Bluetooth Low Energy Smartphone Application Example

TryBT for Android

Introduction

TryBT is an Android sample application that can communicate with the evaluation boards for RX23W, RA4W1, or RE01B which are Renesas Electronics' MCU by Bluetooth® Low Energy wireless technology. This app is distributed as a sample project including source code, so users can customize and reuse the source code.

This document will explain how to create a development environment as well as how to perform basic customization of TryBT.

Target Devices

- Android device (Android OS 6.0 or later)

Related Documents

- RX23W Group Target Board for RX23W Quick Start Guide (R20QS0014)
- RX23W Group Target Board for RX23W module Quick Start Guide (R20QS0022)
- RA4W1 Group Evaluation Kit for RA4W1 EK-RA4W1 Quick Start Guide (R20QS0015)
- RE01B Group Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)
- Renesas Flash Programmer V3.08 Flash memory programming software User's Manual (R20UT4813)

Contents

1. Overview

TryBT works on Android devices running Android OS 6.0 or later. This app connects to an evaluation board and displays a sample screen to perform demonstration. Also, source code of this app is distributed as an Android project that can be modified.

1.1 Operational Environment

Hardware required for checking TryBT operation:

- Android device: Android OS 6.0 or later
- Windows PC
- any of the evaluation boards below:
  - Target Board for RX23W or Target Board for RX23W module NOTE1
  - EK-RA4W1 NOTE2
  - EB-RE01B NOTE3

NOTE1 A firmware that can communicate with TryBT is written to Target Board for RX23W and Target Board for RX23W module at the factory. If you write a firmware for communicating with TryBT to the board again, write the prebuilt firmware included in the quick start guide below to the Target Board for RX23W.

RX23W Group Target Board for RX23W Quick Start Guide (R20QS0014)
→ble_demo_tbrx23w_profile_server_preinstall_yyyymmdd.mot file in mot folder
RX23W Group Target Board for RX23W module Quick Start Guide (R20QS0022)
→ble_demo_mtbrx23w_profile_server_preinstall_yyyymmdd.mot file in mot folder

NOTE2 A firmware that can communicate with TryBT is written to EK-RA4W1 at the factory. If you write a firmware for communicating with TryBT to the board again, write the prebuilt firmware included in the quick start guide below to the EK-RA4W1.

RA4W1 Group Evaluation Kit for RA4W1 EK-RA4W1 Quick Start Guide (R20QS0015)
→Restore_Factory/r20qs0015.srec included in bin.zip

NOTE3 A firmware is not written to EB-RE01B at the factory. If you write a firmware for communicating with TryBT, write the firmware included in the document below to the EB-RE01B.

RE01B Group Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)
→ble_project_server.hex in ROM_Files folder

Hereinafter, this document will describe operations using Target Board for RX23W. These operations are same as when either EK-RA4W1 or EB-RE01B is used.

Android device tested:

- Google Pixel 3a (Android OS 11)

Software required for checking TryBT operation:

- Android Studio (refer to Section 2.1 for more information)

  Note: Use Android Gradle Plugin 4.1.0 or later, because some libraries have been removed in Java 11 or later environment.
- Renesas Flash Programmer (refer to Chapter 5 for more information)

TryBT Project:

- Implementation Language: Kotlin
- Build Configuration (excerpted from build.gradle)

```
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 NOTES

- This document was created based on the status on 3rd Mar 2021. It is not guaranteed that information described in this document supports all of the future versions of software and tools provided by our company or third parties.
- Renesas Electronics disclaims any and all liability arising from the use of information in this document and related software. Please also refer to the "Notice" in the last page of this document.

2. Environment Setup

Install Android Studio that is an IDE for developing Android apps. This document describes how to install Android Studio on Windows PC for the first time. Note that screens in this document are of Android Studio 4.1.1, so it is possible that screens are different when you use different version of Android Studio.

2.1 Downloading Android Studio

1. Access the URL below and click "DOWNLOAD ANDROID STUDIO".
   https://developer.android.com/studio

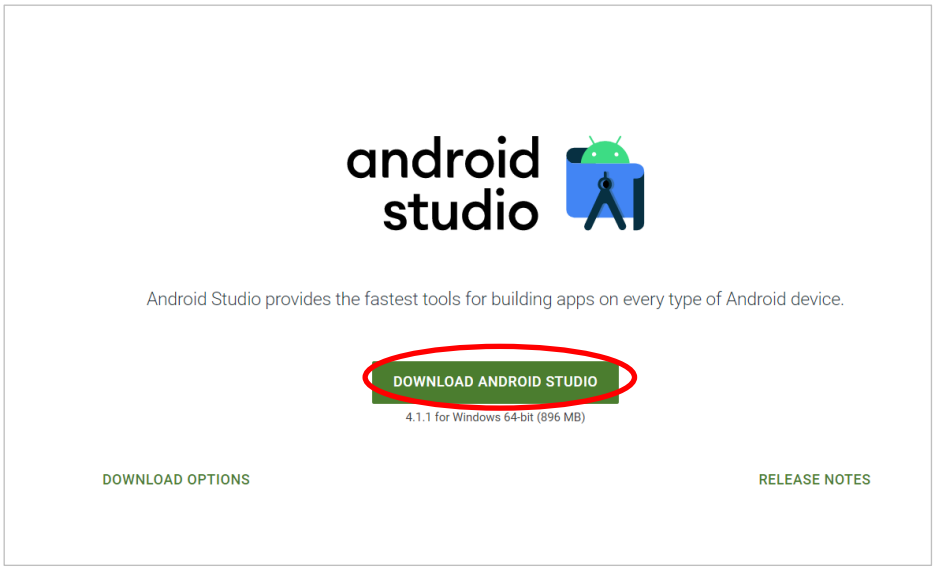

Figure 2.1 Downloading Android Studio (1)

2. Agree to the Terms of Use and click "DOWNLOAD".

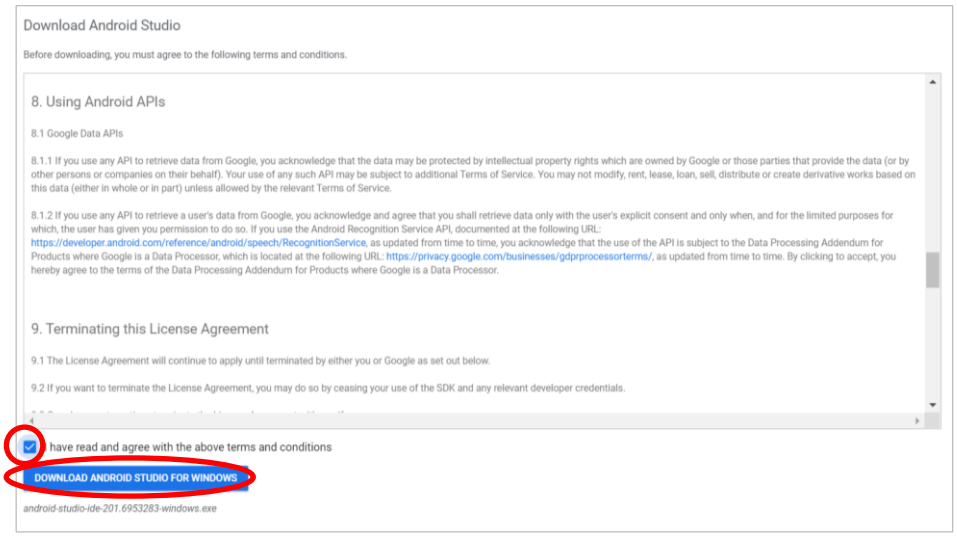

Figure 2.2 Downloading Android Studio (2)

2.2 Installing Android Studio

1. Execute the downloaded Android Studio installer.
2. Click "Next".

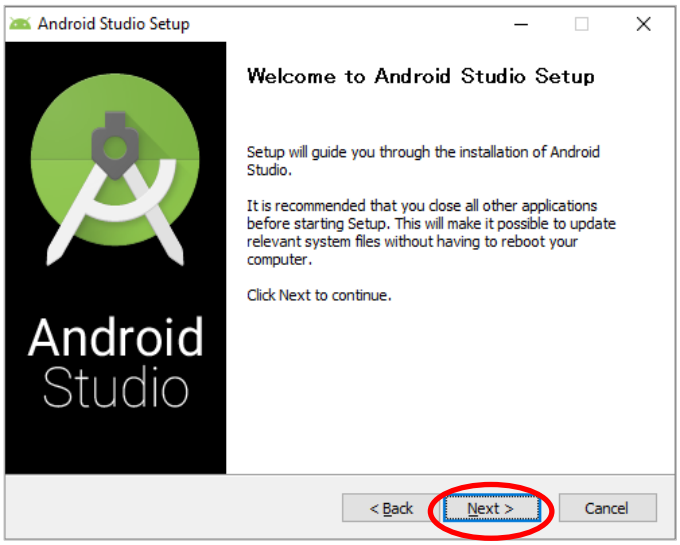

Figure 2.3 Installing Android Studio (2)

3. Uncheck "Android Virtual Device" and click "Next".

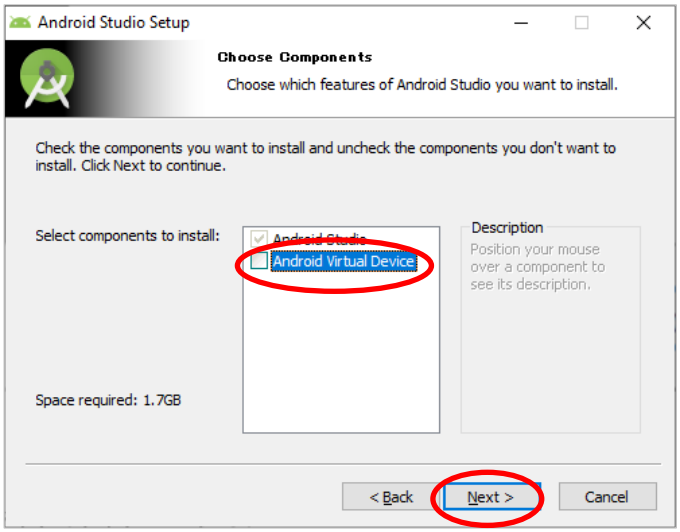

Figure 2.4 Installing Android Studio (3)

4. Set the Installation Location and click "Next".

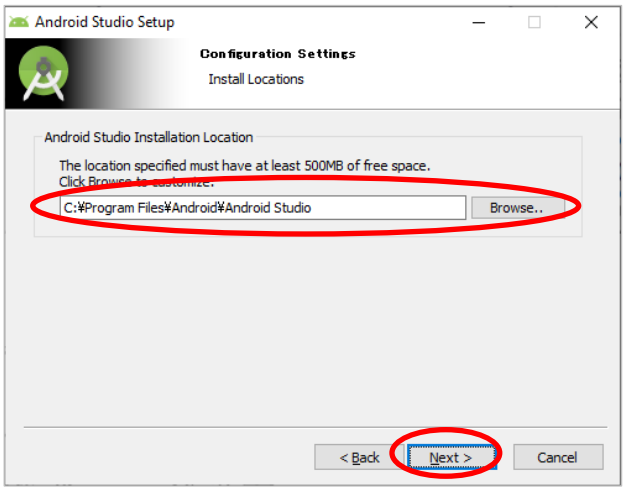

Figure 2.5 Installing Android Studio (4)

5. Click "Install".

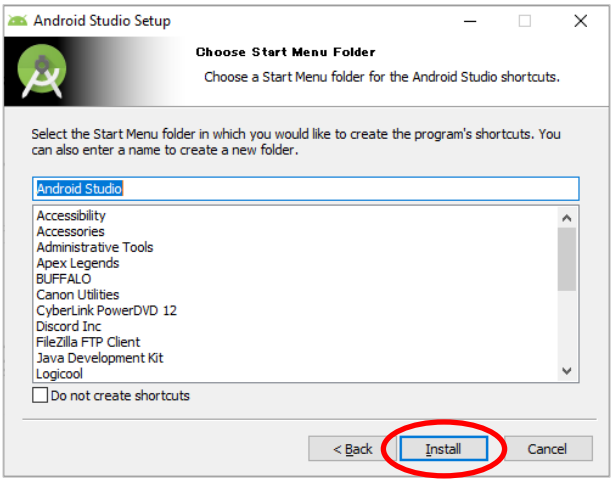

Figure 2.6 Installing Android Studio (5)

6. Click "Next" after the installation finished.

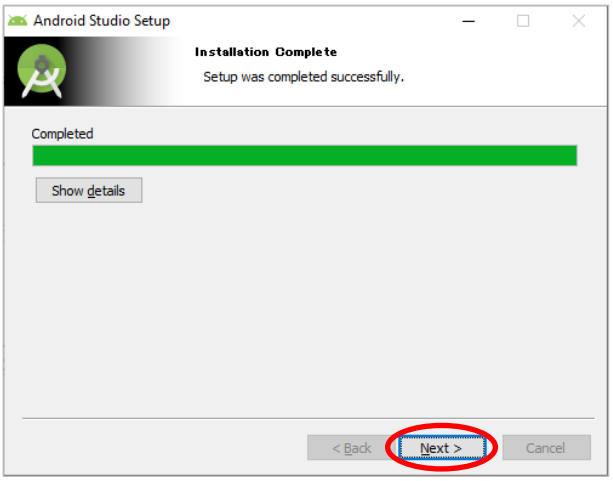

Figure 2.7 Installing Android Studio (6)

7. Click "Finish". Android Studio will start.

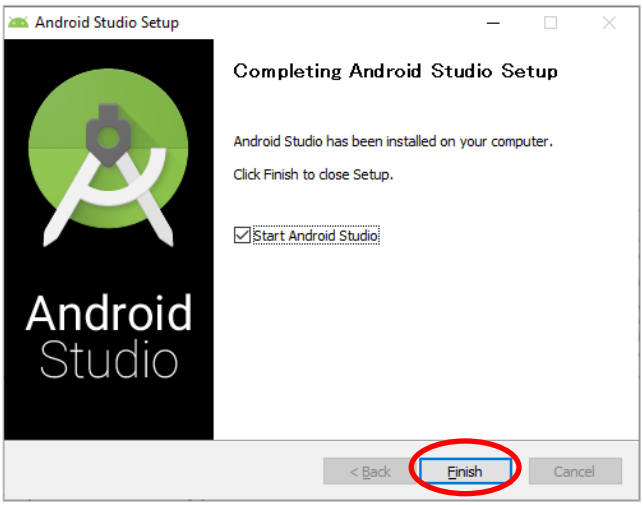

Figure 2.8 Installing Android Studio (7)

8. Choose "Do not import settings" and click "OK".

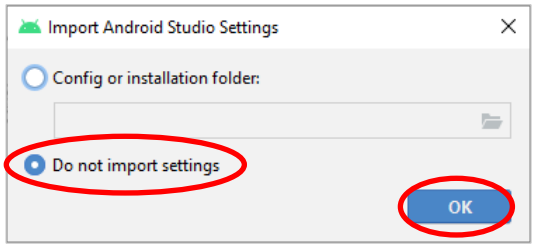

Figure 2.9 Installing Android Studio (8)

9. Click "Next".

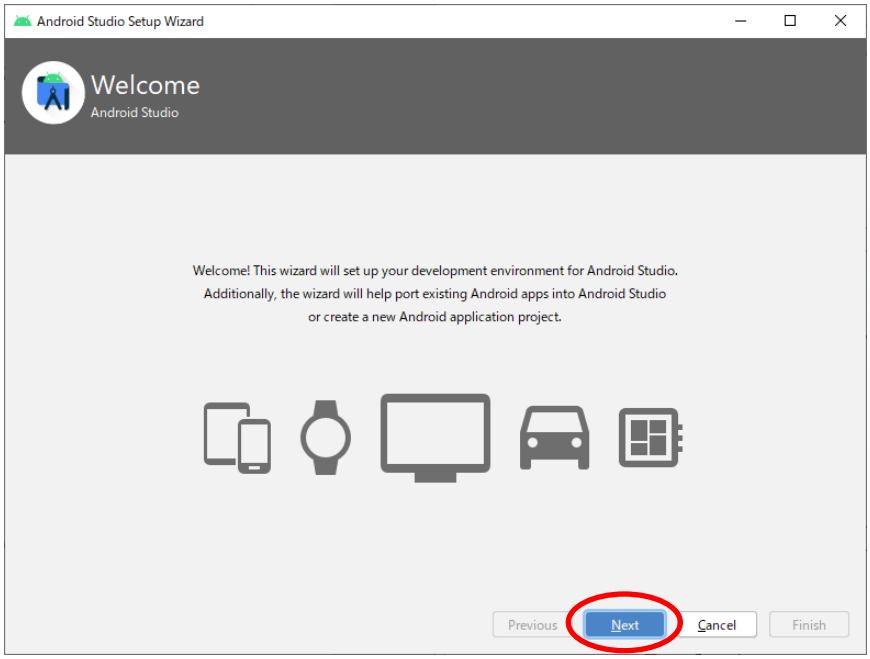

Figure 2.10 Installing Android Studio (9)

10. Click "Next".

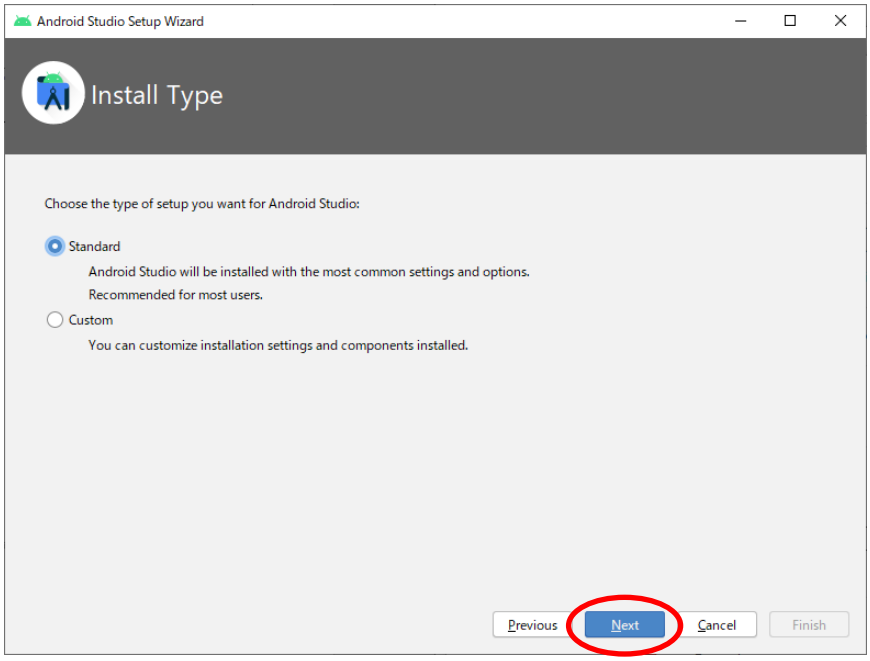

Figure 2.11 Installing Android Studio (10)

11. Click "Next".

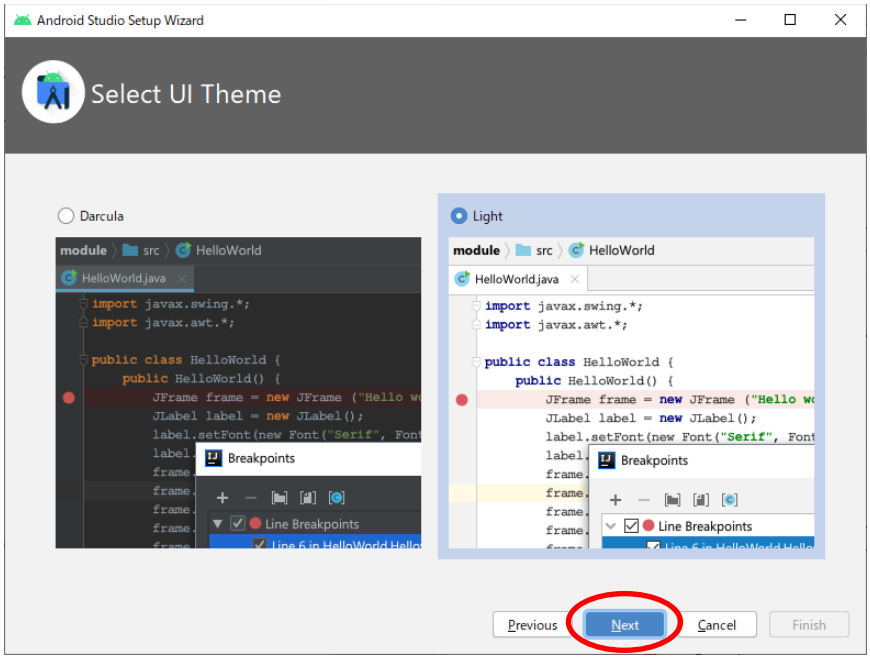

Figure 2.12 Installing Android Studio (11)

12. Click "Finish" after downloading components.

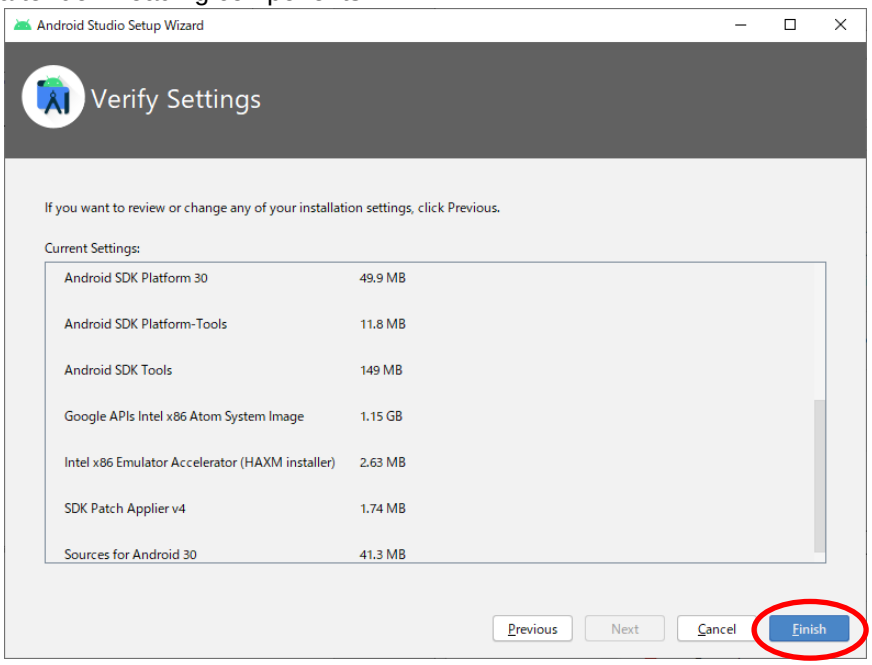

Figure 2.13 Installing Android Studio (12)

2.3 Importing TryBT Project

1. Unzip the TryBT zip file attached to this document.
2. Move the unzipped folder to any folder.
3. Start Android Studio.
4. Select "Open an Existing Project".

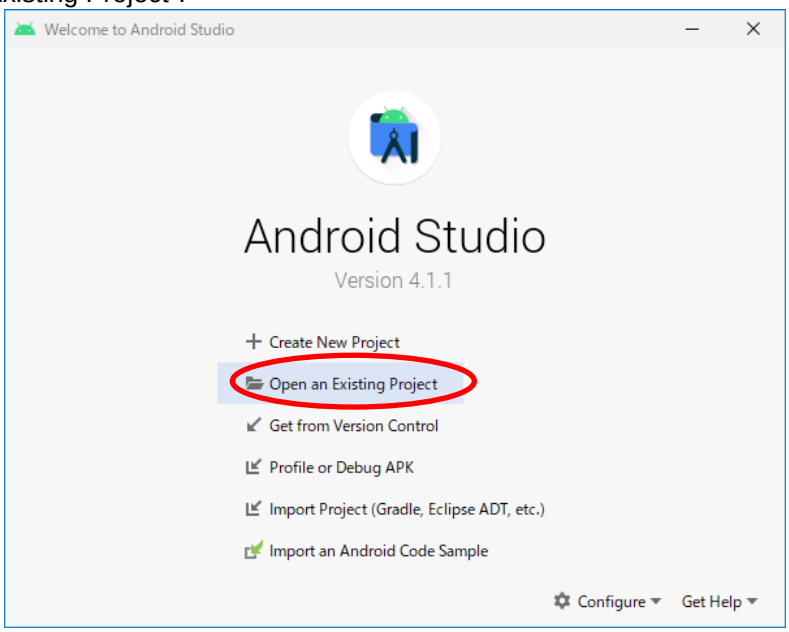

Figure 2.14 Importing TryBT Project (1)

5. Specify the TryBT folder unzipped by the step 2.

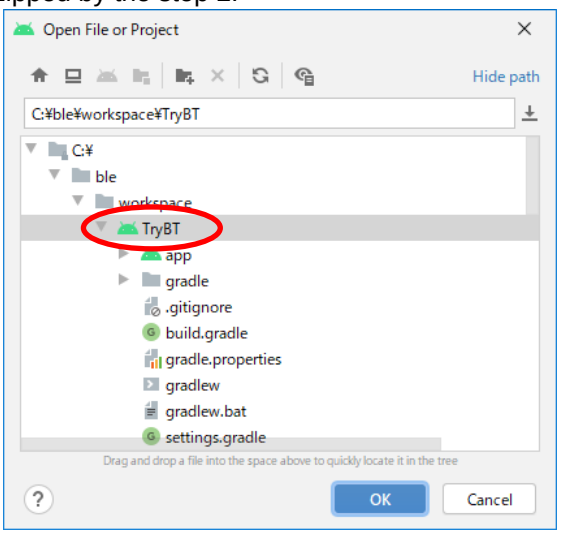

Figure 2.15 Importing TryBT Project (2)

2.4 Installing Android SDK Platform

1. Start Android Studio.
2. To launch SDK Manager, click "SDK Manager" button.

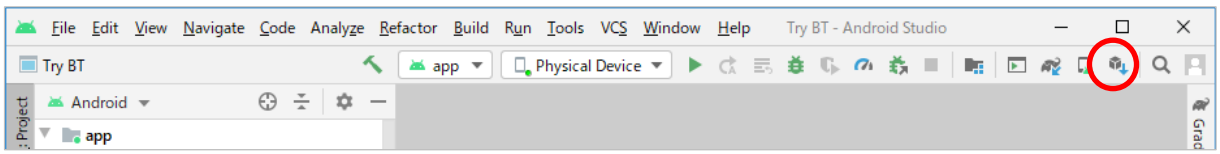

Figure 2.16 Launching SDK Manager

3. To support old versions of Android OS, installing old version of Android SDK by using SDK manager. For example, check "Android 6.0 (Marshmallow): API Level 23 and click "Apply" to support versions of Android OS 6.0 or later.

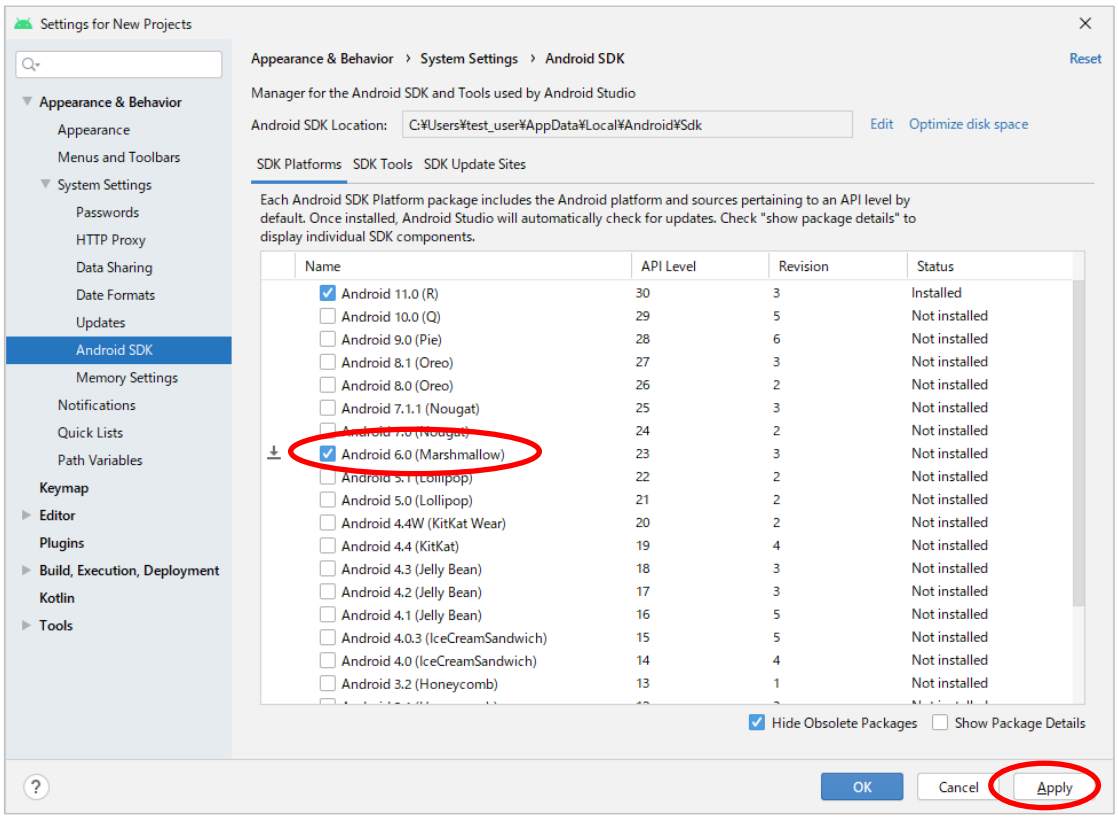

Figure 2.17 Installing Android SDK Platform (1)

4. Click "OK" on Confirm Change dialog.

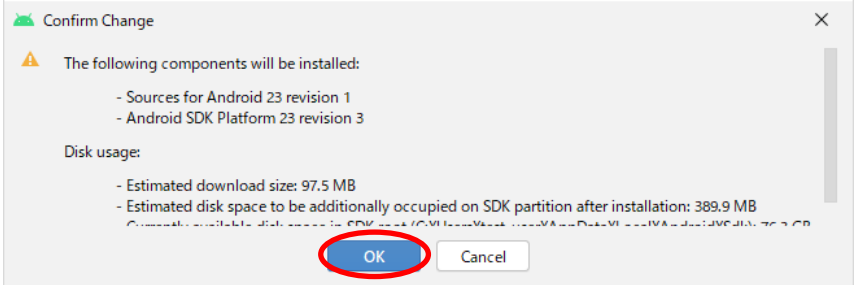

Figure 2.18 Installing Android SDK Platform (2)

5. Select "Accept" on License Agreement dialog and click "Next".

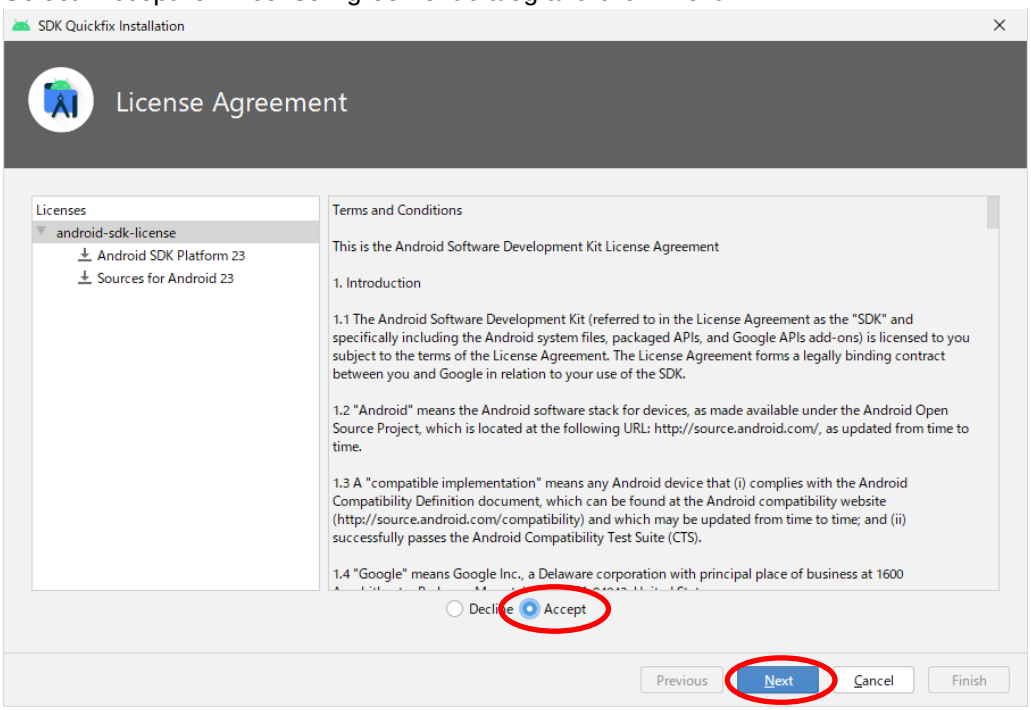

Figure 2.19 Installing Android SDK Platform (3)

6. Click "Finish" after installing the SDK.
7. Click "OK" on SDK Manager.

3. Configuring for using Android device

TryBT works on devices of Android OS 6.0 or later. This chapter describes the setting procedure with Google Pixel 3a as an example.

3.1 USB driver installation for Android

Installing a USB driver to Windows PC is required to develop app with real Android device.

1. Open https://developer.android.com/studio/run/win-usb?hl=en in your browser.
2. Click "Click here to download the Google USB Driver ZIP file" and agree to the Terms of Use and then download.

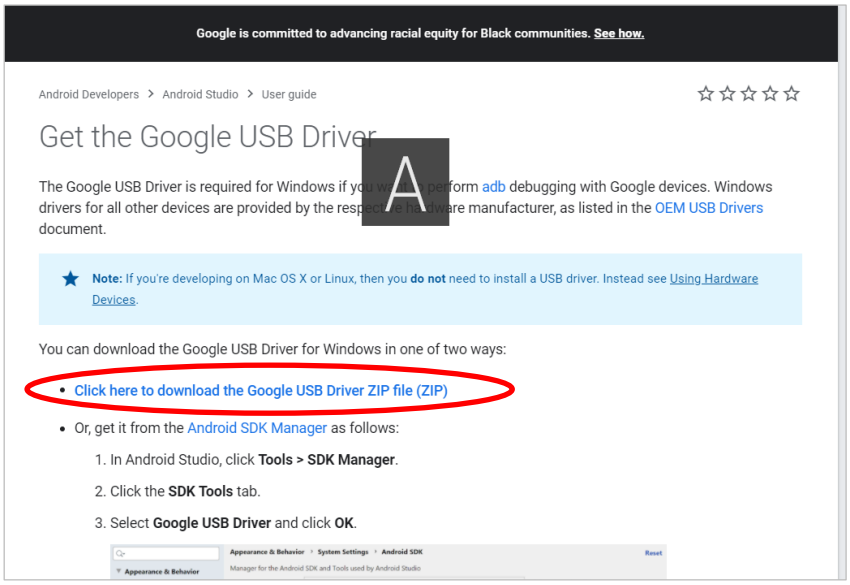

Figure 3.1 Downloading USB driver

3. Unzip the downloaded USB driver zip file.
4. Connect the Android device to the Windows PC via USB.
5. Open Device Manager and confirm that Pixel3 is displayed in "Portable Devices".

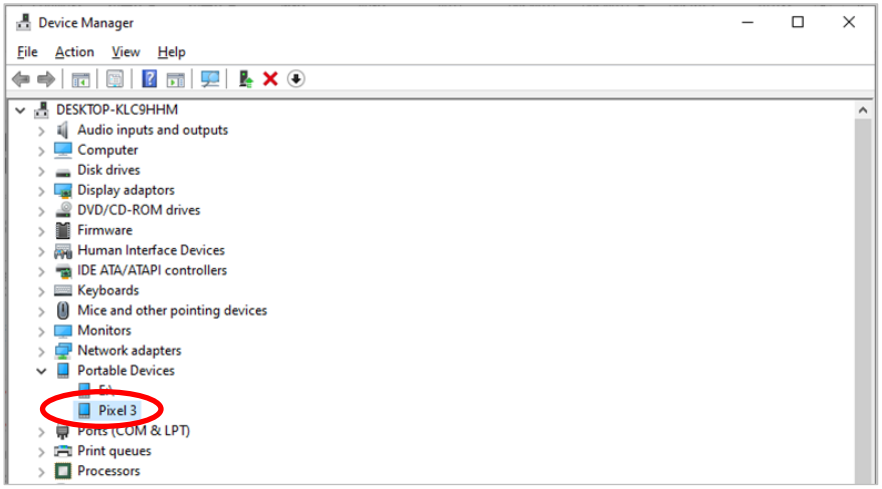

Figure 3.2 Installing USB driver (1)

6. Right-click the Pixel3 and select "Update drivers". Then, select "Browse my computer for driver software".

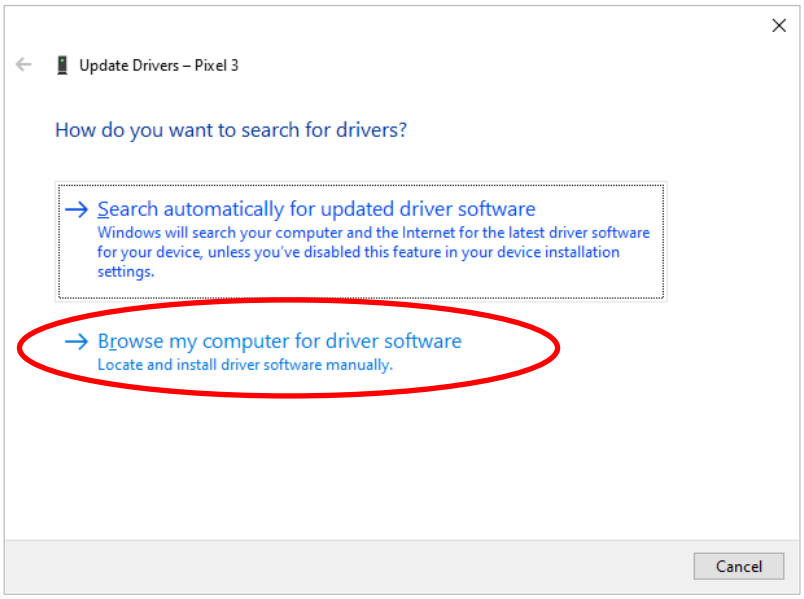

Figure 3.3 Installing USB driver (2)

7. Select the unzipped folder of step 3 and select "Next".

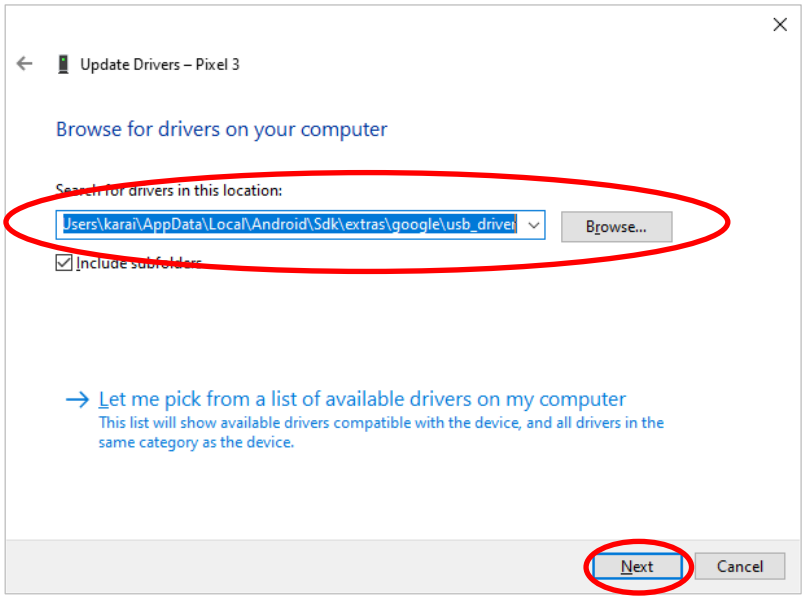

Figure 3.4 Installing USB driver (3)

NOTE: When installing USB driver is not required, "The best drivers for your device are already installed" will be displayed.
NOTE: When installing USB driver fails, see also https://developer.android.com/studio/run/oem-usb?hl=en.

3.2 Setting up Developer Mode on the Android Device

To install the app, Android device used to run TryBT needs to be changed to developer mode. This document describes the procedure for changing the Google Pixel 3a to developer mode.

1. Open Settings and select "About Phone"

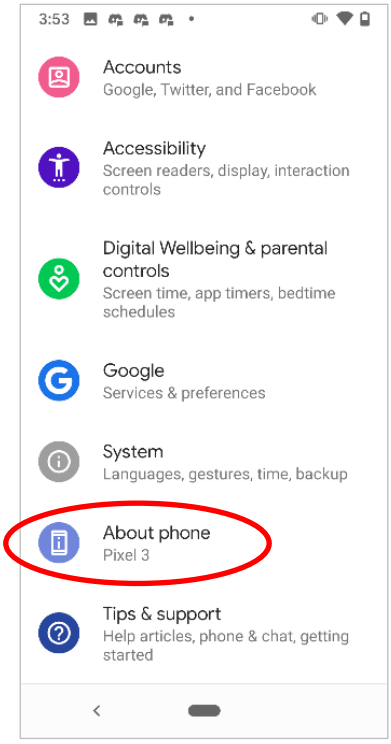

Figure 3.5 Setting up Developer Mode on the Android Device (1)

2. Tap "Build number" repeatedly. Enter your password to unlock Developer Mode. When unlocking is successful, "You are now a developer!" is displayed.

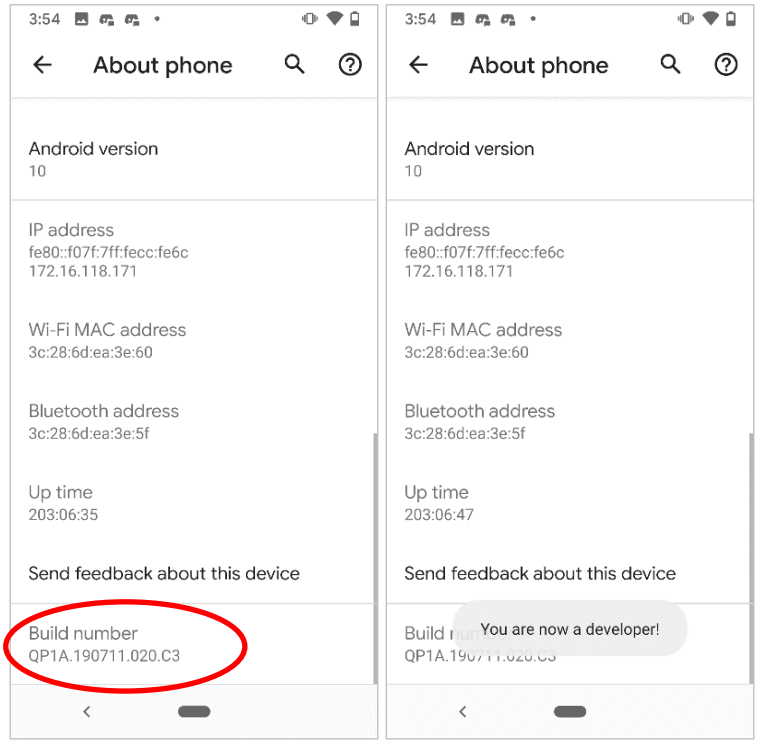

Figure 3.6 Setting up Developer Mode on the Android Device (2)

3. Return to the Settings screen and select "System"→"Developer options".

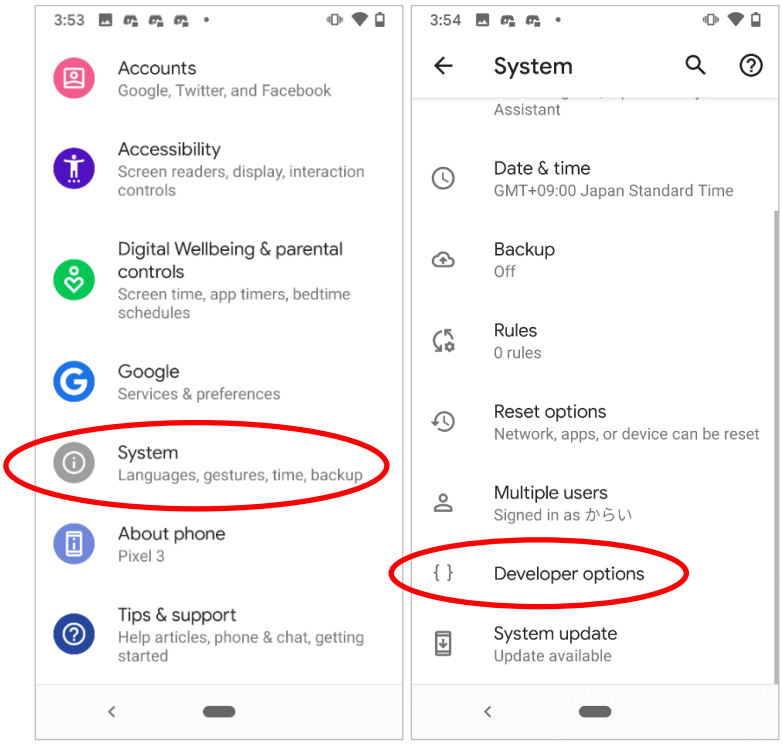

Figure 3.7 Setting up Developer Mode on the Android Device (3)

4. Enable "USB debugging".

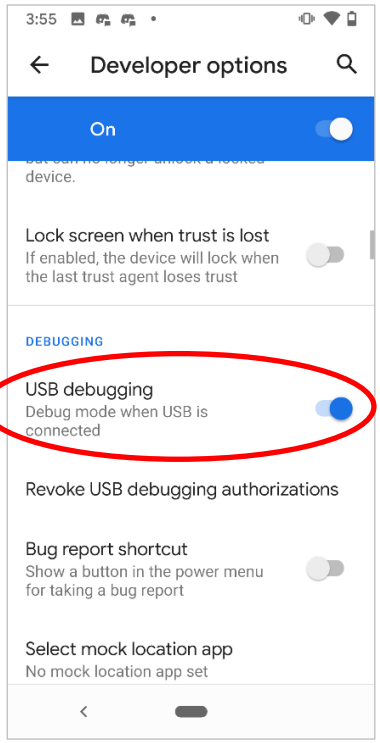

Figure 3.8 Setting up Developer Mode on the Android Device (4)

4. Installing TryBT

1. Start Android Studio and open TryBT project.
2. Connect the Android device to your PC.
3. Allow USB debugging in a dialog displayed on the Android device.
4. The Android device's name will be displayed next to the Android Studio "app".
   NOTE: It may take a long time until Android device's name is displayed at the first time.

Figure 4.1 Installing TryBT (1)

5. When you click the execute button next to the device name, installing TryBT to the Android device will begin.
   NOTE: It may take a long time until installing starts at the first time.

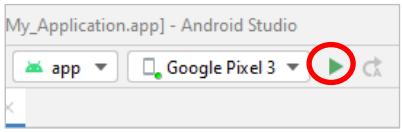

Figure 4.2 Installing TryBT (2)

6. Installing completes when TryBT starts.

Figure 4.3 Splash Screen of TryBT

5. Writing Firmware to Evaluation board

Here is an example of using the Target Board for RX23W as the evaluation board. As for the other boards, refer to documents introduced by section 1.1.

A firmware that can communicate with TryBT is written to Target Board for RX23W at the factory. This chapter describes how to write a firmware to Target Board for RX23W again.

1. Access the URL below. After logging in by My Renesas account and agreeing to the disclaimer, you can download a zip file.
   https://www.renesas.com/document/scd/rx23w-group-target-board-rx23w-quick-start-guide-sample-code

2. Unzip the zip file downloaded by the step 1. Pre-built firmware is the mot file below.
   ./mot/ble_demo_tbrx23w_profile_server_preinstall_20191009.mot

   On the following pages, Steps to write a pre-built firmware to Target Board for RX23W. To write the firmware, the tool below can be used.
   Renesas Flash Programmer (Programming GUI)
   https://www.renesas.com/software-tool/renesas-flash-programmer-programming-gui

3. When you write a firmware, change the ESW 1-2 to ON and connect the ECN1 connector to PC via USB cable.

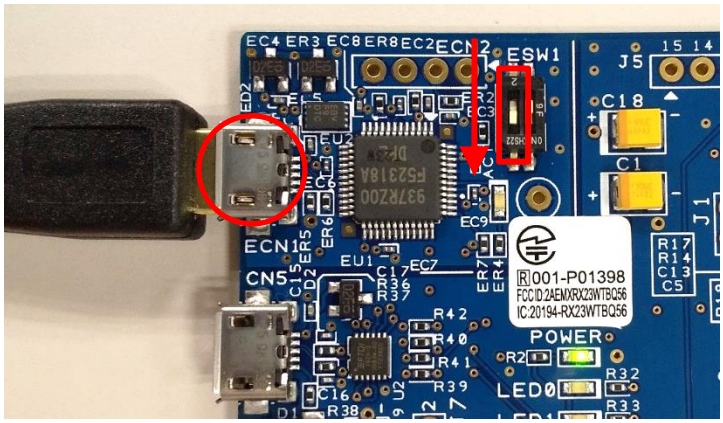

Figure 5.1 Target Board for RX23W Setting for Writing Firmware

4. Start the Renesas Flash Programmer and select "File"→"New Project".

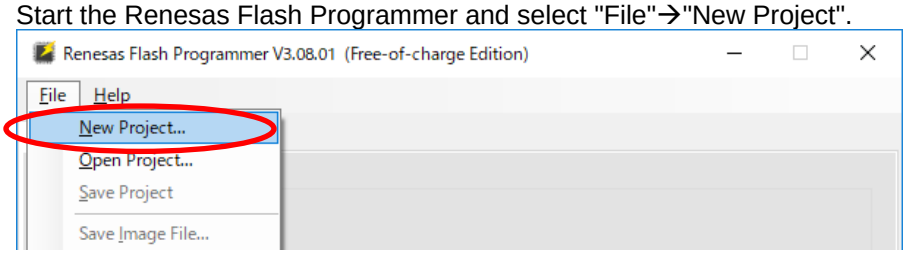

Figure 5.2 Writing Firmware to Target Board for RX23w (1)

5. In the Create New Project dialog, set the following settings and click Connect button.
   Microcontroller:RX200
   Project Name:any name
   Project Folder: any place
   Tool: E2 emulator Lite
   Interface: FINE
   Power: None

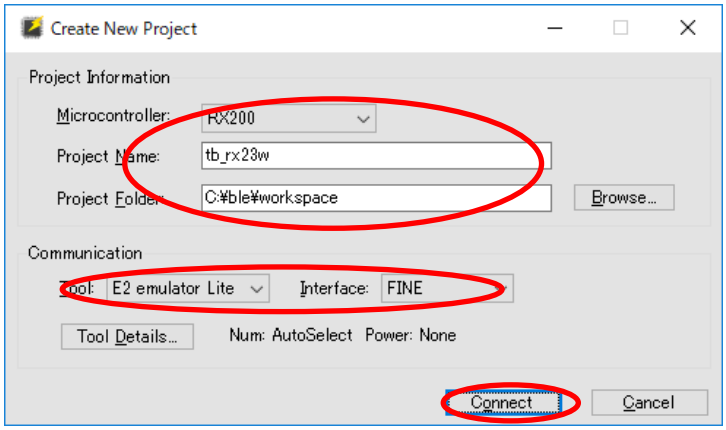

Figure 5.3 Writing Firmware to Target Board for RX23w (2)

6. Configuration steps complete when "Operation completed" is displayed.
7. Specify the following firmware included in the folder of step2 and click Start button.
   Program File: mot/ble_demo_tbrx23w_profile_server_preinstall_20191009.mot

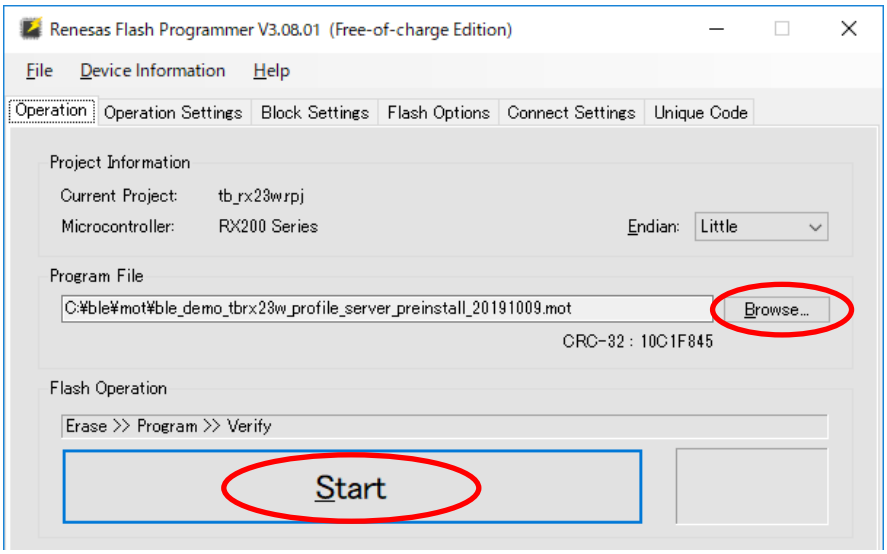

Figure 5.4 Writing Firmware to Target Board for RX23w (3)

8. "Operation completed" will be displayed after writing a firmware.
9. Detach Target Board for RX23W from PC.
10. When you run the firmware, change the ESW 1-2 to OFF and connect the CN5 connector to PC via USB cable.

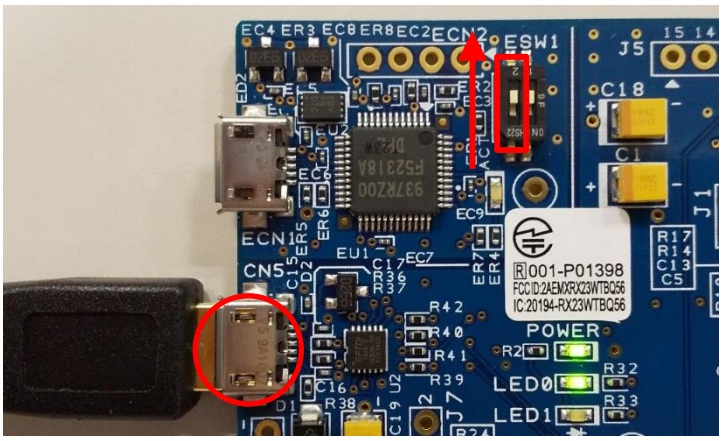

Figure 5.5 Target Board for RX23W Setting for Running Firmware

6. Basic Operation of TryBT

Here is an example of using the Target Board for RX23W as the evaluation board.

6.1 Device List Screen

When the app is launched, Device List screen is displayed. This screen shows a list of connectable devices and its connection status.

Target Board for RX23W is displayed as "RBLE-DEV" in Device List Screen. Tapping "RBLE-DEV" will establish a connection to Target Board for RX23W and display Connected Device Detail Screen.

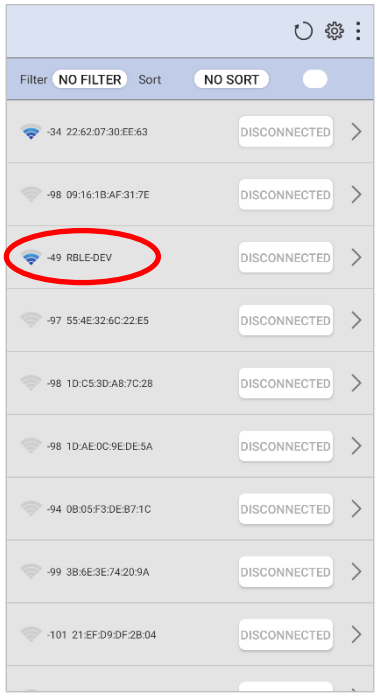

Figure 6.1 Device List Screen (1)

Filtering devices can be performed by selecting Filter type.

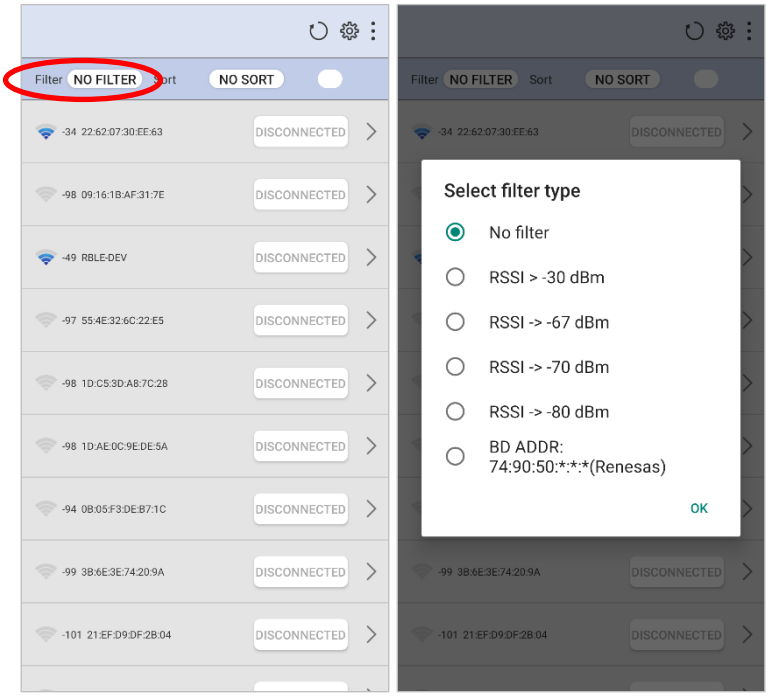

Figure 6.2 Device List Screen (2)

Order of the device list can be specified by selecting Sort order.

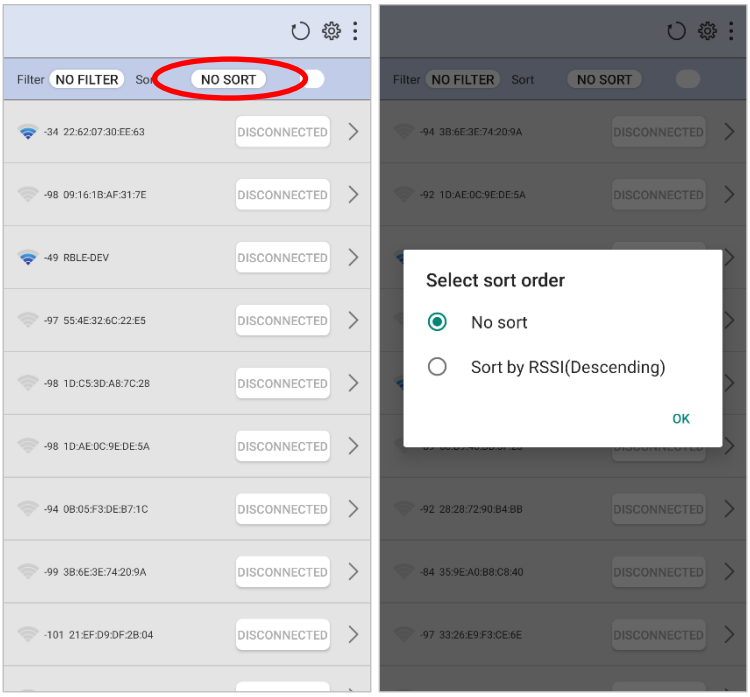

Figure 6.3 Device List Screen (3)

Device list is reloaded by tapping Reload button.

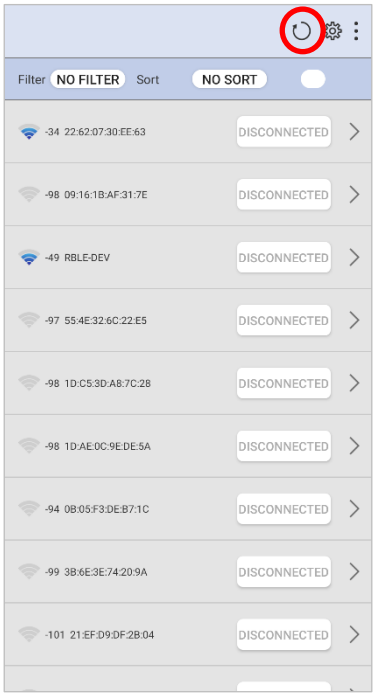

Figure 6.4 Device List Screen (4)

Tapping setting button will display "Register UUID name" and "Bluetooth Settings".
Service Name of GATT Profile and its UUID can be registered by selecting "Register UUID name". Service name registered is displayed in Detailed Information of Connected Device Detail Screen. Note that only one UUID can be registered in this setting. For display example, see Figure 6.11.

UUIDs of Service implemented in Target Board for RX23W are shown below.

To display Service Name implemented in Target Board for RX23W, register any of the following combinations of UUID and Service Name.

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

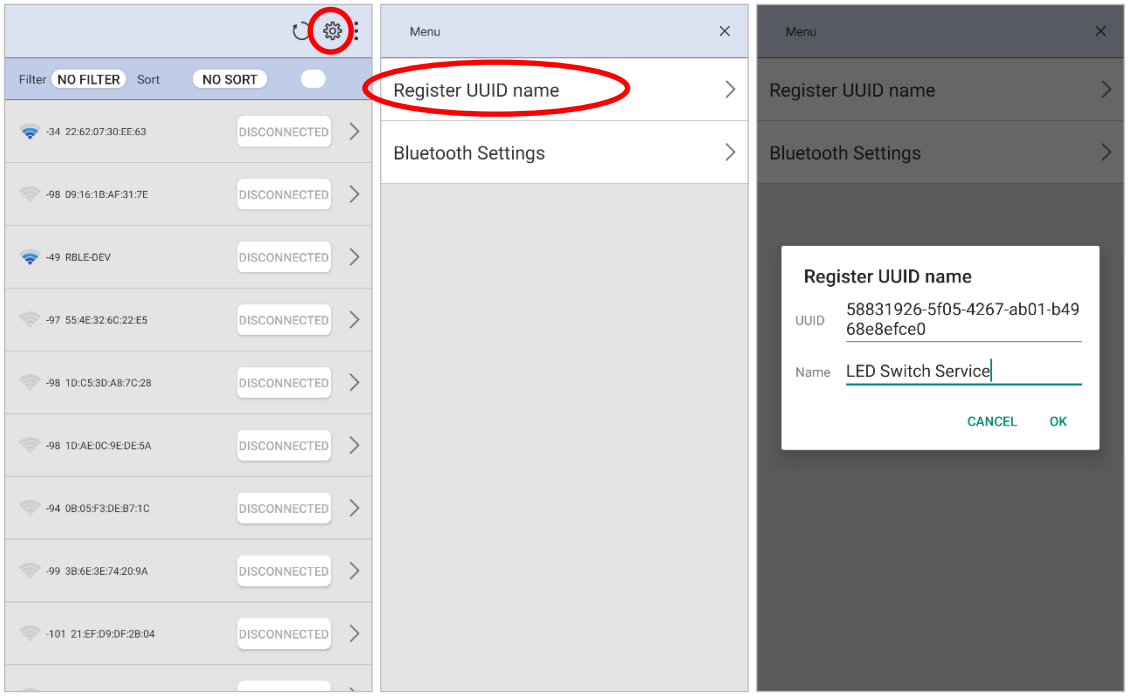

Figure 6.5 Device List Screen (5)

Bluetooth Setting screen of OS can be displayed by selecting "Bluetooth Settings".

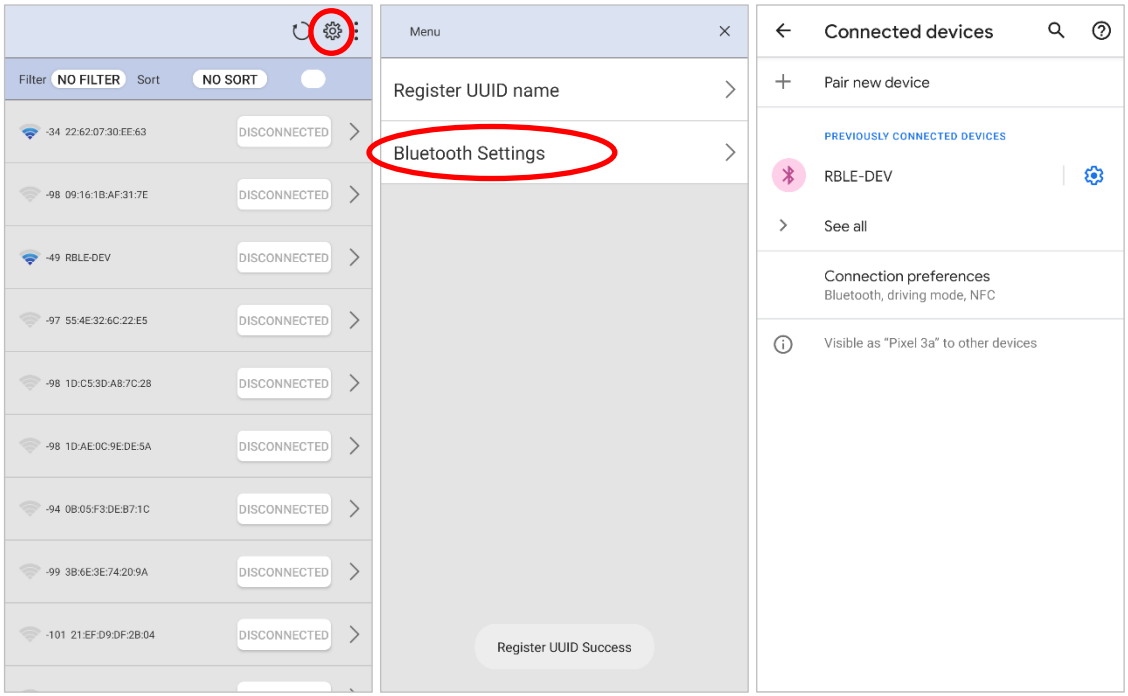

Figure 6.6 Device List Screen (6)

6.2 Connected Device Detail Screen

Connected Device Detail Screen shows the connection status of Target Board for RX23W.

"CONNECTED" indicates the board is connected. Tapping this will terminate this connection. Similarly, "DISCONNECTED" indicates the board is disconnected. Tapping this will establish a connection again.

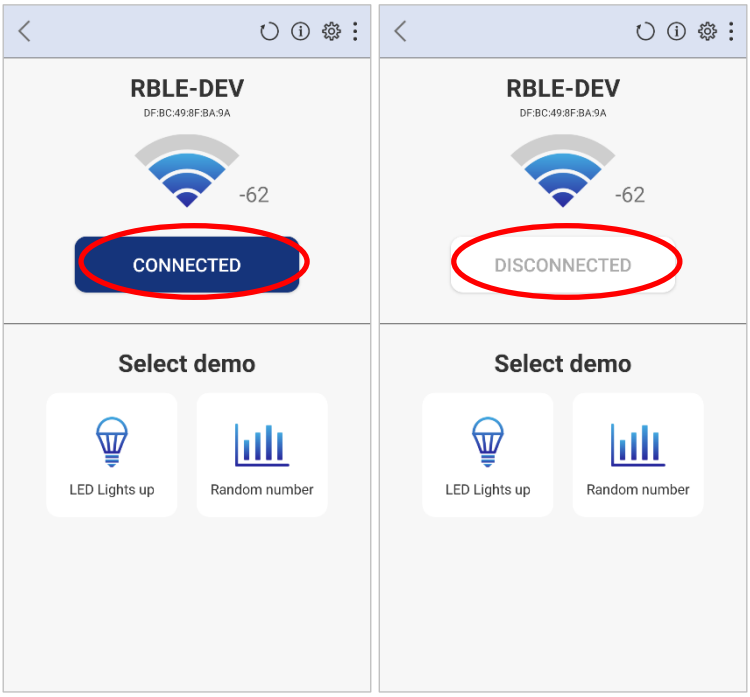

Figure 6.7 Connected Device Detail Screen (1)

Tapping the LED Lights up button will display the Light Demo Screen. To go back to the Connected Device Detail Screen, tap back button on the top left.

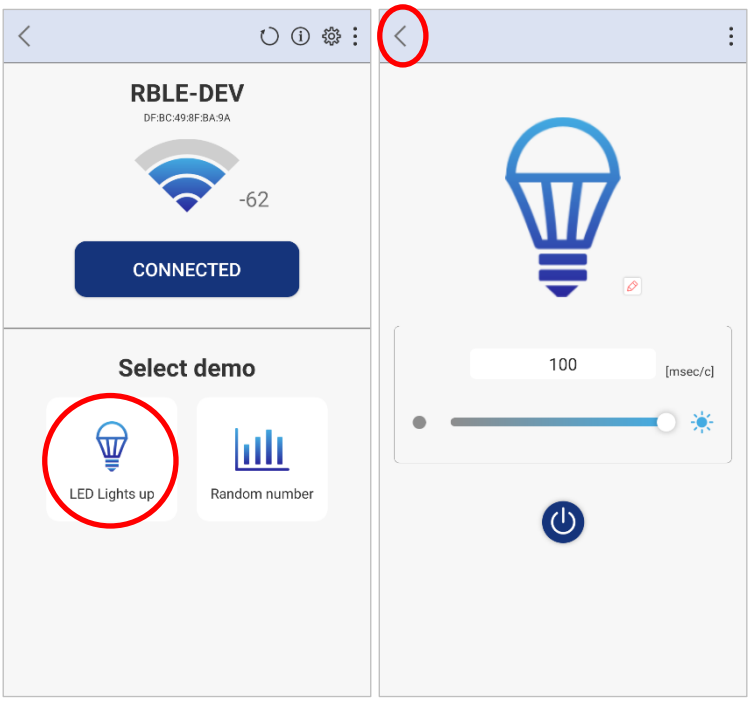

Figure 6.8 Connected Device Detail Screen (2)

Tapping the Random number button will display the data demo screen. To go back to the Connected Device Detail Screen, tap back button on the top left.

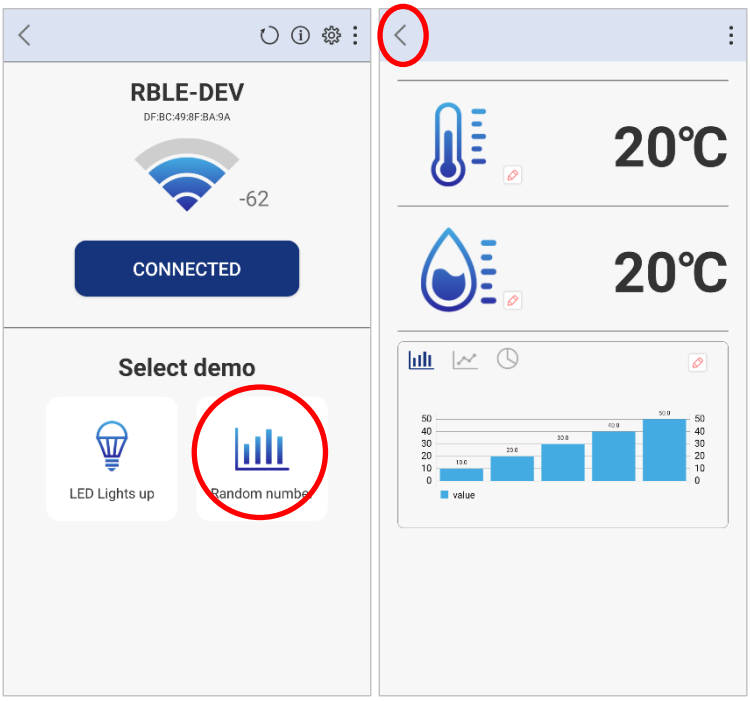

Figure 6.9 Connection Details Screen (3)

Tapping the Reload button will reload the information of Target Board for RX23W connected.

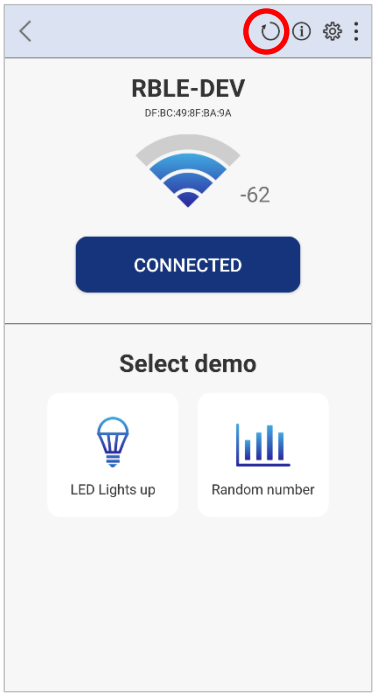

Figure 6.10 Connection Details Screen (4)

Tapping the Info button will display detailed information of evaluation board.

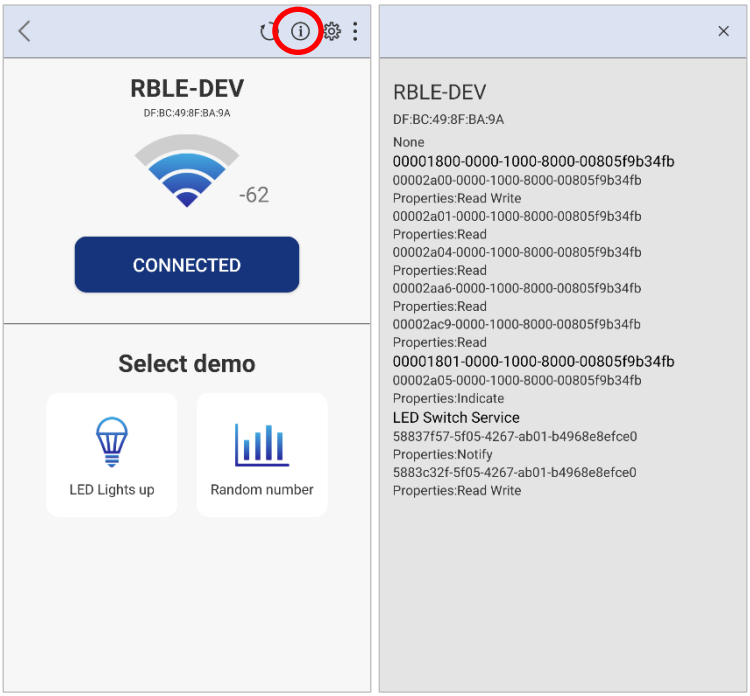

Figure 6.11 Connection Details Screen (5)

Tapping setting button will display "Create bond" and "Bluetooth Settings".
Bluetooth Setting screen of Android OS can be displayed by selecting "Bluetooth Settings".

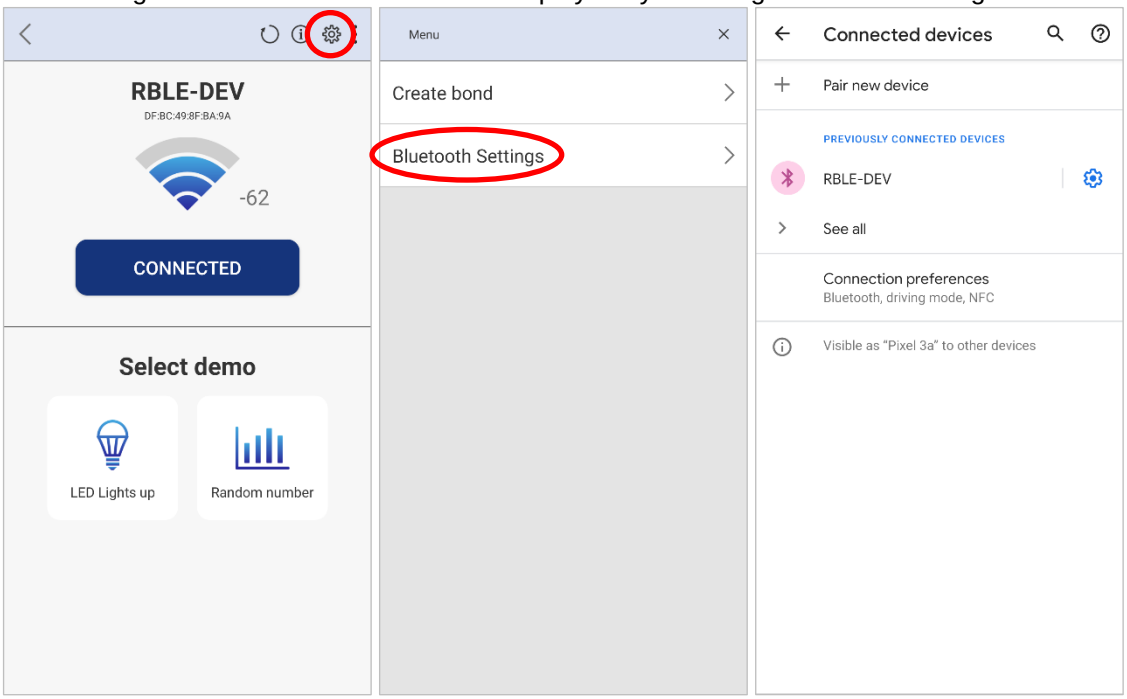

Figure 6.12 Connection Details Screen (6)

Pairing will be performed by selecting "Create bond".

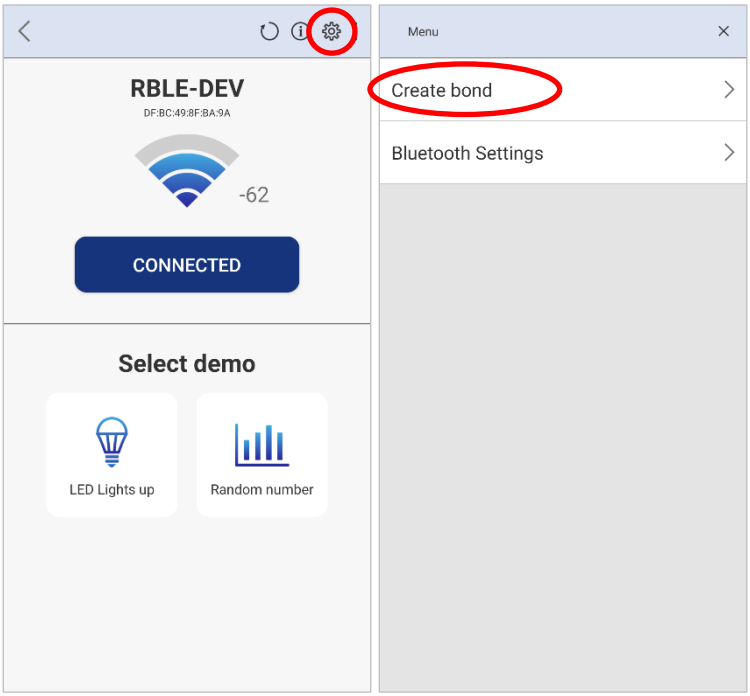

Figure 6.13 Connection Details Screen (7)

NOTE: After paring, RX23W stores bonding information to Data Flash memory. When the bonding information in Data Flash memory is deleted by rewriting a firmware to Target Board for RX23W, it is possible that TryBT becomes unable to reconnect to Target Board for RX23W. In the case that bonding information is deleted, delete bonding information stored in Android device too.

Deleting bonding information in Android device can be performed in Bluetooth Setting Screen of Android OS.

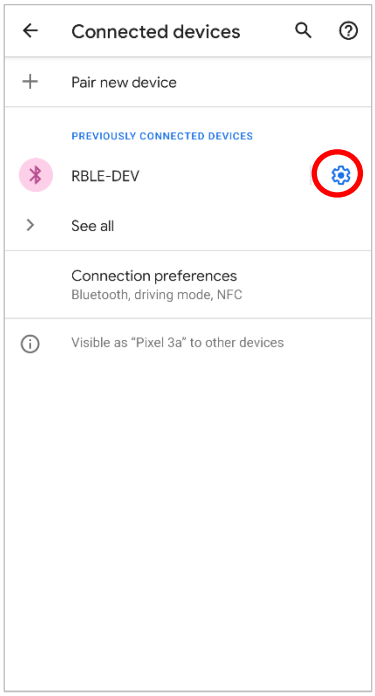

Figure 6.14 Bluetooth Setting Screen of Android OS

6.3 Light Demo Screen

Light Demo screen can operate the LED blinking of Target Board for RX23W LED.

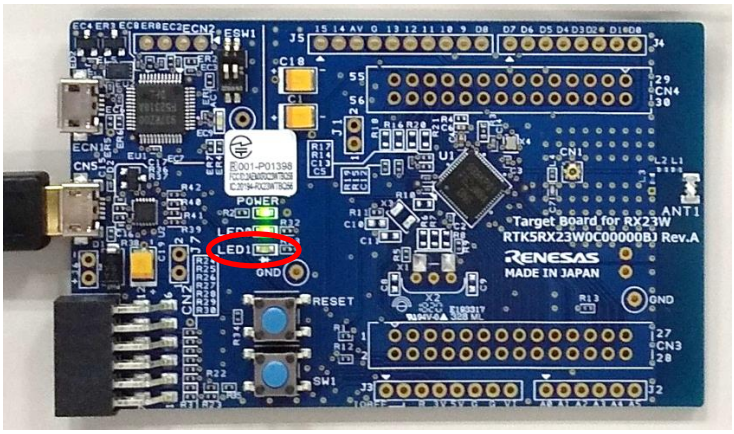

Figure 6.15 User LED on Target Board for RX23W

Specify the blinking interval in milliseconds by using text edit or the slider. (100ms to 10000ms)

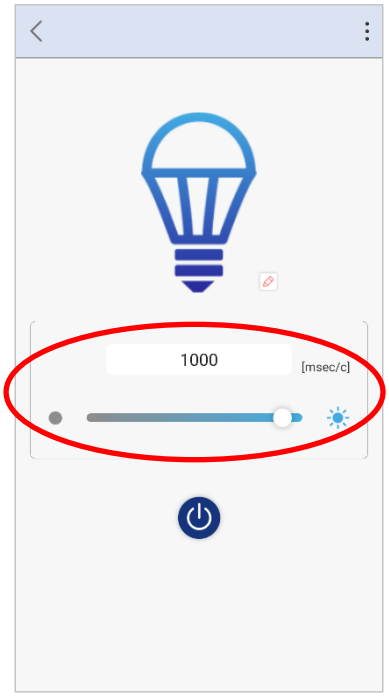

Figure 6.16 Light Demo Screen (1)

Switch on and off by tapping the power button.

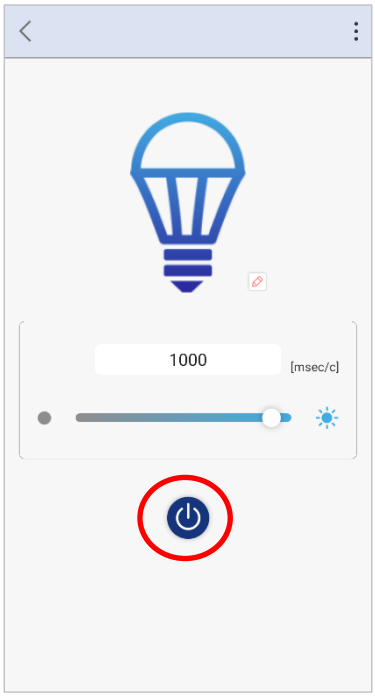

Figure 6.17 Light Demo Screen (2)

TryBT has Customization Mode that can change icon and graph color dynamically. When you tap the pen mark near lamp icon, TryBT changes into Customization Mode and the lamp icon can be changed.

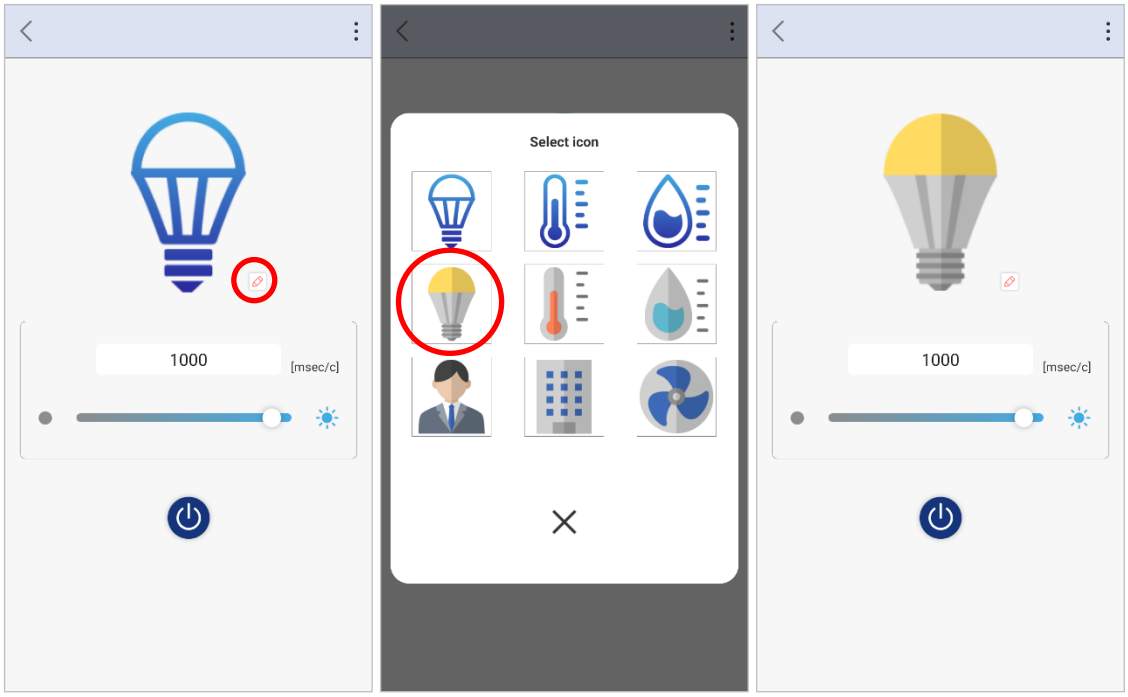

Figure 6.18 Light Demo Screen (3)

6.4 Data Demo Screen

Data Demo screen generates a randomized number and displays temperature, humidity, and graph each time the switch on the Target Board for RX23W is pressed. The graph stores up to ten data points.

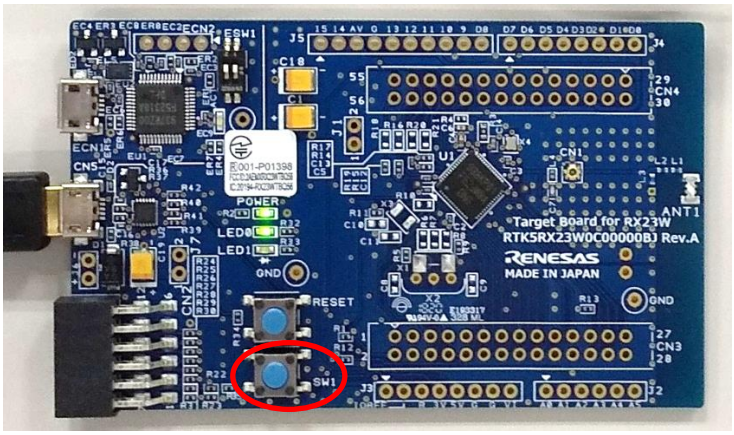

Figure 6.19 User Switch on Target Board for RX23W

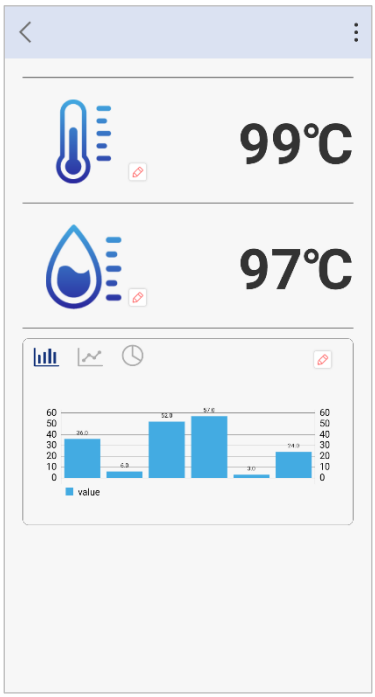

Figure 6.20 Data Demo Screen (1)

Graph format can be switched among bar graph, line graph, and pie chart.

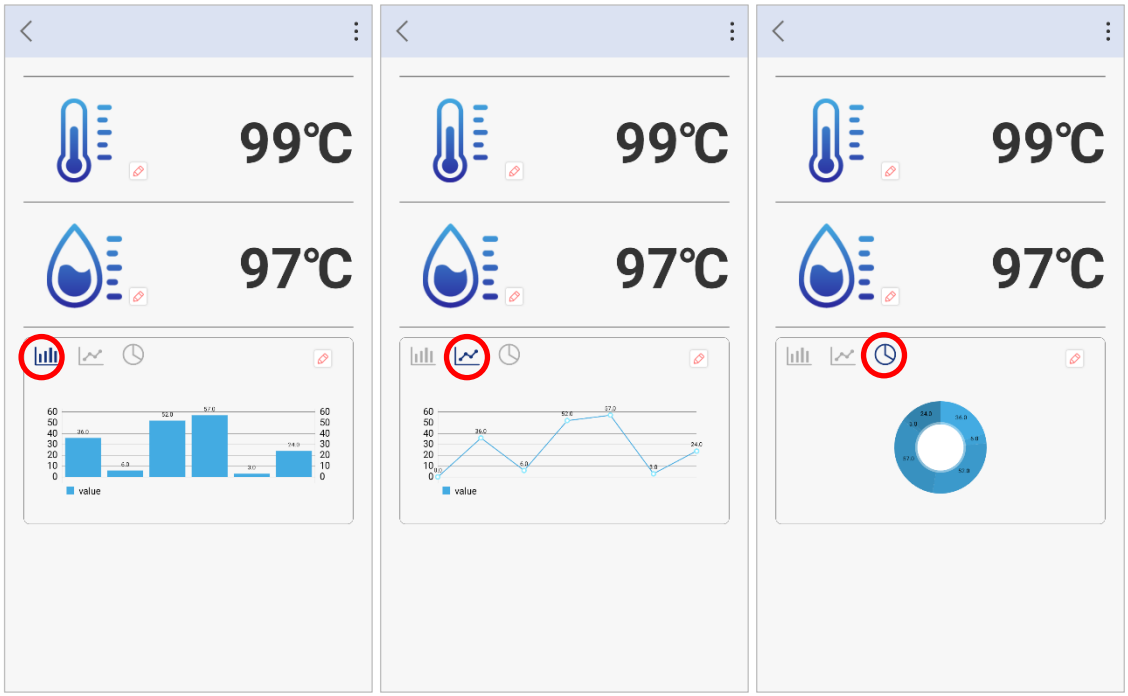

Figure 6.21 Data Demo Screen (2)

When you tap the pen mark near thermometer and hygrometer icons, TryBT changes into Customization Mode and thermometer and hygrometer icons can be changed.

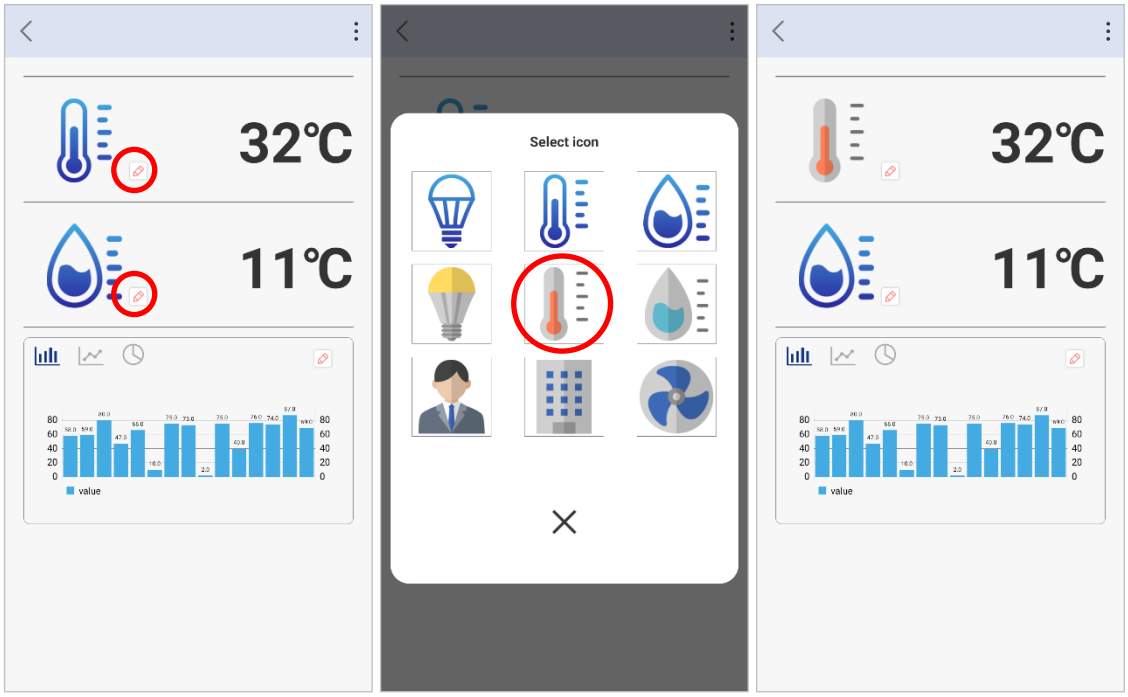

Figure 6.22 Data Demo Screen (3)

When you tap the pen mark near the graph, TryBT changes into Customization Mode and color of the graph can be changed.

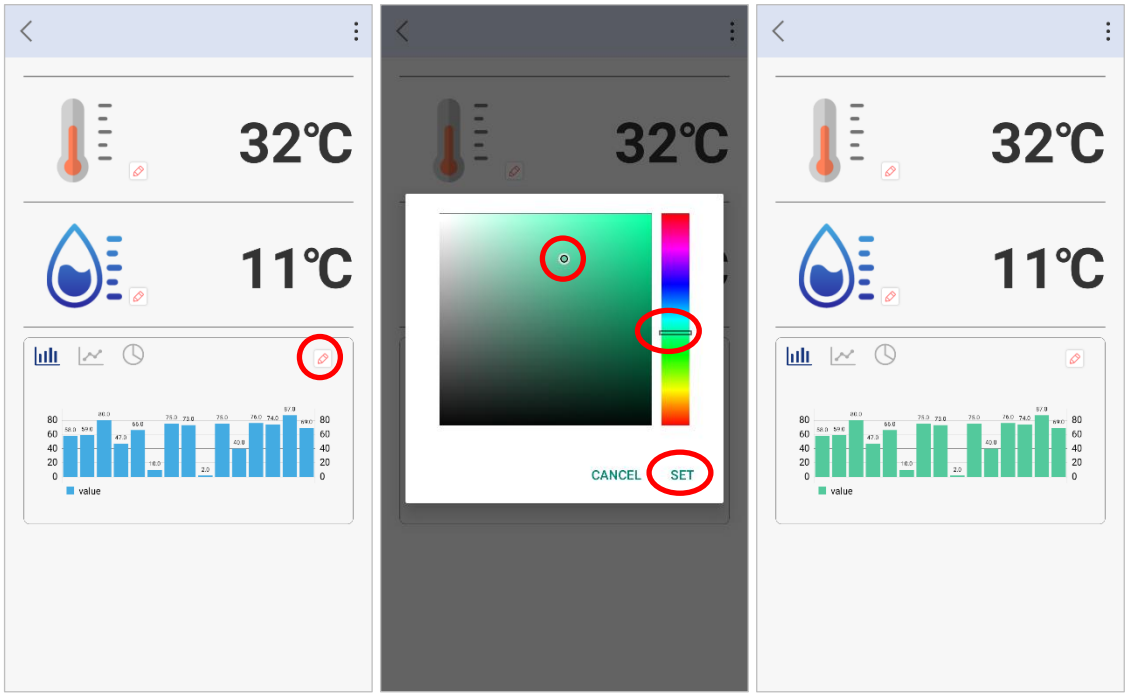

Figure 6.23 Data Demo Screen (4)

7. Customizing TryBT

This chapter describes how to change application title, splash screen, and Icon data as well as how to enable/disable customization mode of TryBT. When you customize software implementation of TryBT, also refer to description regarding TryBT project described in the following chapters.

7.1 Customizing Application Title

The title of this application can be changed.

1. Open the TryBT project in Android Studio. Set the upper-left pane of the screen to "Android" and double-click "app→manifests→AndroidManifest.xml" to open it.

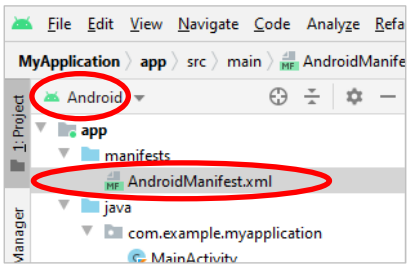

Figure 7.1 Customizing Application Title (1)

2. Changing the android:label attribute of the application tag in AndroidManifest.xml will change the title of the app.

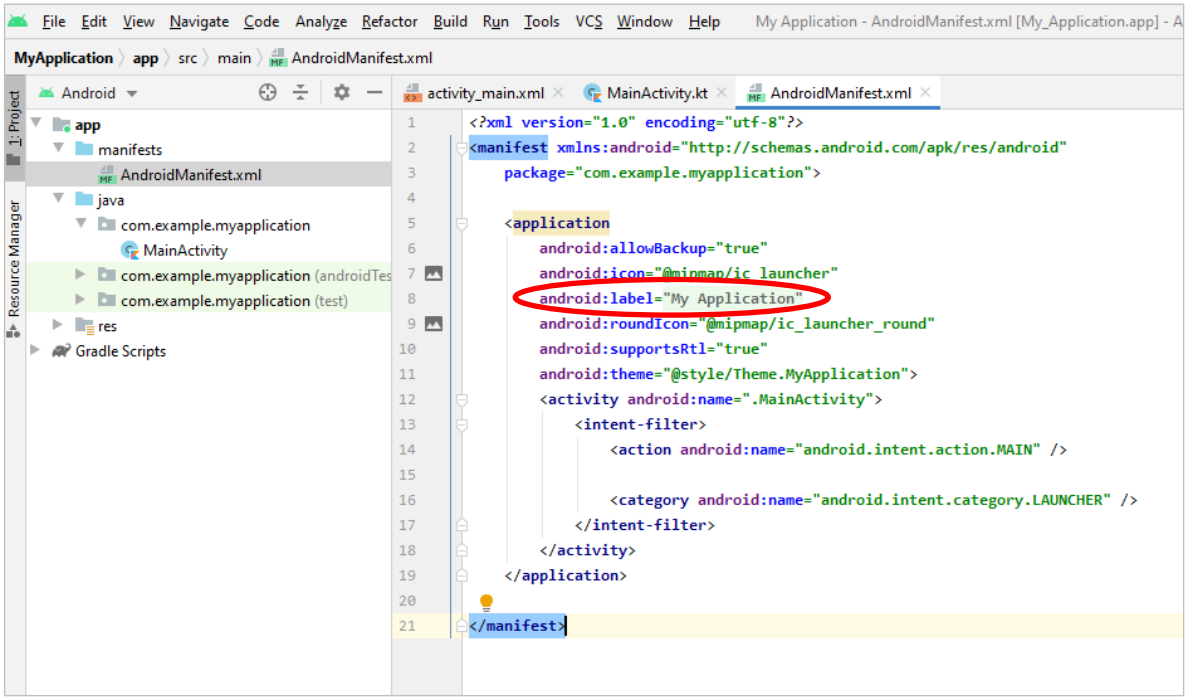

Figure 7.2 Customizing Application Title (2)

3. Re-install the application by the steps described in the Chapter 4.

7.2 Customizing Splash Screen

Splash screen can be changed.

Figure 7.3 Customizing Splash Screen

1. Open the TryBT project in Android Studio. Set the upper-left pane of the screen to "Android" and change "res→drawable→splash_background.png" and "res→drawable→splash_logo.png".

   NOTE: Set the resolution of splash_background.png to 1080x2160 and the resolution of splash_logo.png to 860x287.

2. Re-install the application by the steps described in the Chapter 4.

7.3 Customizing Icon Data on the Demo Screen

Customizable icon on the Light Demo screen and Data Demo screen can be changed.

NOTE: Up to nine icons.

NOTE: Prepare png format icons with a resolution of up to 200x200.

1. Open the TryBT project in Android Studio. Set the upper-left pane of the screen to "Android" and open "res"→"drawable".
2. Replace icon01.png to icon09.png with the icons you prepared.
3. Re-install the application by the steps described in the Chapter 4.

7.4 Enabling/Disabling Customization Mode

This application has a customization mode that can be changed such as icons while the application is running.

1. Open the TryBT project in Android Studio. Set the upper-left pane of the screen to "Android" and double-click "res→values→values.xml" to open.
2. Change the "is_customize_mode" attribute value in values.xml to either true or false.

   true: Enable Customize mode
   false: Disable Customize mode

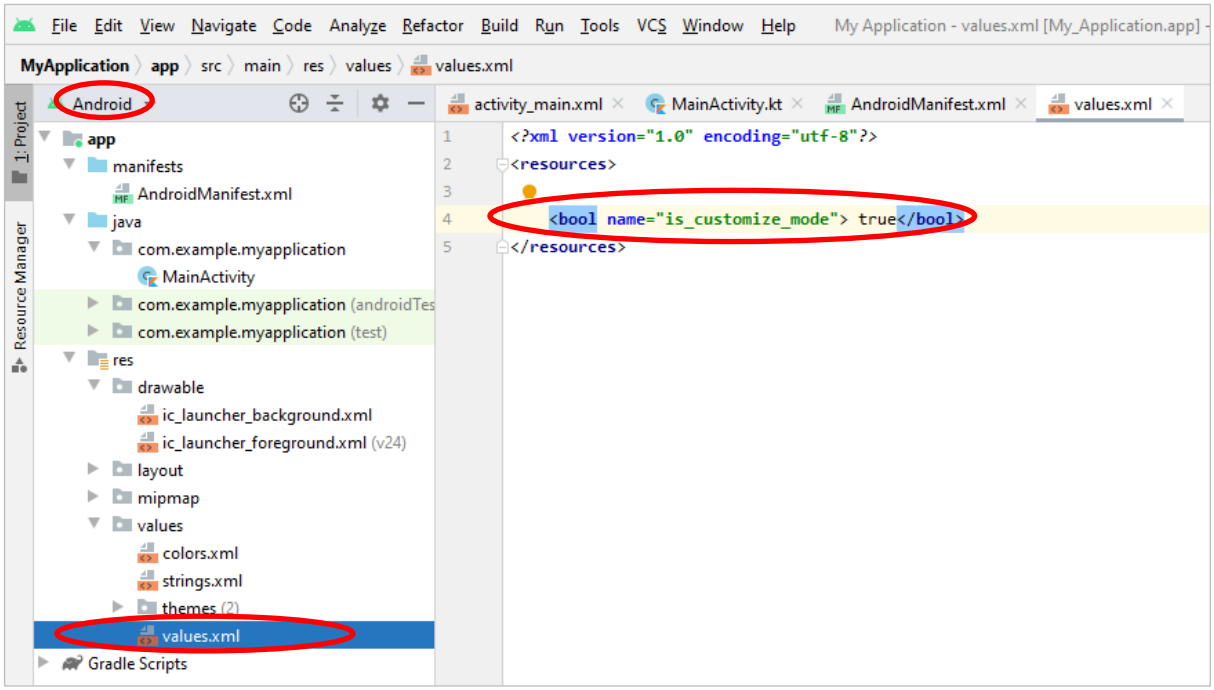

Figure 7.4 Enabling/Disabling Customization Mode

3. Re-install the application by the steps described in the Chapter 4.

8. File Composition of TryBT

Folder and File Composition of TryBT is shown below. build.gradle, AndroidManifest.xml, and kt files are described in this chapter.

Regarding Android Project overview, also refer to https://developer.android.com/studio/projects?hl=en.

```
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/
```

Figure 8.1 Folder and File Composition of TryBT

8.1 About build.gradle

build.gradle is a file to set build configurations of TryBT. There are two build.gradle: one is for a project and the other is for a module.

8.1.1 ./build.gradle (for project)

Build configurations of the whole project is described in this file. It is usually used when there are multiple modules such as app and/or library in a project. TryBT project includes only one app, so its build configuration leaves most of default configurations.

8.1.2 ./app/build.gradle (for module)

Each build configuration for app and library can be described in this file when there are multiple modules in a project. TryBT describes a build configuration for one app to it. Major configuration items are as follows.

Table 8-1 Attributes of build.gradle

Attribute	Description	
compileSdkVersion	Android SDK version for building app is specified by API level. e.g.) 29 Note: Specify API level greater than or equal to the level specified by minSdkVersion. For Android SDK version and API level, refer to the following web page. https://developer.android.com/studio/releases/platforms?hl=en	
buildToolsVersion	Build tool version is specified. e.g.) "29.0.3" For build tool version, refer to the following web page. https://developer.android.com/studio/releases/build-tools?hl=en	
defaultConfig	applicationId	Application ID is specified. Usually, a company domain name of inverse order followed by an app name is used. e.g.) "com.renesas.trybt"
	minSdkVersion	Minimum API level that is supported by app is specified. e.g.) 23
	versionCode	Version of app is specified by a serial number. This is not shown to users. e.g.) 2
	versionName	Version of app for showing to users is specified by a string. e.g.) "1.0.0
dependencies	implementation	External libraries used by a project are specified. e.g.) 'com.google.firebase:firebase-analytics-ktx'

8.2 About ./app/src/main/AndroidManifest.xml

AndroidManifest.xml is a file to define TryBT's screen composition and necessary permissions.

Table 8-2 Attributes of AndroidManifest.xml

Attribute	Description	
users-permission	android:name	Necessary permissions from OS such as Bluetooth and Internet Access are specified. e.g.) "android.permission.BLUETOOTH"
application	android:name	Deriving class from android.app.Application for controlling the whole app is specified. e.g.) ".MainApplication"
	android:icon	Icon of app is specified. Icon is placed to mipmap folder or drawable folder. e.g.) "@mipmap/ic_launcher"
	android:label	Application name is specified. Application name described in string.xml is referred. e.g.) "@string/app_name"
activity	android:name	Activities used in app are declared. e.g.) "ui.menu.MenuListActivity" NOTE: If screen is added without declaring its activity, app might crash.

8.3 About folder composition and .kt files in ./app/src/main/java

Logic of TryBT is implemented by Kotlin language, and its file extension is .kt. Usually, single class is implemented in each kt file, and each file name is same as class name.

kt files are managed by a group called as package. Each package is included in ./app/src/main/java folder. In the case that package name is com.renesas.trybt.ui.bluetoothDetail, its package folder is com/renesas/trybt/ui/bluetoothDetail.

This section describes overview of class implemented in each file.

- com.renesas.trybt package

This package contains classes to manage a lifecycle of TryBT.

Table 8-3 Classes in com.renesas.trybt

Class	Description
`MainApplication`	This class manages lifecycle of the whole app. `MainApplication` of TryBT also manages status change events of Bluetooth device.
`MyFirebaseMessagingService`	This class describes processing for push notification.
`SplashActivity`	This class describes splash screen.

- com.renesas.trybt.ui package

This package contains packages and classes for each screen of TryBT.

Table 8-4 Classes in com.renesas.trybt.ui

Class	Description
`AppSettingManager`	This class stores and reads information of each screen such as sorting setting.

- com.renesas.trybt.ui.bluetoothDetail package

This package contains classes related to Connected Device Detail Screen.

Table 8-5 Classes in com.renesas.trybt.ui.bluetoothDetail

Class	Description
`BluetoothDetailActivity`	This class defines processing for Connected Device Detail Screen.
`BluetoothDeviceInfoActivity`	This class defines processing for Device Information Screen.

- com.renesas.trybt.ui.bluetoothList package

This package contains classes related to Device List Screen.

Table 8-6 Classes in com.renesas.trybt.ui.bluetoothList

Class	Description
`BluetoothListActivity`	This class defines processing for Device List Screen.
`BluetoothListAdapter`	This class defines cell of each device in a table of Device List Screen.
`BluetoothListFilterDialogFragment`	This class defines processing for Filter Dialog of Device List.
`BluetoothListSortDialogFragment`	This class defines processing for Sort Dialog of Device List.
`BluetoothUUIDDialogFragment`	This class defines processing for UUID Registration Dialog of Device List.

- com.renesas.trybt.ui.colorPicker package

This package contains a picker class to select color used in Connected Device Detail Screen.

Table 8-7 Classes in com.renesas.trybt.ui.colorPicker

Class	Description
`ColorPickerDialogFragment`	This class defines Picker Dialog to select color.

- com.renesas.trybt.ui.iconSelect package

This package contains classes to select icon used in some screens.

Table 8-8 Classes in com.renesas.trybt.ui.iconSelect

Class	Description
`IconSelectDialogFragment`	This class defines Icon Select Dialog.
`IconSelectPageAdapter`	This class defines a layout of each icon in Icon Select Dialog.

- com.renesas.trybt.ui.ligthsUpDemo package

This package contains a class for Light Demo Screen.

Table 8-9 Classes in com.renesas.trybt.ui.lightsUpDemo

Class	Description
`LightsUpDemoActivity`	This class defines Light Demo Screen.

- com.renesas.trybt.ui.menu package

This package contains classes for Menu List Screen.

Table 8-10 Classes in com.renesas.trybt.ui.menu

Class	Description
`MenuListActivity`	This class defines Menu List Screen.
`MenuListAdapter`	This class defines description of each menu.
`MenuInfo`	This class defines description element of each menu.

- com.renesas.trybt.ui.randomNumberDemo package

This package contains a class for Data Demo Screen.

Table 8-11 Classes in com.renesas.trybt.ui.randomNumberDemo

Class	Description
`RandomNumberDemoActivity`	This class defines Data Demo Screen.

- com.renesas.trybt.ui.setting package

This package contains classes for some setting screens.

Table 8-12 Classes in com.renesas.trybt.ui.setting

Class	Description
`BluetoothDetailSettingActivity`	This class a setting screen for Connected Device Detail Screen.
`BluetoothListSettingActivity`	This class a setting screen for Device List Screen.

- com.renesas.trybt.ui.versionInfo package

This package contains a class for Version Information Screen.

Table 8-13 Classes in com.renesas.trybt.ui.versionInfo

Class	Description
`VersionInfoActivity`	This class defines Version Information Screen.

- com.renesas.trybt.util package

This package contains a utility class.

Table 8-14 Classes in com.renesas.trybt.util

Class	Description
`TryBTUtil`	This class defines generic processing such as decisioning antenna icon in accordance with RSSI value.

9. Screen Transition of TryBT

Screen and class transition of TryBT is shown below.

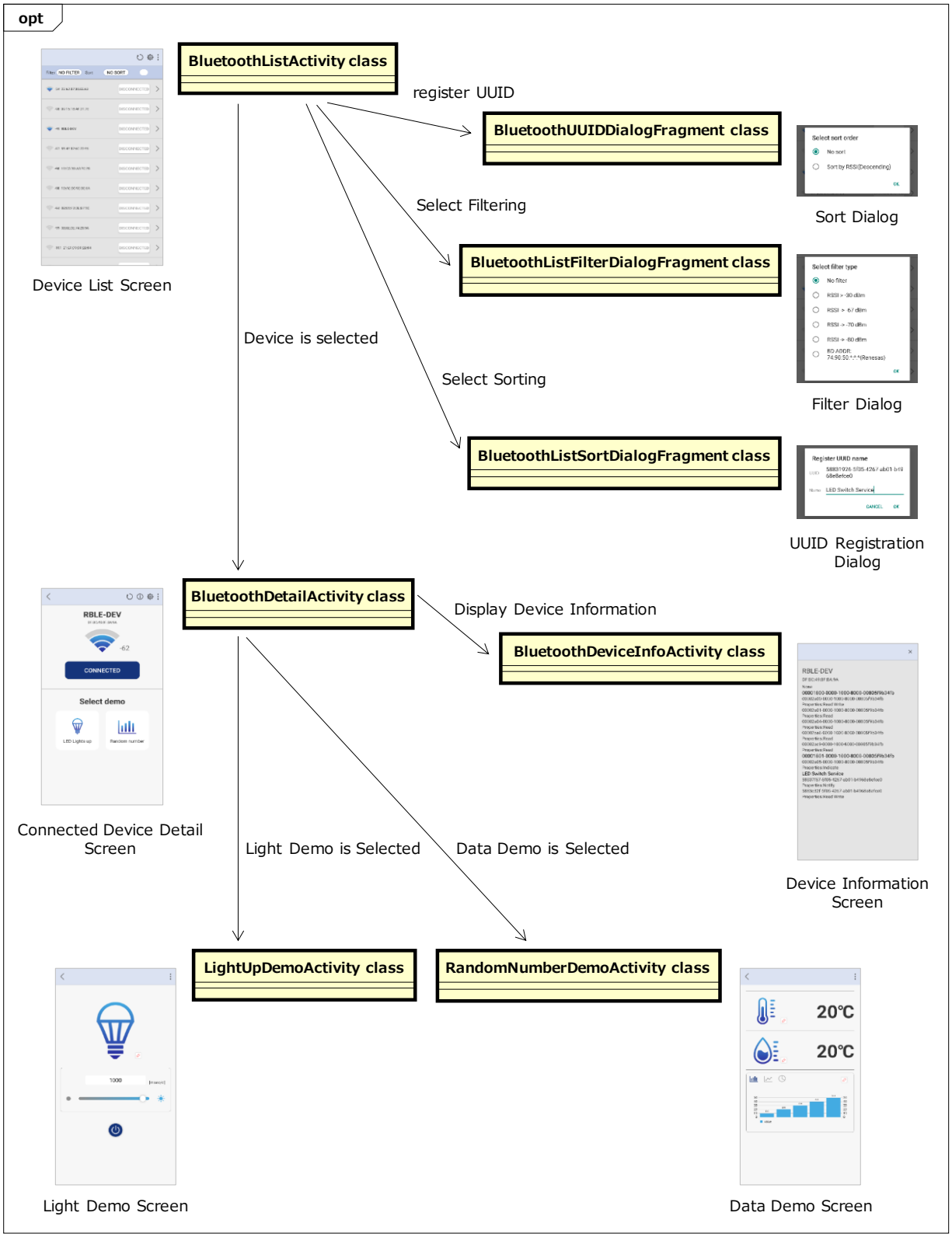

Figure 9.1 Screen and Class Transition of TryBT

10. Bluetooth communication of TryBT

This chapter describes Bluetooth usage of TryBT.

10.1 Enabling Bluetooth and Checking Fine Location Permission of Android device

`BluetoothListActivity` class checks if Bluetooth of Android device is enabled. When Bluetooth is disabled, this class displays a screen to enable Bluetooth.

Fine Location permission is also required to use Bluetooth. `BluetoothListActivity` class checks Fine Location permission of the app when Bluetooth is enabled.

An instance variable `mBluetoothAdapter` of `BluetoothAdapter` class to scan devices is defined.

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

Figure 10.1 BluetoothListActivity.kt (1)

The instance variable `mBluetoothAdapter` is initialized by `onCreate` method.

```
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
    }
```

Figure 10.2 BluetoothListActivity.kt (2)

Bluetooth permission and Fine Location permission are checked by `onResume` method each time Device List Screen is displayed.

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

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

Figure 10.3 BluetoothListActivity.kt (3)

If `BluetoothAdapter` class cannot be generated, system settings to enable Bluetooth is launched by `Intent`.

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

Figure 10.4 BluetoothListActivity.kt (4)

When Fine Location permission was granted, Scan starts. When the permission has not been granted, the permission is requested.

BluetoothListActivity.kt

```
// 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()
    }
}
```

Figure 10.5 BluetoothListActivity.kt (5)

10.2 Starting Device Scan

`BluetoothListActivity` class starts Scan by `startScan` method.

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

Figure 10.6 BluetoothListActivity.kt (6)

The point is that it is necessary to implement a processing to receive callback for scan result, because scanning device is performed asynchronously. TryBT implements a callback method to receive scan result in the same class.

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

Figure 10.7 BluetoothListActivity.kt (7)

The callback method is `mLeScanCallback`. Device found by scan is added to an instance variable `mDeviceListAdapter`.

```
/**
 * 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()
        }
    }
}
```

Figure 10.8 BluetoothListActivity.kt (8)

10.3 Stopping Device Scan

`BluetoothListActivity` class stops Scan by `stopScan` method.

```
/**
 * 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()
}
```

Figure 10.9 BluetoothListActivity.kt (9)

10.4 Connecting to Device

`BluetoothDetailActivity` class establishes a connection to a device by `connect` method.

```
/**
 * 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
    }
}
```

Figure 10.10 BluetoothDetailActivity.kt (1)

10.5 Terminating a Connection to Device

`BluetoothDetailActivity` class disconnects a connection to a device by `disconnect` method.

```
/**
 * 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
    }
}
```

Figure 10.11 BluetoothDetailActivity.kt (2)

10.6 Changed Notification of Device Connection Status

When a connection state changes, it is notified by onConnectionStateChange method in BluetoothDetailActivity class. TryBT controls button status and internal flags in accordance with the state notified.

```
/**
  * 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
    }
}
```

Figure 10.12 BluetoothDetailActivity.kt (3)

10.7 Service Discovery of Devices and Moving to Demo Screens

`BluetoothDetailActivity` class checks if a connected device supports the GATT services that are used for Light Demo and Data Demo of TryBT. TryBT gets a GATT service each time a push event of Lights Demo button or Graph Demo button is notified. If this class cannot get a GATT service, it displays error message.

```
//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)

        }
    }
}
```

Figure 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)

        }
    }

}
```

Figure 10.14 BluetoothDetailActivity.kt (5)

10.8 Change Blink Interval of LED on Evaluation board

`LightUpDemoActivity` class sets BLE blink interval input from light demo screen to the evaluation board by `write` method.

```
/**
 * 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()
    }
}
```

Figure 10.15 LightsUpDemoActivity.kt

10.9 Notification from switch on Evaluation board

When switch on the evaluation board is pushed, it is notified by `onCharacteristicChanged` method of `MainApplication` class. When receiving a notification, Data Demo of TryBT generates randomized number and updates graph.

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

Figure 10.16 MainApplication.kt

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	31. May, 2021	-	First edition
1.01	15. Oct, 2021	P.1	Changed the document name to "Bluetooth Low Energy Smartphone Application Example TryBT for Android"
		P.1	Added EK-RA4W1 and EB-RE01B as “Related documents”
		P.4	Added EK-RA4W1 and EB-RE01B as “1.1 Operational Environment”
		Overall	Updated some section name, description

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.