Bluetooth LE microcomputer / module

Windows 10 Bluetooth LE Application

Introduction

This application note explains the operation and processing flow of the Bluetooth LE API of Windows® 10 API using the Windows 10 Bluetooth LE application that can wirelessly communicate with an evaluation board equipped with a Renesas Electronics Bluetooth® Low Energy technology compatible microcomputer or module.

Bluetooth LE applications that run on Windows 10 are created in Microsoft Visual Studio Express 2017 for Windows Desktop as Windows Presentation Foundation (WPF) applications for the .NET Framework to pair, connect, search for services and characteristics, and communicate data.

Target Device

Target Board for RX23W

Target Board for RX23W module

EK-RA4W1

EB-RE01B

RL78/G1D Evaluation Board (RTK0EN0001D01001BZ)

Related Documents

RX23W Group

- RX23W Group Target Board for RX23W Quick Start Guide (R20QS0014)
- RX23W Group Target Board for RX23W User's Manual (R20UT4634)
- RX23W Group Target Board for RX23W module Quick Start Guide (R20QS0022)
- RX23W Group Target Board for RX23W module User's Manual (R20UT4890)

RA4W1 Group

- RA4W1 Group Evaluation Kit for RA4W1 Microcontroller Group EK-RA4W1 Quick Start Guide (R20QS0015)
- RA4W1 Group EK-RA4W1 User's Manual (R20UT4683)

RE01B Group

- EB-RE01B Hardware Manual (TS-TUM09734) (TESSERA TECHNOLOGY INC.)
- RE01B Group Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)

RL78/G1D Group

- Bluetooth® Low Energy Protocol Stack Virtual UART Application (R01AN3130)
- Bluetooth® Low Energy Protocol Stack Quick Start Guide (R01AN2767)
- RL78/G1D Evaluation Board User's Manual (R30UZ0048)

Contents

1. Overview 5
1.1 LED Switch Service Client 6
1.2 Virtual UART Client 6
2. Operation Check 7
2.1 Executable File 7
2.1.1 LED Switch Service Client 7
2.1.2 Virtual UART Client 7
2.2 Windows Application Specification 8
2.2.1 LED Switch Service Client 8
2.2.2 Virtual UART Client 10
2.3 Preparation of Operation Check Environment 12
2.3.1 Windows PC 12
2.3.2 Target Board for RX23W 12
2.3.3 Target Board for RX23W module 12
2.3.4 EK-RA4W1 12
2.3.5 EB-RE01B 12
2.3.6 RL78/G1D Evaluation Board (RTK0EN0001D01001BZ) 13
2.4 Operation Check 14
2.4.1 Operation check using LED Switch Service Client 14
2.4.2 Operation check using Virtual UART Client 20
3. Build 27
3.1 Build Environment Setup 27
3.1.1 Install Visual Studio 2017 27
3.1.2 Install Windows 10 SDK 30
3.2 File Structure 33
3.2.1 LED Switch Service Client 33
3.2.2 Virtual UART Client 33
3.3 Build Windows Application 34
4. Implementation Details 35
4.1 Program file of Windows application 35
4.2 Use Bluetooth LE API of UWP with WPF 35
4.3 LED Switch Service Client 36
4.3.1 MainWindow class 36
4.3.1.1 Field (Member variable) 36
4.3.1.2 Constructor 37
4.3.1.3 Method/Event handler 38
4.3.2 AdvertisementWatcherInformation class 43
4.3.2.1 Property 43

4.3.2.2 Method 43
4.3.3 DeviceWatcherInformation class 43
4.3.3.1 Constructor 43
4.3.3.2 Property 43
4.3.3.3 Field (Member variable) 43
4.3.3.4 Method 44
4.3.4 ViewModel class 44
4.3.4.1 Field (Member variable) 44
4.3.4.2 Constructor 44
4.3.4.3 Property 44
4.3.4.4 Method 45
4.3.5 Flowchart 46
4.3.5.1 Launch application 46
4.3.5.2 Pair/Unpair button 47
4.3.5.3 Connect/Disconnect button 49
4.3.5.4 LED Blink Rate button (Send write) 50
4.3.5.5 Receive SW state (Receive indication) 50
4.3.5.6 Disconnect from evaluation board 50
4.4 Virtual UART Client 51
4.4.1 MainWindow class 51
4.4.1.1 Field (Member variable) 51
4.4.1.2 Constructor 52
4.4.1.3 Method/Event handler 52
4.4.2 AdvertisementWatcherInformation class 55
4.4.2.1 Property 55
4.4.2.2 Method 56
4.4.3 ViewModel class 56
4.4.3.1 Field (Member variable) 56
4.4.3.2 Constructor 56
4.4.3.3 Property 56
4.4.3.4 Method 57
4.4.4 Flowchart 58
4.4.4.1 Launch application 58
4.4.4.2 Connect/Disconnect button 59
4.4.4.3 Log button 61
4.4.4.4 File button (Send write) 61
4.4.4.5 Send button (Send write) 62
4.4.4.6 Send textbox, key down (Send write) 62
4.4.4.7 Receive string data (Receive indication) 63
4.4.4.8 Disconnect from evaluation board 63
4.5 Windows 10 Bluetooth LE communication 64

4.5.1 BluetoothLEAdvertisementWatcher 64
4.5.2 DeviceWatcher 65
4.5.3 Pairing 67
4.5.4 Unpairing 68
4.5.5 Connection 68
4.5.6 Search for Service 68
4.5.7 Search for Characteristic 69
4.5.8 Client Characteristic Configuration Descriptor setting 69
4.5.9 Disconnection 70
4.5.10 Receive Notification 70
4.5.11 Send Write 70

5. Notes 71
5.1 Client Characteristic Configuration Descriptor write error 71

6. Appendix 72
6.1 Service and characteristic customization 72
6.2 Create a new project 73

Revision History 78

1. Overview

This application note explains the operation and processing flow of the Bluetooth LE API of Windows 10 API using the Windows 10 Bluetooth LE application (hereinafter called "Windows application") that can wirelessly communicate with an evaluation board equipped with a Renesas Electronics Bluetooth® Low Energy technology compatible microcomputer or module.

There are two Windows applications that communicate with evaluation boards are shown below.

- LED Switch Service Client

  Communicates with the evaluation board of RX23W Group/RA4W1 Group/RE01B Group.

- Virtual UART Client

  Communicates with the evaluation board of RL78/G1D Group.

The evaluation boards and software that communicate with Windows applications are shown below.

- RX23W Group
  - Evaluation Board
    - Target Board for RX23W
    - Target Board for RX23W module
  - Software
    - LED Switch Service (Software stored at the time of shipment of the evaluation board.)

- RA4W1 Group
  - Evaluation Board
    - EK-RA4W1
  - Software
    - LED Switch Service (Software stored at the time of shipment of the evaluation board.)

- RE01B Group
  - Evaluation Board
    - EB-RE01B
  - Software
    - GATT Server
      (Same function as the LED Switch service.)
      (This is the software included in "RE01B Group Bluetooth Low Energy Sample code (using CMSIS Driver Package)" (R01AN5606).)

- RL78/G1D Group
  - Evaluation Board
    - RL78/G1D Evaluation Board (RTK0EN0001D01001BZ)
  - Software
    - Virtual UART Application
      (This is the software included in " Bluetooth® Low Energy Protocol Stack Virtual UART Application" (R01AN3130). (Use the Character data transmission and reception.))

1.1 LED Switch Service Client

The LED Switch Service Client Windows application can communicate with the LED Switch Service running on the RX23W Group/RA4W1 Group/RE01B Group evaluation boards. By pairing, the information of the evaluation board is registered in the Windows system, the LED blinking interval on the evaluation board is controlled, and the number of times the switch is pressed is counted.

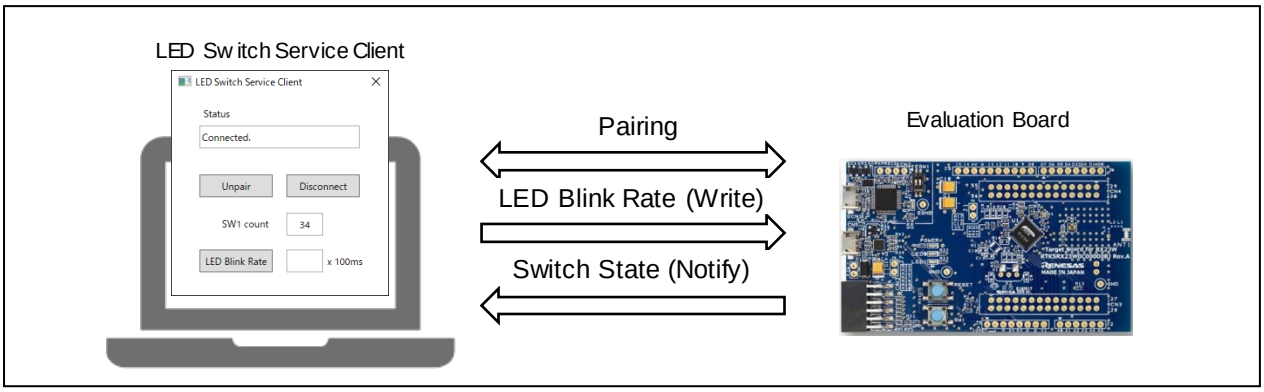

Figure 1-1 LED Switch Service Client

1.2 Virtual UART Client

The Virtual UART Client Windows application can communicate with the virtual UART application running on the evaluation board of the RL78/G1D group. Character data is sent and received between the Windows application and the terminal software on the PC connected to the evaluation board via USB.

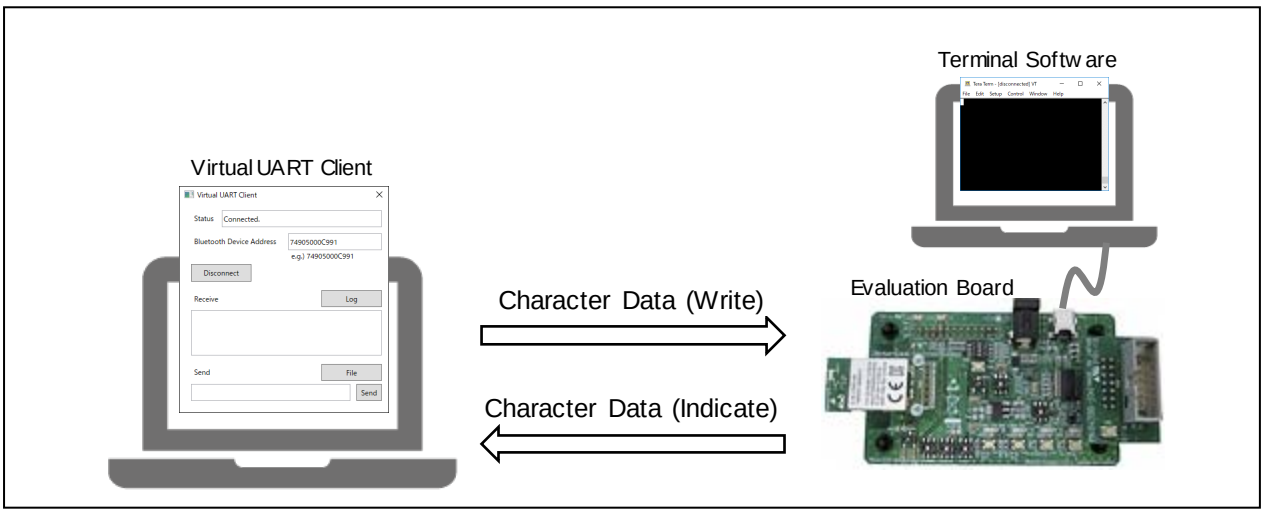

Figure 1-2 Virtual UART Client

2. Operation Check

2.1 Executable File

You can check the communication operation with the evaluation board using the Windows application executable file included in this application note.

2.1.1 LED Switch Service Client

Table 2-1 shows the LED Switch Service Client executable files, and Table 2-2 shows the evaluation boards that can be used.

Table 2-1 LED Switch Service Client executable file

Folder Name	File Name
r01an6004jj0100-ble-mcu-win\LED_Switch_Service_Client\exe	LED_Switch_Service_Client.exe

Table 2-2 Evaluation board that can be used with the LED Switch Service Client

Windows Application File Name	Evaluation Board
LED_Switch_Service_Client.exe	• Target Board for RX23W • Target Board for RX23W module • EK-RA4W1 • EB-RE01B

2.1.2 Virtual UART Client

Table 2-3 shows the Virtual UART Client executable files, and Table 2-4 shows the evaluation boards that can be used.

Table 2-3 Virtual UART Client executable file

Folder Name	File Name
r01an6004jj0100-ble-mcu-win\Virtual_UART_Client\exe	Virtual_UART_Client.exe

Table 2-4 Evaluation board that can be used with the Virtual UART Client

Windows Application File Name	Evaluation Board
Virtual_UART_Client.exe	• RL78/G1D Evaluation Board (RTK0EN0001D01001BZ)

2.2 Windows Application Specification

Describes the specifications of the LED Switch Service Client and Virtual UART Client.

2.2.1 LED Switch Service Client

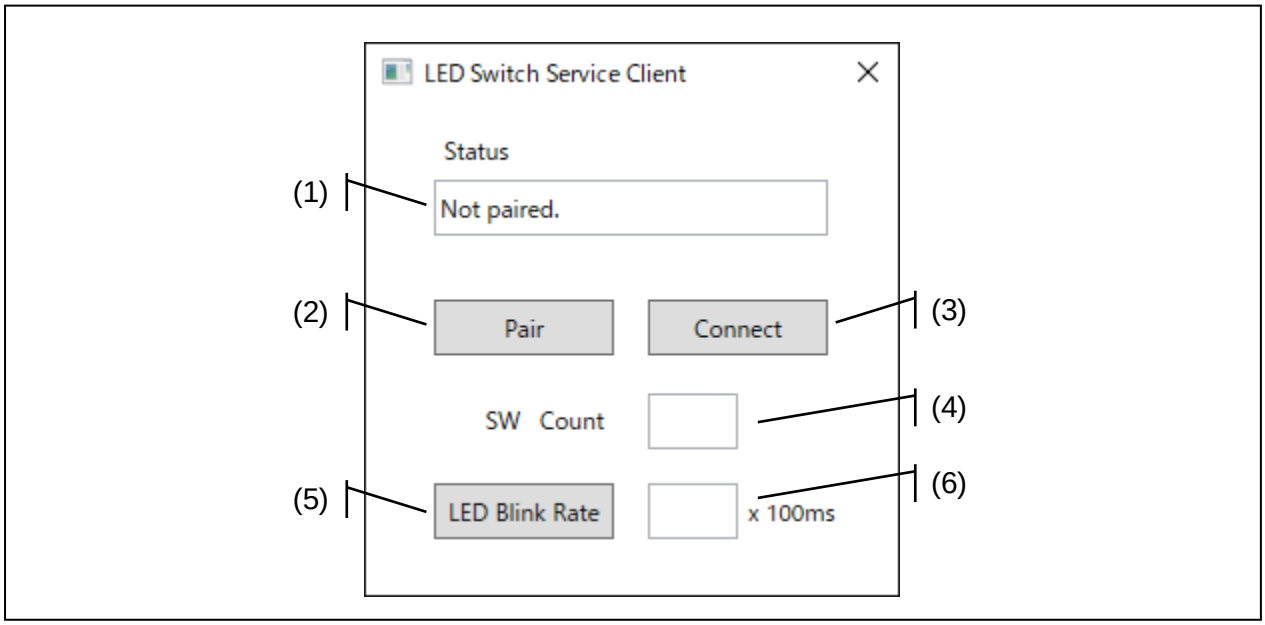

Figure 2-1 LED Switch Service Client

(1) Status display textbox

Displays the operating status of the application.

(2) Pair/Unpair button

When not paired with the evaluation board, it works as a Pair button and displays "Pair". Click the button to pair with the evaluation board.

When paired with the evaluation board, it works as an Unpair button and displays "Unpair". Click the button to cancel the pairing with the evaluation board.

(3) Connect/Disconnect button

When not connected to the evaluation board, it operates as a Connect button and "Connect" is displayed. Click the button to connect to the evaluation board.

When connected to the evaluation board, it operates as a Disconnect button and displays "Disconnect". Click the button to disconnect from the evaluation board.

(4) SW Count textbox

Displays the number of notifications sent by the connected evaluation board.

(5) LED Blink Rate send button

Sends the time inputted in the LED Blink Rate input textbox to the connected evaluation board.

(6) LED Blink Rate input textbox

Input the time for the LED Blink Rate.

2.2.2 Virtual UART Client

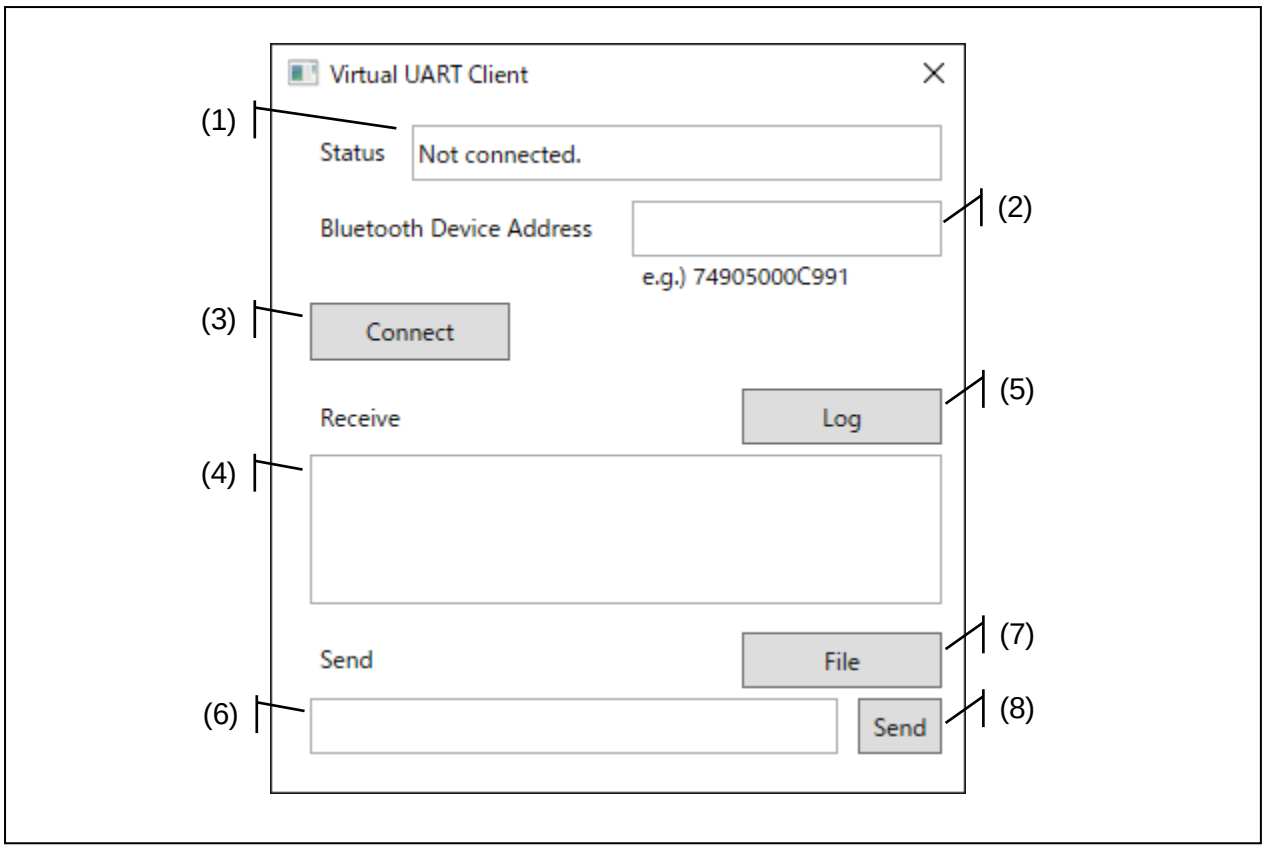

Figure 2-2 Virtual UART Client

(1) Status display textbox

Displays the operating status of the application.

(2) Bluetooth Device Address textbox

Empty: Click the Connect button to connect to the evaluation board with the UUID of the virtual UART profile, and then display the Bluetooth Device Address.

Inputted: Click the Connect button to connect to the input Bluetooth Device Address evaluation board.

(3) Connect/Disconnect button

When not connected to the evaluation board, it operates as a Connect button and "Connect" is displayed. Click the button to connect to the evaluation board.

When connected to the evaluation board, it operates as a Disconnect button and "Disconnect" is displayed. Click the button to disconnect from the evaluation board.

The Bluetooth Device Address textbox is empty:

Click the Connect button to connect to the evaluation board with the UUID of the virtual UART profile.

Bluetooth Device Address is inputted in the Bluetooth Device Address textbox:

Click the Connect button to connect to the written Bluetooth Device Address evaluation board.

(4) Receive data textbox

Receives and displays the character data sent by the connected evaluation board.

(5) Receive data save button

Receives the character data sent by the connected evaluation board and saves it in a file. "Logging" is displayed while saving to a file.

(6) Send data textbox

Input the character data to be sent to the connected evaluation board. Character data that can be inputted is 20 characters (20 bytes).

(7) Send file button

Sends the character data in the file to the connected evaluation board.

(8) Send data button

The character data inputted in the send data textbox is sent to the connected evaluation board.

2.3 Preparation of Operation Check Environment

As an environment for checking the operation, prepare a PC running Windows 10 and one of the evaluation boards shown in Table 2-2 and Table 2-4. Also, write the software to be used for each evaluation board.

2.3.1 Windows PC

Table 2-5 shows information on Windows 10 PC that have been confirmed to work with Windows applications. For Windows 10 PC, use a PC equipped with Bluetooth function of Bluetooth 4.0 or later.

Windows applications may not work depending on the state of Windows 10. In this case, refer to "5. Notes".

Table 2-5 Windows information

Edition	Windows 10 Home
Version	21H1
OS build	19043.1110

2.3.2 Target Board for RX23W

Target Board for RX23W uses "Factory Software (LED Switch Service)". If you have written other programs to the Target Board for RX23W, refer to the following documents and restore to "Factory Software".

- RX23W Group Target Board for RX23W Quick Start Guide (R20QS0014)
  - 5.1 Restoring to Factory Software

2.3.3 Target Board for RX23W module

Target Board for RX23W module uses "Factory Software (LED Switch Service)". If you have written other programs to the Target Board for RX23W module, refer to the following documents and restore to "Factory Software".

- RX23W Group Target Board for RX23W module Quick Start Guide (R20QS0022)
  - 5.1 Restoring to Factory Software

2.3.4 EK-RA4W1

EK-RA4W1 uses "factory software (LED Switch Service)". If you have written other programs to the EK-RA4W1, refer to the following documents and restore to "factory software".

- RA4W1 Group Evaluation Kit for RA4W1 Microcontroller Group EK-RA4W1 – Quick Start Guide (R20QS0015)
  - 6. Restoring Factory Settings

2.3.5 EB-RE01B

EB-RE01B uses "GATT Server". Refer to the following document and write the program of "GATT Server".

- RE01B Group Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)
  - 2.4 Programming firmware

2.3.6 RL78/G1D Evaluation Board (RTK0EN0001D01001BZ)

RL78G1D Evaluation Board uses "Virtual UART Application (Character data transmission and reception)". Refer to the following document and write the program of "Virtual UART Application (Character data transmission and reception)".

- Bluetooth® Low Energy Protocol Stack Virtual UART Application (R01AN3130)
  - 7.3 Preparation for Execution Environment
- Bluetooth® Low Energy Protocol Stack Quick Start Guide (R01AN2767)
  - 5. Writing Programs

Character data transmission and reception of the Virtual UART Application is performed by "Communication with response". Set the DIP switch 6-4 on the evaluation board as shown in Table 2-6. For more information on DIP switch, refer to the following document.

- Bluetooth® Low Energy Protocol Stack Virtual UART Application (R01AN3130)
  - 3.2 Selection of data communication with response or without response

Table 2-6 Dip-SW setting of RL78/G1D Evaluation Board

Number	Switch	Description
SW6-4	OFF (left side)	With response transmission. Client->Server: Write Request Server->Client: Indication

Input the character data to be sent and display the received character data with the terminal software on the PC connected to the evaluation board via USB. Refer to the following document to connect the evaluation board to the PC and set up the terminal software.

The following document states that you should prepare two evaluation boards, but in this application note, prepare only one.

- Bluetooth® Low Energy Protocol Stack Virtual UART Application (R01AN3130)
  - 7.3 Preparation for Execution Environment

2.4 Operation Check

2.4.1 Operation check using LED Switch Service Client

(1) Power on the evaluation board. Advertising will start automatically when the power is turned on. For how to power on the evaluation board, refer to the following documents according to the evaluation board to be used.

- RX23W Group Target Board for RX23W Quick Start Guide (R20QS0014)
  - 2.1 Power on
- RX23W Group Target Board for RX23W module Quick Start Guide (R20QS0022)
  - 2.1 Power on
- RA4W1 Group Evaluation Kit for RA4W1 Microcontroller Group EK-RA4W1 Quick Start Guide (R20QS0015)
  - 3.1 Connecting and Powering Up the EK-RA4W1 Board
- RE01B Group Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)
  - 2.2 Startup procedure

(2) Launch the LED Switch Service Client in Table 22. When it starts up, it checks whether the pairing information with the evaluation board is registered in the Windows system.

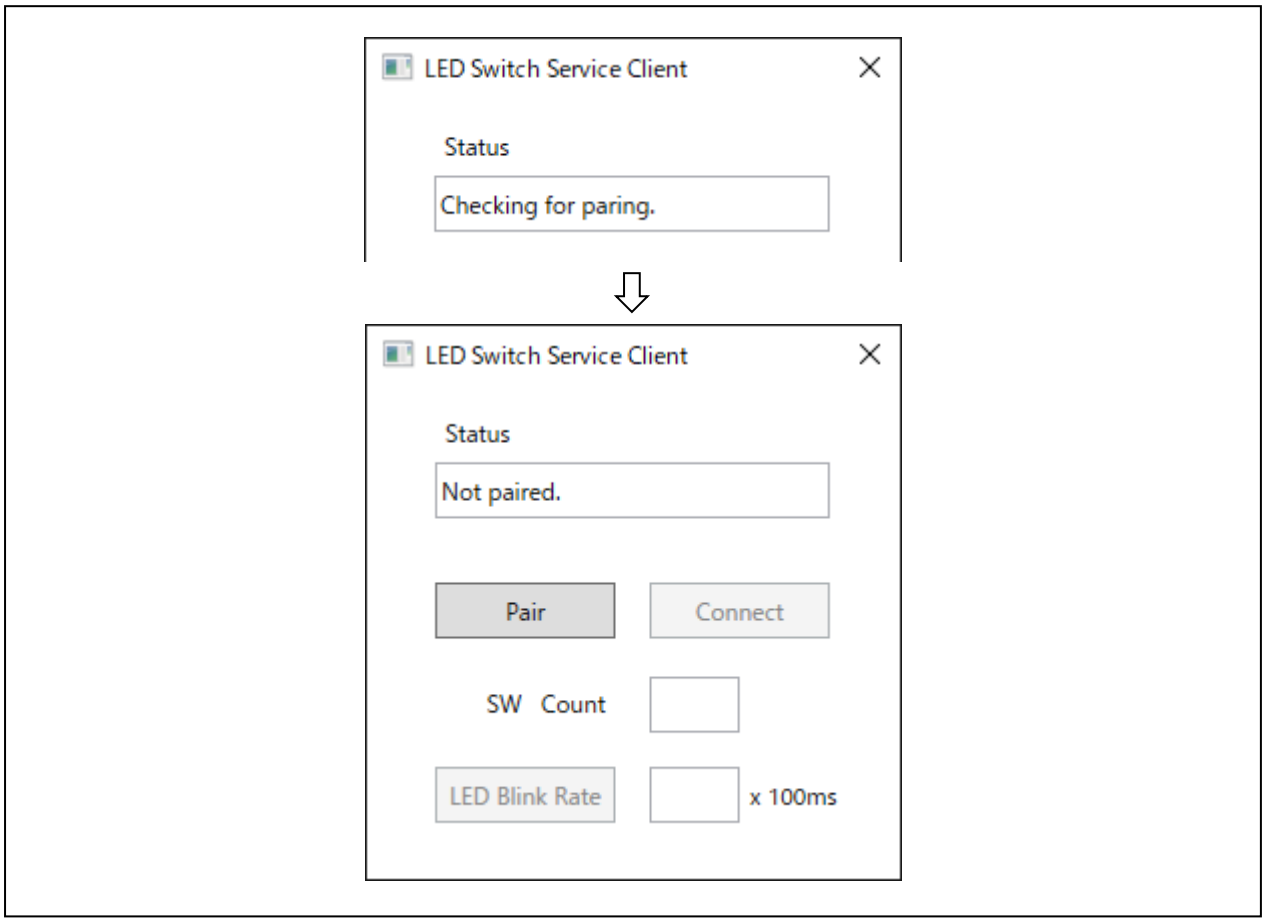

Figure 2-3 Operation check of LED Switch Service Client (1)

(3) Click the [Pair] button to pair with the evaluation board. If the pairing is successful, the [Connect] button will be enabled. In addition, the evaluation board (RBLE-DEV) is displayed as a paired device in [Settings] - [Devices] - [Bluetooth & other devices] in Windows.
If the evaluation board cannot be found or pairing fails, it may be difficult to find it because a certain amount of time has passed since the evaluation board was turned on and the advertising transmission interval becomes longer. Press Reset Switch on the evaluation board to restart to shorten the advertising transmission interval.

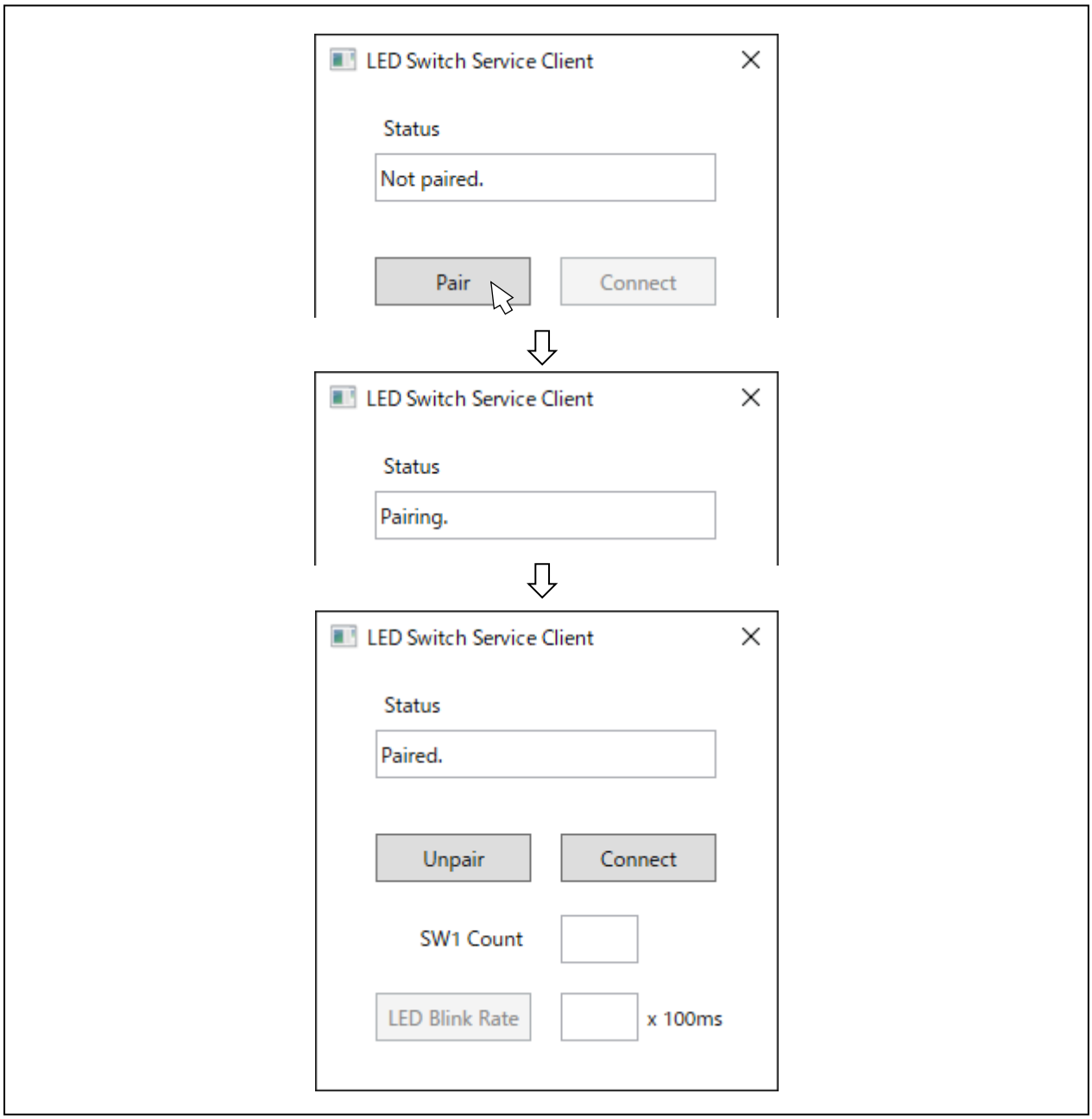

Figure 2-4 Operation check of LED Switch Service Client (2)

Figure 2-5 Operation check of LED Switch Service Client (3)

(4) Click the [Connect] button to connect to the evaluation board. Once connected, the [LED Blink Rate] button will be enabled.

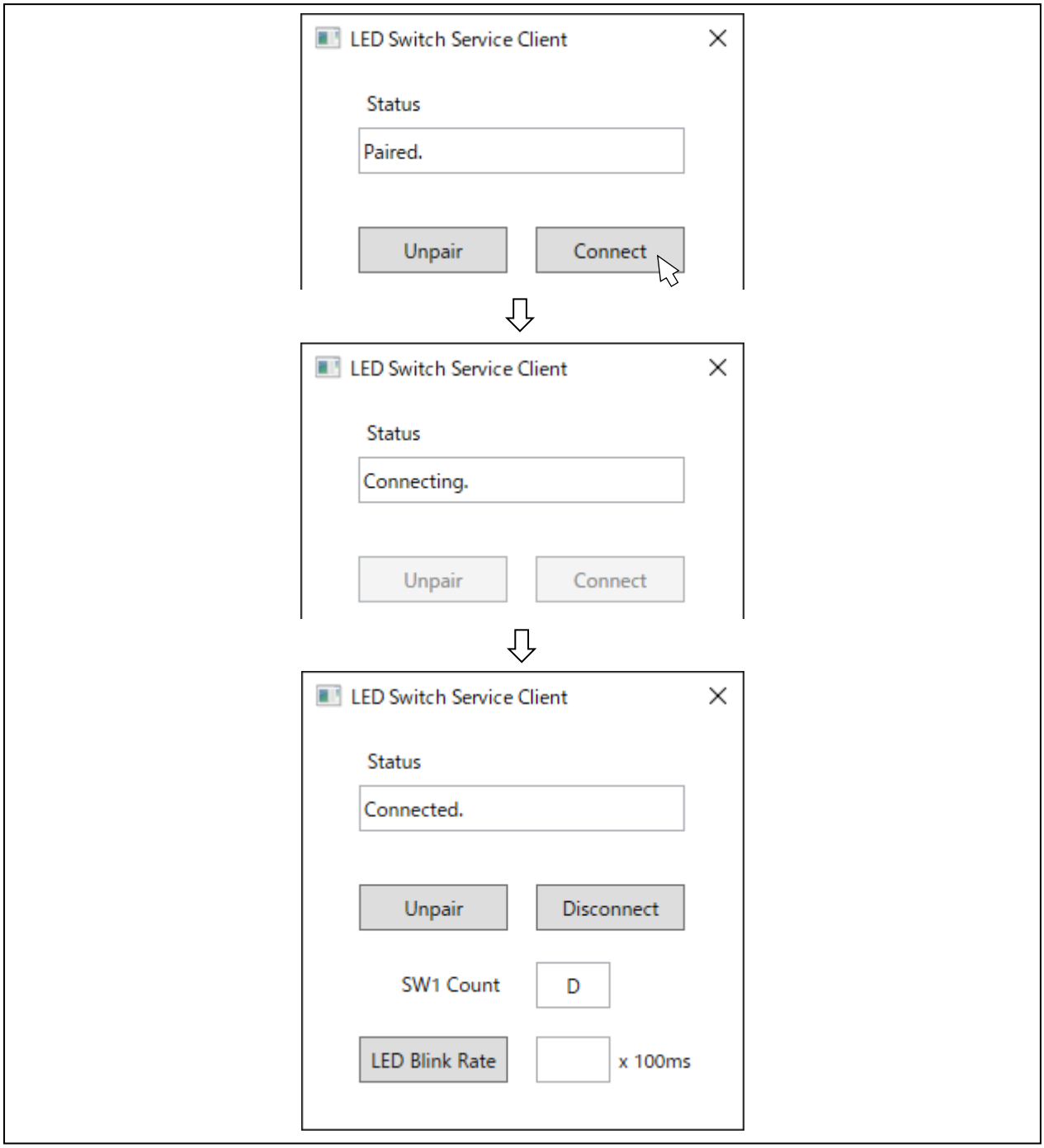

Figure 2-6 Operation check of LED Switch Service Client (4)

(5) When you press SW on the evaluation board, a Notification will be sent and counted in the SW Count textbox.
(Press SW1 for Target Board for RX23W/Target Board for RX23W module/EK-RA4W1. Press SW2 for EB-RE01B.)

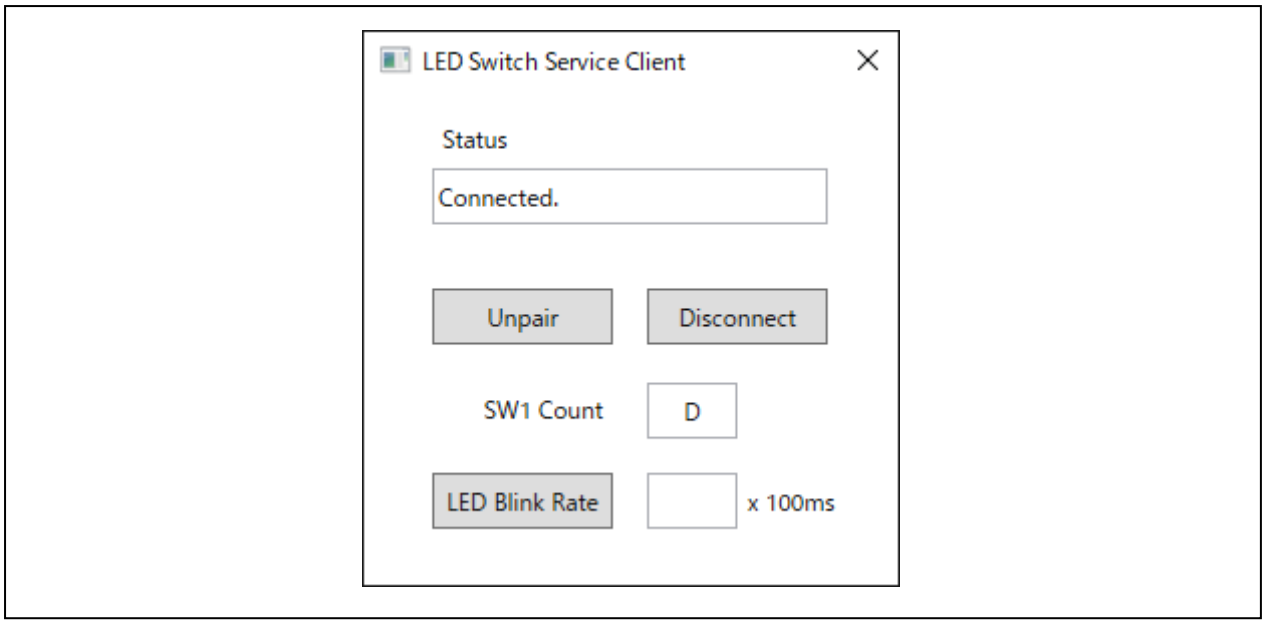

Figure 2-7 Operation check of LED Switch Service Client (5)

(6) Enter the LED blinking interval in the LED Blink Rate textbox and click the LED Blink Rate button. The blinking speed of the LED on the evaluation board changes.

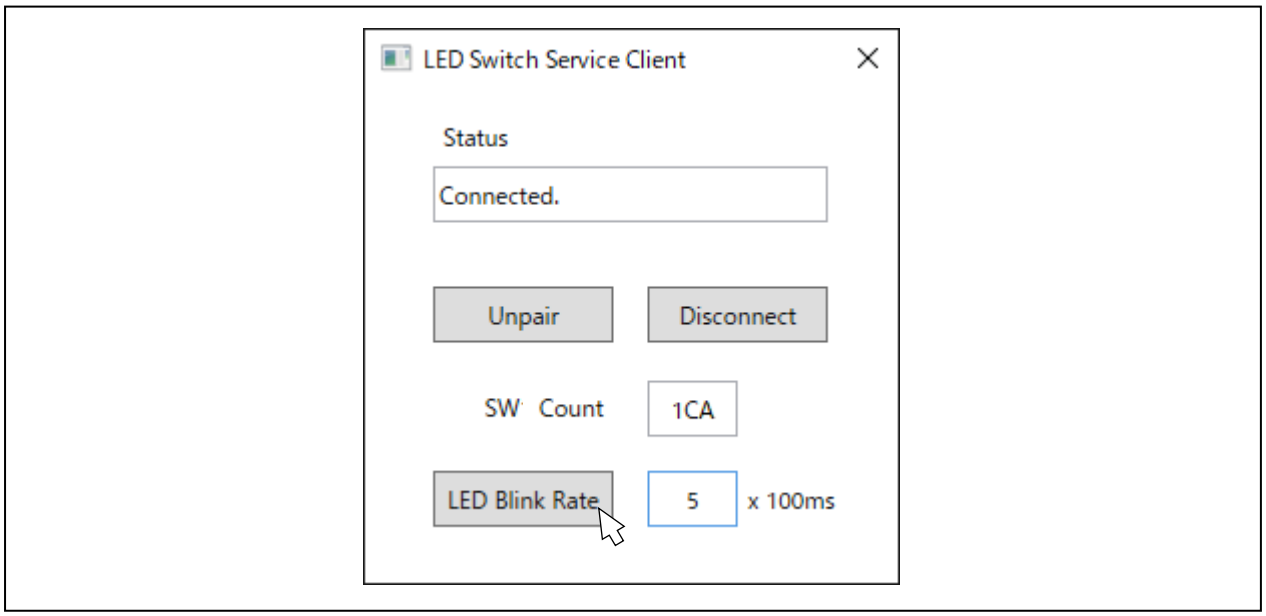

Figure 2-8 Operation check of LED Switch Service Client (6)

(7) Click the [Disconnect] button to disconnect from the evaluation board.

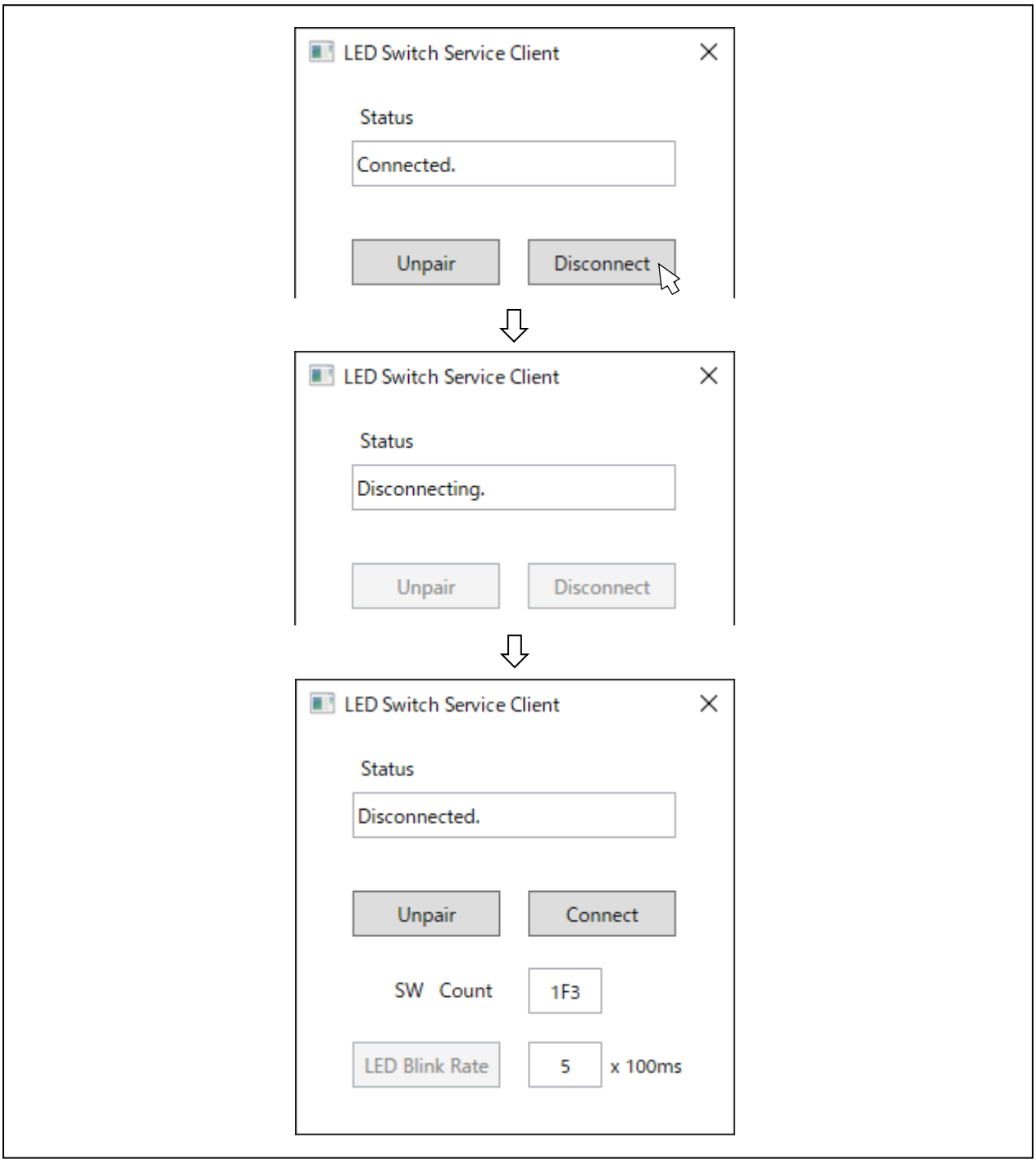

Figure 2-9 Operation check of LED Switch Service Client (7)

(8) Click the [Unpair] button to unpair. The evaluation board (RBLE-DEV) is deleted from [Settings] - [Devices] - [Bluetooth & other devices] in Windows.

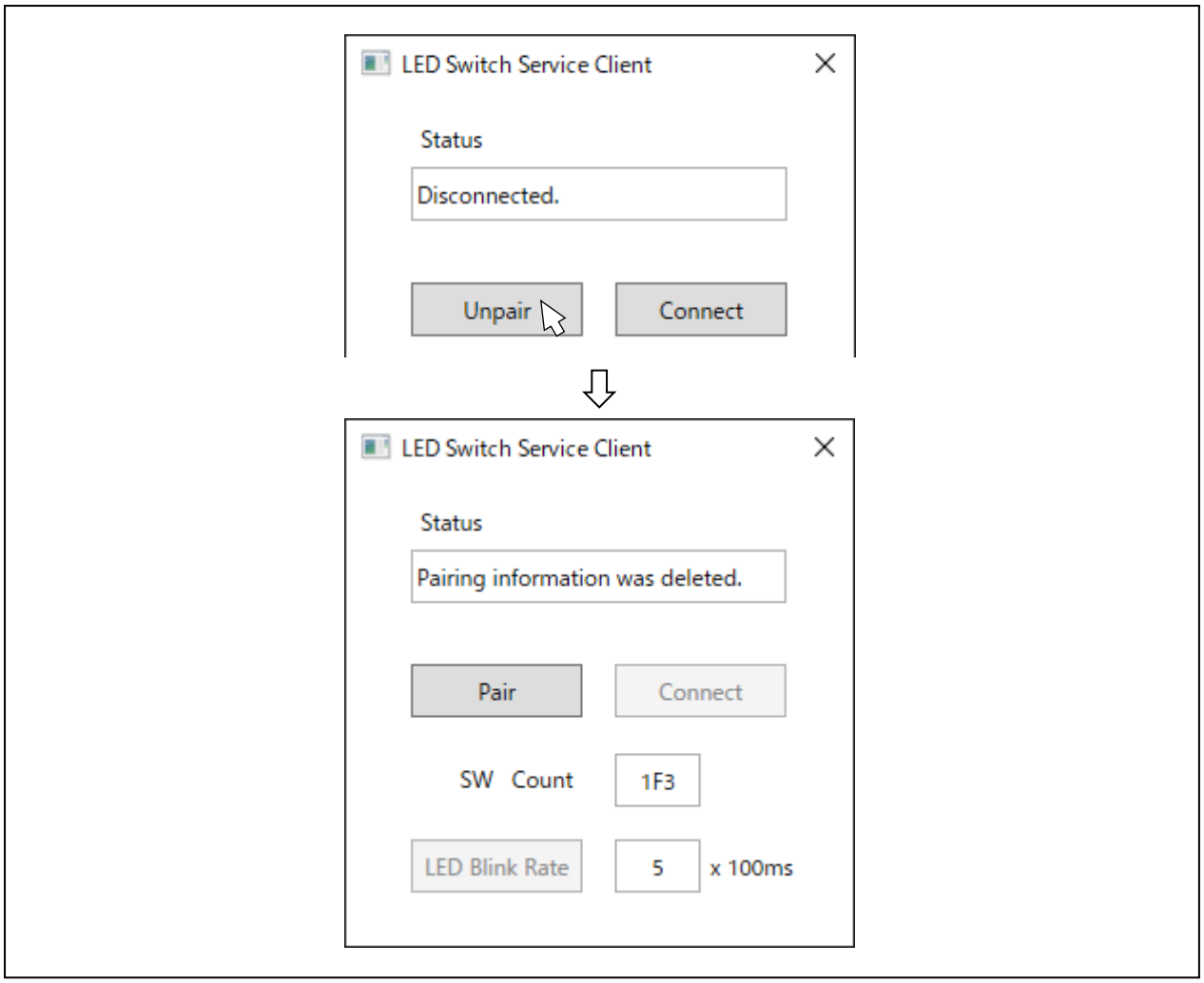

Figure 2-10 Operation check of LED Switch Service Client (8)

2.4.2 Operation check using Virtual UART Client

(1) Connect the CN3 (USB mini-B) of the RL78/G1D Evaluation board to the PC with a USB cable, and turn on the power to the RL78/G1D Evaluation board. Advertising will start automatically when the power is turned on.

(2) Start the terminal software on the PC to which the RL78/G1D Evaluation board is connected. In this operation check, Tera Term is used as the terminal software.

(3) Start the Virtual UART Client in Table 2-3.

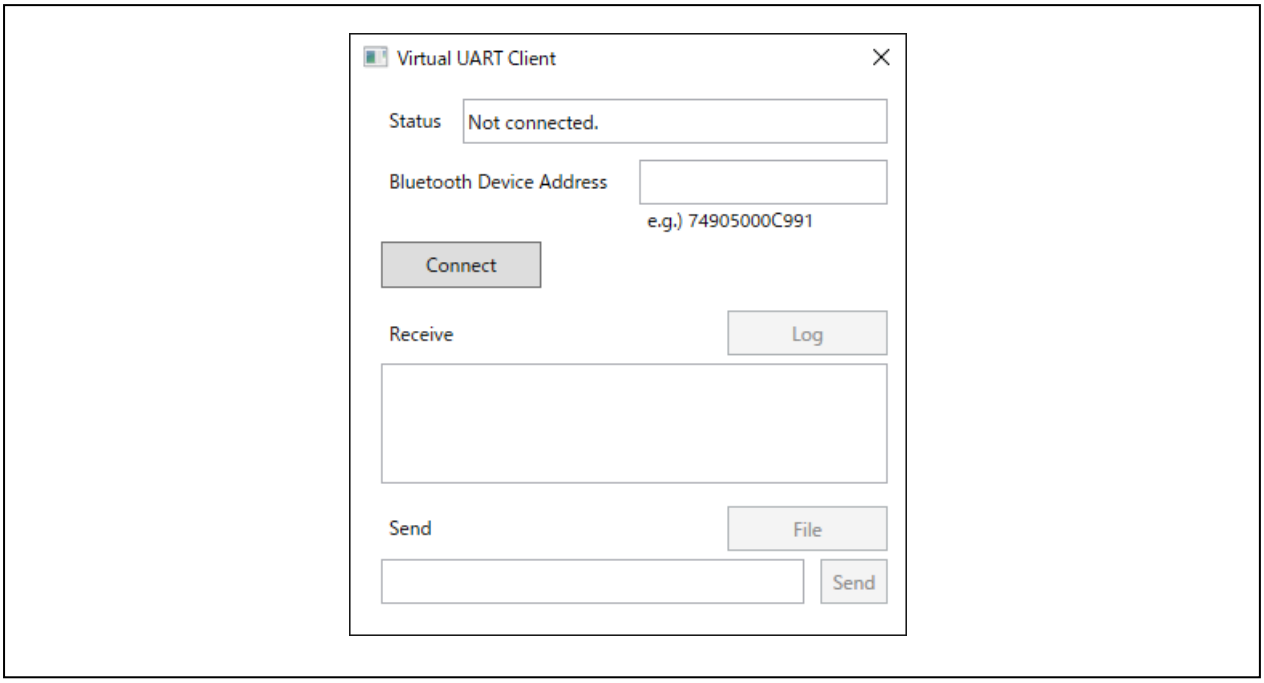

Figure 2-11 Operation check of Virtual UART Client (1)

(4) Click the [Connect] button to connect to the evaluation board. The address of the evaluation board connected to the Bluetooth Device Address textbox is displayed, and the [Log] button, [File] button, and [Send] button are enabled. "CONNECT" is displayed in Tera Term.
If the evaluation board cannot be found, it may be difficult to find it because a certain amount of time has passed since the power was turned on to the evaluation board and the advertising transmission interval becomes longer. Press Reset Switch on the evaluation board to restart to shorten the advertising transmission interval.

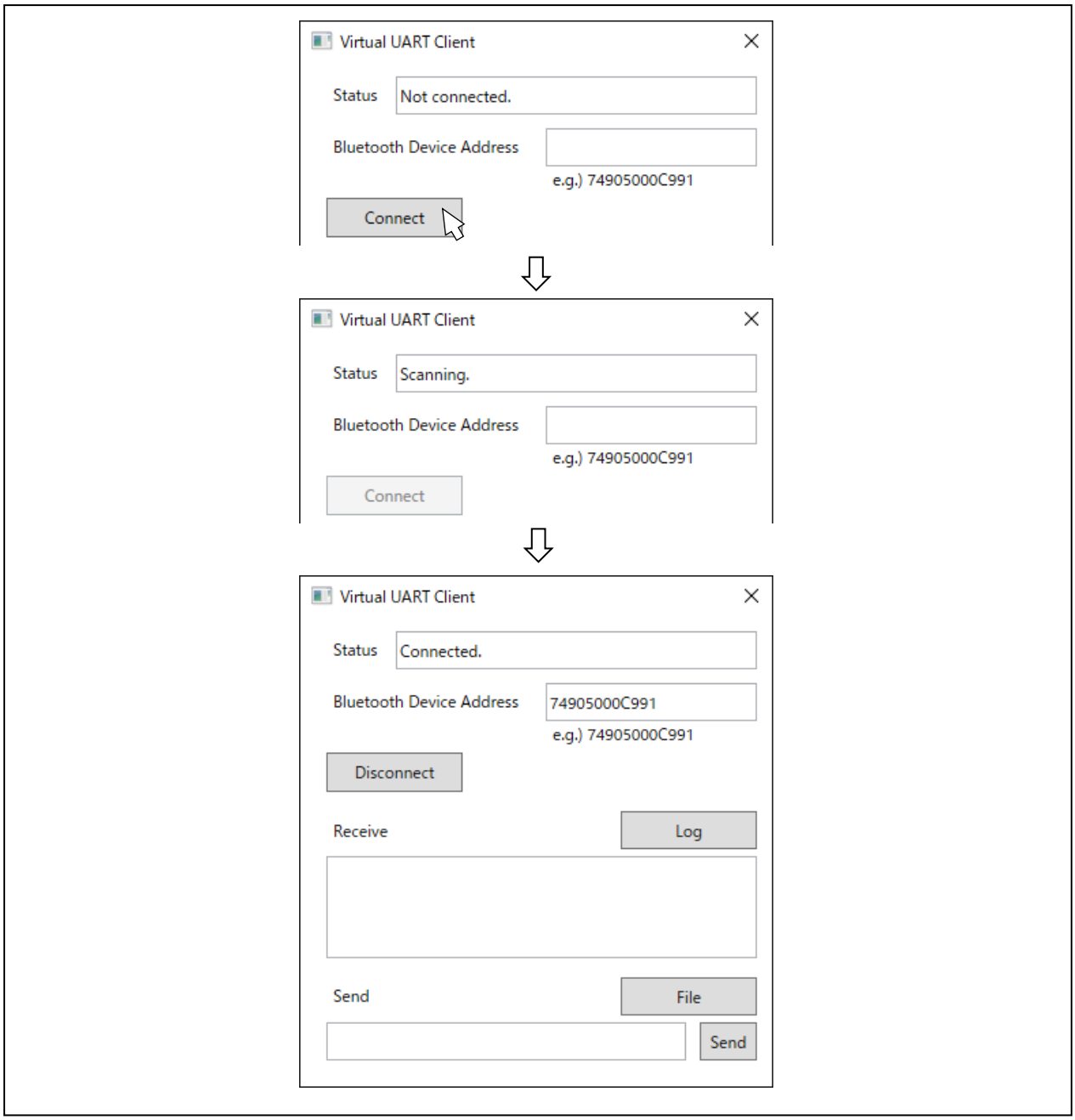

Figure 2-12 Operation check of Virtual UART Client (2)

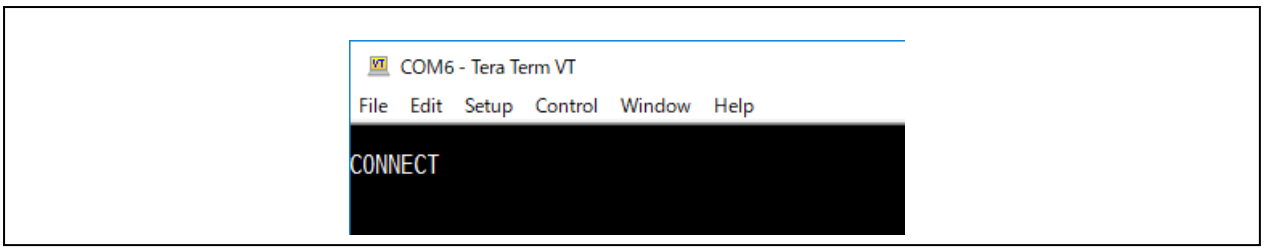

Figure 2-13 Operation check of Virtual UART Client (3)

(5) Click the [Log] button to display the "Save As" dialog. Select the folder where you want to create the file, and enter an arbitrary name in "File name". Click the [Save] button to create a log file and start logging.

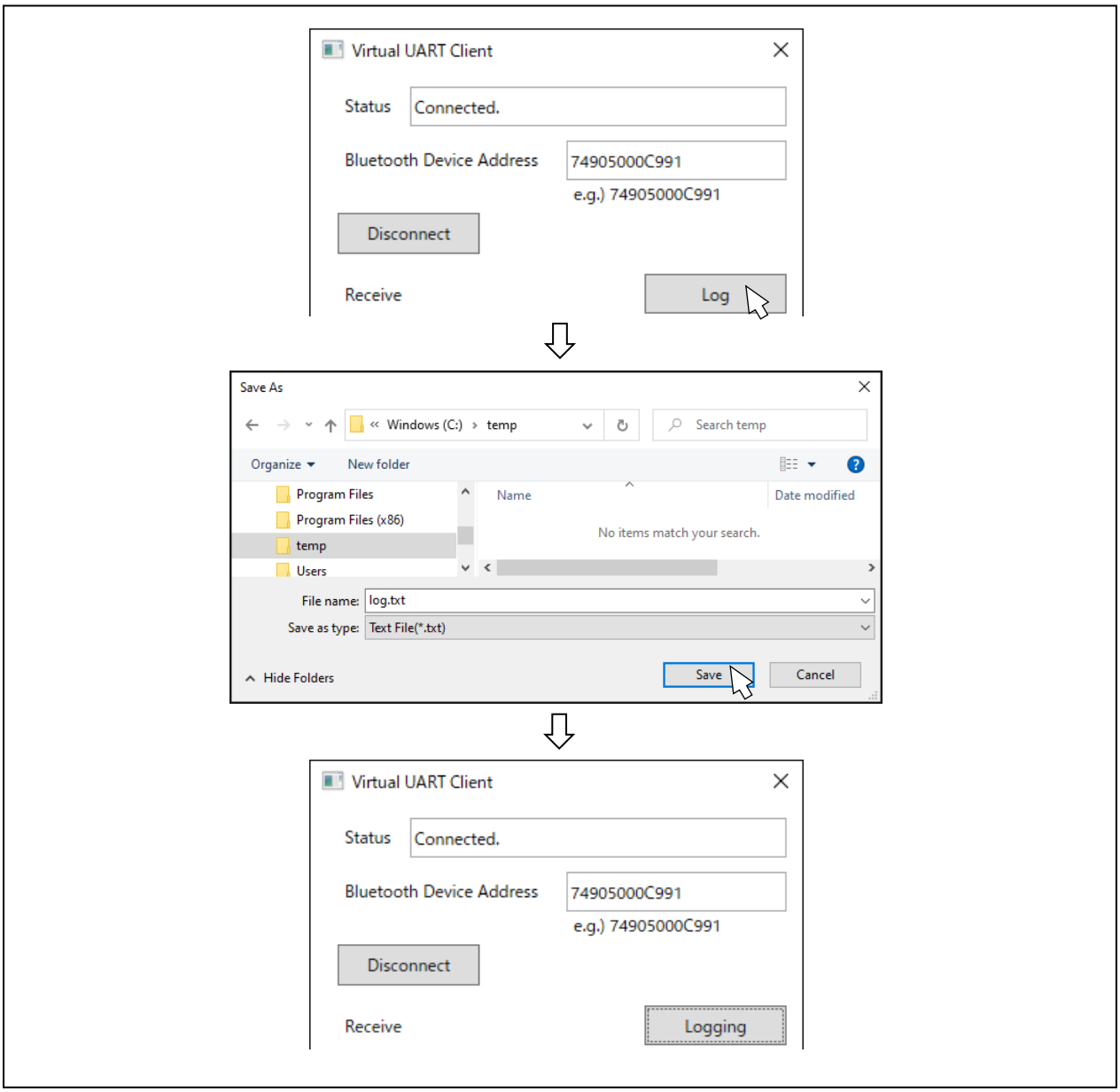

Figure 2-14 Operation check of Virtual UART Client (4)

(6) Input characters after pressing the ESC key in Tera Term to enter Virtual UART Mode.

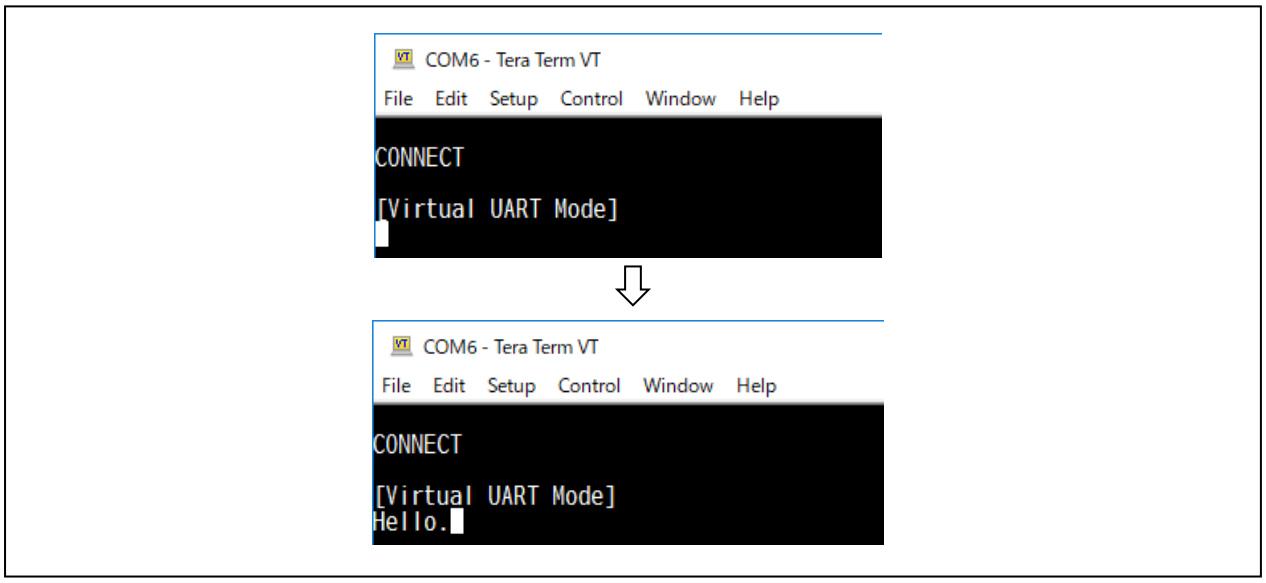

Figure 2-15 Operation check of Virtual UART Client (5)

(7) The characters inputted in Tera Term are displayed in the "Receive" textbox of the Virtual UART Client.

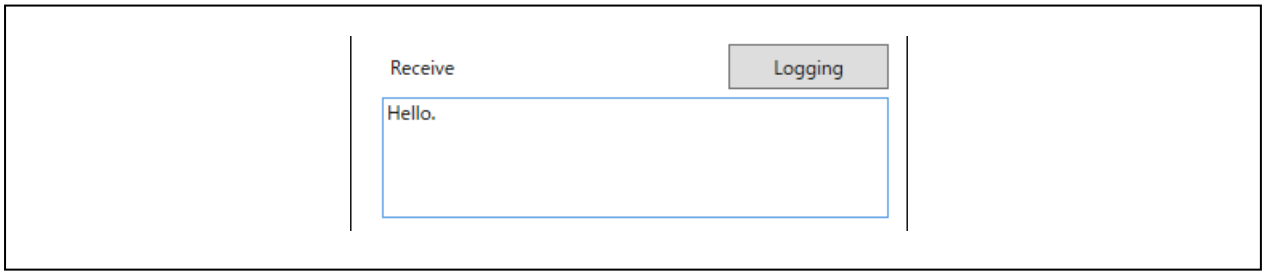

Figure 2-16 Operation check of Virtual UART Client (6)

(8) Click the [Logging] button to stop logging. If you open the log file with a text editor, you can see that the received characters are saved.

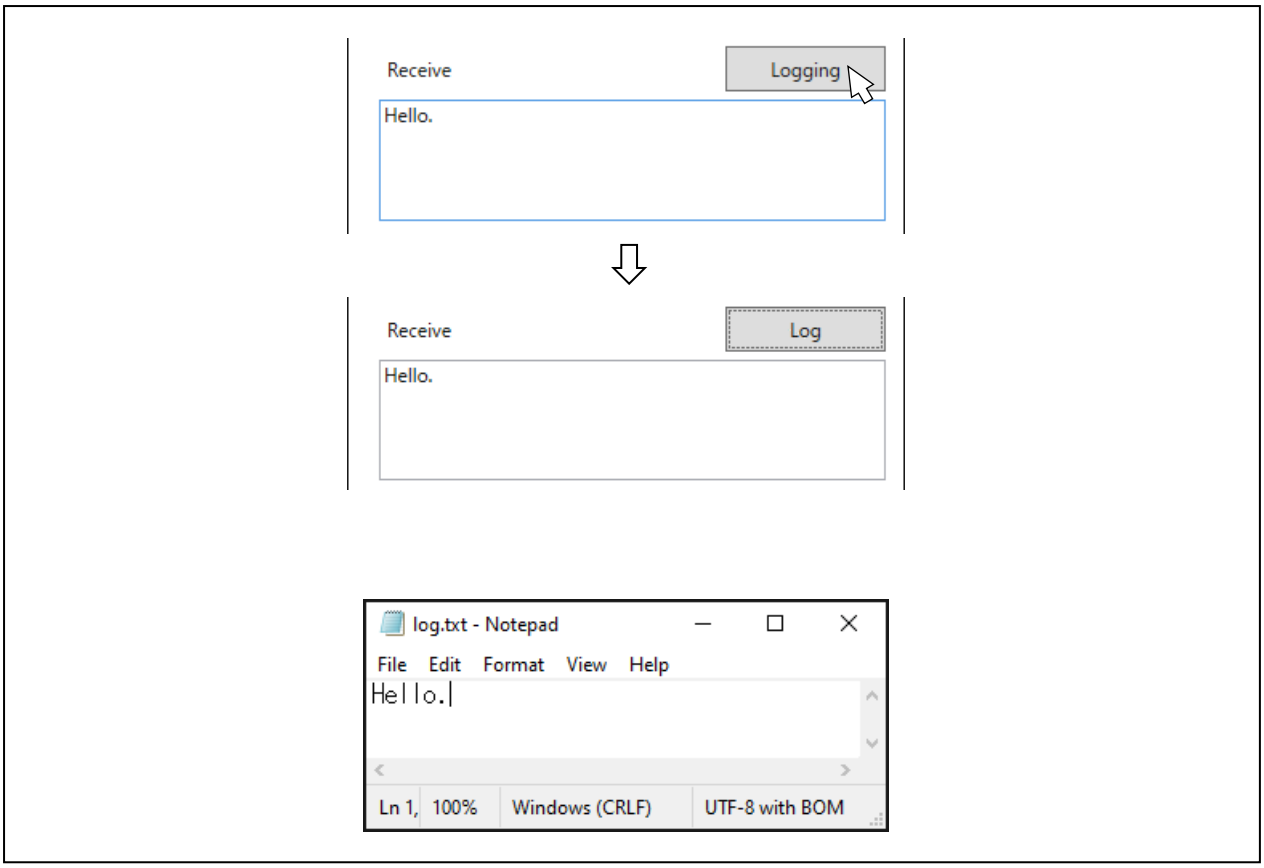

Figure 2-17 Operation check of Virtual UART Client (7)

(9) In the Virtual UART Client, input characters in the Send textbox and click the [Send] button. The characters sent to Tera Term are displayed.

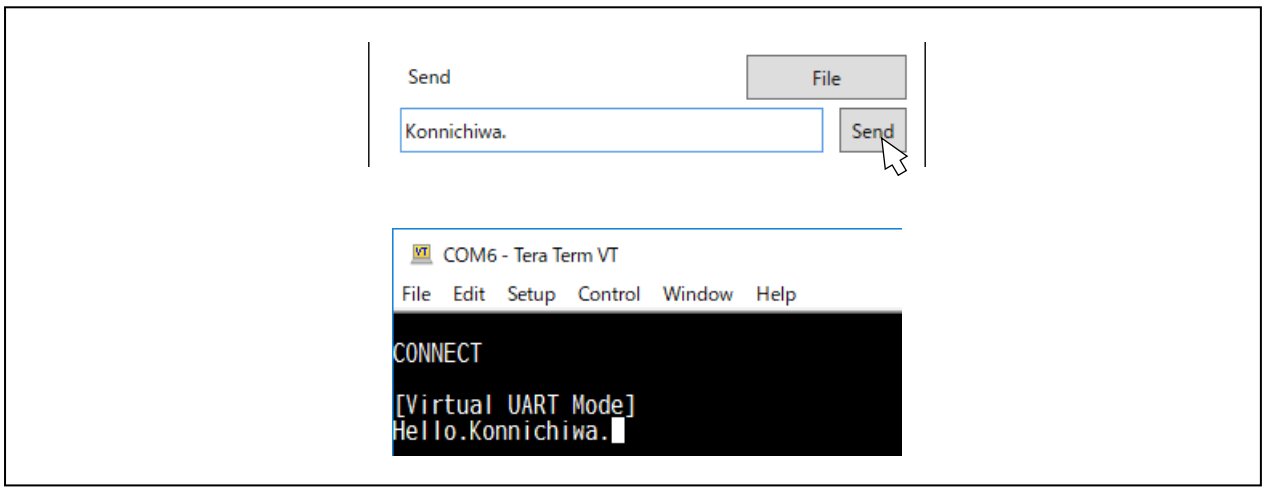

Figure 2-18 Operation check of Virtual UART Client (8)

(10) Prepare a text file in which characters are written. Click the [File] button to display the "Open" dialog. Select the text file and click the [Open] button to send the data of the text file.

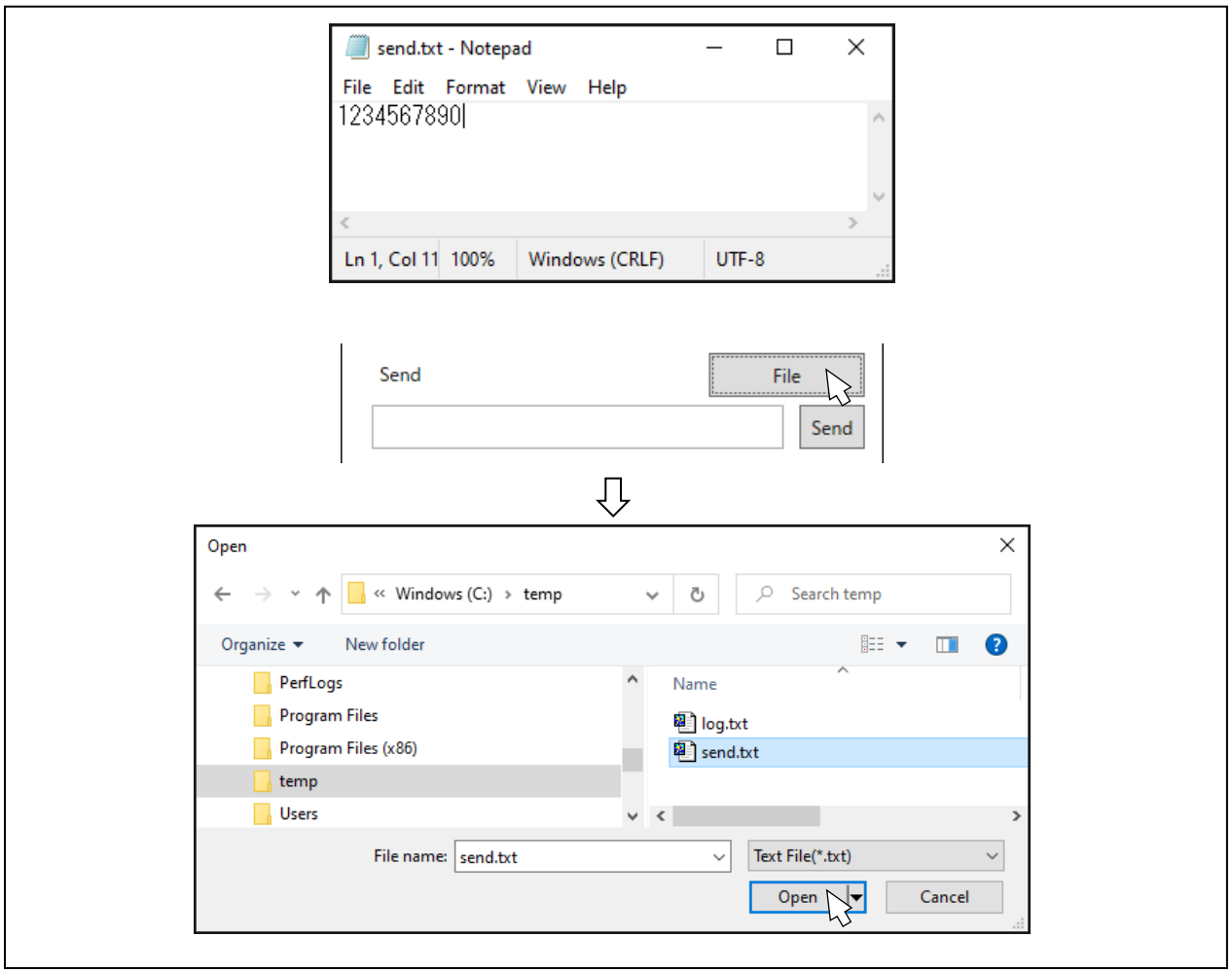

Figure 2-19 Operation check of Virtual UART Client (8)

(11) The data of the text file is displayed in Tera Term.

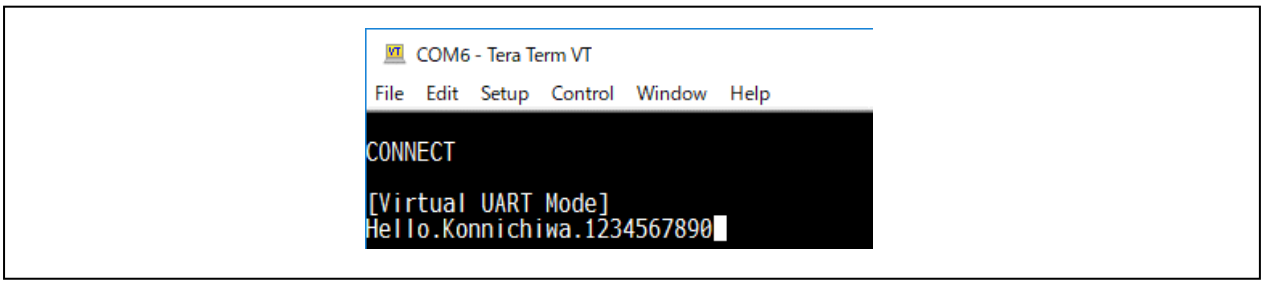

Figure 2-20 Operation check of Virtual UART Client (8)

(12) Click the [Disconnect] button of the Virtual UART Client to disconnect from the evaluation board.
"DISCONNECT" is displayed in Tera Term.

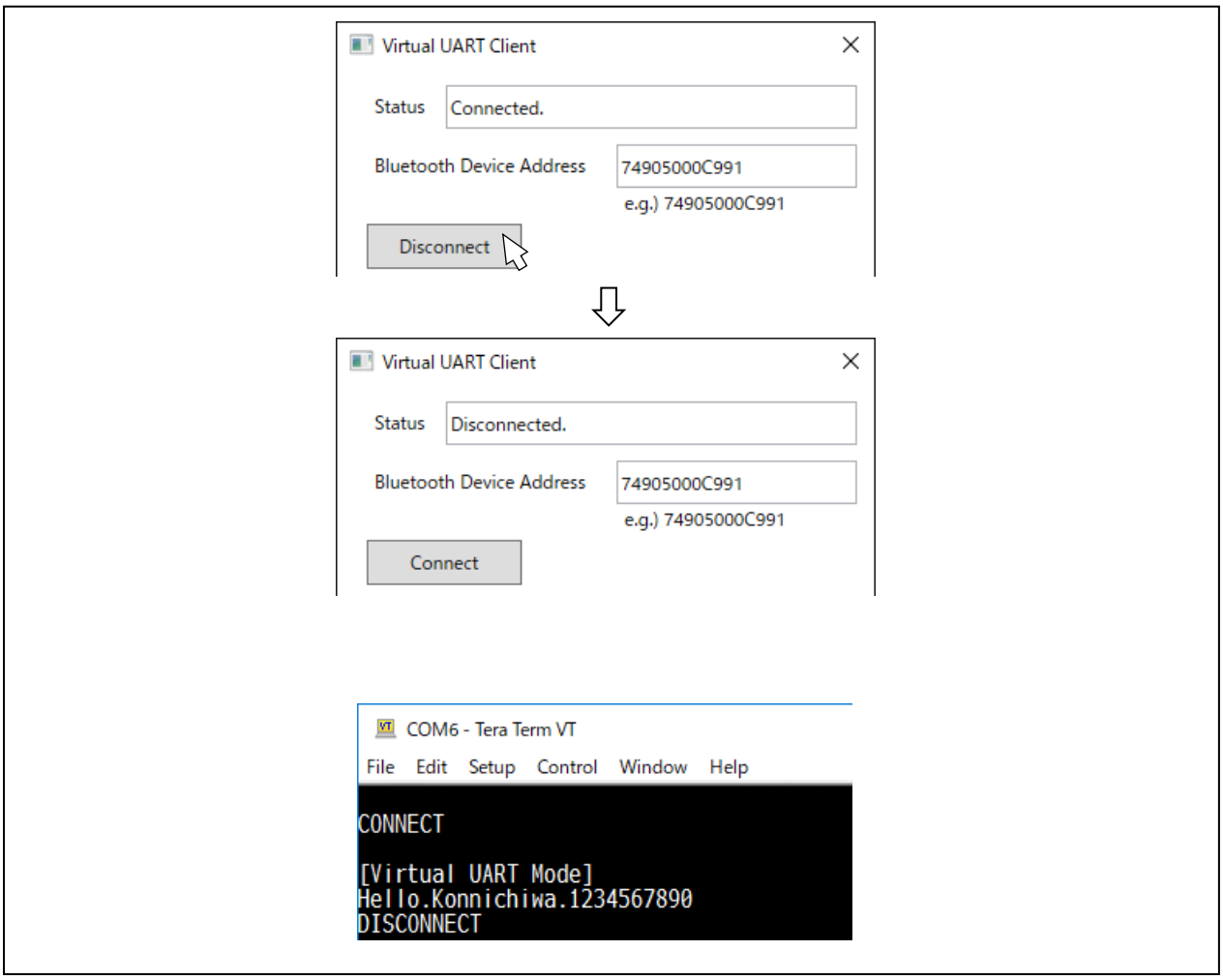

Figure 2-21 Operation check of Virtual UART Client (11)

3. Build

Describes how to build a build environment for Windows applications.

In this application note, "Microsoft Visual Studio Express 2017 for Windows Desktop" (hereinafter called "Visual Studio 2017") is used for the purpose of checking the operation of Windows applications. (For product development, please install the edition of Visual Studio that meets your requirements.)

Table 3-1 shows the build environment information for Windows applications.

Table 3-1 Build environment information

Visual Studio version	Microsoft Visual Studio Express 2017 for Windows Desktop Version 15.9.38
.NET Framework version	.NET Framework 4.6.1
Project Template	WPF App (.NET Framework)
Windows 10 SDK version	10.0.19041.0

3.1 Build Environment Setup

3.1.1 Install Visual Studio 2017

Download the installer from the following homepage and install Visual Studio 2017.

Visual Studio Express | Now Visual Studio Community

(1) Download the "Express 2017 for Windows Desktop" installer from the homepage.

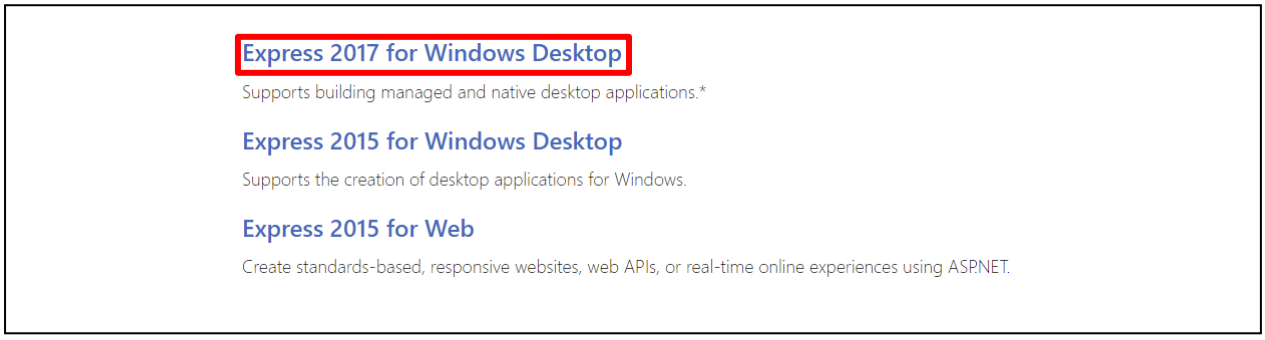

Figure 3-1 Install Visual Studio 2017 (1)

(2) Run the installer and click the [Continue] button.

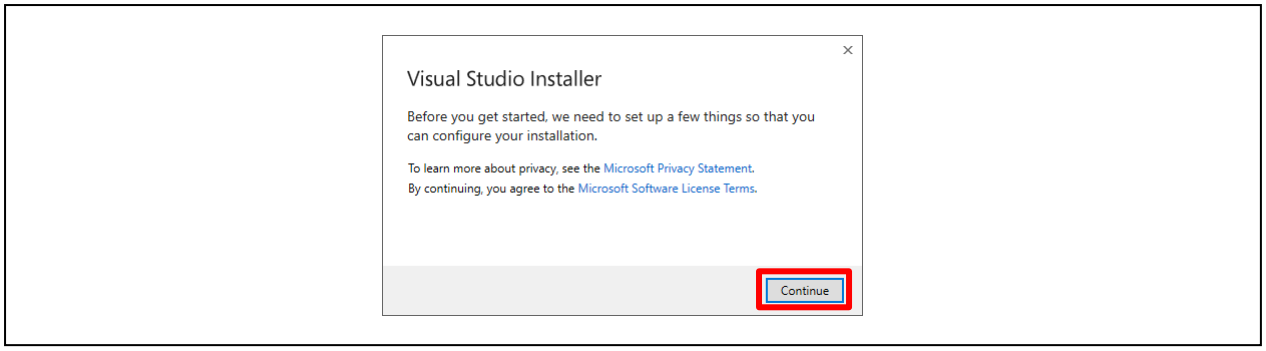

Figure 3-2 Install Visual Studio 2017 (2)

(3) When the download of the files required for Visual Studio Installer is completed, the following window will be launched. Click the "Install" button to start the installation.

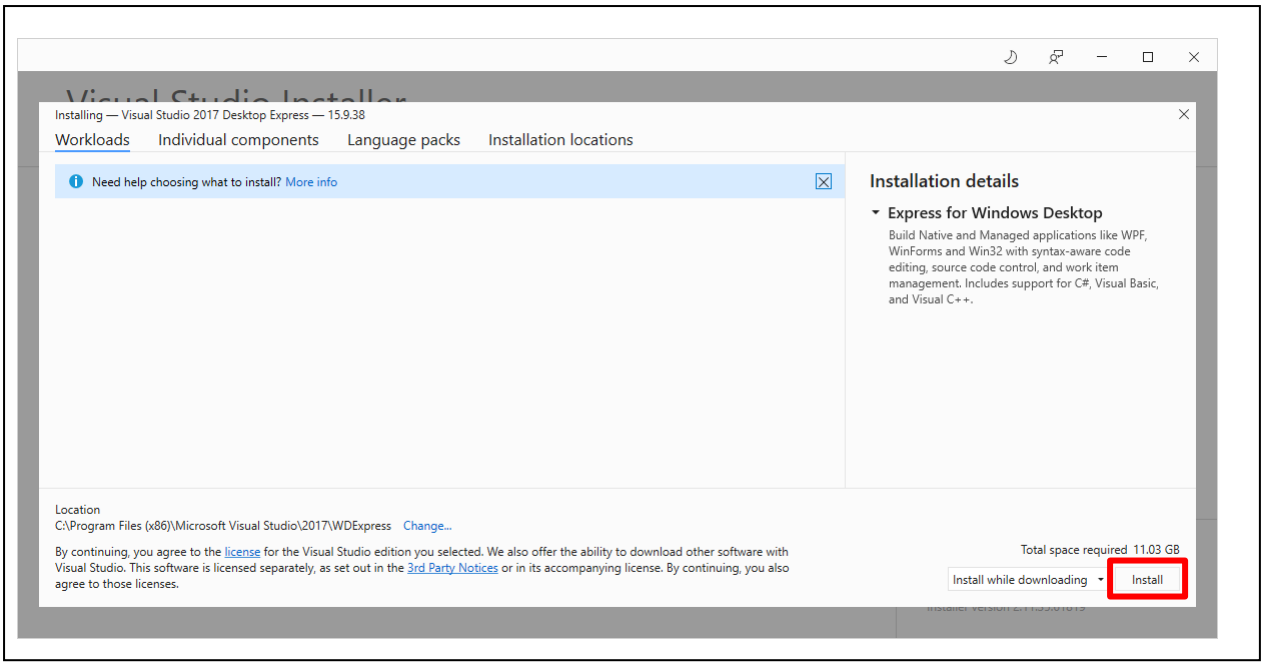

Figure 3-3 Install Visual Studio 2017 (3)

(4) When the installation is complete, a window for signing in to your account will open. Here, click "Not now, maybe later." If Visual Studio 2017 starts, click the "x" button in the upper right to quit Visual Studio 2017.

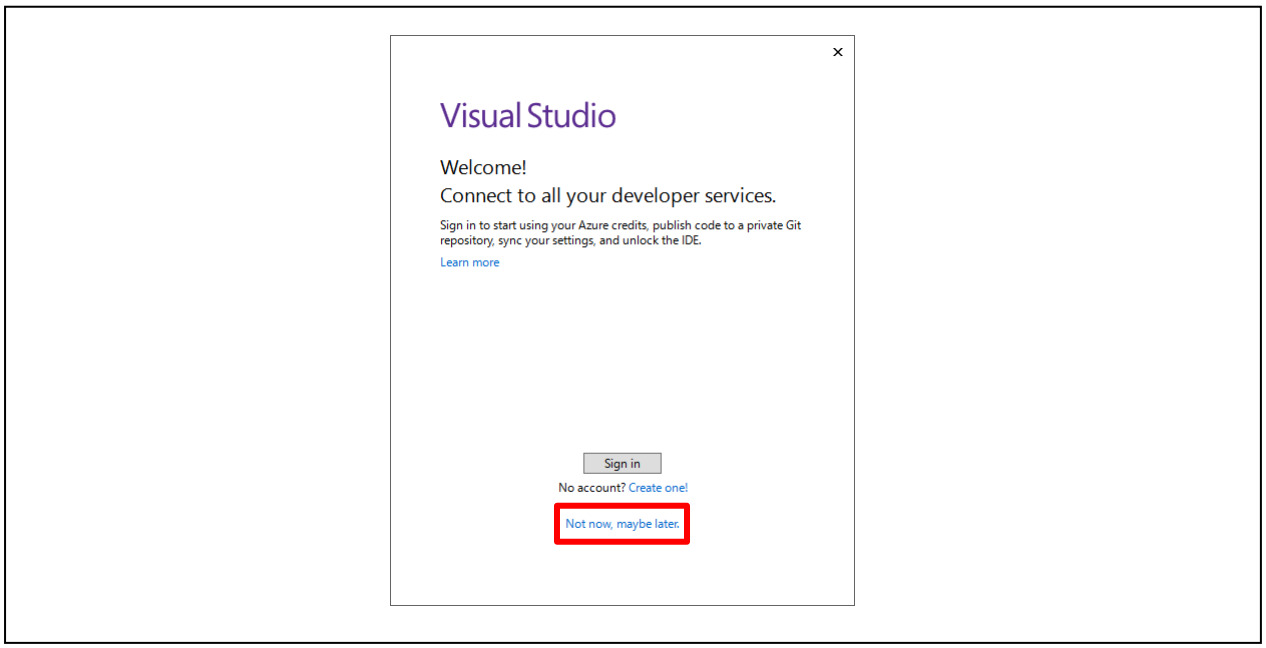

Figure 3-4 Install Visual Studio 2017 (4)

(5) Click the "x" button on the upper right of the installer to close the window.

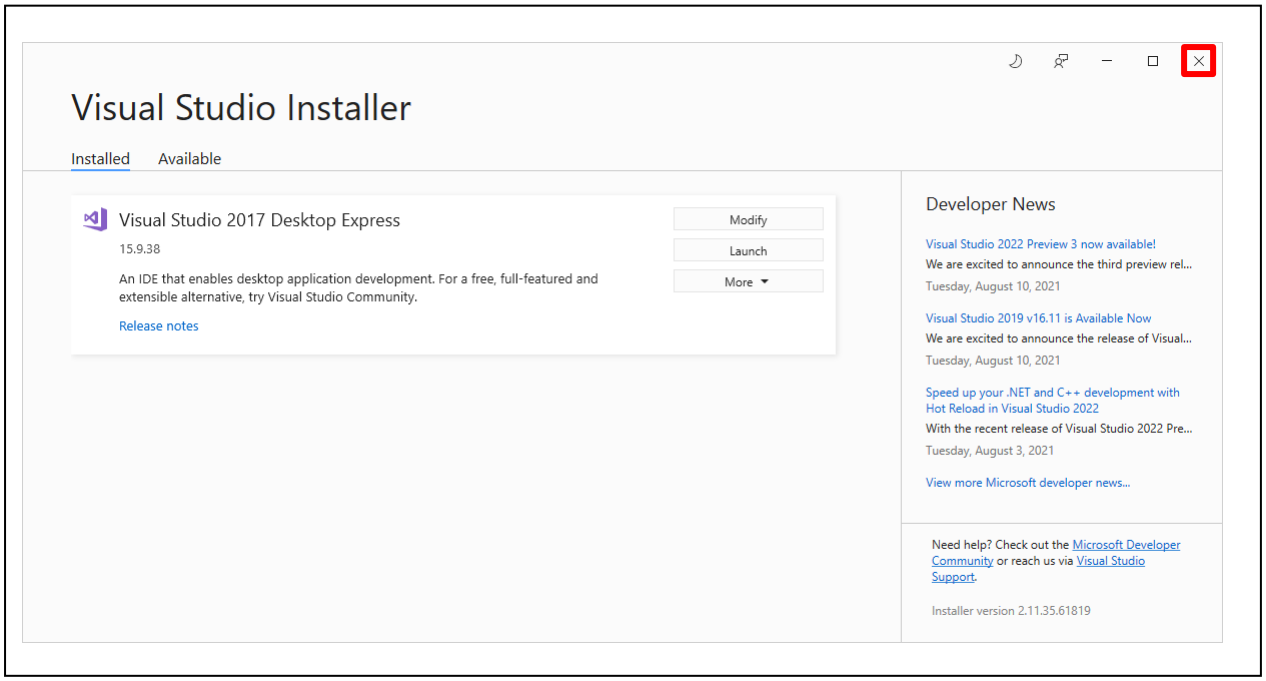

Figure 3-5 Install Visual Studio 2017 (5)

3.1.2 Install Windows 10 SDK

Download the installer from the following homepage and install the Windows 10 SDK. This Windows application uses the "Updated 12/16/20" installer.

Windows 10 SDK - Windows app development

(1) Download the installer indicated by the red line from the Windows 10 SDK home page.

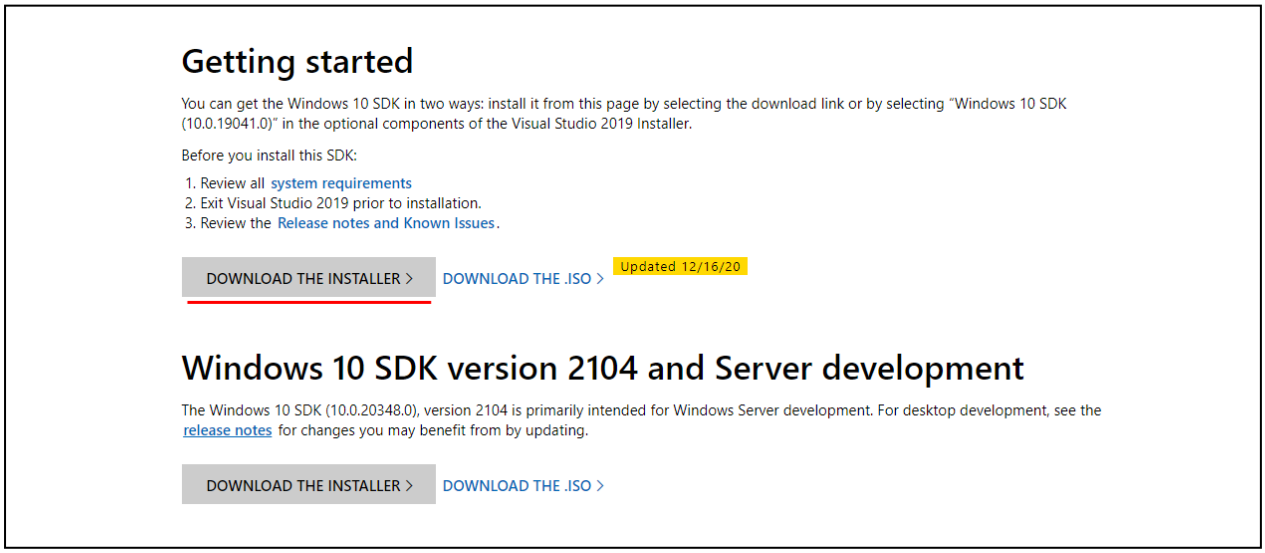

Figure 3-6 Install Windows 10 SDK (1)

(2) Run the installer, select the "Install new or update ..." radio button and click the "Next" button.

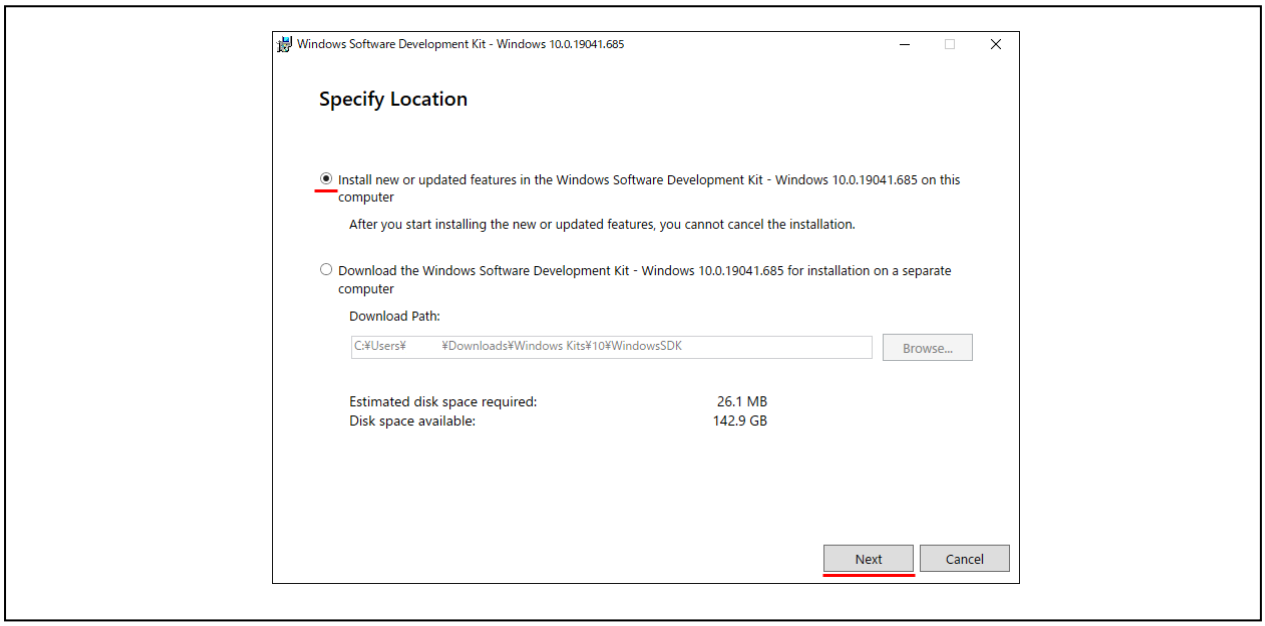

Figure 3-7 Install Windows 10 SDK (2)

(3) Accept the license agreement and click the "Accept" button.

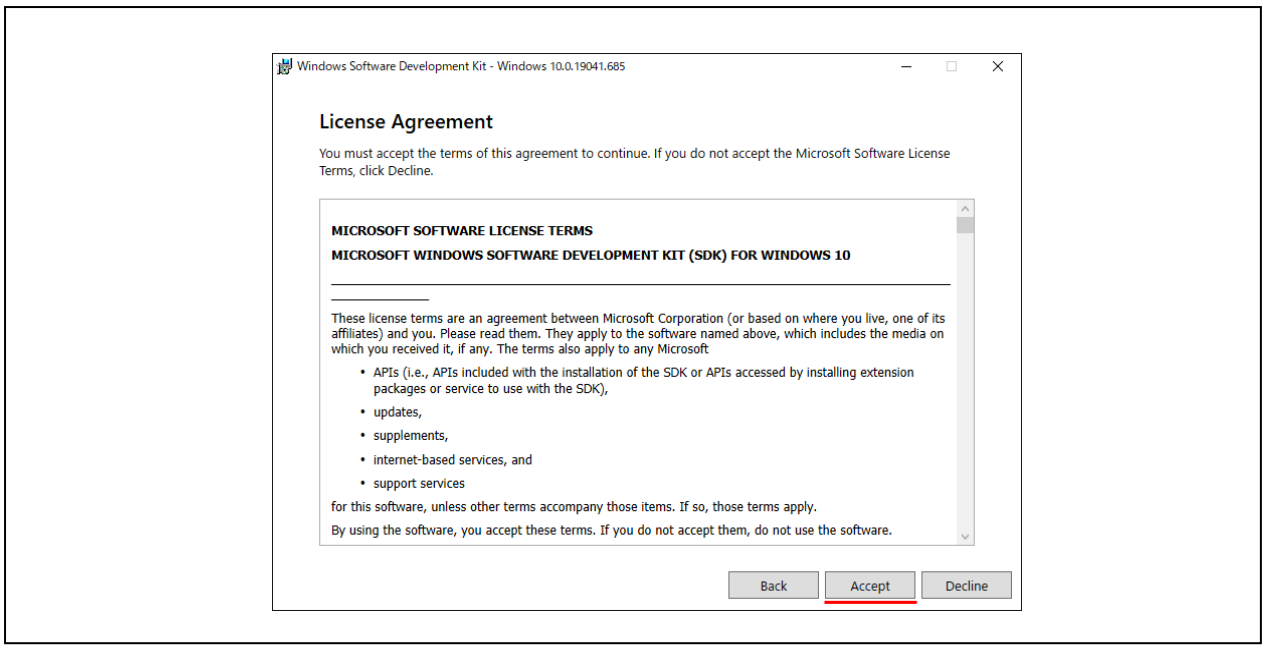

Figure 3-8 Install Windows 10 SDK (3)

(4) Select the features you want to install. All functions are selected in the default state, so click the "Install" button as it is.

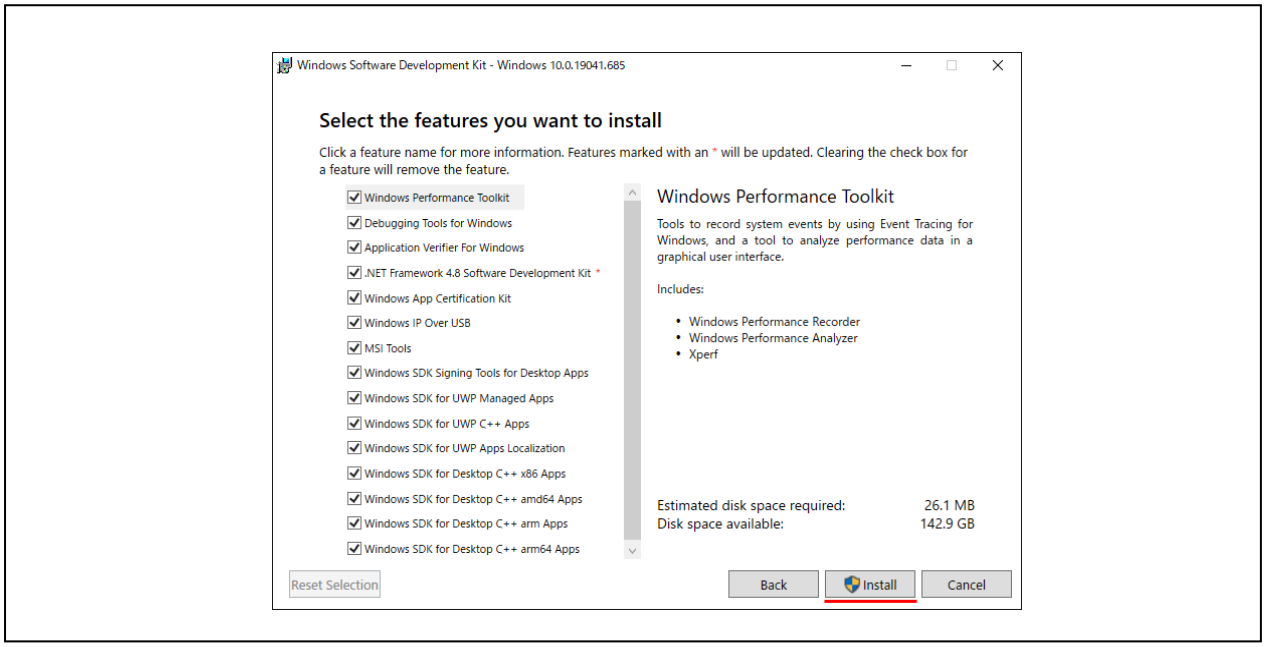

Figure 3-9 Install Windows 10 SDK (4)

(5) When the installation is complete, click the "Close" button to close the window.

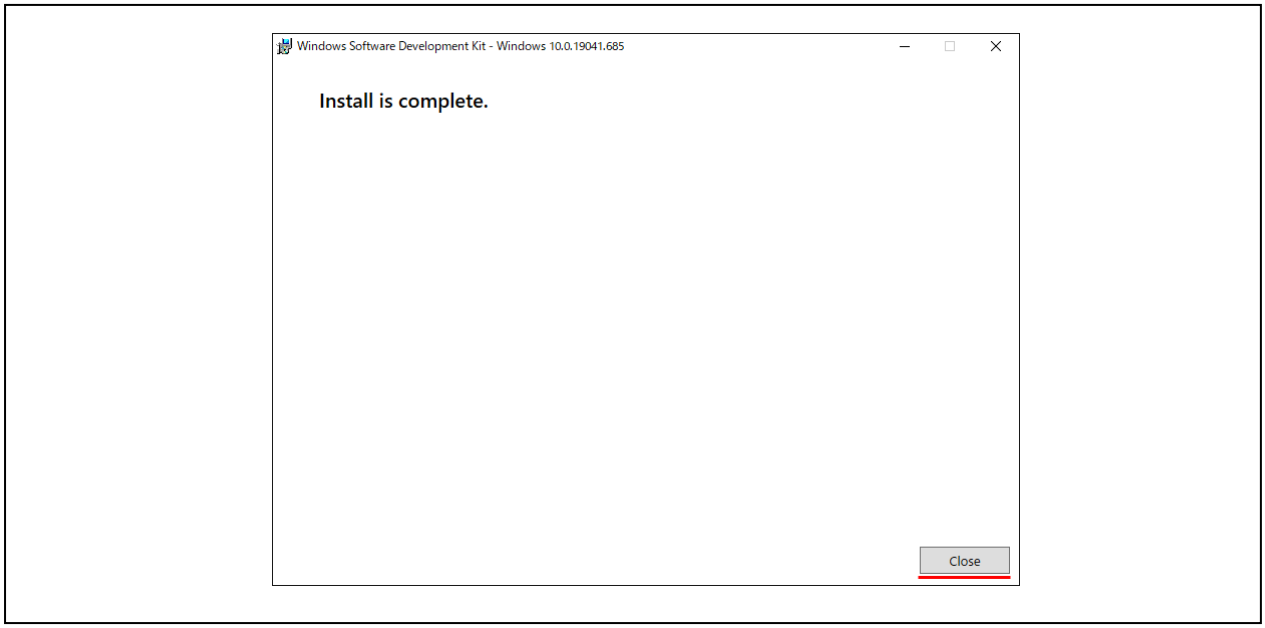

Figure 3-10 Install Windows 10 SDK (5)

3.2 File Structure

3.2.1 LED Switch Service Client

The file structure of the LED Switch Service Client is shown below.

```
r01an6004jj0100-ble-mcu-win
 │
 └─────LED_Switch_Service_Client
    │  App.config                          Application configuration file
    │  App.xaml                            App class file
    │  App.xaml.cs                         App class file
    │  LED_Switch_Service_Client.csproj    LED Switch Service Client program file
    │  LED_Switch_Service_Client.sln       LED Switch Service Client program file
    │  MainWindow.xaml                     Project file
    │  MainWindow.xaml.cs                  Solution file
    │
    ├─────exe
    │      LED_Switch_Service_Client.exe   Executable file
    │
    └─────Properties
         AssemblyInfo.cs                   Assembly information file
         Resources.Designer.cs             Resources file
         Resources.resx                    Resources file
         Settings.Designer.cs              Application settings file
         Settings.settings                 Application settings file
```

3.2.2 Virtual UART Client

The file structure of the Virtual UART Client is shown below.

```
r01an6004jj0100-ble-mcu-win
 │
 └─────Virtual_UART_Client
    │  App.config                          Application configuration file
    │  App.xaml                            App class file
    │  App.xaml.cs                         App class file
    │  MainWindow.xaml                     Virtual UART Client program file
    │  MainWindow.xaml.cs                  Virtual UART Client program file
    │  Virtual_UART_Client.csproj          Project file
    │  Virtual_UART_Client.sln             Solution file
    │
    ├─────exe
    │      Virtual_UART_Client.exe         Executable file
    │
    └─────Properties
         AssemblyInfo.cs                   Assembly information file
         Resources.Designer.cs             Resources file
         Resources.resx                    Resources file
         Settings.Designer.cs              Application settings file
         Settings.settings                 Application settings file
```

3.3 Build Windows Application

In Visual Studio 2017, build the LED Switch Service Client or Virtual UART Client executable file.

(1) Double-click the solution file (LED_Switch_Service_Client.sln or Virtual_UART_Client.sln) shown in "3.2 File Structure" to start Visual Studio 2017.

(2) Set the solution configurations to [Release].

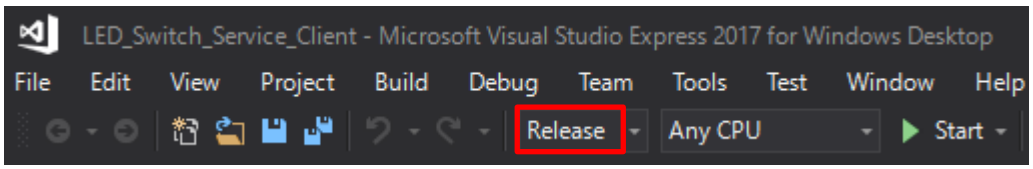

Figure 3-11 Release build setting

(3) For LED Switch Service Client, select [Build] - [Build Solution] from the menu bar to build.
For Virtual UART Client, select [Build] - [Build Solution] from the menu bar to build.

(4) The executable file is created in the following folder.

Table 3-2 Executable file

Folder Name	File Name
r01an6004jj0100-ble-mcu-win\LED_Switch_Service_Client\bin\Release	LED_Switch_Service_Client.exe
r01an6004jj0100-ble-mcu-win\Virtual_UART_Client\bin\Release	Virtual_UART_Client.exe

4. Implementation Details

4.1 Program file of Windows application

The user interface (hereinafter called "UI") and application processing of LED Switch Service Client and Virtual UART Client are described in the following two files. For the folder where the files are stored, refer to "3.2 File Structure".

- MainWindow.xaml

  An XML-formatted XAML code file that describes the Window user interface. The appearance of the window and the arrangement of controls such as buttons and textboxes are defined.

- MainWindow.xaml.cs

  A C# code file that describes the processing of the application. A program that processes pairing, connection, data communication, etc. is described.

4.2 Use Bluetooth LE API of UWP with WPF

In this application note, the LED Switch Service Client and Virtual UART Client are created as WPF applications, but the Windows Bluetooth LE API is an API that can be used with UWP. Normally, it is not available from WPF applications, but it uses the UWP API by referencing the Windows metadata files (winmd) and .NET assemblies (dll) that make up the UWP API from the project.

- Windows.winmd

  A Windows metadata (WinMD) file that enables you to use the Windows Runtime API (WinRT API).

  The files are stored in the following folders.

  C:\Program Files (x86)\Windows Kits\10\UnionMetadata\10.0.19041.0

  (10.0.19041.0 is version number of Windows 10 SDK)

- System.Runtime.WindowsRuntime.dll

  A .NET assembly file that is a component of a .NET application.

  The files are stored in the following folders.

  C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETCore\v4.5

  (For Windows 10 .NET Core V4.5)

4.3 LED Switch Service Client

This section describes the classes and processing contents defined in the LED Switch Service Client.

Table 4-1 LED Switch Service Client Class Overview

Class Name	Description
MainWindow	A class that describes the application processing of the LED Switch Service Client.
AdvertisementWatcherInformation	A class that stores information obtained from advertising sent by the evaluation board to be connected.
DeviceWatcherInformation	A class that stores the Bluetooth LE device information listed by DeviceWatcher.
ViewModel	A class used for data exchange between the program and the UI.

4.3.1 MainWindow class

4.3.1.1 Field (Member variable)

Guid UUID_LEDSW_SERVICE
Define LED Switch Service UUID. After connecting to the evaluation board, use it to search for Services.
Guid UUID_LEDSW_SW_STATE_NOTIFICATION
Define the Switch State Characteristic UUID of the LED Switch Service. After connecting to the rating board, use it for Characteristic search.
Guid UUID_LEDSW_LED_BLINK_RATE_WRITE
Define LED Blink Rate Characteristic UUID for LED Switch Service. After connecting to the rating board, use it for Characteristic search.
const string DEVNAME_RBLE
It is used to find the evaluation board to be connected by comparing with the Complete Local Name of the advertisement received by the BluetoothLEAdvertisementWatcher.
BluetoothLEAdvertisementWatcher advertisementWatcher
Saves an object of class BluetoothLEAdvertisementWatcher and uses it to perform a scan and receive advertising.
DeviceWatcher deviceWatcher
Saves an object of the DeviceWatcher class and lists the devices registered in the Windows system and the external devices found. This application is used to enumerate paired Bluetooth LE devices registered in Windows and Bluetooth LE devices sending advertisements.
AdvertisementWatcherInformation peripheralDevice
Saves the object of AdvertisementWatcherInformation class and stores the information of the evaluation board to be connected received by the scan of BluetoothLEAdvertisementWatcher class.

DeviceWatcherInformation pairedDeviceInfomation
Saves the DeviceWatcherInformation class object and stores the DeviceInformation class information of the paired evaluation board.
ObservableCollection<DeviceWatcherInformation> RBLEDevices
A collection that stores objects of the ObservableCollection<T> class. Stores the DeviceInformation class information of the devices listed in the DeviceWatcher class.
BluetoothLEDevice bluetoothLeConnectedDevice
Save the objects of the BluetoothLEDevice class when you start the connection with the evaluation board.
GattDeviceService gattPrimaryService
After connecting to the evaluation board, save the object of the LED Switch Service's BluetoothLEDevice class.
GattCharacteristic characteristicNotify
After connecting to the evaluation board, save the object of the GattCharacteristic class of Switch State Characteristic.
GattCharacteristic characteristicWrite
After connecting to the evaluation board, save the object of the GattCharacteristic class of LED Blink Rate Characteristic.
bool isLunchedApp
Used to determine the DeviceWatcher start position when DeviceWatcher is stopped. This flag is set when the application is started.
bool isClickedPairButton
Used to determine the DeviceWatcher start position when DeviceWatcher is stopped.
bool isClickedConnectButton
Used to determine the DeviceWatcher start position when DeviceWatcher is stopped. This flag is set when the Connect button is clicked.
ViewModel viewModel
Saves an object of the ViewModel class and uses it to exchange data between the program and the UI.
Timer timer
Saves an object of Timer class and uses it to stop the operation (timeout) of BluetoothLEAdvertisementWatcher and DeviceWatcher.
int notifyCount
It is used to count the number of Notifications received from the evaluation board.

4.3.1.2 Constructor

MainWindow()
Instantiate the ViewModel class and set it in the DataContext property so that you can exchange data with the UI. Start DeviceWatcher to see if it is paired with the evaluation board.

4.3.1.3 Method/Event handler

void ButtonPair_Click(object sender, RoutedEventArgs e)	
Click event handler for the Pair/Unpair button. When the Pair button is clicked, pairing with the evaluation board is started. When the Unpair button is clicked, the pairing with the evaluation board is canceled.	
Parameters	
object sender	The object that sent the event.
RoutedEventArgs e	Event parameters.

void ButtonConnect_Click(object sender, RoutedEventArgs e)	
Click event handler for the Connect/Disconnect button. When the Connect button is clicked, it connects to the paired evaluation board. When the Disconnect button is clicked, the connection with the evaluation board is disconnected.	
Parameters	
object sender	The object that sent the event.
RoutedEventArgs e	Event parameters.

void ButtonLEDBlinkRate_Click(object sender, RoutedEventArgs e)	
Click event handler for LED Blink Rate button. The value entered in the LED Blink Rate textbox is sent to the evaluation board by Write.	
Parameters	
object sender	The object that sent the event.
RoutedEventArgs e	Event parameters.

void AdvertisementWatcher_Start(int timeout)	
Run the scan with the BluetoothLEAdvertisementWatcher. BluetoothLEAdvertisementWatcher will exit if it finds a rating board to connect to or if the rating board is not found within the timeout period.	
Parameters	
int timeout	Timeout time. (Unit: milliseconds)

void AdvertisementWatcher_Stop()
Stop the BluetoothLEAdvertisementWatcher.

void AdvertisementWatcher_Received(BluetoothLEAdvertisementWatcher watcher, BluetoothLEAdvertisementReceivedEventArgs eventArgs)	
Advertising receive event handler for BluetoothLEAdvertisementWatcher. Called every time an advertisement is received. If it finds the name of the evaluation board to connect to, it starts DeviceWatcher.	
Parameters	
BluetoothLEAdvertisementWatcher watcher	The object of the BluetoothLEAdvertisementWatcher that sent the event.
BluetoothLEAdvertisementReceivedEventArgs eventArgs	Data of received events.

void AdvertisementWatcherTimer_Callback(object state)	
Callback method for Timer. Called when the operating time of BluetoothLEAdvertisementWatcher has expired, it stops BluetoothLEAdvertisementWatcher.	
Parameters	
object state	Information object for the application.

void DeviceWatcher_Start(int timeout)	
Start DeviceWatcher and list the Bluetooth LE devices registered in the system and external Bluetooth LE devices. Executes DeviceWatcher for the time specified by the timeout.	
Parameters	
int timeout	Timeout time. (Unit: milliseconds)

void DeviceWatcher_Added(DeviceWatcher sender, DeviceInformation deviceInfo)	
Added event handler for DeviceWatcher. Called when a device is added by DeviceWatcher, it adds information to the collection of the DeviceWatcherInformation class.	
Parameters	
DeviceWatcher sender	The object that sent the event.
DeviceInformation deviceInfo	DeviceInformation information for the added device.

void DeviceWatcher_Updated(DeviceWatcher sender, DeviceInformationUpdate deviceInfoUpdate)	
Updated event handler for DeviceWatcher. Called when the device is updated by DeviceWatcher, it updates the information stored in the collection of the DeviceWatcherInformation class.	
Parameters	
DeviceWatcher sender	The object that sent the event.
DeviceInformationUpdate deviceInfoUpdate	DeviceInformationUpdate information for the updated device.

void DeviceWatcher_Removed(DeviceWatcher sender, DeviceInformationUpdate deviceInfoUpdate)	
Removed event handler for DeviceWatcher. Called when the device is deleted by DeviceWatcher, it deletes the information stored in the DeviceWatcherInformation class collection.	
Parameters	
DeviceWatcher sender	The object that sent the event.
DeviceInformationUpdate deviceInfoUpdate	DeviceInformationUpdate information for the deleted device.

void DeviceWatcher_EnumerationCompleted(DeviceWatcher sender, object e)	
EnumerationCompleted event handler for DeviceWatcher. Called when DeviceWatcher enumeration is complete. This application does not process.	
Parameters	
DeviceWatcher sender	The object that sent the event.
object e	Event data.

void DeviceWatcher_Stopped(DeviceWatcher sender, object e)	
Stopped event handler for DeviceWatcher. Called when DeviceWatcher is stopped. If DeviceWatcher is started when the application starts, it will check if it is already paired. If DeviceWatcher is started by clicking the Pair button, pairing will start. If DeviceWatcher is started by clicking the Connect button, it will start connecting.	
Parameters	
DeviceWatcher sender	The object that sent the event.
object e	Event data.

void DeviceWatcherTimer_Callback(object state)	
Callback method for Timer. Called when the operating time of DeviceWatcher has expired, it stops Timer and DeviceWatcher.	
Parameters	
object state	Information object for the application.

DeviceWatcherInformation DeviceWatcherInformation_Search(string id)	
Search the collection of DeviceWatcherInformatio classes for information that matches the device ID.	
Parameters	
string id	Device ID.
Return Parameter	
DeviceWatcherInformation	An object of the DeviceWatcherInformation class.

DeviceWatcherInformation DeviceWatcherInformation_CheckBDA(string bda)	
Search the collection of DeviceWatcherInformatio classes for information that matches the Bluetooth Device Address.	
Parameters	
string bda	Bluetooth Device Address of device.
Return Parameter	
DeviceWatcherInformation	An object of the DeviceWatcherInformation class.

DeviceWatcherInformation DeviceWatcherInformation_CheckPaired()	
Search for paired devices in the collection of DeviceWatcherInformation class.	
Return Parameter	
DeviceWatcherInformation	An object of the DeviceWatcherInformation class.

void Connect()
Performs connection processing. Connect to the evaluation board, search for Services and Characteristics, and set Notification permissions on the CCCD.

void Disconnect()
Performs disconnection processing.

void ConnectionStatusChanged(BluetoothLEDevice sender, object e)	
An event handler that notifies you of changes in connection status. Called when the disconnection process with the evaluation board is completed.	
Parameters	
BluetoothLEDevice sender	The object that sent the event.
object e	Event data.

void Pair(DeviceInformation deviceInfo)	
Perform pairing processing.	
Parameters	
DeviceInformation deviceInfo	DeviceInformation information for the device to be paired.

void Unpair()
Perform unpairing processing.

void PairingRequestedHandler(DeviceInformationCustomPairing sender, DevicePairingRequestedEventArgs eventArgs)	
Event handler for pairing.	
Parameters	
DeviceInformationCustomPairing sender	The object that sent the event.
DevicePairingRequestedEventArgs eventArgs	Event data.

void ReceiveNotification(GattCharacteristic sender, GattValueChangedEventArgs eventArgs)	
Receives a Notification and displays the number of times it has been received in the SW count text box.	
Parameters	
GattCharacteristic sender	The object that sent the event.
GattValueChangedEventArgs eventArgs	Receive data.

void SendWrite(string str)	
Send the value of the LED Blink Rate textbox with Write.	
Parameters	
string str	LED Blink Rate string.

4.3.2 AdvertisementWatcherInformation class

4.3.2.1 Property

string BluetoothDeviceAddrss
Save the advertising Bluetooth Device Address. It is used to search the evaluation board to be connected from the devices listed in Device Watcher.
string Name
Save the Advertising Local Name. It is used to determine the evaluation board to be connected.

4.3.2.2 Method

void Clear()
Initialize the property.

4.3.3 DeviceWatcherInformation class

4.3.3.1 Constructor

public DeviceWatcherInformation(DeviceInformation deviceInfo)	
Set the DeviceInformation information in the deviceInformation property.	
Parameters	
DeviceInformation deviceInfo	DeviceInformation information.

4.3.3.2 Property

DeviceInformation DeviceInformation
Saves Device Information information.

4.3.3.3 Field (Member variable)

string Id
Saves the IDs of the listed devices.
string Name
Saves the names of the listed devices.
bool IsPaired
Indicates whether the enumerated devices are paired.

4.3.3.4 Method

void Update(DeviceInformationUpdate deviceInfoUpdate)	
Update the deviceInformation property.	
Parameters	
DeviceInformationUpdate deviceInfoUpdate	Device Information Update information for the updated device.

4.3.4 ViewModel class

4.3.4.1 Field (Member variable)

event PropertyChangedEventHandler PropertyChanged
Declare the PropertyChanged event.

4.3.4.2 Constructor

ViewModel()
Initialize the property.

4.3.4.3 Property

string textbox_status
It is used to display characters in the Status textbox and to get characters. The string _textbox_status field is used to hold data.
string button_pair_content
It is used to display characters on the Pair/Unpair button and to acquire characters. The string _button_pair_content field is used to hold data.
string button_connect_content
It is used to display characters on the Connect/Disconnect button and to get characters. The string _button_connect_content field is used to hold the data.
string textbox_sw_count
It is used to display characters in the SW Count textbox and to get characters. The string _textbox_sw_count field is used to hold data.
string textbox_led_blink_rate
It is used to display characters in the LED Blink Rate textbox and acquire characters. The string _textbox_led_blink_rate field is used to hold data.
bool button_pair_isenabled
It is used to enable or disable the Pair/Unpair button. The bool _button_pair_isenabled field is used to hold data.

bool button_connect_isenabled
It is used to enable or disable the Connect/Disconnect button. The bool _button_connect_isenabled field is used to hold data.
bool button_led_blink_rate_isenabled
It is used to enable or disable the LED Blink Rate send button. The bool _button_led_blink_rate_isenabled field is used to hold data.

4.3.4.4 Method

void NotifyPropertyChanged(string propName)	
The PropertyChanged event notifies the UI that is bound to each property of the data change.	
Parameters	
string propName	Property name string.

4.3.5 Flowchart

The operation when starting the LED Switch Service Client and operating the UI is explained using a flowchart.

4.3.5.1 Launch application

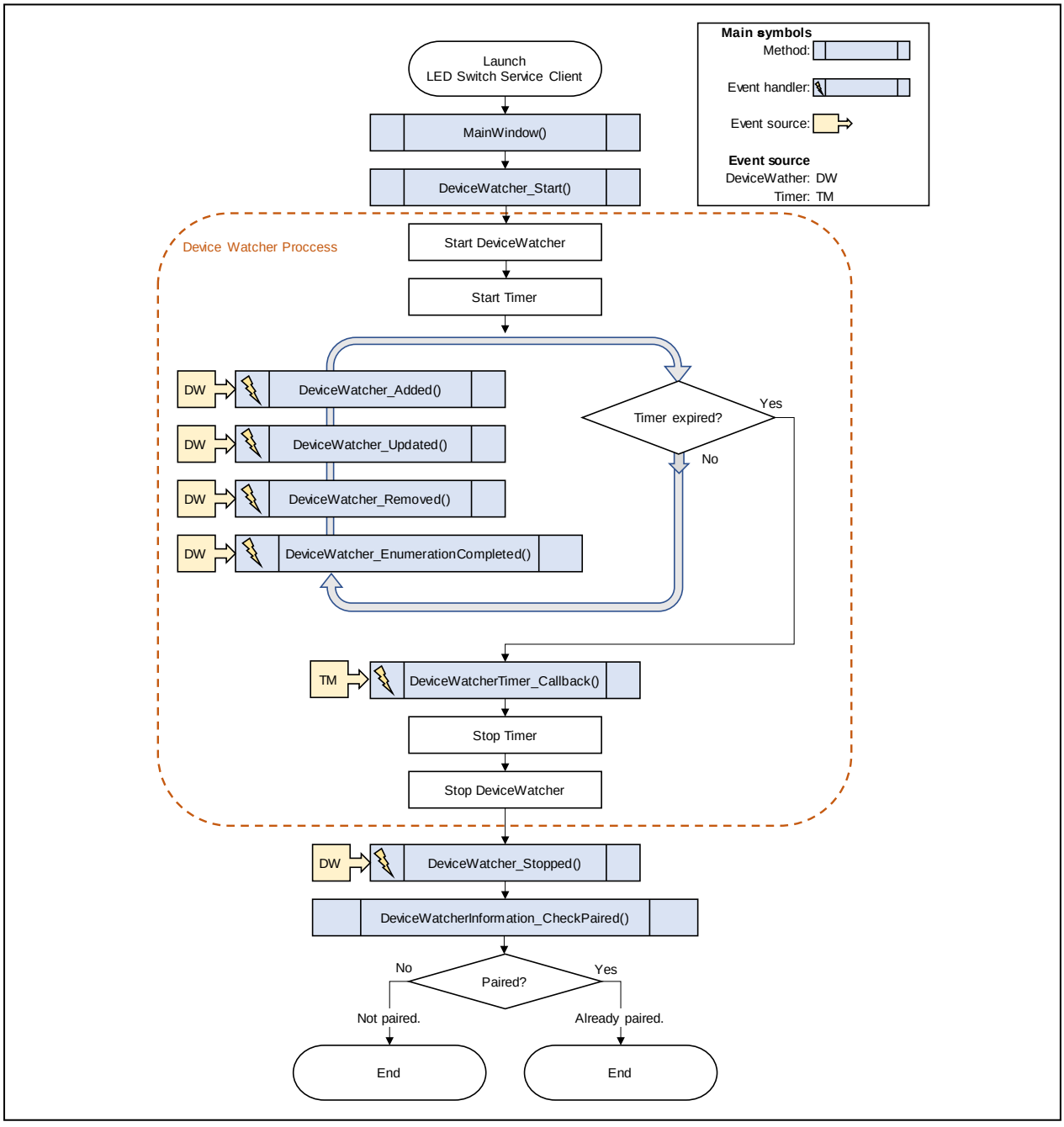

Figure 4-1 Flowchart of Launch application

When you launch the application, it checks if it has been paired with the evaluation board. Use the DeviceWatcher class to enumerate the devices registered in the Windows system and search for saved pairing information with the evaluation board.

If pairing information is not found, enable the Pair button.

If pairing information is found, enable the Connect button to connect to the evaluation board and the Unpair button to unpair.

4.3.5.2 Pair/Unpair button

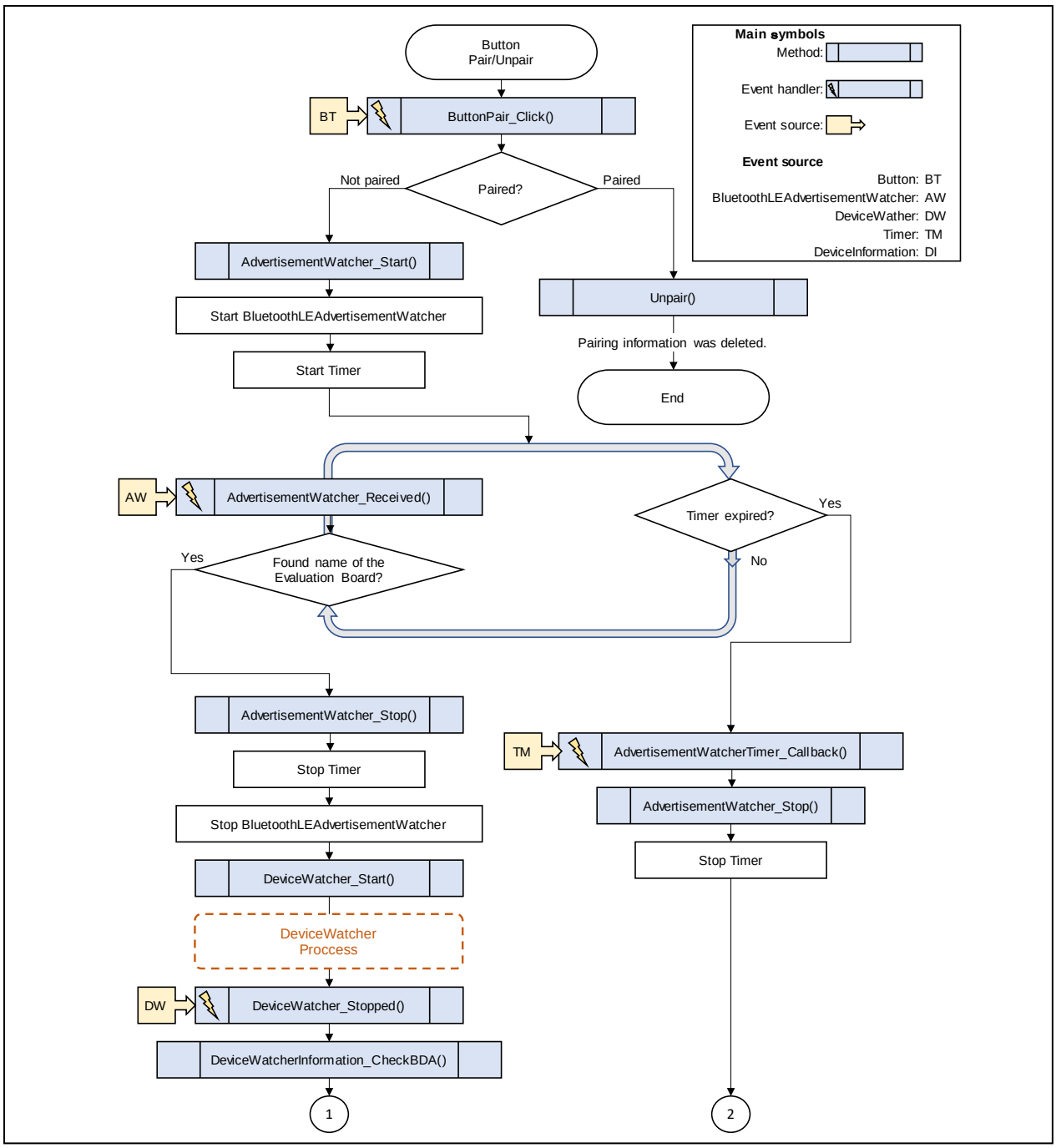

Figure 4-2 Flowchart of Pair/Unpair button (1)

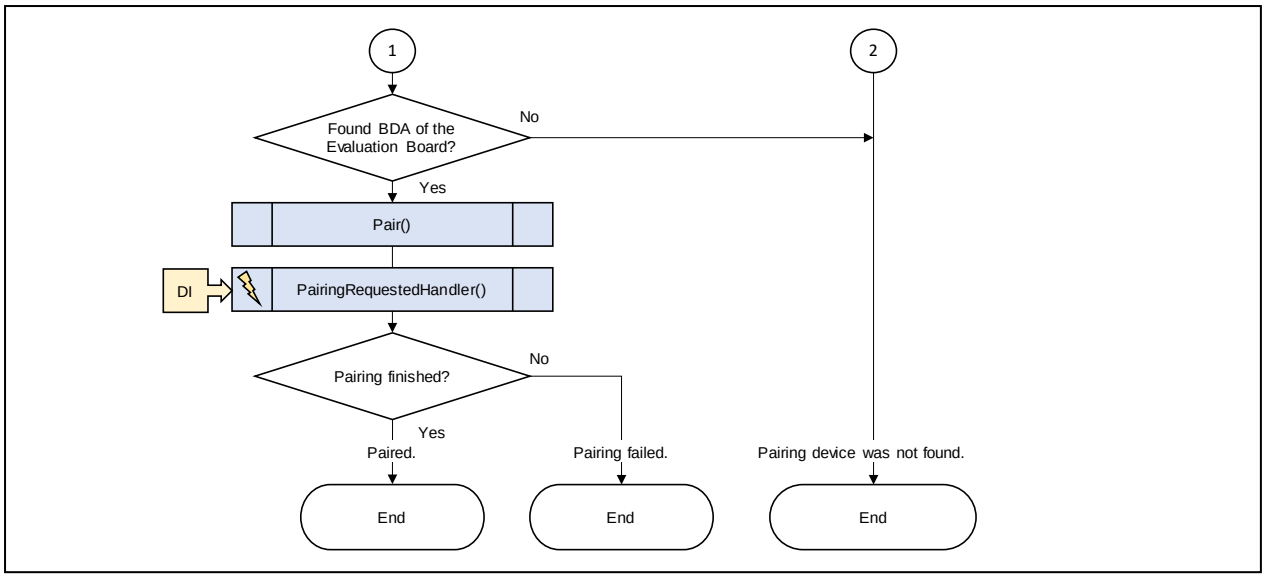

Figure 4-3 Flowchart of Pair/Unpair button (2)

It operates as a Pair button when not paired with the evaluation board, and as an Unpair button when paired with the evaluation board.

When the Pair button is clicked, it starts scanning using the BluetoothLEAdvertisementWatcher class to find the evaluation board and receives the advertisements sent by the evaluation board. When it finds the name of the evaluation board in the advertising data, it uses the DeviceWatcher class to list peripheral devices and devices registered on the Windows system. If a device that matches the Bluetooth Device Address of the evaluation board found by scanning is found among the listed, pairing will start. Pairing uses the PairAsync method of the DeviceInformation class. As the pairing process progresses, the PairingRequested event occurs and the registered PairingRequestedHandler event handler is called. When the Accept method of the DevicePairingRequestedEventArgs class passed in the event parameter is executed, pairing with the evaluation board is completed and the pairing information is registered in the Windows system.

When the Unpair button is clicked, the UnpairAsync method of the DeviceInformation class is used to remove the pairing information from the Windows system.

4.3.5.3 Connect/Disconnect button

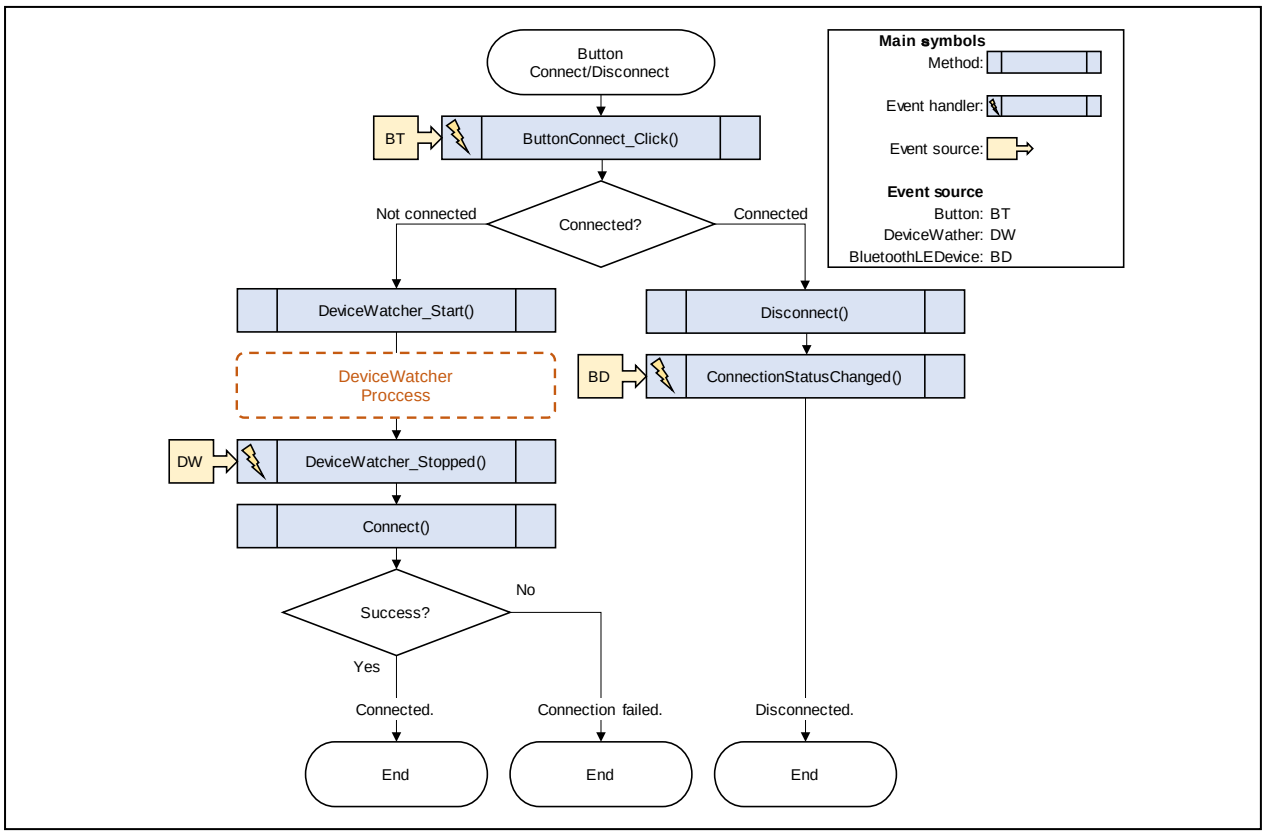

Figure 4-4 Flowchart of Connect/Disconnect button

It operates as a Connect button when not connected to the evaluation board, and as a Disconnect button when connected to the evaluation board.

When the Connect button is clicked, it starts DeviceWatcher, updates the pairing information of the DeviceWatcherInformation class, and connects to the evaluation board using the updated pairing information. Once connected, it searches for LED Switch Service, Notify characteristic, and Write characteristic, and saves Service and Characteristic objects in fields for use in data communication. It also writes a enable notirication to the Client Characteristic Configuration Descriptor so that the evaluation board can send Notifications.

When the Disconnect button is clicked, Dispose is executed in the order of Characteristic, Service, and BluetoothLEDevice to release resources. When the resource is released, the ConnectionStatusChanged event of the BluetoothLEDevice class occurs and the connection with the evaluation board is disconnected. As a process after disconnection, unregister the ConnectionStatusChanged event handler and initialize the field that stores Service and Characteristic.

4.3.5.4 LED Blink Rate button (Send write)

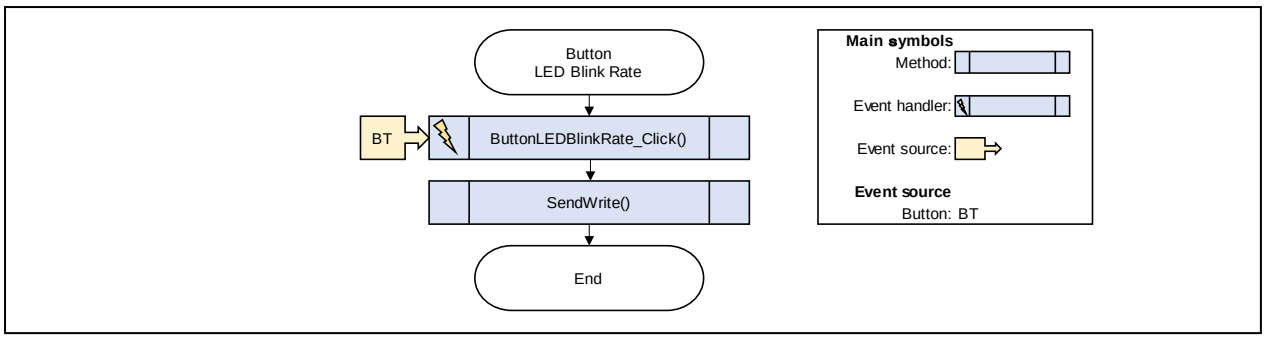

Figure 4-5 Flowchart of LED Blink Rate button (Send write)

Sends the data inputted in the LED Blink Rate textbox to the evaluation board. Data is sent by the WriteValueAsync method of the GattCharacteristic class.

4.3.5.5 Receive SW state (Receive indication)

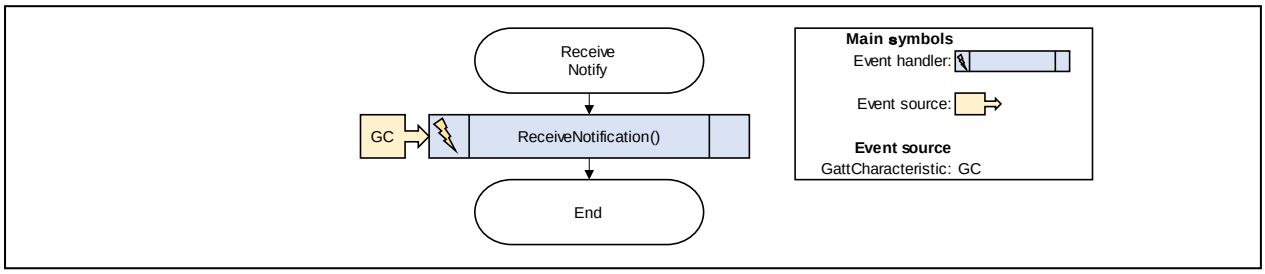

Figure 4-6 Flowchart of Receive SW state (Receive indication)

Receiving a Notification from the evaluation board, the ValueChanged event of the GattCharacteristic class occurs. Receives SW State data as an event parameter and displays the number of Notifications received in the SW Count textbox.

4.3.5.6 Disconnect from evaluation board

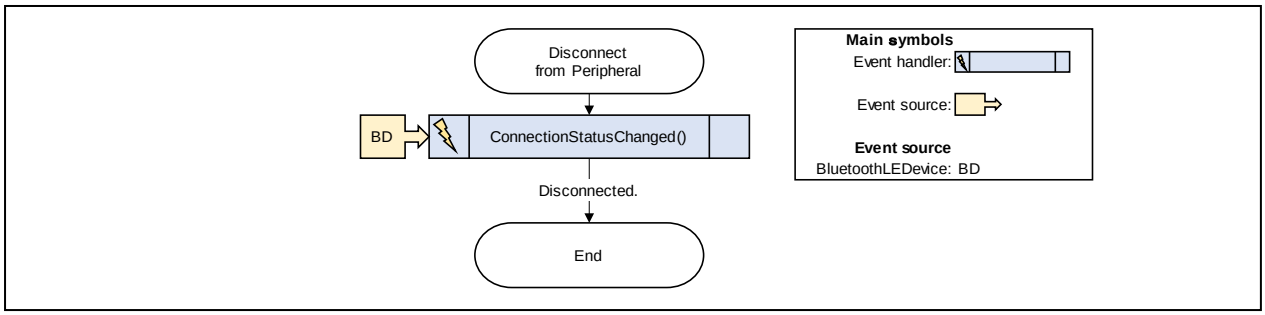

Figure 4-7 Flowchart of Disconnect from evaluation board

The ConnectionStatusChanged event of the BluetoothLEDevice class occurs when communication is not possible due to the power of the evaluation board being turned off or the evaluation board being reset.

As a disconnection process, unregister the ConnectionStatusChanged event handler and initialize the fields that store the Service and Characteristic.

4.4 Virtual UART Client

This section describes the classes and processing contents defined in the LED Switch Service Client.

Table 4-2 Virtual UART Client Class Overview

Class Name	Description
MainWindow	A class that describes the application processing of the Virtual UART Client.
AdvertisementWatcherInformation	A class that stores information obtained from advertising sent by the evaluation board to be connected.
ViewModel	A class used for data exchange between the program and the UI.

4.4.1 MainWindow class

4.4.1.1 Field (Member variable)

Guid UUID_VUART_SERVICE
Define the Primary Service UUID of the Virtual UART Profile. After connecting to the evaluation board, use it to search for Services.
Guid UUID_VUART_INDICATION
Define the Indication Characteristic UUID of the Virtual UART Profile. After connecting to the evaluation board, use it for Characteristic search.
Guid UUID_VUART_WRITE
Define the Write Characteristic UUID of the Virtual UART Profile. After connecting to the evaluation board, use it for Characteristic search.
const int VUART_WRITE_LENGTH
Used to determine the number of data transmitted from the Virtual UART Client to the evaluation board.
BluetoothLEAdvertisementWatcher advertisementWatcher
Saves the object of BluetoothLEAdvertisementWatcher class and uses it to perform a scan and receive advertising.
AdvertisementWatcherInformation vuartDevice
Saves the object of AdvertisementWatcherInformation class and stores the information of the evaluation board equipped with the virtual UART profile.
BluetoothLEDevice bluetoothLeConnectedDevice
Save the objects of the BluetoothLEDevice class when you start the connection with the evaluation board.
GattDeviceService gattPrimaryService
After connecting to the evaluation board, save the object of the BluetoothLEDevice class of the Virtual UART Profile Primary Service.
GattCharacteristic characteristicIndicate
After connecting to the evaluation board, save the object of the GattCharacteristic class of the Virtual UART Profile Indication Characteristic.

GattCharacteristic characteristicWrite
After connecting to the evaluation board, save the object of the GattCharacteristic class of the Virtual UART Profile Write Characteristic.
bool isFoundVuartDevice
It is used to judge the transition to connection processing when BluetoothLEAdvertisementWatcher is stopped. This flag is set when the Connect button is clicked.
ViewModel viewModel
Save the ViewModel class object and use it to exchange data between the program and the UI.
Timer timer
Saves an object of Timer class and uses it to stop the operation (timeout) of BluetoothLEAdvertisementWatcher.
StreamWriter streamWriter
Saves an object of the StreamWriter class that writes to the log file.
string pathLogFile
Saves the path information of the log file.
StreamWriter streamReader
Saves the StreamReader class object that reads from the send file.
string pathSendFile
Saves the path information of the send file.

4.4.1.2 Constructor

MainWindow()
Instantiate the ViewModel class and set it in the DataContext property so that you can exchange data with the UI.

4.4.1.3 Method/Event handler

void ButtonConnect_Click(object sender, RoutedEventArgs e)	
Click event handler for the Connect/Disconnect button. When the Connect button is clicked, BluetoothLEAdvertisementWatcher is started to search for the evaluation board to be connected. When the Disconnect button is clicked, the connection with the evaluation board is disconnected.	
Parameters	
object sender	The object that sent the event.
RoutedEventArgs e	Event parameters.

void ButtonLog_Click(object sender, RoutedEventArgs e)	
Click event handler for the Log button. Displays a file save dialog and creates an instance of the StreamWrite class for the specified file.	
Parameters	
object sender	The object that sent the event.
RoutedEventArgs e	Event parameters.

void ButtonFile_Click(object sender, RoutedEventArgs e)	
Click event handler for File button. Displays the file open dialog and sends the character data of the specified file to the evaluation board.	
Parameters	
object sender	The object that sent the event.
RoutedEventArgs e	Event parameters.

void ButtonSend_Click(object sender, RoutedEventArgs e)	
Click event handler for the Send button. The character data inputted in the Send Data input textbox is sent to the evaluation board.	
Parameters	
object sender	The object that sent the event.
RoutedEventArgs e	Event parameters.

void TextBoxSend_KeyDown(object sender, RoutedEventArgs e)	
Keydown event handler for Send Data input textbox. If you press the Enter (Return) key in the Send Data input textbox, the character data inputted in the textbox will be sent to the evaluation board.	
Parameters	
object sender	The object that sent the event.
RoutedEventArgs e	Event parameters.

void SendTextBoxData()
Check the number of characters in the data inputted in the Send Data input textbox and send.

void AdvertisementWatcher_Start(int timeout)	
Run the scan with the BluetoothLEAdvertisementWatcher. BluetoothLEAdvertisementWatcher will exit if it finds a evaluation board to connect to or if the evaluation board is not found within the timeout period.	
Parameters	
int timeout	Timeout time. (Unit: milliseconds)

void AdvertisementWatcher_Stop()
Stop the BluetoothLEAdvertisementWatcher. If it finds an evaluation board to connect to, it will start connecting with the evaluation board.

void AdvertisementWatcher_Received(BluetoothLEAdvertisementWatcher watcher, BluetoothLEAdvertisementReceivedEventArgs eventArgs)	
Advertising receive event handler for BluetoothLEAdvertisementWatcher. Called every time an advertisement is received. If you find an evaluation board with the address entered in the Bluetooth Device Address textbox or an evaluation board with the UUID of the Virtual UART Profile, stop BluetoothLEAdvertisementWatcher.	
Parameters	
BluetoothLEAdvertisementWatcher watcher	The object of the BluetoothLEAdvertisementWatcher that sent the event.
BluetoothLEAdvertisementReceivedEventArgs eventArgs	Data of received events.

void AdvertisementWatcherTimer_Callback(object state)	
Callback method for Timer. Called when the operating time of BluetoothLEAdvertisementWatcher has expired, it stops BluetoothLEAdvertisementWatcher.	
Parameters	
object state	Information object for the application.

void Connect()
Perform connection processing, search for Service and Characteristic, and set Indicate permission to CCCD.

void Disconnect()
Performs disconnection processing.

void ConnectionStatusChanged(BluetoothLEDevice sender, object e)	
An event handler that notifies you of changes in connection status. Called when the disconnection process with the evaluation board is completed.	
Parameters	
BluetoothLEDevice sender	The object that sent the event.
object e	Event data.

void ReceiveIndiation(GattCharacteristic sender, GattValueChangedEventArgs eventArgs)	
Receives Indicate and displays the received character string data in the Receive data display textbox.	
Parameters	
GattCharacteristic sender	The object that sent the event.
GattValueChangedEventArgs eventArgs	Receive data.

void SendWrite(string str)	
Send character data with Write.	
Parameters	
string str	String data.

void FileAppend(string str)	
Add the string to the file.	
Parameters	
string str	A string to add to the file.

void FileSend()
Read the character string data from the file and send it to the evaluation board by Write.

4.4.2 AdvertisementWatcherInformation class

4.4.2.1 Property

string BluetoothDeviceAddrss
Saves the advertising Bluetooth Device Address (string). It is used to determine the evaluation board to be connected.
string Name
Save the Advertising Local Name.
Guid ServiceUuid
Saves the advertising 128bit UUID. It is used to determine the evaluation board to be connected.

ulong BluetoothDeviceAddrss_ulong
Save the advertising Bluetooth Device Address (ulong). It is used as a parameter to specify the connection target at the start of connection.

4.4.2.2 Method

void Clear()
Initialize the property.

4.4.3 ViewModel class

4.4.3.1 Field (Member variable)

event PropertyChangedEventHandler PropertyChanged
Declare the PropertyChanged event.

4.4.3.2 Constructor

ViewModel()
Initialize the property.

4.4.3.3 Property

string textbox_status
It is used to display characters in the Status textbox and to get characters. The string _textbox_status field is used to hold data.
string textbox_bdaddress
It is used to display characters in the Bluetooth Device Address textbox and to get characters. The string _textbox_bdaddress field is used to hold data.
string textbox_receive_text
It is used to display characters in the Receive Data textbox and to get characters. The string _textbox_receive_text field is used to hold data.
string button_connect_content
It is used to display characters on the Connect/Disconnect button and to get characters. The string _button_connect_content field is used to hold the data.
string button_log_content
It is used to display characters on the Receive Data Save button and to get characters. The string _button_log_content field is used to hold data.
bool button_connect_isenabled
It is used to enable or disable the Connect/Disconnect button. The bool _button_connect_isenabled field is used to hold data.

bool button_log_isenabled
It is used to enable or disable the Receive Data Save button. The bool _button_log_isenabled field is used to hold data.
bool button_file_isenabled
It is used to enable or disable the Send File button. The bool _button_file_isenabled field is used to hold data.
bool button_send_isenabled
It is used to enable or disable the Send Data button. The bool _button_send_isenabled field is used to hold data.

4.4.3.4 Method

void NotifyPropertyChanged(string propName)	
The PropertyChanged event notifies the UI that is bound to each property of the data change.	
Parameters	
string propName	Property name string.

4.4.4 Flowchart

The operation when starting the Virtual UART Client and operating the UI is explained using a flowchart.

4.4.4.1 Launch application

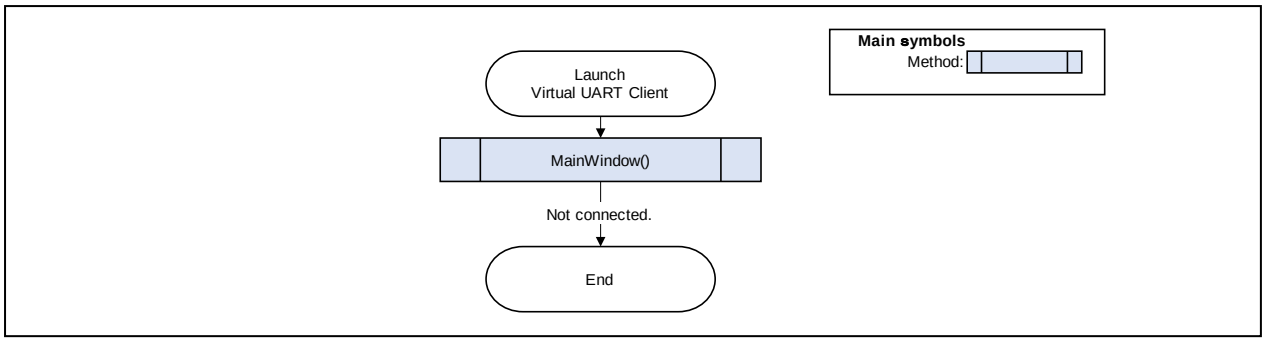

Figure 4-8 Flowchart of Launch application

When you launch the application, it will be in a state waiting for the Connect button to be clicked.

4.4.4.2 Connect/Disconnect button

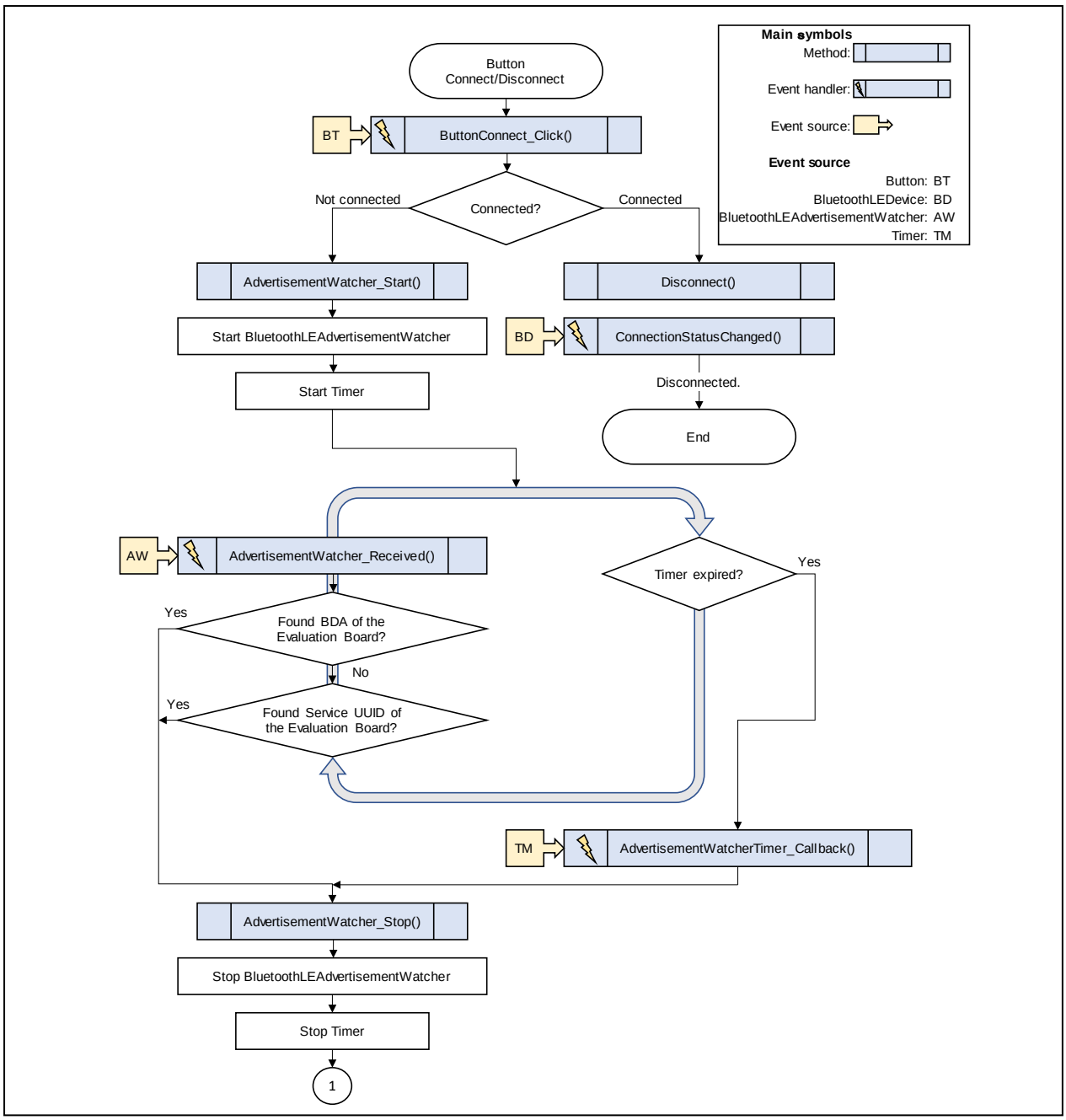

Figure 4-9 Flowchart of Connect/Disconnect button (1)

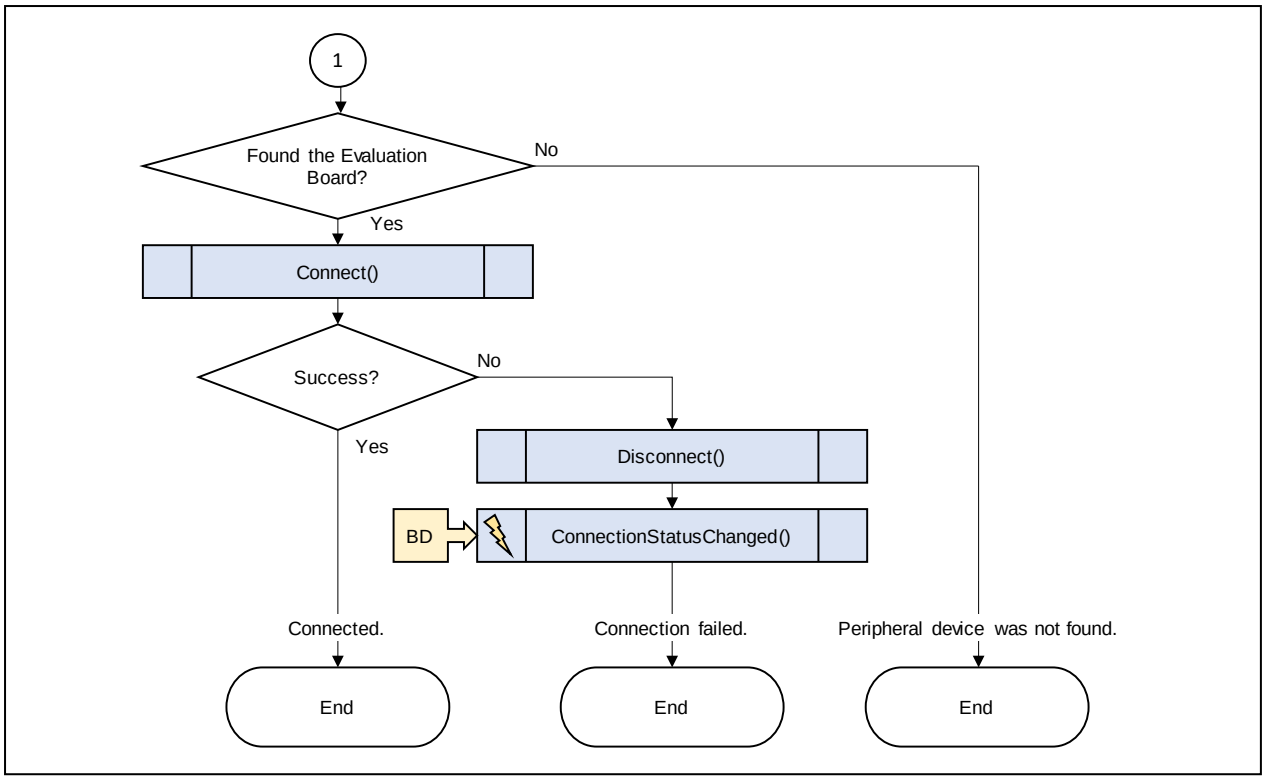

Figure 4-10 Flowchart of Connect/Disconnect button (2)

It operates as a Connect button when not connected to the evaluation board, and as a Disconnect button when connected to the evaluation board.

When the Connect button is clicked, the BluetoothLEAdvertisementWatcher will start scanning and receive advertising. If it finds an address entered in the Bluetooth Device Address textbox or an advertising with the UUID of the Virtual UART profile, it will connect.

When the Disconnect button is clicked, Dispose is executed in the order of Characteristic, Service, and BluetoothLEDevice to release resources. When the resource is released, the ConnectionStatusChanged event of the BluetoothLEDevice class occurs and the connection with the evaluation board is disconnected. As a process after disconnection, unregister the ConnectionStatusChanged event handler and initialize the field that stores Service and Characteristic.

4.4.4.3 Log button

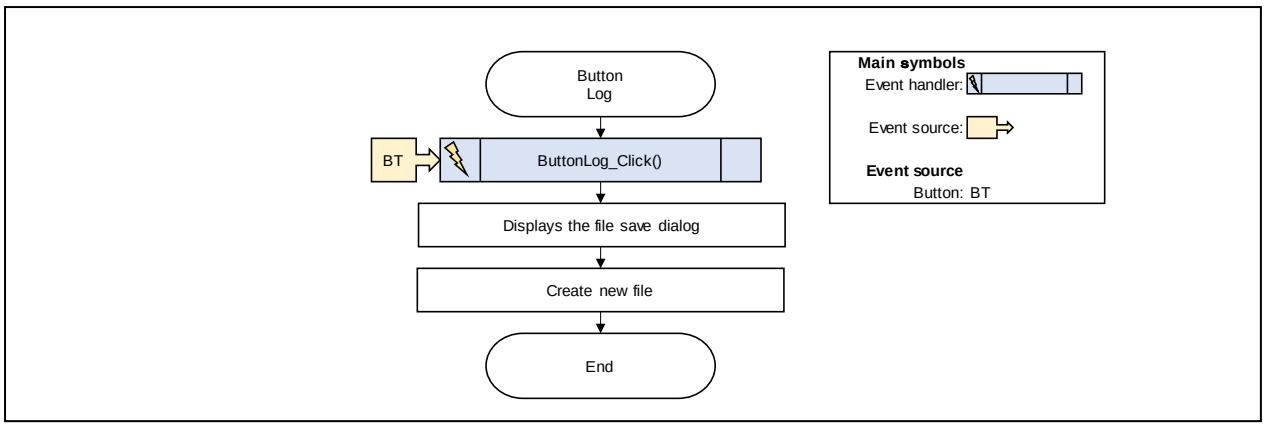

Figure 4-11 Flowchart of Log button

It operates as a Log button when not logging, and as a Logging button (log stop button) when logging.

When the Log button is clicked, a file save dialog will be displayed, saving the file path in the field and creating a new file.

When the Logging button is clicked, it initializes the field that stores the file path so that it will not be logged when character data is received.

4.4.4.4 File button (Send write)

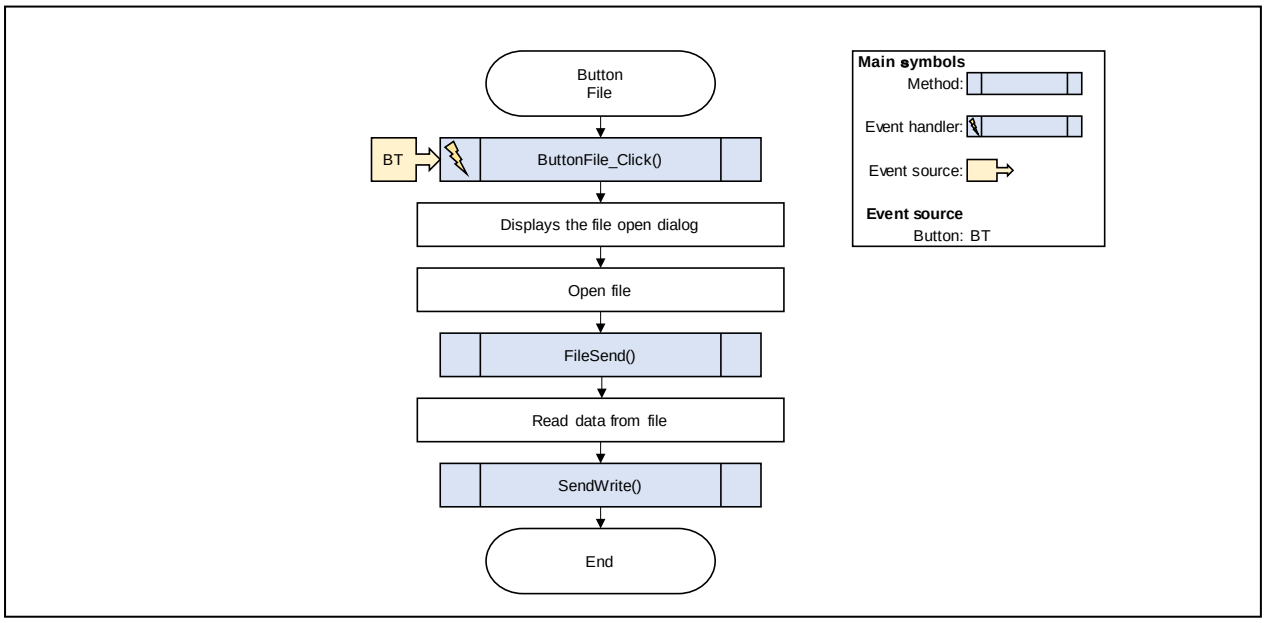

Figure 4-12 Flowchart of File button (Send write)

Displays the file open dialog and sends the character data of the specified file to the evaluation board. Data is sent every 20 bytes with the WriteValueAsync method of the GattCharacteristic class.

4.4.4.5 Send button (Send write)

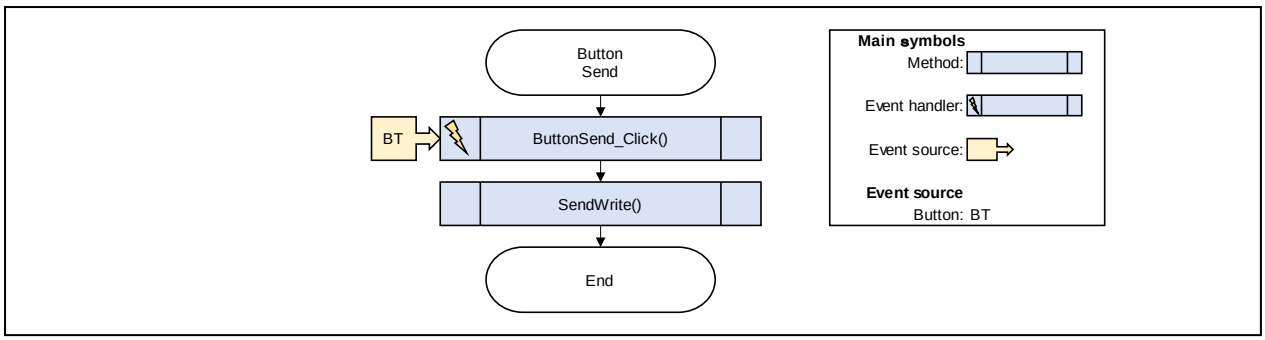

Figure 4-13 Flowchart of Send button (Send write)

Send The data entered in the data textbox is sent to the evaluation board. Data is sent by the WriteValueAsync method of the GattCharacteristic class. The data length that can be sent is 20 bytes.

4.4.4.6 Send textbox, key down (Send write)

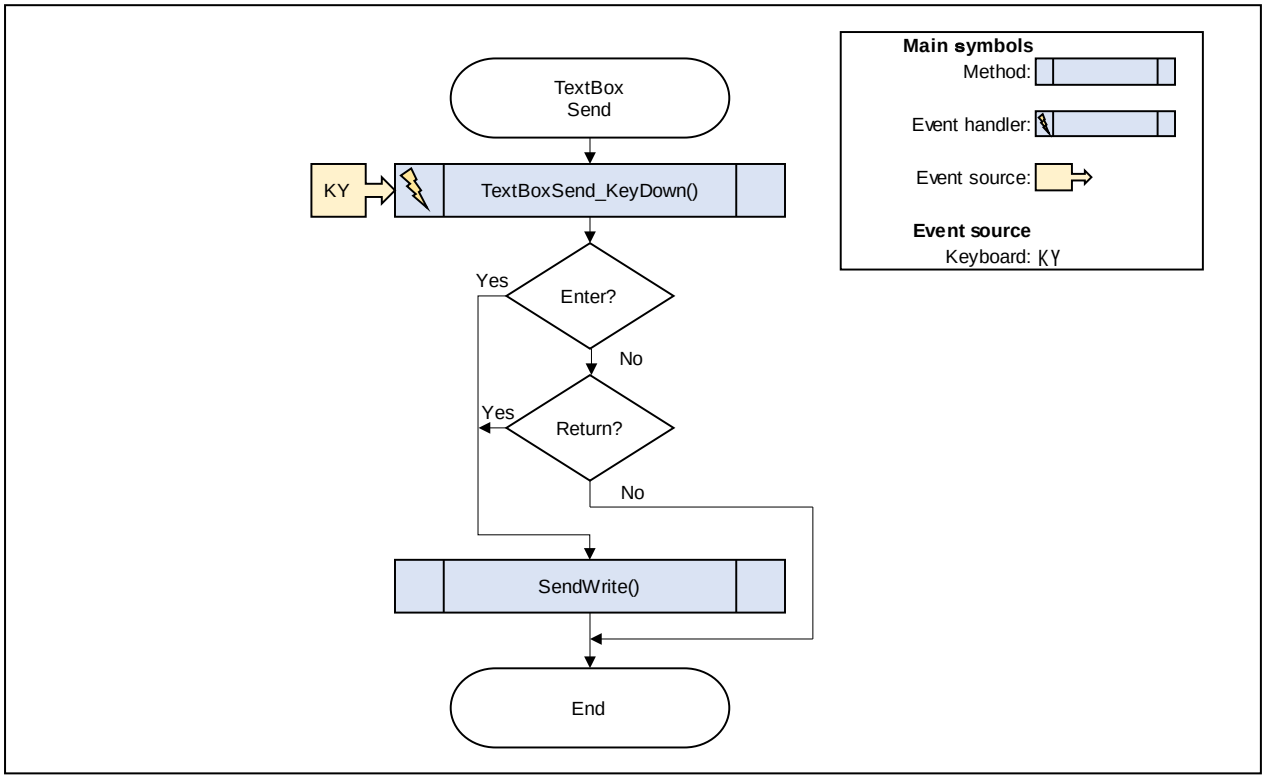

Figure 4-14 Flowchart of Send textbox, key down (Send write)

Send data inputted in the data textbox is sent to the evaluation board by pressing the Enter key or the Return key. Data is sent by the WriteValueAsync method of the GattCharacteristic class. The data length that can be sent is 20 bytes.

4.4.4.7 Receive string data (Receive indication)

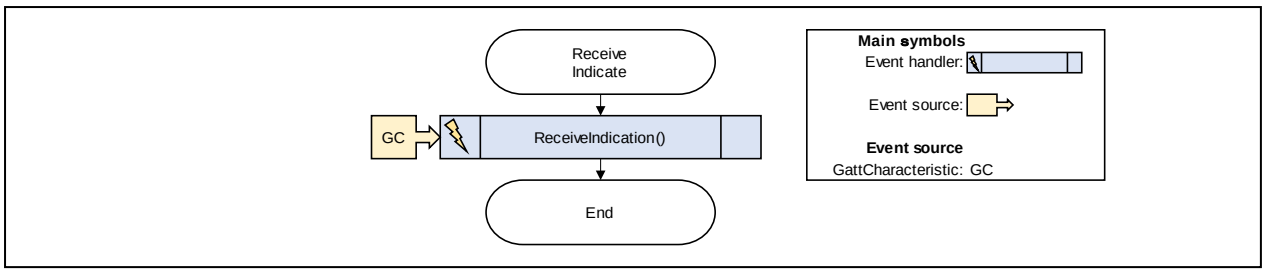

Figure 4-15 Flowchart of Receive string data (Receive indication)

Receiving the Indication from the evaluation board, the ValueChanged event of the GattCharacteristic class is occurs. Receives character data as an event parameter and displays the received character data in the Receive textbox.

4.4.4.8 Disconnect from evaluation board

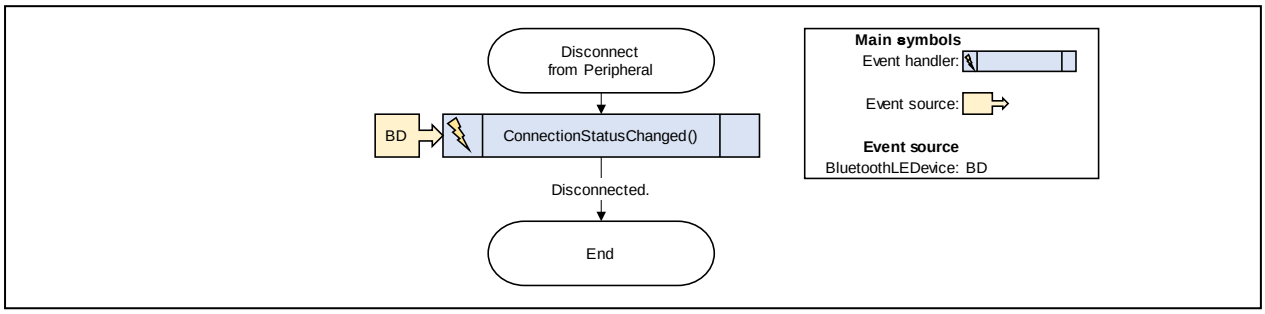

Figure 4-16 Flowchart of Disconnect from evaluation board

The ConnectionStatusChanged event of the BluetoothLEDevice class occurs when communication is not possible due to the power of the evaluation board being turned off or the evaluation board being reset.

As a disconnection process, unregister the ConnectionStatusChanged event handler and initialize the fields that store the Service and Characteristic.

4.5 Windows 10 Bluetooth LE communication

The source code related to Bluetooth LE communication of Windows 10 is explained by excerpting from the source code of LED Switch Service Client. The source code related to Bluetooth LE communication of Virtual UART Client is almost the same as the source code of LED Switch Service Client, so the explanation here is omitted.

The source code shown in this section removes unnecessary code for debug output and Bluetooth LE communication.

4.5.1 BluetoothLEAdvertisementWatcher

BluetoothLEAdvertisementWatcher is a class that can perform scanning and receive advertising (beacon) sent by Bluetooth LE devices.

To start scanning, create a BluetoothLEAdvertisementWatcher object, set the scanning mode (Active mode), and register the event handler (AdvertisementWatcher_Received) that will be called when the advertisement is received. Then call the Start method to start the scan.

```
private void AdvertisementWatcher_Start(int timeout)
{
    advertisementWatcher = new BluetoothLEAdvertisementWatcher();
    advertisementWatcher.ScanningMode = BluetoothLEScanningMode.Active;
    advertisementWatcher.Received += AdvertisementWatcher_Received;
    advertisementWatcher.Start();
}
```

Figure 4-17 Start BluetoothLEAdvertisementWatcher

Called when an advertisement is received. Here, the name of the evaluation board to be connected is determined from the received advertising data, and if it is the connection target, the necessary parameters are saved.

```
public void AdvertisementWatcher_Received(BluetoothLEAdvertisementWatcher watcher,
                                          BluetoothLEAdvertisementReceivedEventArgs eventArgs)
{
    if (advertisementWatcher != null)
    {
        if (eventArgs.Advertisement.LocalName == DEVNAME_RBLE)
        {
            peripheralDevice.BluetoothDeviceAddrss = String.Format("{0:X}", eventArgs.BluetoothAddress);
            peripheralDevice.Name = eventArgs.Advertisement.LocalName;
        }
    }
}
```

Figure 4-18 Advertising reception event handler

Call the Stop method to stop the scan. Unregister the event handler and initialize the field that stored the BluetoothLEAdvertisementWatcher object.

```
private void AdvertisementWatcher_Stop()
{
    advertisementWatcher.Stop();
    advertisementWatcher.Received -= AdvertisementWatcher_Received;
    advertisementWatcher = null;
}
```

Figure 4-19 Stop BluetoothLEAdvertisementWatcher

4.5.2 DeviceWatcher

DeviceWatcher is a class that detects devices registered in the Windows system and devices registered in the system and dynamically enumerates them. The difference from BluetoothLEAdvertisementWatcher is that you can also list the devices that have been paired and registered in the system. Instead of notifying the application many times like BluetoothLEAdvertisementWatcher, it notifies only once, and if there is a change in the notified information, it will notify again.

To start DeviceWatcher, set the parameters for using DeviceWatcher for Bluetooth LE devices and pass it to the CreateWatcher method to create an object. Then, after registering the event handler that receives the notification, call the Start method to start DeviceWatcher.

```
private void DeviceWatcher_Start(int timeout)
{
    // DeviceWatcher Bluetooth LE setting.
    string aqsAllBluetoothLEDevices =
        "(System.Devices.Aep.ProtocolId:=\"{bb7bb05e-5972-42b5-94fc-76eaa7084d49}\")";

    string[] requestedProperties = {"System.Devices.Aep.DeviceAddress",
                                    "System.Devices.Aep.IsConnected",
                                    "System.Devices.Aep.Bluetooth.Le.IsConnectable"};

    deviceWatcher = DeviceInformation.CreateWatcher(aqsAllBluetoothLEDevices,
                                                    requestedProperties,
                                                    DeviceInformationKind.AssociationEndpoint);

    // Register event handlers before starting the watcher.
    deviceWatcher.Added   += DeviceWatcher_Added;
    deviceWatcher.Updated += DeviceWatcher_Updated;
    deviceWatcher.Removed += DeviceWatcher_Removed;
    deviceWatcher.EnumerationCompleted += DeviceWatcher_EnumerationCompleted;
    deviceWatcher.Stopped += DeviceWatcher_Stopped;

    // Start DeviceWatcher.
    deviceWatcher.Start();
}
```

Figure 4-20 Start DeviceWatcher

Called when device information is added to the Windows system. The name of the evaluation board to be connected is determined from the notified information, and if the name of the evaluation board to be connected and the information with the same Id are not included in the RBLEDevices collection, it is added to the RBLEDevices collection.

```
private void DeviceWatcher_Added(DeviceWatcher sender, DeviceInformation deviceInfo)
{
    if (deviceInfo.Name == DEVNAME_RBLE)
    {
        DeviceWatcherInformation device = DeviceWatcherInformation_Search(deviceInfo.Id);
        if (device == null)
        {
            // The deviceInfo.Id was not found in the RBLEDevices ObservableCollection.

            // Add the found RBLE device to the RBLEDevices ObservableCollection.
            RBLEDevices.Add(new DeviceWatcherInformation(deviceInfo));
        }
    }
}
```

Figure 4-21 DeviceWatcher information addition event handler

Called when device information is updated on a Windows system. If the RBLEDevices collection contains information with the same Id from the notified information, the RBLEDevices collection is updated.

```
private void DeviceWatcher_Updated(DeviceWatcher sender, DeviceInformationUpdate deviceInfoUpdate)
{
    DeviceWatcherInformation device = DeviceWatcherInformation_Search(deviceInfoUpdate.Id);
    if (device != null)
    {
        // The deviceInfo.Id was found in the RBLEDevices ObservableCollection.

        // Update the RBLE device information in the RBLEDevices ObservableCollection.
        device.Update(deviceInfoUpdate);
    }
}
```

Figure 4-22 DeviceWatcher information update event handler

Called when device information is deleted from the Windows system. If the RBLE Devices collection contains information with the same Id from the notified information, it is deleted from the RBLE Devices collection.

```
private void DeviceWatcher_Removed(DeviceWatcher sender, DeviceInformationUpdate deviceInfoUpdate)
{
    DeviceWatcherInformation device = DeviceWatcherInformation_Search(deviceInfoUpdate.Id);
    if (device != null)
    {
        // The deviceInfo.Id was found in the RBLEDevices ObservableCollection.

        // Remove the RBLE device information in the RBLEDevices ObservableCollection.
        RBLEDevices.Remove(device);
    }
}
```

Figure 4-23 DeviceWatcher information deletion event handler

Called when DeviceWatcher enumeration is complete. This application does not process.

```
private void DeviceWatcher_EnumerationCompleted(DeviceWatcher sender, object e)
{
    // do nothing.
}
```

Figure 4-24 DeviceWatcher enumeration complete event handler

To stop DeviceWatcher, call the Stop method. This application uses Timer to operate DeviceWatcher for a certain period of time and then stop it due to a timeout.

Calling the Stop method calls the DeviceWatcher stop event handler. Release the event handler and initialize the field that stored the DeviceWatcher object.

```
private void DeviceWatcherTimer_Callback(object state)
{
    deviceWatcher.Stop();
}

private void DeviceWatcher_Stopped(DeviceWatcher sender, object e)
{
    // Unregister the event handlers.
    deviceWatcher.Added -= DeviceWatcher_Added;
    deviceWatcher.Updated -= DeviceWatcher_Updated;
    deviceWatcher.Removed -= DeviceWatcher_Removed;
    deviceWatcher.EnumerationCompleted -= DeviceWatcher_EnumerationCompleted;
    deviceWatcher.Stopped -= DeviceWatcher_Stopped;

    deviceWatcher = null;
```

Figure 4-25 Stop DeviceWatcher

4.5.3 Pairing

Pairing can register Pairing information with the evaluation board in the Windows system.

Pairing is executed by registering an event handler (PairingRequestedHandler) that receives Pairing request notification and calling the PairAsync method of the DeviceInformation class. Then unregister the event handler and exit.

```
private async void Pair(DeviceInformation deviceInfo)
{
    deviceInfo.Pairing.Custom.PairingRequested += PairingRequestedHandler;
    DevicePairingResult result =
            await deviceInfo.Pairing.Custom.PairAsync(DevicePairingKinds.ConfirmOnly,
                                                      DevicePairingProtectionLevel.Encryption);
    deviceInfo.Pairing.Custom.PairingRequested -= PairingRequestedHandler;
}
```

Figure 4-26 Pairing

An event handler that is called when Pairing is executed. Pairing information is registered in the Windows system by calling the Accept method of the DevicePairingRequestedEventArgs class of the event parameter.

```
private static void PairingRequestedHandler(DeviceInformationCustomPairing sender,
                                            DevicePairingRequestedEventArgs eventArgs)
{
    switch (eventArgs.PairingKind)
    {
        case DevicePairingKinds.ConfirmOnly:
            eventArgs.Accept();
            break;
        default:
            break;
    }
}
```

Figure 4-27 Pairing request event handler

4.5.4 Unpairing

To unregister Pairing, call the UnpairAsync method of the DeviceInformation class. Calling the UnpairAsync method removes the Pairing information from the Windows system.

```
private async void Unpair()
{
    DeviceUnpairingResult result =
            await pairedDeviceInfomation.DeviceInformation.Pairing.UnpairAsync();
}
```

Figure 4-28 Unpairing

4.5.5 Connection

Connection is started by calling the FromIdAsync method of the BluetoothLEDevice class. The FromIdAsync method only triggers the Connection, and the Connection (Sending a Connection request) is executed when the GetGattServicesForUuidAsync method of the Service search is called.

```
private async void Connect()
{
    bluetoothLeConnectedDevice = await BluetoothLEDevice.FromIdAsync(pairedDeviceInfomation.Id);
```

Figure 4-29 Connection

Register the ConnectionStatusChanged event handler when all Connection processing is completed (Enable notification setting to Client Characteristic Configuration Descriptor in this application). This event handler is called when the state of Connection changes (when disconnected).

```
bluetoothLeConnectedDevice.ConnectionStatusChanged += ConnectionStatusChanged;
```

Figure 4-30 ConnectionStatusChanged event handler

4.5.6 Search for Service

Service search is executed by calling the GetGattServicesForUuidAsync method of the BluetoothLEDevice class with the UUID to be searched specified as a parameter. If the GetGattServicesForUuidAsync method is successful, it reads the Services property of the GattDeviceServicesResult class and saves the Service object (the object of the GattDeviceService class) in the field.

```
var serviceResult =
    await bluetoothLeConnectedDevice.GetGattServicesForUuidAsync(UUID_LEDSW_SERVICE);
if (serviceResult.Status == GattCommunicationStatus.Success)
{
    gattPrimaryService = serviceResult.Services[0];
```

Figure 4-31 Serch for Service

4.5.7 Search for Characteristic

Characteristic search is executed by calling the GetCharacteristicsForUuidAsync method of the GattDeviceServicesResult class with the UUID to be searched as a parameter. If the GetCharacteristicsForUuidAsync method is successful, it reads the Characteristics property of the GattCharacteristicsResult class and saves the Characteristic object (the object of the GattCharacteristic class) in the field.

If Characteristic is Notification, register the ReceiveNotification event handler that will be called when Notification is received.

```
var notificationCharacteristicsResult =
    await serviceResult.Services[0].GetCharacteristicsForUuidAsync(UUID_LEDSW_SW_STATE_NOTIFICATION);
if (notificationCharacteristicsResult.Status == GattCommunicationStatus.Success)
{
    characteristicNotify = notificationCharacteristicsResult.Characteristics[0];
    characteristicNotify.ValueChanged += ReceiveNotification;
```

Figure 4-32 Serch for Characteristic (Notification)

Similarly, Write Characteristic is called by specifying UUID as a parameter in the GetCharacteristicsForUuidAsync method.

```
var writeCharacteristicsResult =
    await serviceResult.Services[0].GetCharacteristicsForUuidAsync(UUID_LEDSW_LED_BLINK_RATE_WRITE);
if (writeCharacteristicsResult.Status == GattCommunicationStatus.Success)
{
    characteristicWrite = writeCharacteristicsResult.Characteristics[0];
```

Figure 4-33 Serch for Characteristic (Write)

4.5.8 Client Characteristic Configuration Descriptor setting

Notification permission is executed by calling the WriteClientCharacteristicConfigurationDescriptorAsync method of the GattCharacteristic class with the Notify permission parameter specified.

```
GattCommunicationStatus status =
    await characteristicNotify.WriteClientCharacteristicConfigurationDescriptorAsync(
                                GattClientCharacteristicConfigurationDescriptorValue.Notify);
```

Figure 4-34 Client Characteristic Configuration Descriptor setting

4.5.9 Disconnection

Disconnection calls the Dispose method of the GattCharacteristic class, the Dispose method of the GattDeviceService class, and the Dispose method of the BluetoothLEDevice class.

```
private void Disconnect()
{
    if (characteristicNotify != null)
    {
        characteristicNotify.Service.Dispose();
    }
    if (gattPrimaryService != null)
    {
        gattPrimaryService.Dispose();
    }
    if (bluetoothLeConnectedDevice != null)
    {
        bluetoothLeConnectedDevice.Dispose();
    }
}

private void ConnectionStatusChanged(BluetoothLEDevice sender, object e)
{
    if (sender.ConnectionStatus == BluetoothConnectionStatus.Disconnected)
    {
        // Disconnect and clear value.
        if (bluetoothLeConnectedDevice != null)
        {
            bluetoothLeConnectedDevice.ConnectionStatusChanged -= ConnectionStatusChanged;
            bluetoothLeConnectedDevice = null;
        }
        if (gattPrimaryService != null)
        {
            gattPrimaryService = null;
            characteristicNotify.ValueChanged -= ReceiveNotification;
            characteristicNotify = null;
            characteristicWrite = null;
        }
    }
}
```

Figure 4-35 Disconnection

4.5.10 Receive Notification

When Notification is received from the connected evaluation board, the registered event handler is called. Received data is passed in the parameter GattValueChangedEventArgs class.

```
private void ReceiveNotification(GattCharacteristic sender, GattValueChangedEventArgs eventArgs)
{

}
```

Figure 4-36 Receive Notification

4.5.11 Send Write

Data is sent by Write to the connected evaluation board. It can be sent by calling the WriteValueAsync method of the GattCharacteristic class with the send data specified as a parameter.

```
private async void SendWrite(string str)
{
    await characteristicWrite.WriteValueAsync(ibuffer, GattWriteOption.WriteWithResponse);
}
```

Figure 4-37 Send Write

5. Notes

5.1 Client Characteristic Configuration Descriptor write error

Depending on the state of Windows 10 you are using, the dialog shown in Figure 5-1 may be displayed whenever you click the Connect button. This is due to an exception in the call to the WriteClientCharacteristicConfigurationDescriptorAsync method that is set in the Client Characteristic Configuration Descriptor. Please upgrade Windows 10 or use another Windows 10 PC.

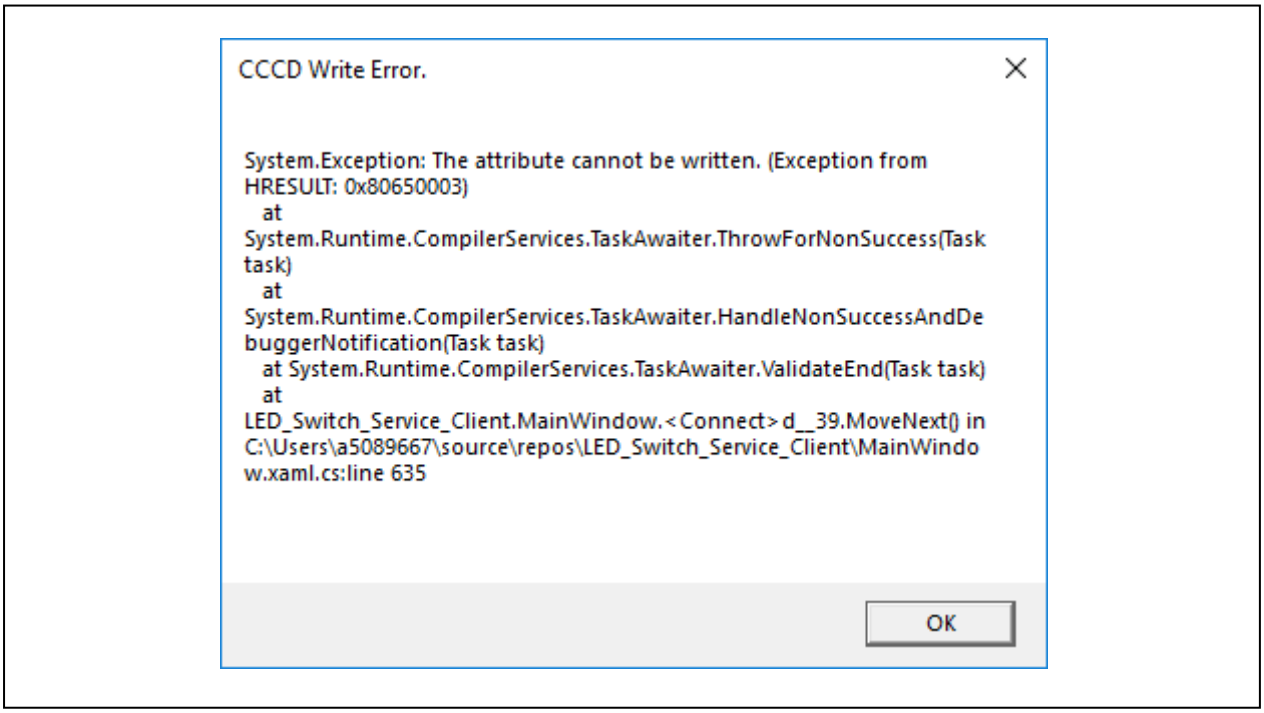

Figure 5-1 CCCD Write Error

6. Appendix

6.1 Service and characteristic customization

e source code shown in "4.5.6 Search for Service" and "4.5.7 Search for Characteristic". The difference is the UUID, field name (variable name), and event handler name, so you can easily customize it.

Figure 6-1 shows sample code to get the Indication Characteristic. (Error handling is excluded.)

```
private Guid UUID_SERVICE    = new Guid("5BC1B9F7-A1F1-40AF-9043-C43692C18D7A");
private Guid UUID_INDICATION = new Guid("5BC18D80-A1F1-40AF-9043-C43692C18D7A");


private BluetoothLEDevice bluetoothLeConnectedDevice;
private GattDeviceService gattPrimaryService;
private GattCharacteristic characteristicIndication;


private async void Connect()
{
    // Start Connection
    bluetoothLeConnectedDevice = await BluetoothLEDevice.FromIdAsync(pairedDeviceInfomation.Id);

    // Service
    var serviceResult =
        await bluetoothLeConnectedDevice.GetGattServicesForUuidAsync(UUID_SERVICE);
    gattPrimaryService = serviceResult.Services[0];

    // Characteristic Indication
    var indicationCharacteristicsResult =
        await serviceResult.Services[0].GetCharacteristicsForUuidAsync(UUID_INDICATION);
    characteristicIndication = indicationCharacteristicsResult.Characteristics[0];
    characteristicIndication.ValueChanged += ReceiveIndication;
}

private void ReceiveIndication(GattCharacteristic sender, GattValueChangedEventArgs eventArgs)
{
    // process
}
```

Figure 6-1 Service and characteristic customization

6.2 Create a new project

Here will explain how to create a project that uses the Bluetooth LE feature in a WPF application.

(1) Launch the Microsoft Visual Studio Express 2017 for Windows Desktop.

(2) Select [File] - [New Project] from the menu bar to display the "New Project" dialog.

(3) In the "New Project" dialog, select "WPF App (.NET Framework)". Input the project name and folder name to be created in [Name] and [Location]. Select the version of .NET Framework to use in Frameworks. (Here, select ".NET Framework 4.6.1".)

Click the [OK] button to close the dialog.

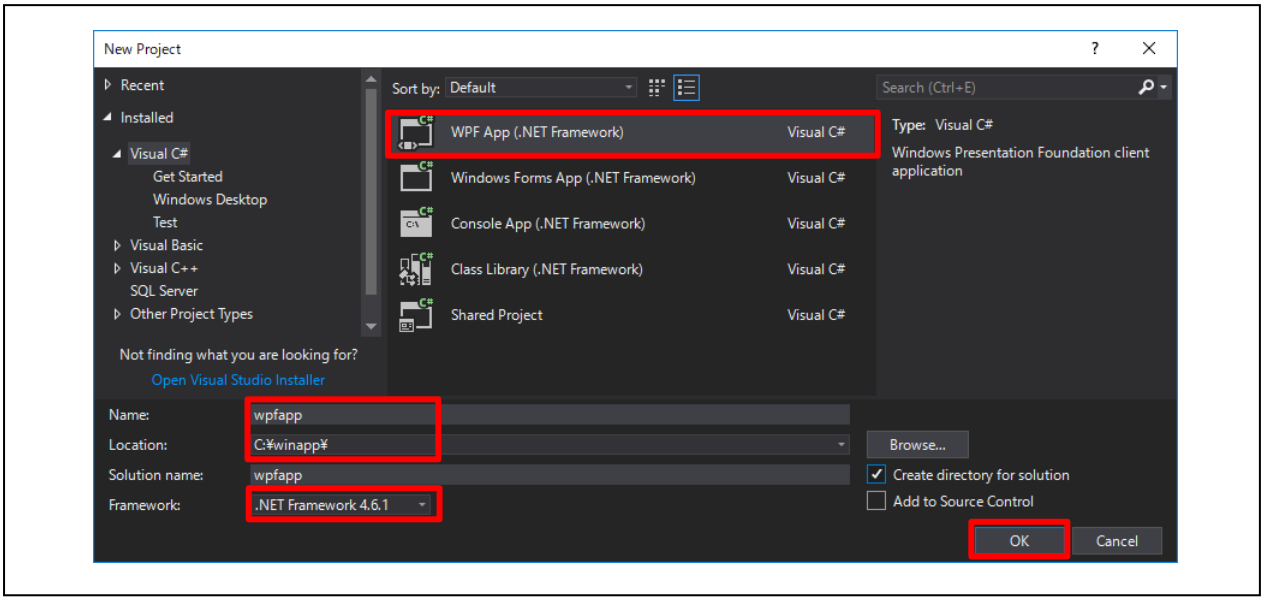

Figure 6-2 Create a new project (1)

(4) A new project window will open.

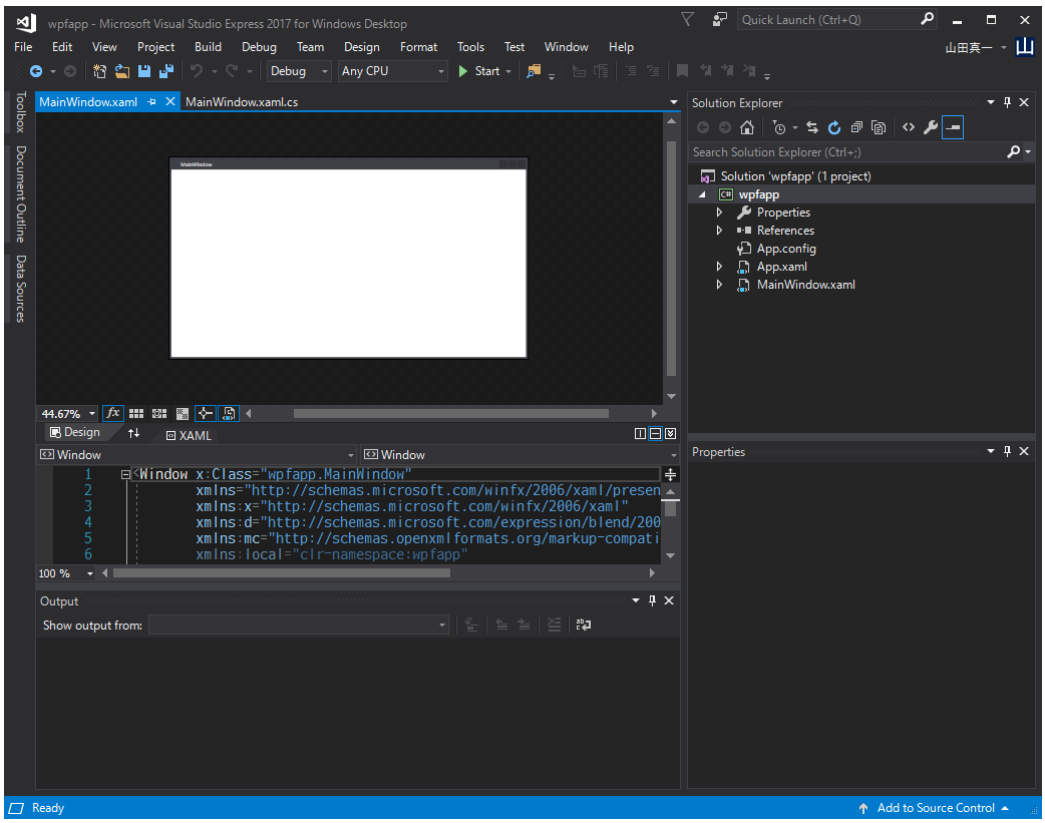

Figure 6-3 Create a new project (2)

(5) Right-click on the project name in "Solution Explorer" to display the menu. Select [Unload Project].

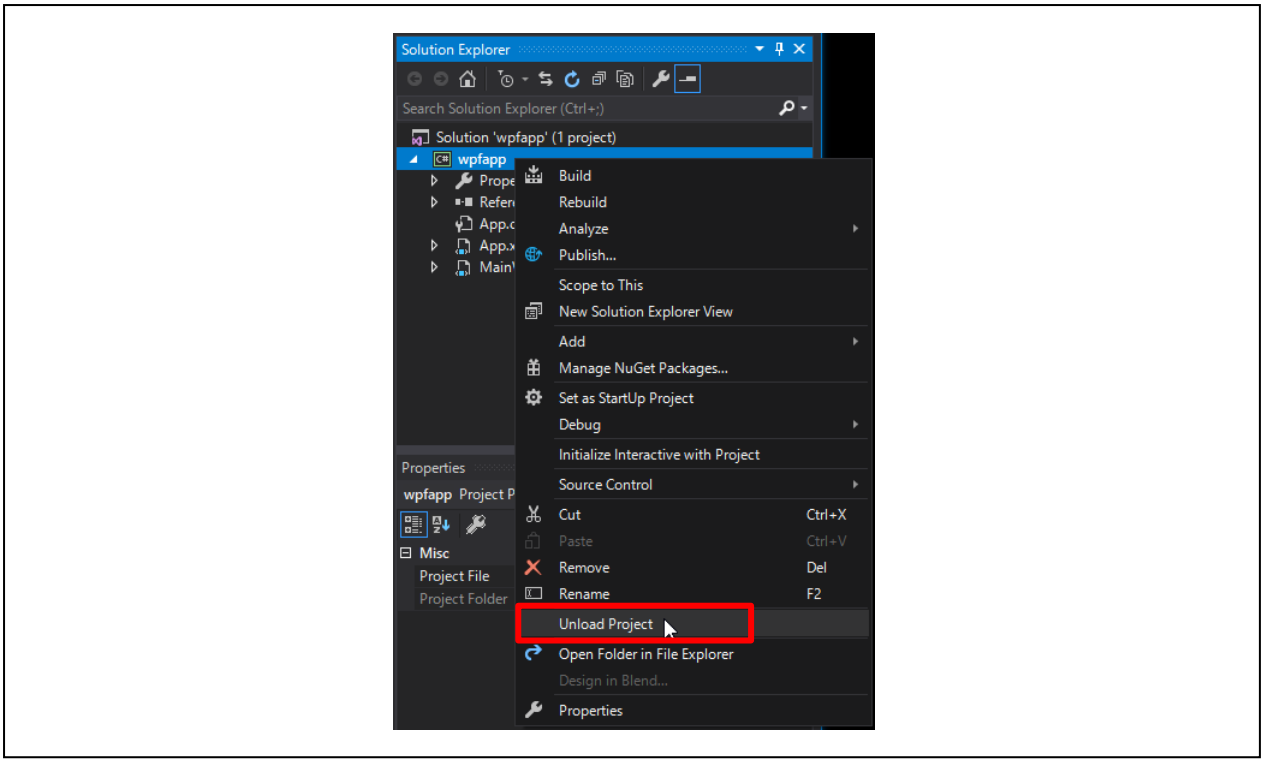

Figure 6-4 Create a new project (3)

(6) Right-click on the project name in "Solution Explorer" to display the menu. Select [Edit wpfapp.csproj].

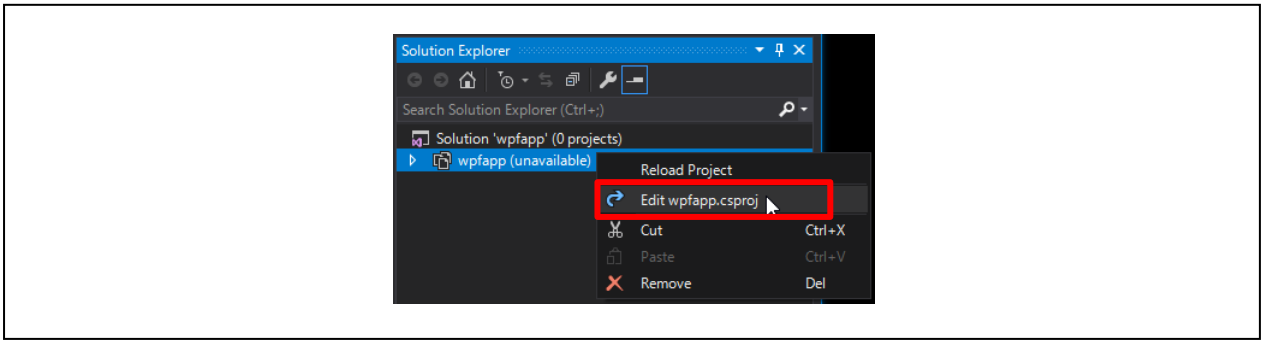

Figure 6-5 Create a new project (4)

(7) Add "<TargetPlatformVersion> 10.0 </ TargetPlatformVersion>" to wpfapp.csproj.

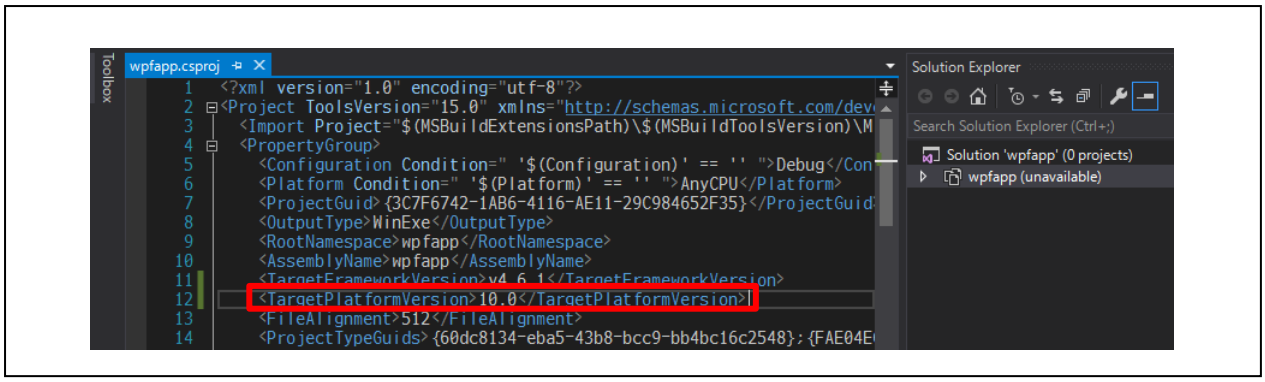

Figure 6-6 Create a new project (5)

(8) Right-click on the project name of "Solution Explorer" to display the menu. Select [Reload Project].

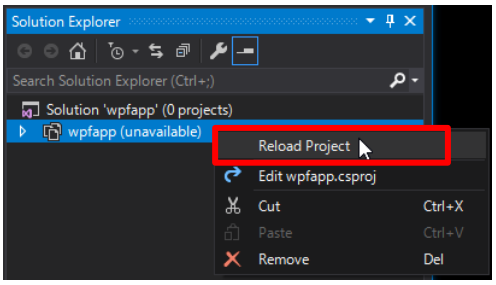

Figure 6-7 Create a new project (6)

(9) Right-click on "Reference" in "Solution Explorer" to display the menu. Select [Add Reference] to display the Reference Manager dialog.

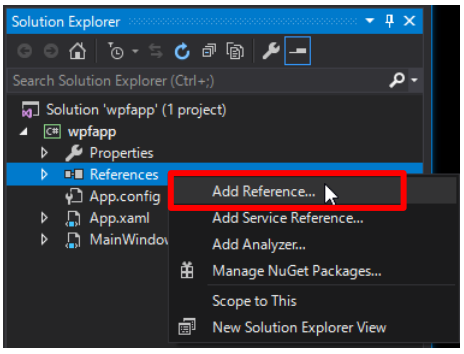

Figure 6-8 Create a new project (7)

(10) Select the "Browse" menu in the "Reference Manager" dialog. Click the [Browse] button and add the following two files.

- C:\Program Files (x86)\Windows Kits\10\UnionMetadata\10.0.19041.0\Windows.winmd
- C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETCore\v4.5\System.Runtime.WindowsRuntime.dll

Click the [OK] button to close the dialog, and the classes and methods of the Bluetooth LE function will be available.

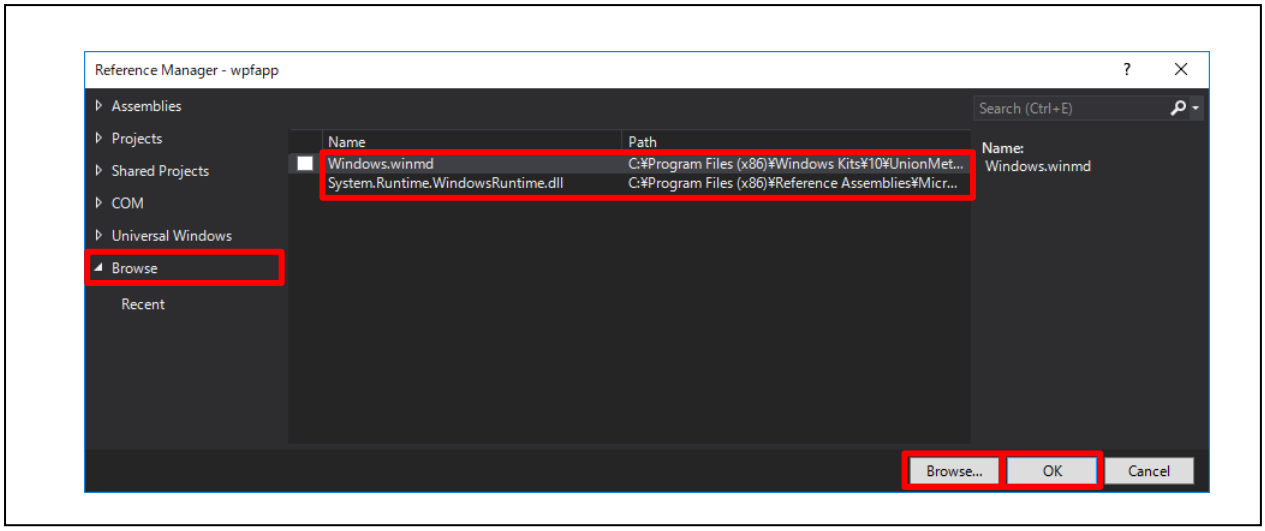

Figure 6-9 Create a new project (8)

Revision History

Rev.	Date	Description	
		Page	Summary
1.0	Aug.31.2021	-	First edition, issued

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/.