ui.bluetoothDetail のクラス

クラス名	概要
BluetoothDetailActivity	接続デバイス詳細画面の処理を定義します。
BluetoothDeviceInfoActivity	デバイス情報画面の処理を定義します。

- com.renesas.trybt.ui.bluetoothList パッケージ

デバイス一覧画面に関するクラスを格納するパッケージです。

表 8-6　com.renesas.trybt.ui.bluetoothList のクラス

クラス名	概要
BluetoothListActivity	デバイス一覧画面の処理を定義します。
BluetoothListAdapter	デバイス一覧画面のテーブルの各デバイスの cell を定義します。
BluetoothListFilterDialogFragment	デバイス一覧のフィルタダイアログの処理を定義します。
BluetoothListSortDialogFragment	デバイス一覧のソートダイアログの処理を定義します。
BluetoothUUIDDialogFragment	デバイス一覧の UUID 登録ダイアログの処理を定義します。

- com.renesas.trybt.ui.colorPicker パッケージ

接続デバイス詳細画面で使用する色を指定するためのピッカークラスを格納するパッケージです。

表 8-7　com.renesas.trybt.ui.colorPicker のクラス

クラス名	概要
ColorPickerDialogFragment	色選択のためのピッカーダイアログを定義します。

- com.renesas.trybt.ui.iconSelect パッケージ

各画面で使用するアイコンを選択するためのクラスを格納するパッケージです。

表 8-8　com.renesas.trybt.ui.iconSelect のクラス

クラス名	概要
IconSelectDialogFragment	アイコン一覧を表示するダイアログを定義します。
IconSelectPageAdapter	アイコン一覧ダイアログの各アイコンのレイアウトを定義します。

- com.renesas.trybt.ui.ligthsUpDemo パッケージ

ライトデモ画面のクラスを格納するパッケージです。

表 8-9　com.renesas.trybt.ui.lightsUpDemo のクラス

クラス名	概要
LightsUpDemoActivity	ライトデモ画面の処理を定義します。

- com.renesas.trybt.ui.menu パッケージ

Menu 画面のクラスを格納するパッケージです。

表 8-10　com.renesas.trybt.ui.menu のクラス

クラス名	概要
MenuListActivity	メニュー画面を定義します。
MenuListAdapter	各メニューの記載を定義します。
MenuInfo	各メニューの記載要素を定義します。

- com.renesas.trybt.ui.randomNumberDemo パッケージ

データデモ画面のクラスを格納するパッケージです。

表 8-11　com.renesas.trybt.ui.randomNumberDemo のクラス

クラス名	概要
RandomNumberDemoActivity	データデモ画面を定義します。

- com.renesas.trybt.ui.setting パッケージ

設定画面のクラスを格納するパッケージです。

表 8-12　com.renesas.trybt.ui.setting のクラス

クラス名	概要
BluetoothDetailSettingActivity	接続デバイス詳細画面に表示する設定画面を定義します。
BluetoothListSettingActivity	デバイス一覧画面に表示する設定画面を定義します。

- com.renesas.trybt.ui.versionInfo パッケージ

バージョン情報画面のクラスを格納するパッケージです。

表 8-13　com.renesas.trybt.ui.vesionInfo のクラス

クラス名	概要
VersionInfoActivity	バージョン情報画面を定義します。

- com.renesas.trybt.util パッケージ

ユーティリティクラスを格納するパッケージです。

表 8-14　com.renesas.trybt.util のクラス

クラス名	概要
TryBTUtil	RSSI の値に応じてアンテナアイコンを決定するといった汎用的な処理を定義します。

9. TryBT の画面遷移

TryBT の画面とクラス遷移を以下に記載します。

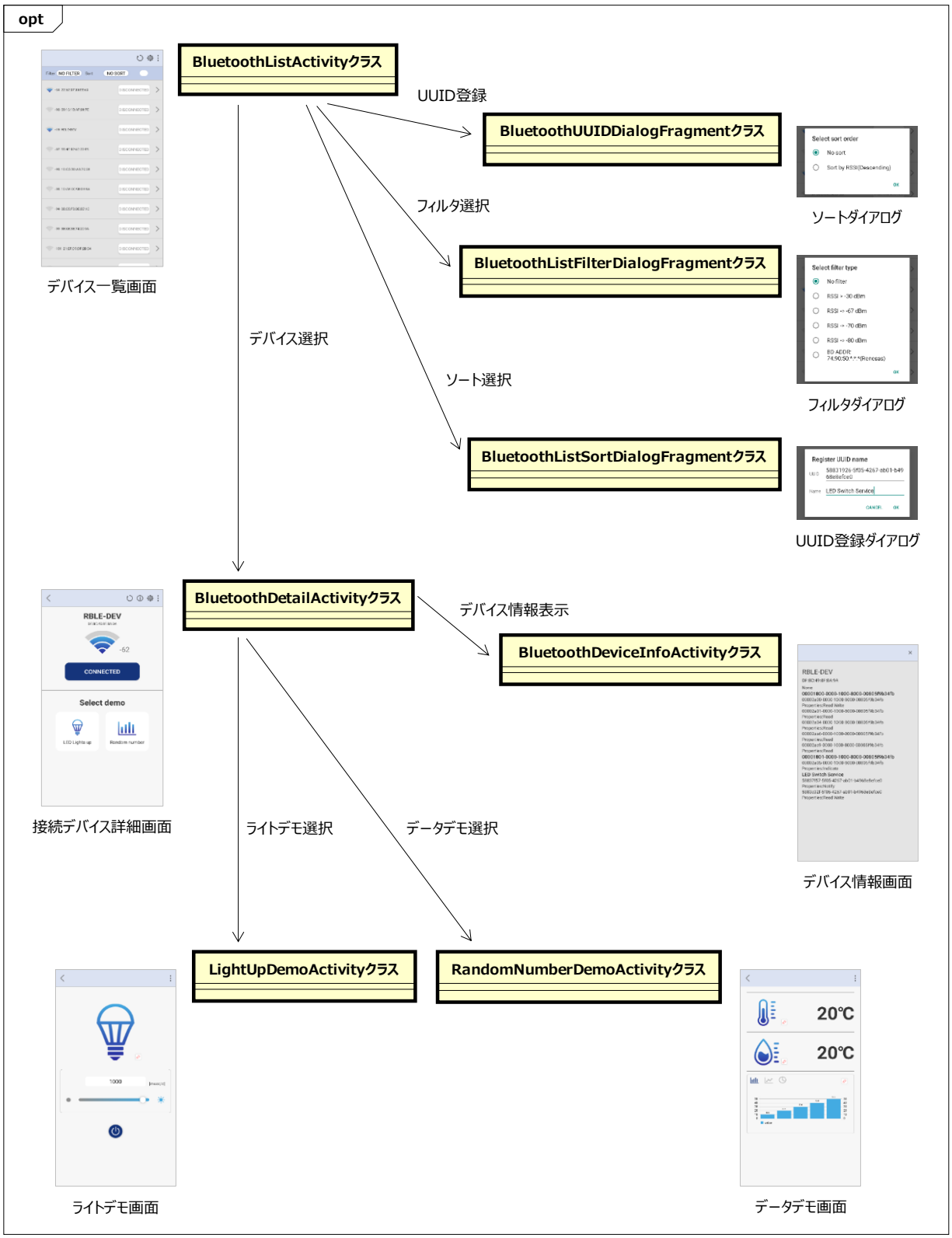

図 9-1　TryBT の画面とクラス遷移

10. TryBT の Bluetooth 通信

本章では TryBT での Bluetooth の使用方法について解説します。

10.1 Android 端末の Bluetooth 有効化と Fine Location 権限チェック

BluetoothListActivity クラスは端末の Bluetooth が有効か否かをチェックして、無効になっている場合は Bluetooth 有効化画面を表示します。

また Bluetooth を使用するためには Fine Location 権限（位置情報権限）が必要です。Bluetooth が有効になっている場合、BluetoothListActivity クラスはアプリの Fine Location 権限をチェックします。

デバイススキャンのための BluetoothAdapter のインスタンス変数 mBluetoothAdapter を定義します。

```
/**
 * Bluetooth Adapter for discovering devices
 */
private var mBluetoothAdapter: BluetoothAdapter? = null
```

図 10-1　BluetoothListActivity.kt (1)

インスタンス変数 mBluetoothAdapter は onCreate メソッドで初期化します。

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    //initialize bluetooth scan
    val bluetoothManager = getSystemService(Context.BLUETOOTH_SERVICE) as BluetoothManager
    mBluetoothAdapter = bluetoothManager.adapter

    //if adapter not found exit application
    if(mBluetoothAdapter == null){
        Toast.makeText( this, "Not Support BLE", Toast.LENGTH_SHORT ).show()
        finish()
        return
    }
```

図 10-2　BluetoothListActivity.kt (2)

デバイス一覧画面が表示されるごとに onResume メソッドで Bluetooth と Fine Location の権限をチェックします。

```
  override fun onResume() {
    super.onResume()

    //request bluetooth permission and start scan.
    requestBluetoothFeature()
    checkPermission()
```

図 10-3　BluetoothListActivity.kt (3)

BluetoothAdapter を生成できない場合、Intent で Bluetooth を有効にするためのシステム設定を起動します。

```
private fun requestBluetoothFeature(){
    if(mBluetoothAdapter != null && mBluetoothAdapter!!.isEnabled){
        return
    }
    val btIntent = Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE )
    startActivityForResult(btIntent, REQUEST_ENABLEBLUETOOTH)
}
```

図 10-4　BluetoothListActivity.kt (4)

Fine Location 権限がある場合はデバイスのスキャンを開始します。権限がない場合はリクエストします。

```
// check location permission
public fun checkPermission() {
    if (ContextCompat.checkSelfPermission(
            this,
            Manifest.permission.ACCESS_FINE_LOCATION
        )
        == PackageManager.PERMISSION_GRANTED
    ) {
        startScan()
    } else {
        //request permission if permission not granted.
        requestLocationPermission()
    }
}
```

図 10-5　BluetoothListActivity.kt (5)

10.2 デバイススキャンの開始

BluetoothListActivity クラスは startScan メソッドでデバイスのスキャンを開始します。

```
/**
 * scan bluetooth device method
 */
private fun startScan(){
```

図 10-6　BluetoothListActivity.kt (6)

特にポイントとなるのは、デバイスのスキャンは非同期処理で行われるため、スキャン結果のコールバックを受け取る処理を実装する必要があります。TryBT ではスキャン結果を同クラスで受け取るためのコールバックメソッドを実装しています。

```
scanner.startScan(mScanFilters,mScanSettings,mLeScanCallback)
```

図 10-7　BluetoothListActivity.kt (7)

コールバックメソッドは mLeScanCallback です。スキャンで発見されたデバイスはインスタンス変数 mDeviceListAdapter に追加されます。

```
/**
 * Object for callback event of Scanning
 */
private val mLeScanCallback: ScanCallback = object : ScanCallback(){
    override fun onScanFailed(errorCode: Int) {
        super.onScanFailed(errorCode)
    }
    override fun onBatchScanResults(results: MutableList<ScanResult>?) {
        super.onBatchScanResults(results)
    }
    override fun onScanResult(callbackType: Int, result: ScanResult?) {
        mnUIHandler.post {
            mDeviceListAdapter!!.addDevice( result!!)
            mDeviceListAdapter!!.notifyDataSetChanged()
        }
    }
}
```

図 10-8　BluetoothListActivity.kt (8)

10.3 デバイススキャンの停止

BluetoothListActivity クラスは stopScan メソッドでスキャンを停止します。

```
/**
 * stop scan method.
 */
private fun stopScan(){
    // remove handler
    mHandler!!.removeCallbacksAndMessages(null)

    // get bluetooth scanner
    val scanner = mBluetoothAdapter!!.bluetoothLeScanner ?: return
    mScanning = false
    scanner.stopScan(mLeScanCallback)

    // invalidate option menu.
    invalidateOptionsMenu()
}
```

図 10-9　BluetoothListActivity.kt (9)

10.4 デバイスとの接続

BluetoothDetailActivity クラスは connect メソッドで検索したデバイスに接続します。

```
/**
 * connect bluetooth device
 */
private fun connect(){
    if(bluetoothInfo != null && bluetoothInfo!!.device != null ){
        val device = bluetoothInfo!!.device
        binding.connectButton.isEnabled = false
        binding.lightDemoButton.isEnabled = false
        binding.randomDemoButton.isEnabled = false
        binding.header.btn_reload.isEnabled = false
        binding.header.btn_setting.isEnabled = false
        binding.header.btn_info.isEnabled = false
        binding.header.btn_menu.isEnabled = false

        val application:MainApplication = getApplication() as MainApplication
        application.bluetoothDetailActivity = this
        mBluetoothGatt = device.connectGatt(this,false,application.bleGattCallback)
        application.bluetoothGatt = mBluetoothGatt
    }
}
```

図 10-10　BluetoothDetailActivity.kt (1)

10.5 デバイスとの接続切断

BluetoothDetailActivity クラスは disconnect メソッドでデバイスとの接続を切断します。

```
/**
 * disconnect bluetooth device
 */
private fun disconnect(){
    if(mBluetoothGatt != null){
        binding.connectButton.isEnabled = false
        binding.lightDemoButton.isEnabled = false
        binding.randomDemoButton.isEnabled = false
        binding.header.btn_reload.isEnabled = false
        binding.header.btn_setting.isEnabled = false
        binding.header.btn_info.isEnabled = false
        binding.header.btn_menu.isEnabled = false
        mBluetoothGatt!!.close()
        mBluetoothGatt = null
        isConnect = false

        val application: MainApplication = getApplication() as MainApplication
        application.isConnected = false
        binding.connectButton.isEnabled = true
        binding.lightDemoButton.isEnabled = true
        binding.randomDemoButton.isEnabled = true
        binding.header.btn_reload.isEnabled = true
        binding.header.btn_setting.isEnabled = true
        binding.header.btn_info.isEnabled = true
        binding.header.btn_menu.isEnabled = true
    }
}
```

図 10-11　BluetoothDetailActivity.kt (2)

10.6 デバイスの接続状態の変更通知

接続の状態が変わった場合、BluetoothDetailActivity クラスの onConnectionStateChange メソッドに通知されます。TryBT では通知状態によってボタンや内部フラグを制御します。

```
/**
  * connection state event
  */
public fun onConnectionStateChange (gatt: BluetoothGatt?, status: Int, newState: Int){
    val application: MainApplication = getApplication() as MainApplication
    //connected state
   if(BluetoothProfile.STATE_CONNECTED == newState){
      application.isConnected = true
      gatt!!.discoverServices()
          val runnable = Runnable{
                  binding.connectButton.isEnabled = true
                  binding.lightDemoButton.isEnabled = true
                  binding.randomDemoButton.isEnabled = true
                  binding.header.btn_reload.isEnabled = true
                  binding.header.btn_setting.isEnabled = true
                  binding.header.btn_info.isEnabled = true
                  binding.header.btn_menu.isEnabled = true

              val application: MainApplication = getApplication() as MainApplication
              if(application.bluetoothInfo != null){
                  bluetoothInfo = application.bluetoothInfo
                  binding.titleView.text = bluetoothInfo!!.device.name
                  binding.detailView.text = bluetoothInfo!!.device.address
                  binding.rssiText.text = ""+bluetoothInfo!!.rssi
                  binding.radioStrength.setImageResource(
                                                  TryBTUtil.createLargeRssiImageId(bluetoothInfo!!.rssi))
              }

          }
          mHandler!!.post(runnable)
          isConnect = true
          return
       }

    //disconnected state
    if(BluetoothProfile.STATE_DISCONNECTED == newState){
        application.isConnected = false
        isConnect = false
        return
    }
}
```

図 10-12　BluetoothDetailActivity.kt (3)

10.7 GATT サービスの取得と各種デモ画面への移動

BluetoothDetailActivity クラスは、接続したデバイスが TryBT のライトデモとグラフデモに必要な GATT サービスに対応しているかを確認します。TryBT はライトデモボタン、グラフデモボタンの押下イベントが通知されると、GATT サービスを取得します。該当するサービスを取得できない場合、エラーメッセージを表示します。

```
//light up button event
binding.lightDemoButton.setOnClickListener {

    //move light up activity when service uuid enabled
    val service = mBluetoothGatt!!.getService(UUID.fromString(getString(R.string.light_up_demo_uuid)))
    if(service == null){
        Toast.makeText(this,getString(R.string.error_invalid_service_uuid),Toast.LENGTH_SHORT).show()
    }else{
        val bleChar = service.getCharacteristic(UUID.fromString(MainApplication.LIGHT_UP_SERVICE_UUID))
        if(bleChar == null){
            Toast.makeText(this,getString(R.string.error_service_not_found),Toast.LENGTH_SHORT).show()
        }else{
            val intent = Intent(this, LightsUpDemoActivity::class.java)
            val application:MainApplication = getApplication() as MainApplication

            application.bluetoothGatt = mBluetoothGatt
            application.bluetoothInfo = bluetoothInfo
            startActivity(intent)

        }
    }
}
```

図 10-13　BluetoothDetailActivity.kt (4)

```
//Random Number button event
binding.randomDemoButton.setOnClickListener {
    //move random number activity when service uuid enabled
    val service = mBluetoothGatt!!.getService(UUID.fromString(getString(R.string.light_up_demo_uuid)))
    if(service == null){
        Toast.makeText(this,getString(R.string.error_invalid_service_uuid),Toast.LENGTH_SHORT).show()
    }else{
        val bleChar = service.getCharacteristic(UUID.fromString(MainApplication.LIGHT_UP_SERVICE_UUID))
        if(bleChar == null){
            Toast.makeText(this,getString(R.string.error_service_not_found),Toast.LENGTH_SHORT).show()
        }else{
            val application:MainApplication = getApplication() as MainApplication
            application.bluetoothGatt = mBluetoothGatt
            application.bluetoothInfo = bluetoothInfo
            val intent = Intent(this, RandomNumberDemoActivity::class.java)
            startActivity(intent)

        }
    }

}
```

図 10-14　BluetoothDetailActivity.kt (5)

10.8 評価ボードの LED 点灯間隔の変更

LightUpDemoActivity クラスは TryBT のライトデモ画面から入力された LED 点灯間隔を、write メソッドで評価ボードに設定します。

```
/**
 * write data rx23w method
 */
fun write(){
    try{
        //convert seek bar value to byte value.
        val convertValue = convertSeekBarValueBoardValue(binding.seekBar.progress)
        val byteValue = convertValue.toByte()

        //create service
        val service = mBluetoothGatt!!.getService(UUID.fromString(getString(R.string.light_up_demo_uuid)))
        if(service == null){
            Toast.makeText(this,getString(R.string.error_invalid_service_uuid),Toast.LENGTH_SHORT).show()
            return
        }

        val bleChar = service.getCharacteristic(UUID.fromString(MainApplication.LIGHT_UP_SERVICE_UUID))
        if(bleChar == null){
            Toast.makeText(this,getString(R.string.error_service_not_found),Toast.LENGTH_SHORT).show()
        }

        //if enableLight == false then light off(0)
        if(enableLight){
            bleChar.setValue(byteArrayOf(byteValue))
        }else{
            bleChar.setValue(byteArrayOf(0))
        }
        mBluetoothGatt!!.writeCharacteristic(bleChar)


    }catch (e:NumberFormatException){
        Toast.makeText(this,
                getString(R.string.error_light_up_range),Toast.LENGTH_SHORT).show()
    }
}
```

図 10-15　LightsUpDemoActivity.kt

10.9 評価ボードからのスイッチ押下の通知

評価ボードのスイッチが押下されると MainApplication クラスの onCharacteristicChanged メソッドで通知されます。TryBT のデータデモはスイッチの押下で乱数を生成してグラフを更新します。

MainApplication.kt

```
override fun onCharacteristicChanged(
    gatt: BluetoothGatt?,
    characteristic: BluetoothGattCharacteristic?
) {
```

図 10-16　MainApplication.kt

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2021.05.31	-	初版発行
1.01	2021.10.15	P.1	文書名を「Bluetooth Low Energy スマートフォンサンプルアプリケーション TryBT for Android」に変更。
		P.1	関連ドキュメントに EK-RA4W1 と EB-RE01B を追加。
		P.4	「1.1 動作環境」に EK-RA4W1 と EB-RE01B を追加。
		全体	一部の節名、記述を更新。

製品ご使用上の注意事項

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

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

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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

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

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

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

www.renesas.com/contact/

商標について

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