

---

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

---

## 要旨

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

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

## 対象デバイス

- iOS 端末 (iOS 13.0 以降)

## 関連ドキュメント

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

Bluetooth®のワードマークおよびロゴは Bluetooth SIG, Inc が所有する登録商標であり、ルネサスエレクトロニクス株式会社はこれらのマークをライセンスに基づいて使用しています。その他の商標および登録商標はそれぞれの所有者に帰属します。

## 目次

1. 概要 .....	4
1.1 動作環境 .....	4
1.2 注意事項 .....	6
2. 環境構築 .....	7
2.1 XCode のインストール .....	7
2.2 Homebrew のインストール .....	7
2.3 Ruby 最新版のインストール .....	9
2.4 CocoaPods のインストール .....	10
2.5 TryBT プロジェクトの起動 .....	10
2.6 Apple Developer アカウントの設定 .....	11
2.7 Certificate Signing Request の作成 .....	13
2.8 Developer Certificate の作成とインストール .....	14
2.9 TryBT 用 App ID の作成 .....	17
2.10 iOS 端末の UDID 確認 .....	20
2.11 iOS 端末の登録 .....	21
2.12 TryBT 用 Profile の作成 .....	22
2.13 TryBT プロジェクトへの Profile の関連付け .....	26
2.14 TryBT のコンパイルとインストール .....	27
3. 評価ボードへのファームウェア書き込み .....	28
4. TryBT の基本操作 .....	31
4.1 デバイス一覧画面 .....	31
4.2 接続デバイス詳細画面 .....	34
4.3 ライトデモ画面 .....	37
4.4 データデモ画面 .....	39
5. TryBT のカスタマイズ .....	42
5.1 アプリタイトルのカスタマイズ .....	42
5.2 アプリアイコンのカスタマイズ .....	42
5.3 スプラッシュ画面のカスタマイズ .....	43
5.4 アイコンデータのカスタマイズ .....	43
5.5 カスタマイズモードの有効化・無効化 .....	44
6. TryBT のファイル構成 .....	45
6.1 Info.plist について .....	46
6.2 .swift .....	46
7. TryBT の画面遷移 .....	47
8. TryBT の Bluetooth 通信 .....	48
8.1 CentralManager によるスキャンの実行 .....	48
8.2 Peripheral デバイスへの接続 .....	49
8.3 Service Discovery の実施 .....	50

---

8.4	ライトデモ画面での値の書き込み .....	52
8.5	データデモ画面での通知待受け .....	52
	改訂記録.....	54

## 1. 概要

TryBT は iOS 13.0 以降の iOS 端末で動作します。評価ボードと Bluetooth Low Energy による接続を確立後、GATT サービスでデータ通信するサンプル画面を表示します。また本アプリのソースコードは改変可能な iOS 用プロジェクトとして配布されます。

### 1.1 動作環境

TryBT の動作確認に必要なハードウェア:

- iOS 端末 iOS 13.0 以降
- Macintosh PC
- 下記のいずれかの評価ボード
  - [Target Board for RX23W](#) または [Target Board for RX23W module](#) <sup>注1</sup>
  - [EK-RA4W1](#) <sup>注2</sup>
  - [EB-RE01B](#) <sup>注3</sup>

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

RX23W グループ Target Board for RX23W クイックスタートガイド ([R20QS0014](#))

→mot フォルダ内の ble\_demo\_tbrx23w\_profile\_server\_preinstall\_yyyymmdd.mot ファイル

RX23W グループ Target Board for RX23W module クイックスタートガイド([R20QS0022](#))

→mot フォルダ内の ble\_demo\_mtbrx23w\_profile\_server\_preinstall\_yyyymmdd.mot ファイル

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

RA4W1 Group Evaluation Kit for RA4W1 EK-RA4W1 Quick Start Guide ([R20QS0015](#))

→bin.zip に含まれる Restore\_Factory/r20qs0015.srec

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

RE01B グループ Bluetooth Low Energy サンプルコード (using CMSIS Driver Package) ([R01AN5606](#))

→ROM\_Files フォルダ内の ble\_project\_server.hex

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

動作確認済みの iOS 端末:

- iPhone SE 第 2 世代 (iOS 14.7)

TryBT の動作確認に必要なソフトウェア:

- MacOS 10.14.4 Mojave 以降
- Xcode 11.0 以降 (入手方法は 2.1 節参照)

TryBT プロジェクト:

- 実装言語: Swift (<https://www.apple.com/jp/swift/>)

事前準備:

TryBT プロジェクトを iOS 端末にインストールして動作確認するには、Apple Developer Program の有料会員に加入する必要があります。

<https://developer.apple.com/jp/programs/>

## 1.2 注意事項

- 本資料は 2021 年 9 月 30 日時点での動作確認結果に基づいて作成されました。本資料に記載された情報は、弊社または第三者が提供するソフトウェアおよびツールの全てのバージョンに対応することを保証するものではありません。
- 本資料に記載された情報およびソフトウェアの利用に起因して Macintosh PC、iOS 端末に損害が発生した場合でも弊社は一切の責任を負いません。本資料末尾の「ご注意書き」もあわせてご確認ください。

## 2. 環境構築

### 2.1 XCode のインストール

統合開発環境である Xcode をインストールします。

1. AppStore を起動して、アプリ検索フィールドで"Xcode"と入力します。
2. インストールを選択します。

### 2.2 Homebrew のインストール

Homebrew は Mac OS 環境に様々なライブラリをインストール・管理するためのパッケージマネージャです。開発環境構築のためにインストールします。

1. 左記の URL にアクセスします。 [https://brew.sh/index\\_ja](https://brew.sh/index_ja)
2. 画面に記載されているシェルスクリプトコマンド横のアイコンを選択して、コマンドをコピーします。



図 2.1 Homebrew のインストール(1)

3. Finder を起動して[Applications]→[Utilities]に移動し、Terminal を起動します。

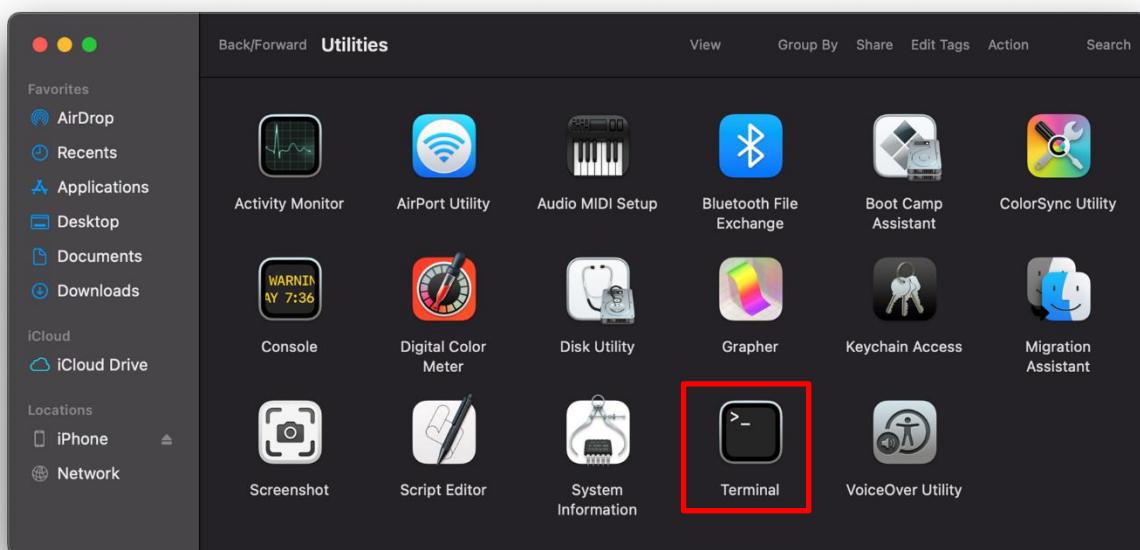
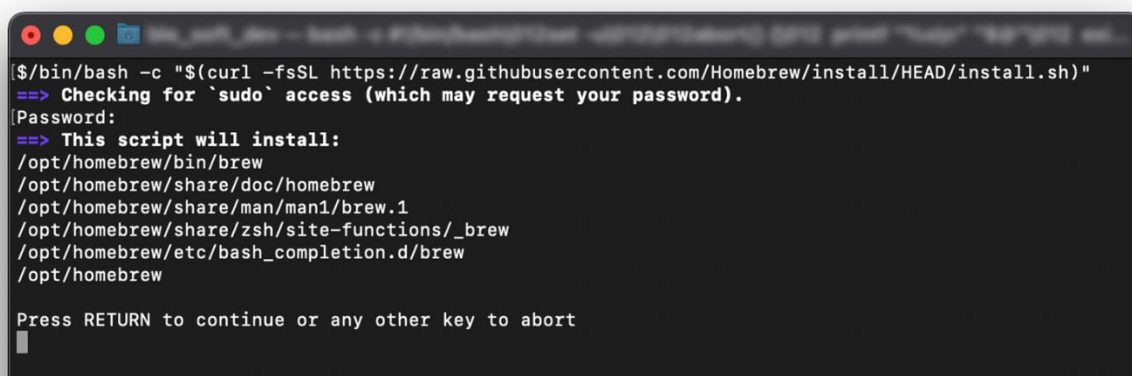


図 2.2 Homebrew のインストール(2)

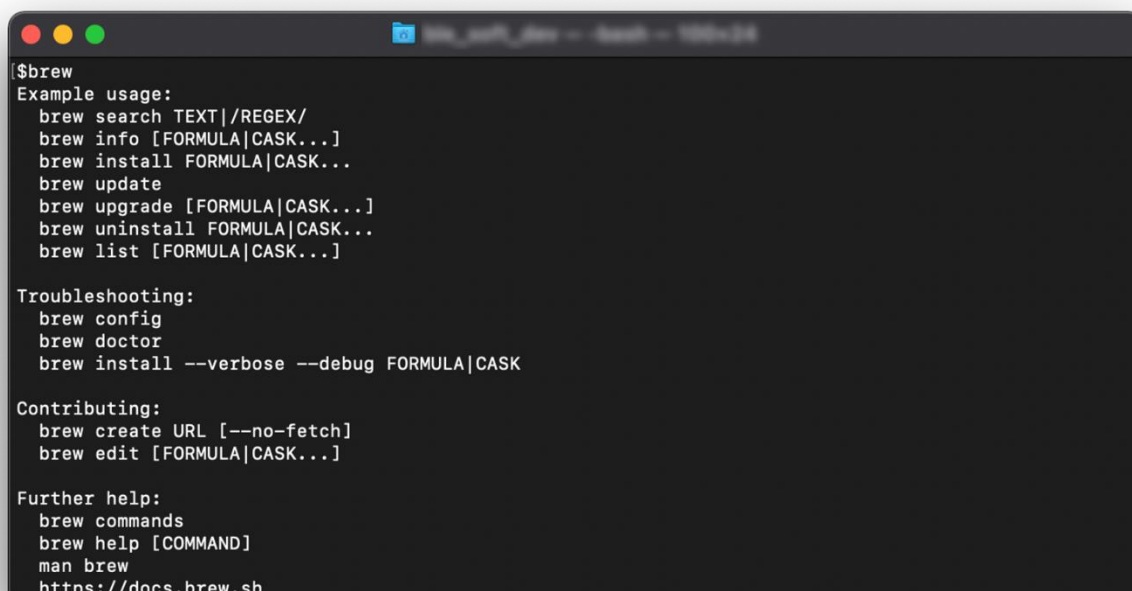
4. 手順2 でコピーしたコマンドをターミナルに貼り付けて実行します。

A terminal window showing the execution of a curl command to install Homebrew. The output indicates that it is checking for sudo access and listing the installation paths for various Homebrew components.

```
$/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
==> Checking for `sudo` access (which may request your password).
Password:
==> This script will install:
/opt/homebrew/bin/brew
/opt/homebrew/share/doc/homebrew
/opt/homebrew/share/man/man1/brew.1
/opt/homebrew/share/zsh/site-functions/_brew
/opt/homebrew/etc/bash_completion.d/brew
/opt/homebrew
Press RETURN to continue or any other key to abort
```

図 2.3 Homebrew のインストール(3)

5. Terminal で brew と入力して tips が表示されることを確認します。

A terminal window showing the output of the brew command. It displays example usage, troubleshooting options, contributing options, and further help information.

```
$/brew
Example usage:
brew search TEXT|REGEX/
brew info [FORMULA|CASK...]
brew install FORMULA|CASK...
brew update
brew upgrade [FORMULA|CASK...]
brew uninstall FORMULA|CASK...
brew list [FORMULA|CASK...]

Troubleshooting:
brew config
brew doctor
brew install --verbose --debug FORMULA|CASK

Contributing:
brew create URL [--no-fetch]
brew edit [FORMULA|CASK...]

Further help:
brew commands
brew help [COMMAND]
man brew
https://docs.brew.sh
```

図 2.4 Homebrew のインストール(4)



## 2.3 Ruby 最新版のインストール

Homebrew を用いて Ruby を最新に更新して、Xcode のライブラリを管理する CocoaPods をインストールします。

1. Finder を起動して Applications→Utilities→Terminal を起動します。
2. 次のコマンドを実行します。brew install ruby-build
3. 次のコマンドを実行します。brew install rbenv
4. PATH に Ruby の設定を追加するため、以下のコマンドを実行します。

```
cd
echo 'export PATH="$HOME/.rbenv/shims:$PATH"' >> .bash_profile
echo 'eval "$(rbenv init -)"' >> .bash_profile
source .bash_profile
```

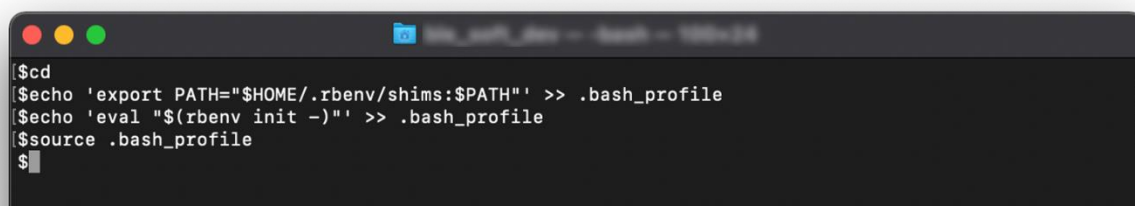


図 2.5 Ruby の設定

※上記ではシェルとして/bin/bash を使用しています。デフォルトシェルを変更する手順は以下の URL を参照してください。

<https://support.apple.com/guide/terminal/trml113/mac>

5. ruby -v を実行して ruby のバージョンが 2.2.2 以降になっていることを確認します。ruby のバージョンが 2.2.2 未満の場合、以下のコマンドで最新のものをインストールしターミナルを再起動します
- |                     |                       |
|---------------------|-----------------------|
| rbenv install -l    | (インストール可能なバージョン一覧の表示) |
| rbenv install x.x.x | (指定したバージョンのインストール)    |
| rbenv global x.x.x  | (指定したバージョンへの切り替え)     |

## 2.4 CocoaPods のインストール

CocoaPods は iOS アプリ向けサードパーティライブラリを管理するためのツールです。TryBT プロジェクトは CocoaPods を使用してライブラリを管理しています。下記の URL にアクセスし CocoaPods をインストールしてください。

<https://guides.cocoapods.org/using/getting-started.html>

## 2.5 TryBT プロジェクトの起動

TryBT プロジェクトを Xcode で起動します。

1. 本文書に添付された TryBT プロジェクトの zip ファイルを展開します。
2. 展開したフォルダを任意のフォルダに移動します。
3. ターミナルを起動して、TryBT プロジェクトを展開したフォルダに `cd` コマンドで移動します。
4. 次のコマンドを実行します。 `pod install`

TryBT プロジェクトに使用しているライブラリが自動インストールされます。

`pod install` コマンドの実行で下記のエラーが発生する場合は、Xcode メニューの [Xcode]→[Preferences...]を選択し、Location タブで Command Line Tools を設定してください。

```
xcode-select: error: tool 'xcodebuild' requires Xcode, but active developer directory '/Library/Developer/CommandLineTools' is a command line tools instance
```

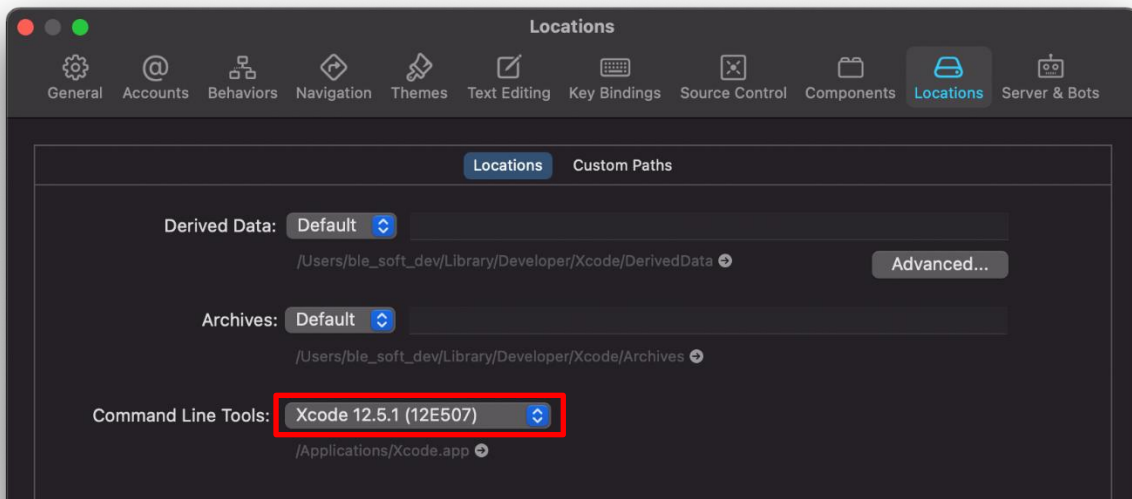


図 2.6 Command Line Tools の設定

5. 同一フォルダ内に TryBT.xcworkspace が作成されます。TryBT.xcworkspace をダブルクリックしてプロジェクトを起動します。

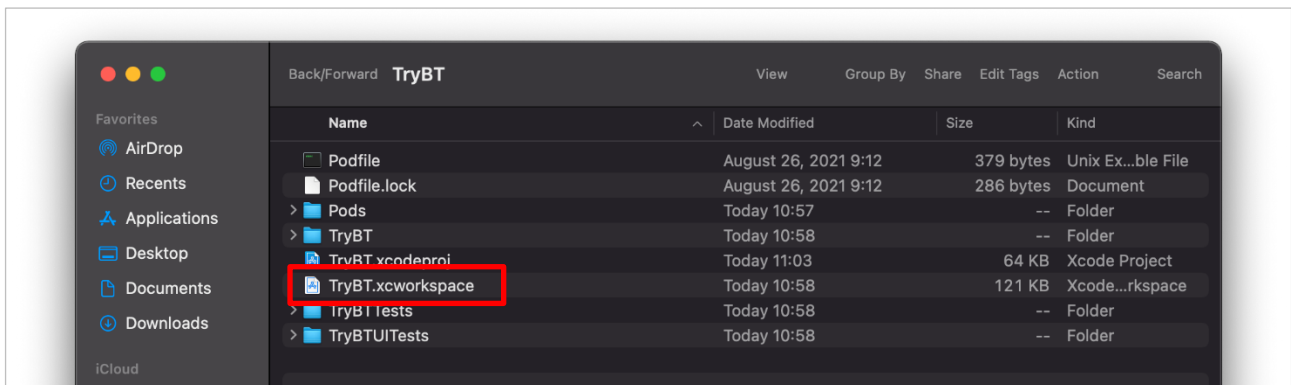


図 2.7 TryBT プロジェクトの起動

## 2.6 Apple Developer アカウントの設定

iOS 端末に TryBT をインストールするためには、加入済みの Apple Developer アカウントを XCode に設定する必要があります。

1. Xcode のメニューで[Xcode] →[Preferences]を選択します。
2. Accounts タブで左下の+ボタンをクリックして、Apple Developer アカウントを追加します。

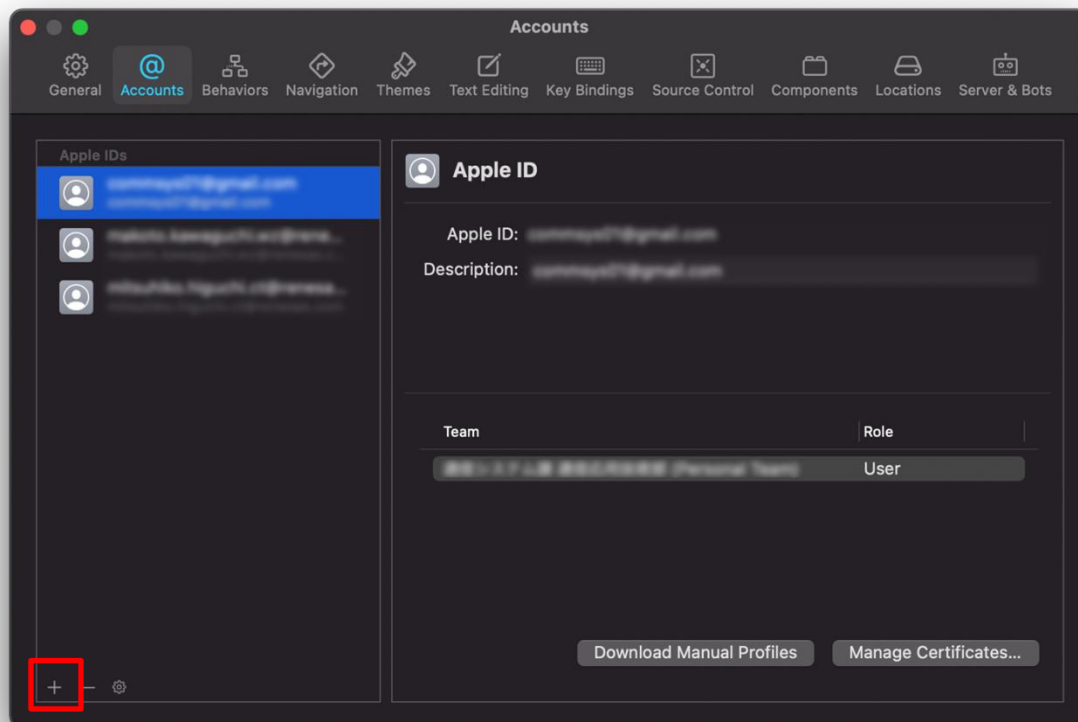


図 2.8 Apple Developer アカウントの設定(1)

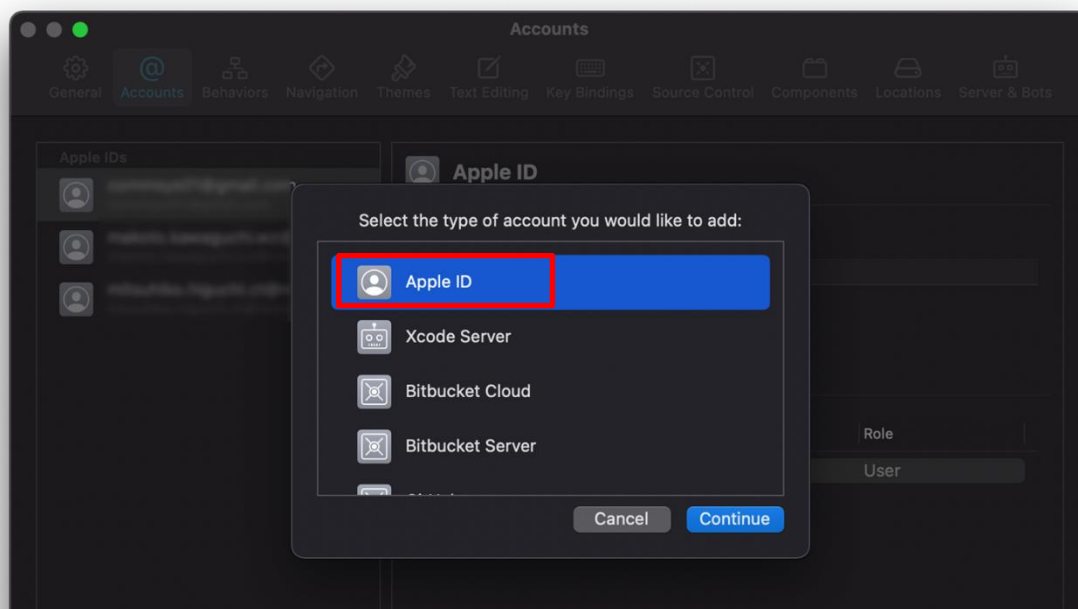


図 2.9 Apple Developer アカウントの設定(2)

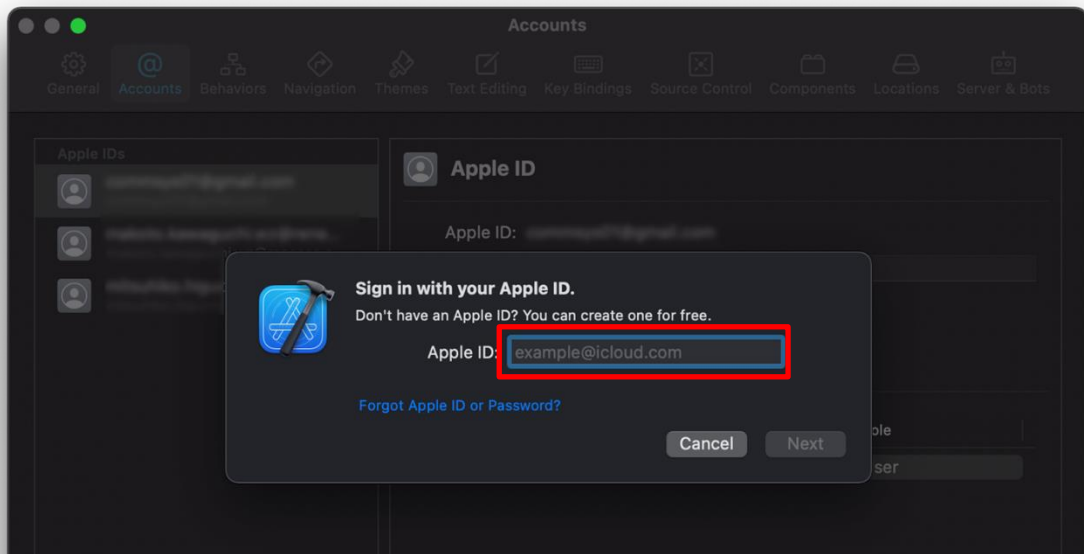


図 2.10 Apple Developer アカウントの設定(3)

## 2.7 Certificate Signing Request の作成

iOS 端末へのインストールに必要な Certificate Signing Request ファイルを作成します。

1. Finder を起動して[Applications]→[Utilities]に移動し、Keychain Access を起動します。

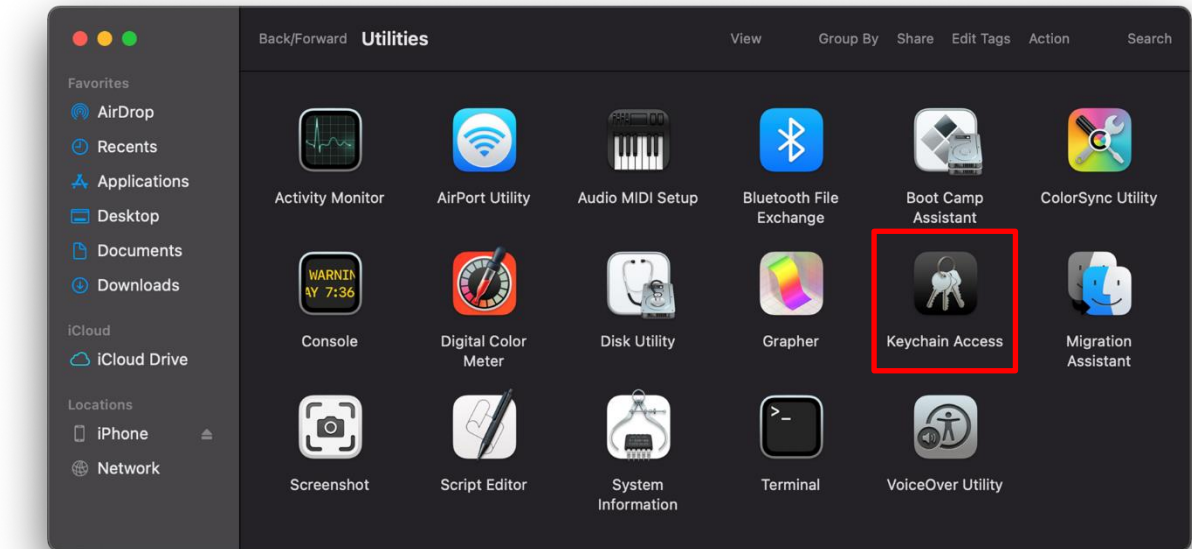


図 2.11 Certificate Signing Request の作成(1)

2. Keychain Access のメニューから[Keychain Access]→[Certificate Assistant]→[Request a Certificate From a Certificate Authority]を選択します
3. User Email Address フィールドに Apple Developer Program で使用している Email アドレス、Common Name フィールドに名前を記載します。"Save to disk"を選択します。Continue ボタンをクリックして Certificate Signing Request を保存します。

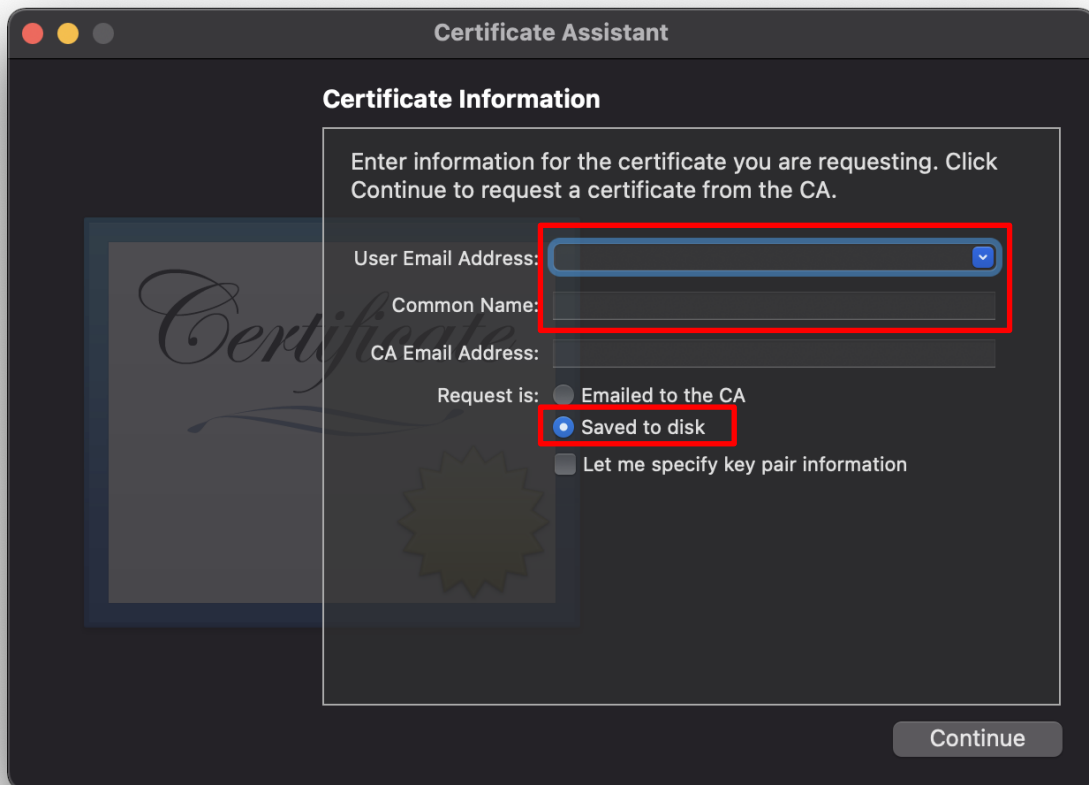


図 2.12 Certificate Signing Request の作成(2)

## 2.8 Developer Certificate の作成とインストール

iOS 端末に TryBT をインストールするために必要な Developer 用の Certificate を作成してインストールします。

1. Apple Developer にサインインして、左メニューの"Certificate ,IDs &Profiles"を選択します。

Apple Developer

<https://developer.apple.com/jp/>

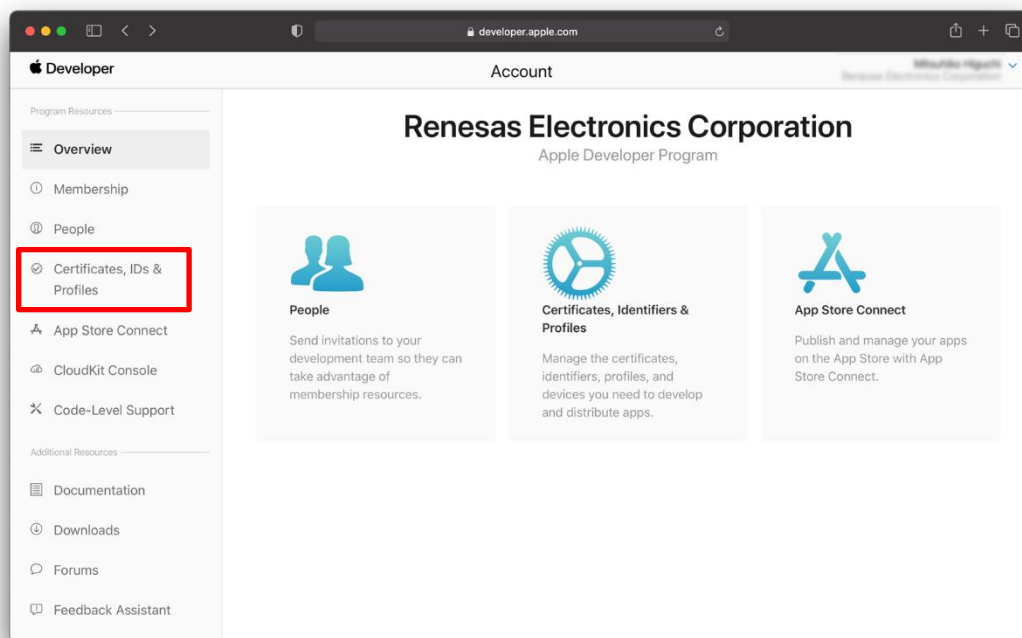


図 2.13 Developer Certificate の作成(1)

2. 左メニューの[Certificates]を選択して、Certificates 横の[+]ボタンを選択します。

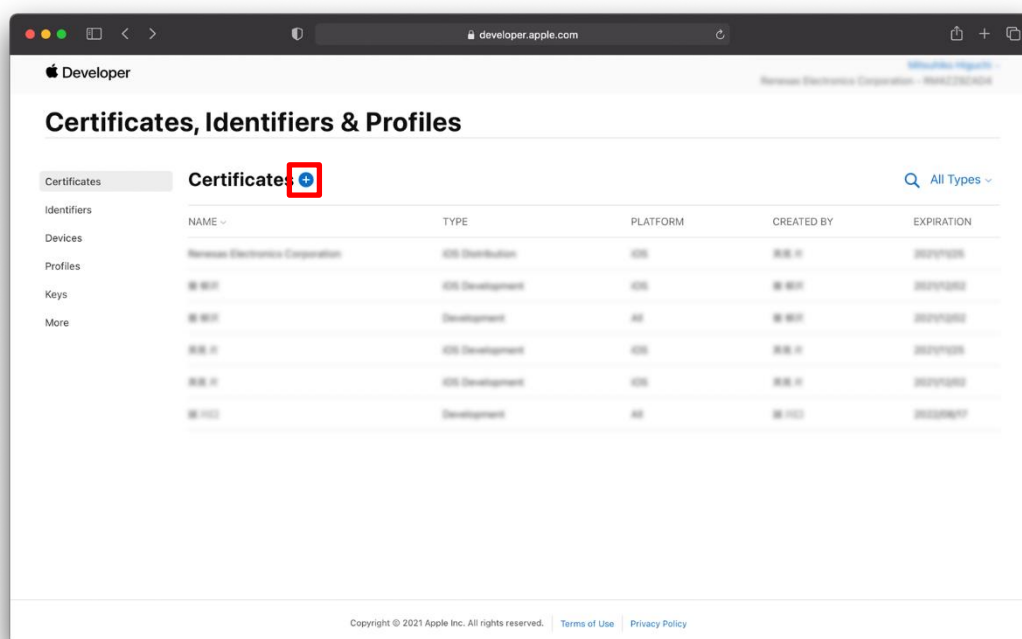


図 2.14 Developer Certificate の作成(2)

3. "Apple Development"を選択して Continue ボタンをクリックします。

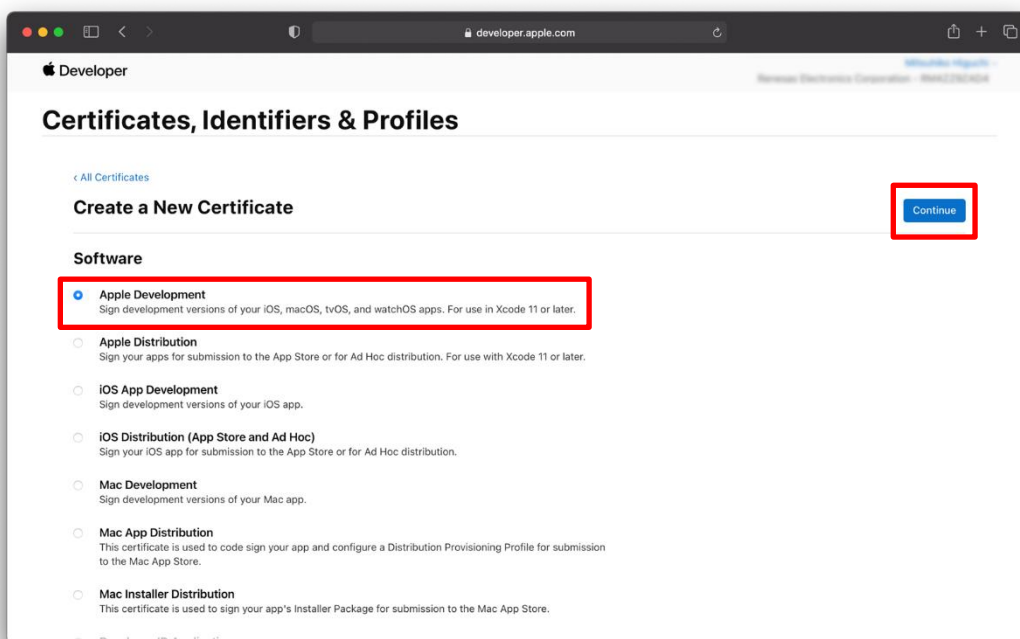


図 2.15 Developer Certificate の作成(3)

4. Choose File で作成済みの Certificate Signing Request を選択して Continue ボタンをクリックします。

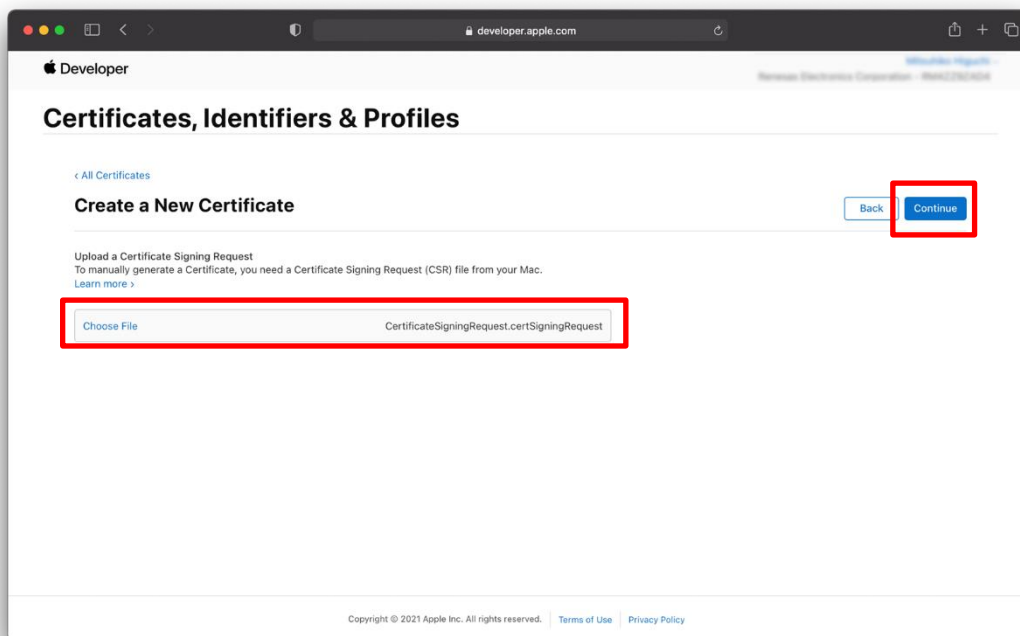


図 2.16 Developer Certificate の作成(4)

5. Certificate が作成されました。右上の Download ボタンをクリックして Certificate をダウンロードします。ダウンロードした Certificate を Finder 上でダブルクリックしてインストールします。

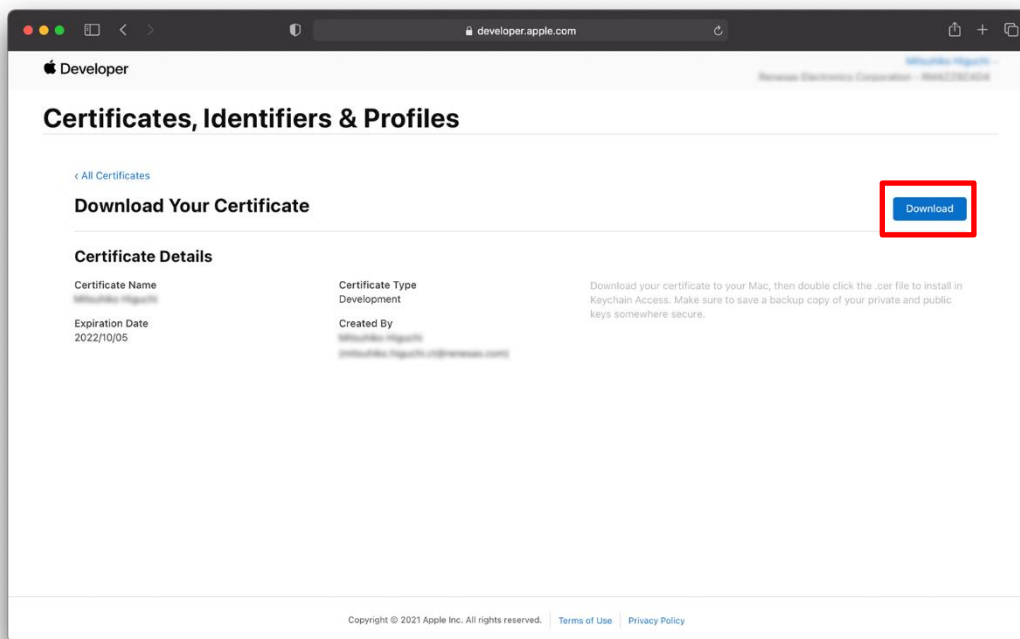


図 2.17 Developer Certificate の作成(5)



## 2.9 TryBT 用 App ID の作成

TryBT の App ID を作成します。App ID は iOS アプリケーション全体で一意的な値です。

1. 左メニューの[Identifiers]を選択して、Identifiers 横の青い[+]ボタンを選択します。

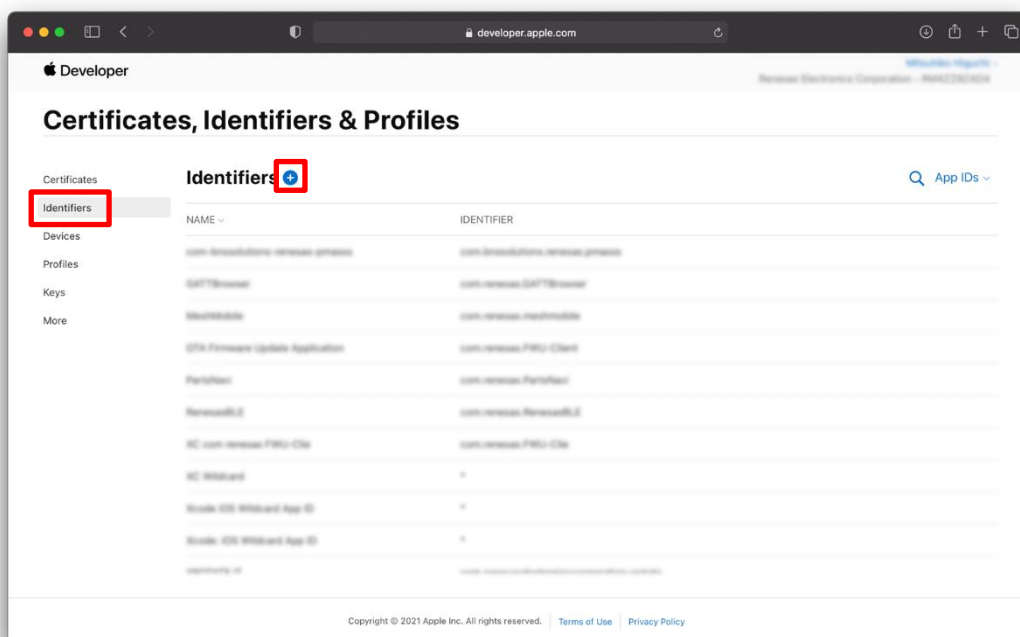


図 2.18 TryBT 用 App ID の作成(1)

2. "App IDs"を選択して Continue ボタンをクリックします。

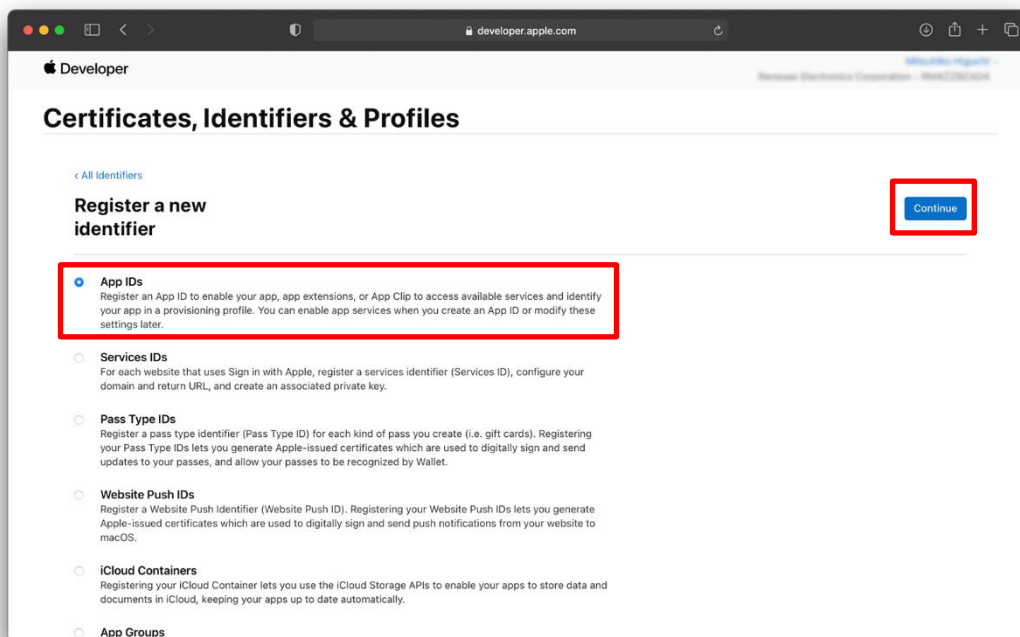


図 2.19 TryBT 用 App ID の作成(2)

3. "App"を選択して Continue ボタンをクリックします。

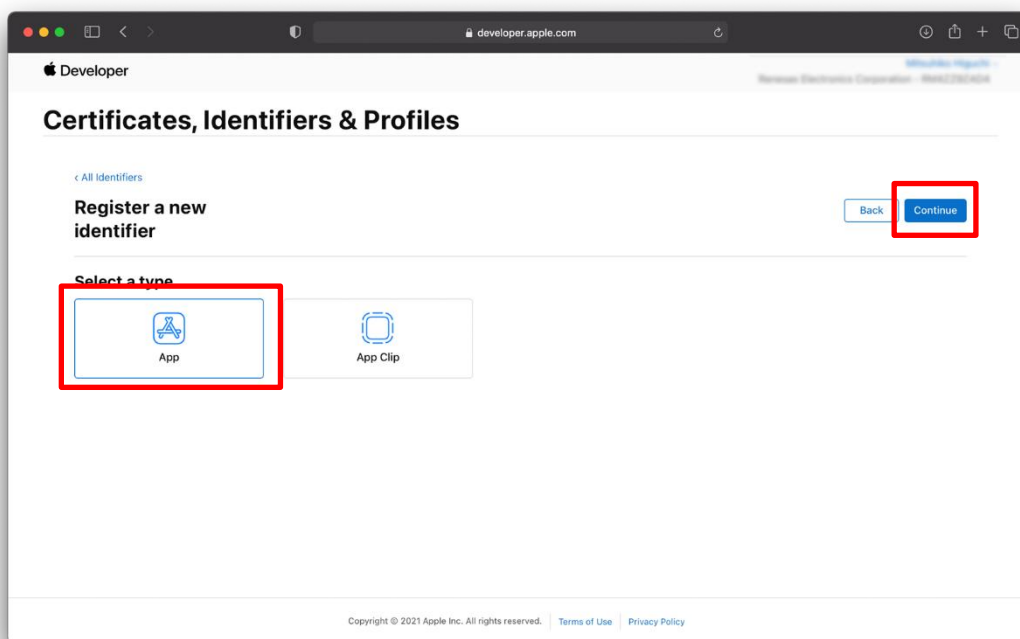


図 2.20 TryBT 用 App ID の作成(3)

4. App ID を作成します。Description に"TryBT Development"を入力します。App ID Prefix はそのままにします。Bundle ID は会社ドメインの逆引きに続けて".TryBT"を入力します。

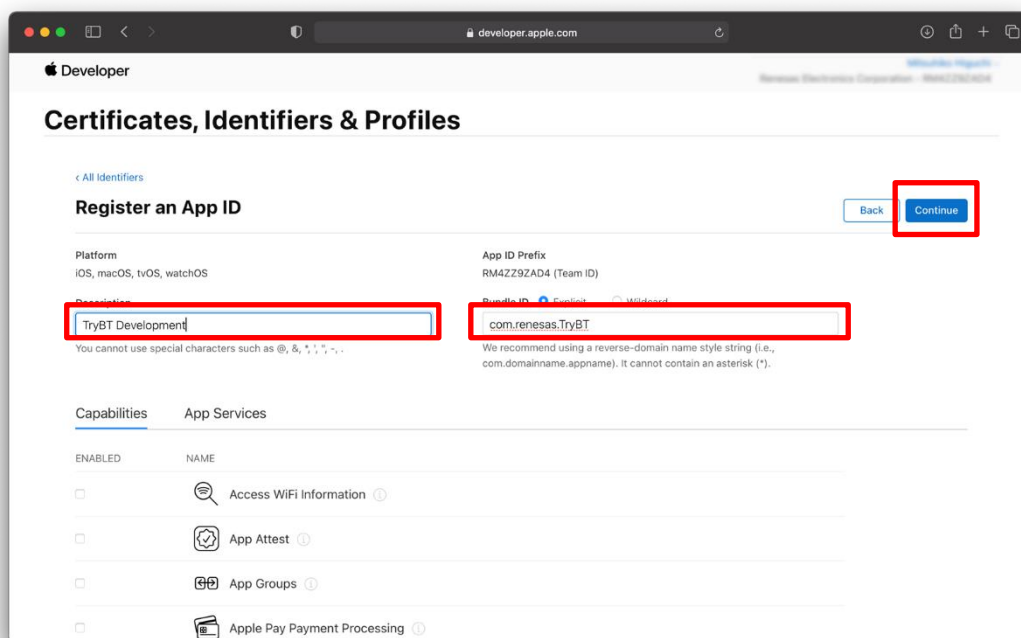


図 2.21 TryBT 用 App ID の作成(4)

5. 画面をスクロールして Push Notification をチェックし、Continue ボタンをクリックします。続けて Register ボタンをクリックします。

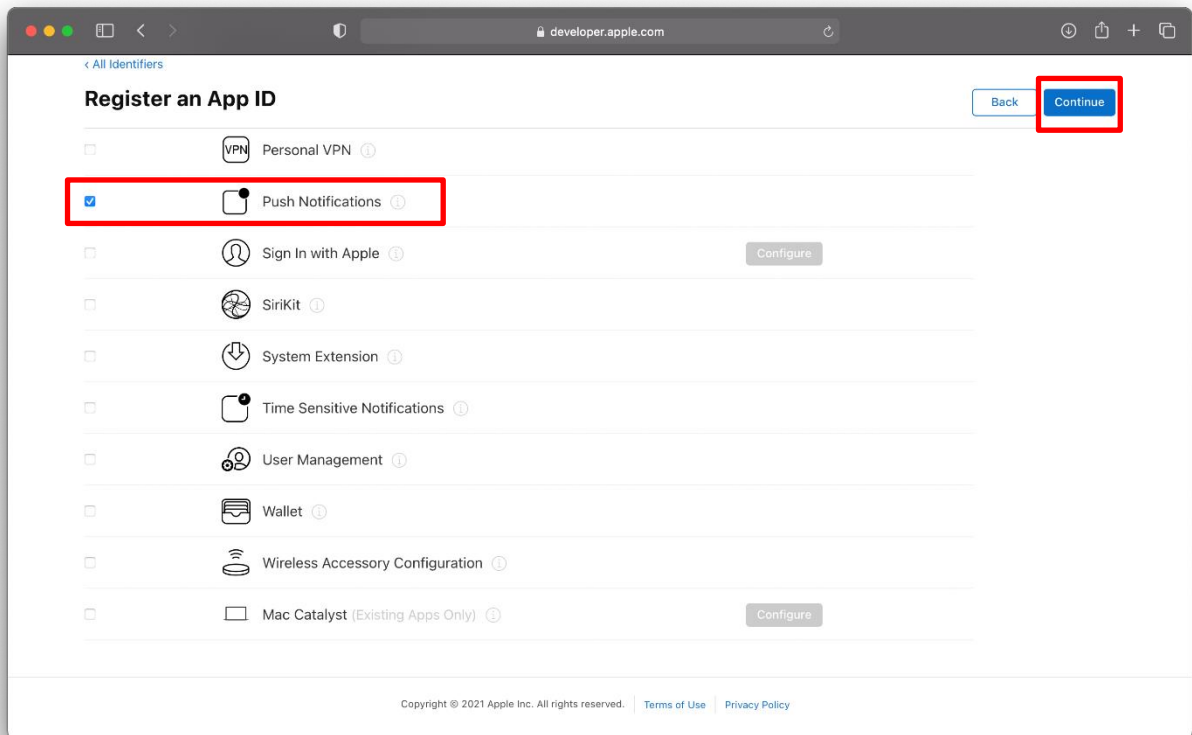


図 2.22 TryBT 用 App ID の作成(5)

## 2.10 iOS 端末の UDID 確認

TryBT をインストールする iOS 端末の UDID を確認します。

1. iOS 端末を Mac と有線接続します。
2. Xcode のメニューで[Window]→[Devices and Simulators]を選択します。
3. Devices で iOS 端末を選択します。Identifier フィールドに表示される値が iOS 端末の UDID です。

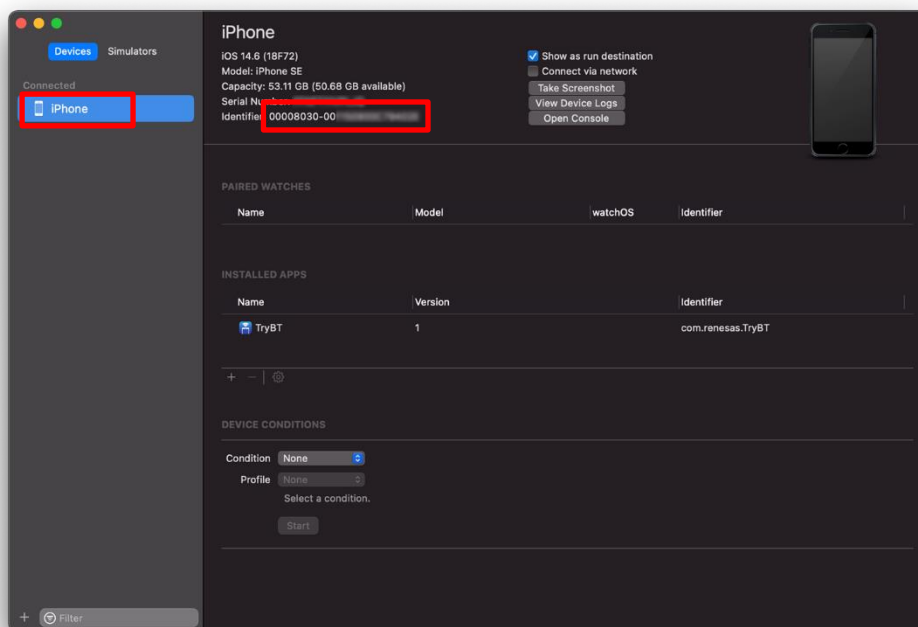


図 2.23 iOS 端末の UDID 確認



## 2.12 TryBT 用 Profile の作成

Profile を作成します。Profile はアプリの AppID や Device などを一元管理するファイルです。

1. 左メニューで Profiles を選択し、Profiles 横の青い[+]ボタンをクリックします。

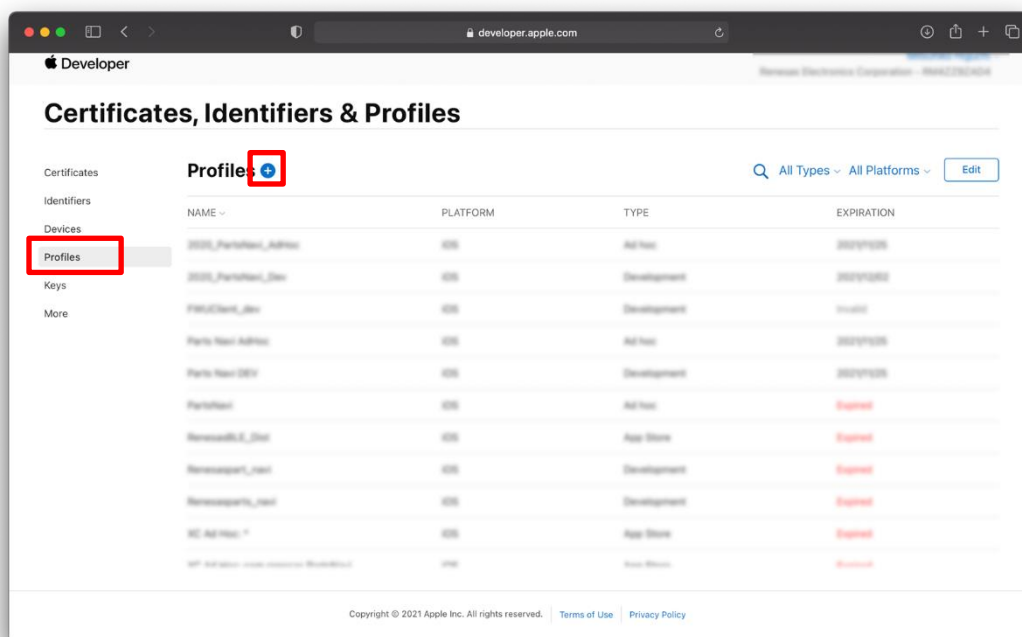


図 2.26 TryBT 用 Profile の作成(1)

2. "iOS App Development"を選択して Continue ボタンをクリックします。

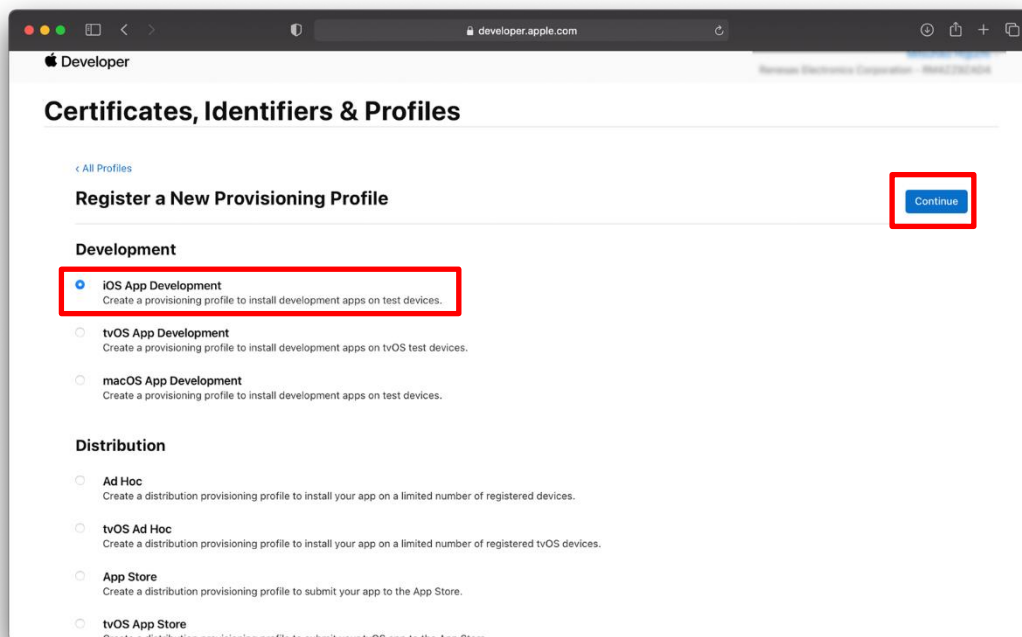


図 2.27 TryBT 用 Profile の作成(2)

3. TryBT の App ID を指定して Continue ボタンをクリックします。

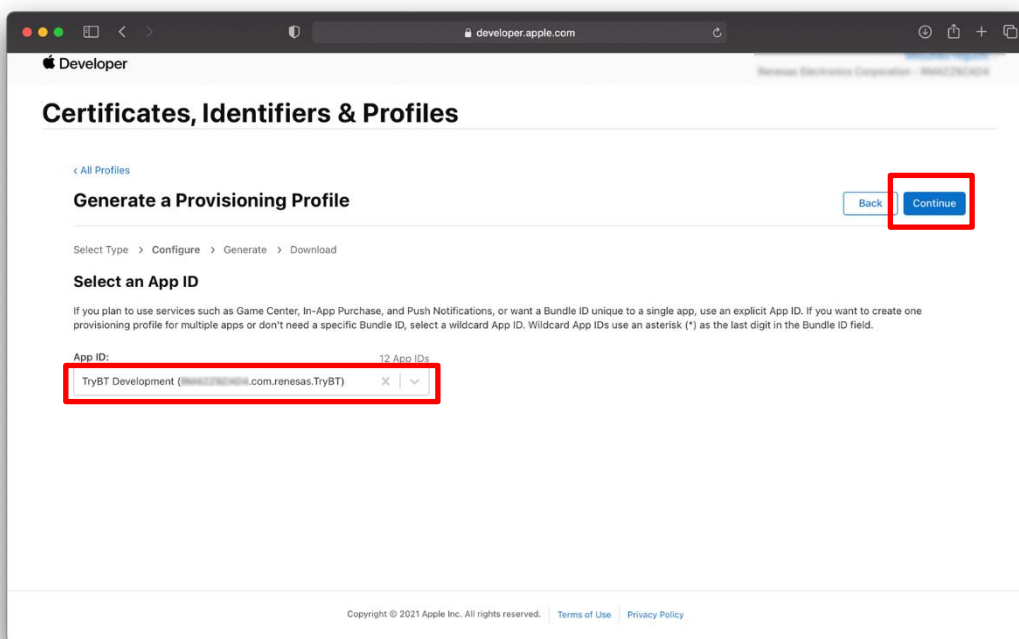


図 2.28 TryBT 用 Profile の作成(3)

4. 作成した Certificate を指定します。

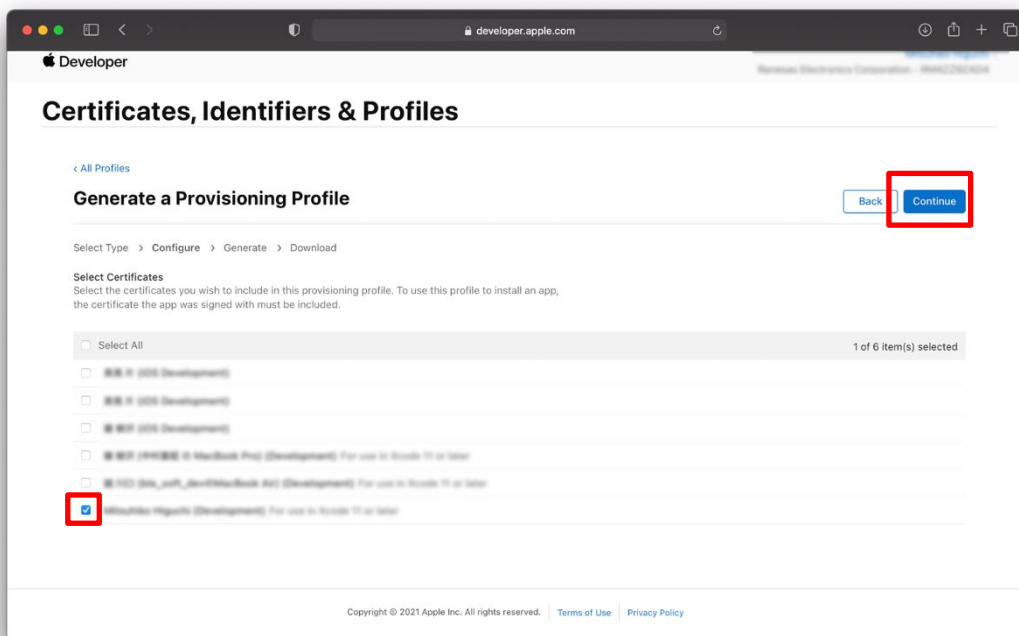


図 2.29 TryBT 用 Profile の作成(4)

5. iOS 端末を選択して Continue ボタンをクリックします。

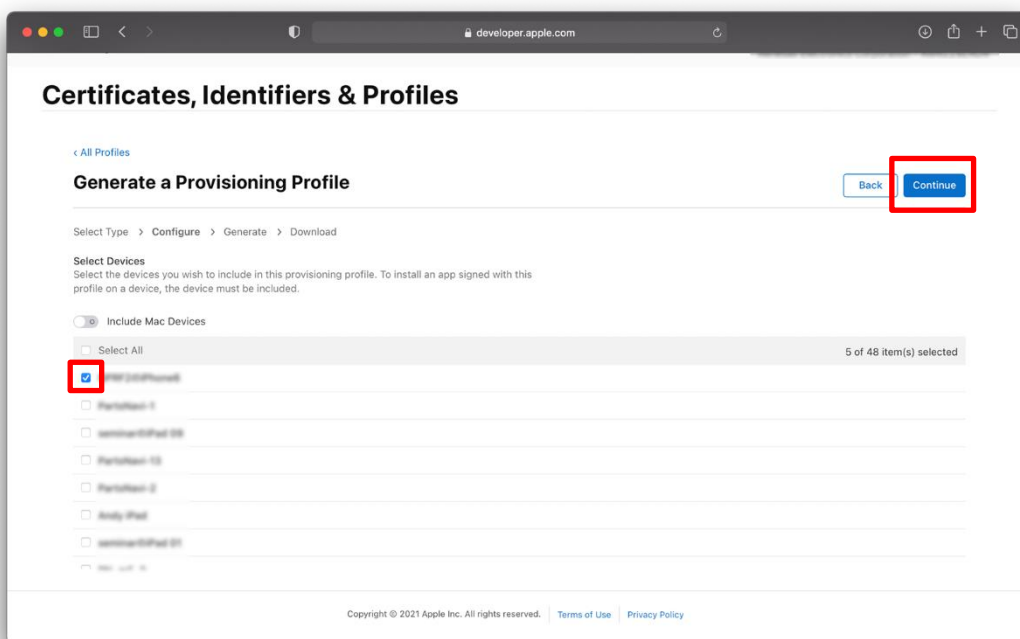


図 2.30 TryBT 用 Profile の作成(5)

6. Profile を識別するための名称を入力します。ここでは"TryBT Profile"と入力しています。入力後、Generate ボタンをクリックします。

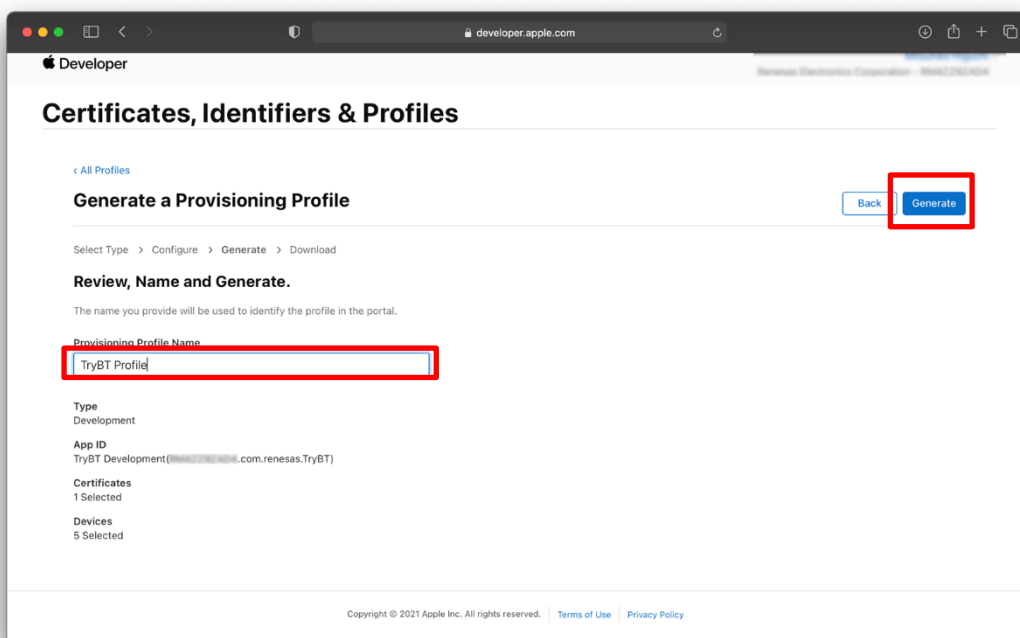


図 2.31 TryBT 用 Profile の作成(6)



7. Download ボタンをクリックして Profile をダウンロードします。

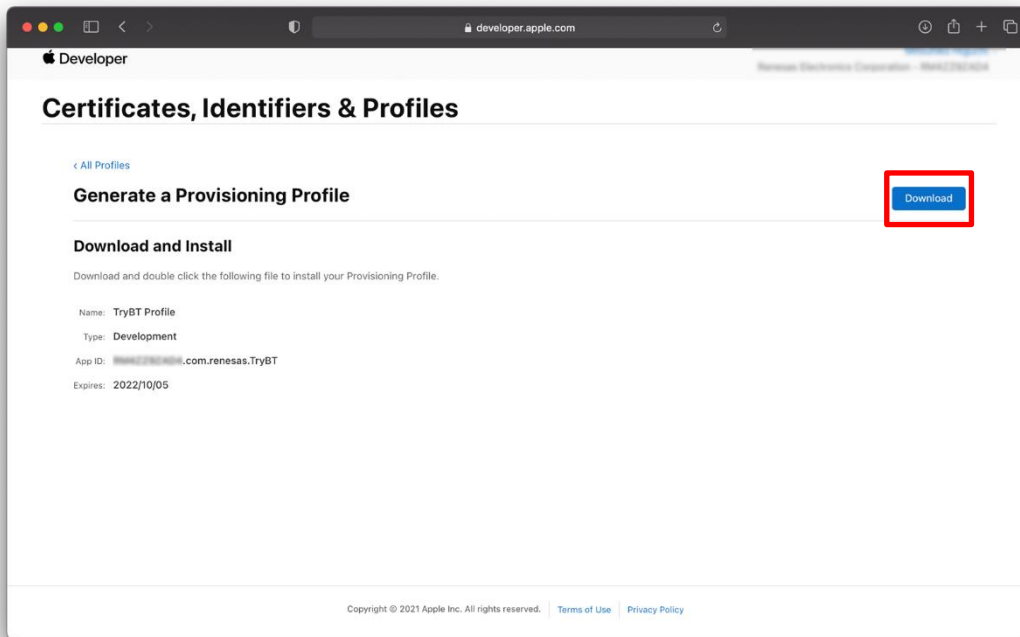


図 2.32 TryBT 用 Profile の作成(7)

### 2.13 TryBT プロジェクトへの Profile の関連付け

ダウンロードした Profile を TryBT プロジェクトに関連付けます。

1. Xcode で TryBT プロジェクトを選択します。
2. [TARGETS]で TryBT を選択し、"Signing & Capability"タブを選択します。
3. "Automatically manage signing"のチェックを外します。
4. Bundle Identifier フィールドに Apple Developer で入力した App ID を入力します。
5. Provisioning Profile フィールドをクリックして"Import Profile"を選択し、2.12 節でダウンロードした Profile を指定します。

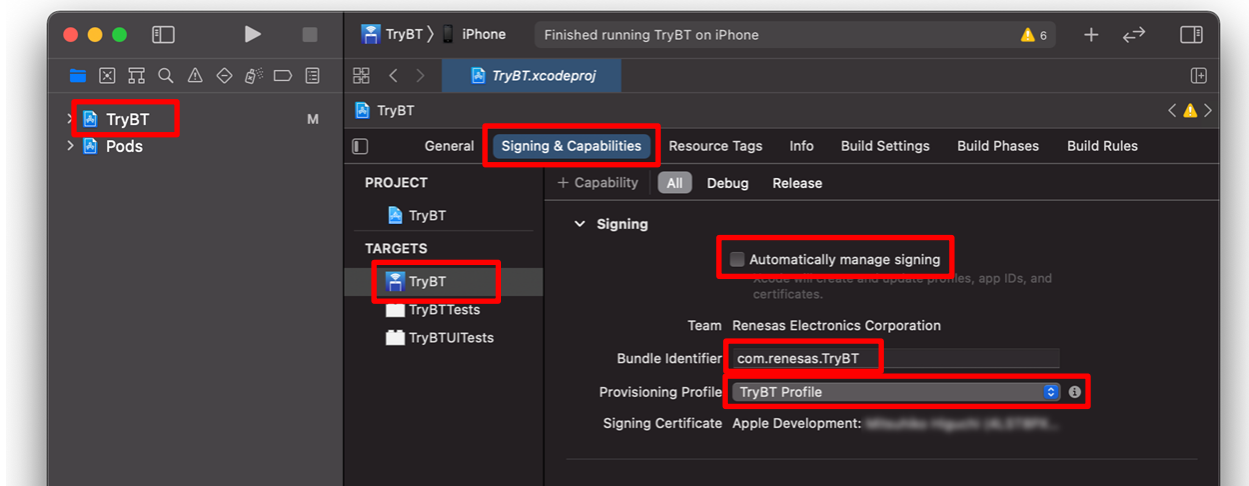


図 2.33 TryBT プロジェクトへの Profile の関連付け

## 2.14 TryBT のコンパイルとインストール

Mac と iOS 端末を有線接続して、iOS 端末に TryBT をインストールします。

1. Xcode を起動して TryBT プロジェクトを開きます。
2. 使用する端末を Mac に接続します。
3. 正しく実機が接続されると Xcode 上部に端末名が表示されます。

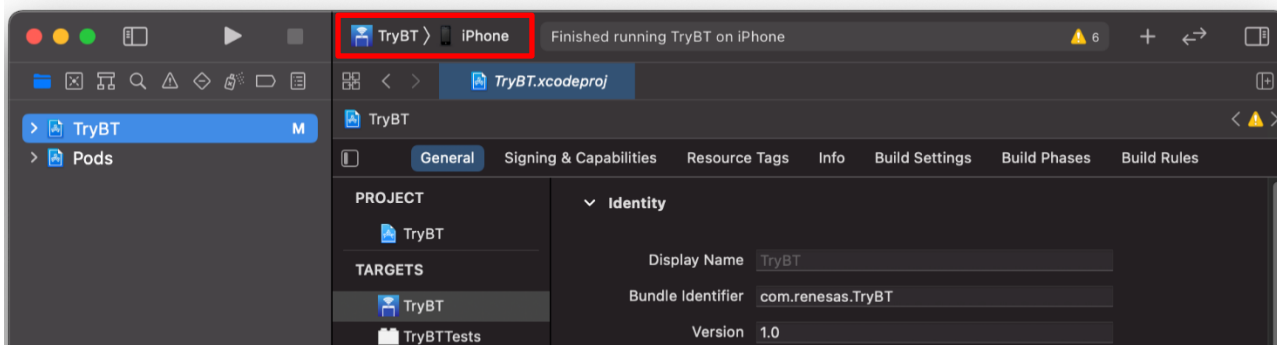


図 2.34 iOS 端末の接続(1)

※iOS 端末の接続後、Xcode 上部の表示が更新されない場合は、Xcode 上部の端末表示部分をクリックして、ドロップダウンリストの[iOS Device]から iOS 端末を選択してください。

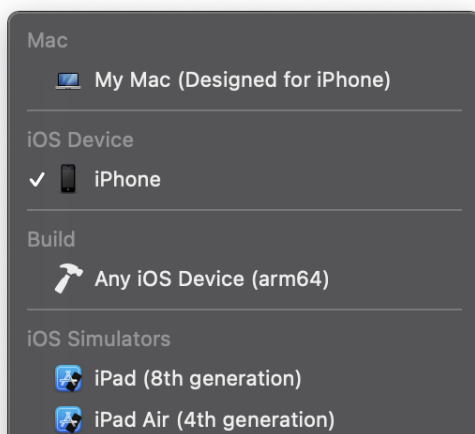


図 2.35 iOS 端末の接続(2)

4. Xcode 左上の再生ボタンをクリックすると、TryBT のコンパイルとインストールが開始されます。

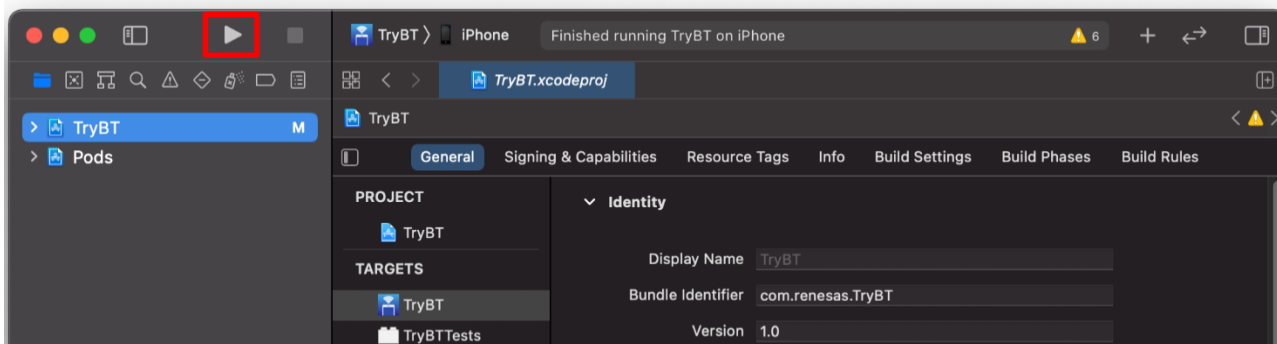


図 2.36 TryBT のコンパイルとインストール

### 3. 評価ボードへのファームウェア書き込み

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

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

1. 以下の URL にアクセスしてください。My Renesas アカウントでログイン後、免責事項に同意することで zip ファイルをダウンロードできます。

<https://www.renesas.com/document/scd/rx23w-group-target-board-rx23w-quick-start-guide-sample-code>

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

./mot/ble\_demo\_tbrx23w\_profile\_server\_preinstall\_20191009.mot

次ページでは、ビルド済みのファームウェアを Target Board for RX23W に書き込む手順を示します。また、ファームウェアの書き込みには以下のツールを使用します。

Renesas Flash Programmer (Programming GUI)

<https://www.renesas.com/software-tool/renesas-flash-programmer-programming-gui>

3. ファームウェアの書き込み時は Target Board for RX23W の ESW1 1-2 を ON に変更して、PC と ECN1 コネクタを USB ケーブルで接続します。

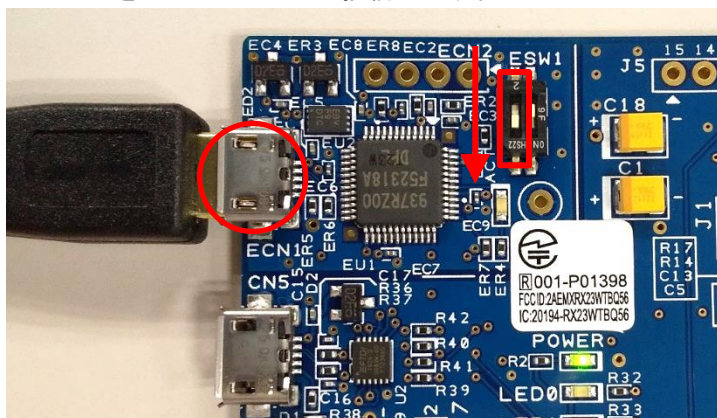


図 3.1 ファームウェア書き込み時の Target Board for RX23W の設定

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

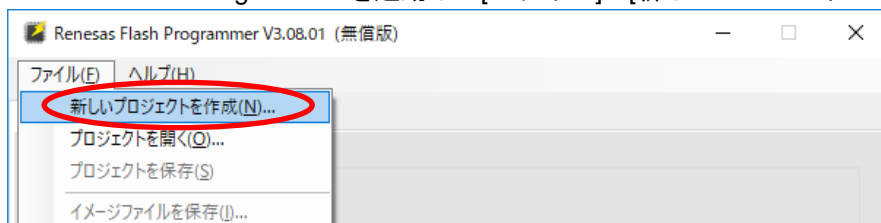


図 3.2 Target Board for RX23W へのファームウェア書き込み(1)

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

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

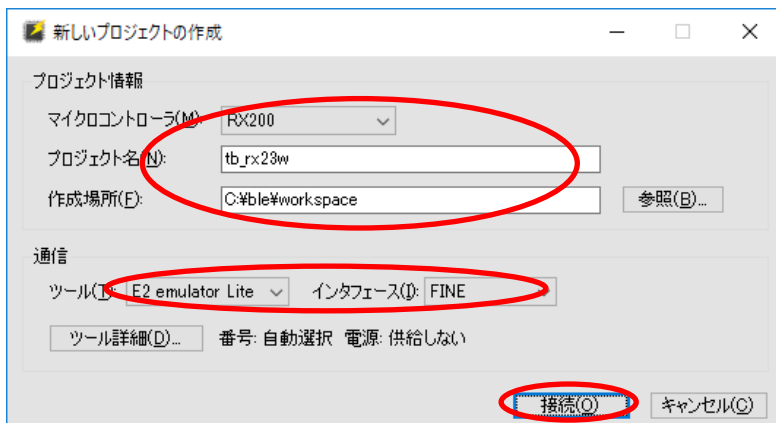


図 3.3 Target Board for RX23W へのファームウェア書き込み(2)



## 4. TryBT の基本操作

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

### 4.1 デバイス一覧画面

アプリを起動するとデバイス一覧画面が表示されます。接続可能なデバイスと接続状況を表示します。

Target Board for RX23W はデバイス一覧画面で"RBLE-DEV"と表示されます。"RBLE-DEV"をタップすると、Target Board for RX23W との接続を確立し、接続デバイス詳細画面を表示します。

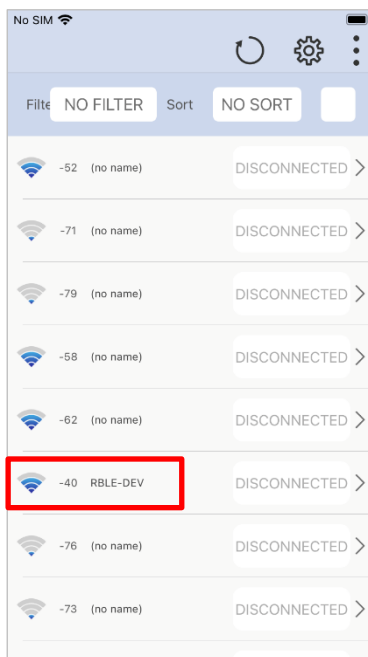


図 4.1 デバイス一覧画面(1)

"Filter"を選択するとデバイスをフィルタリングできます。

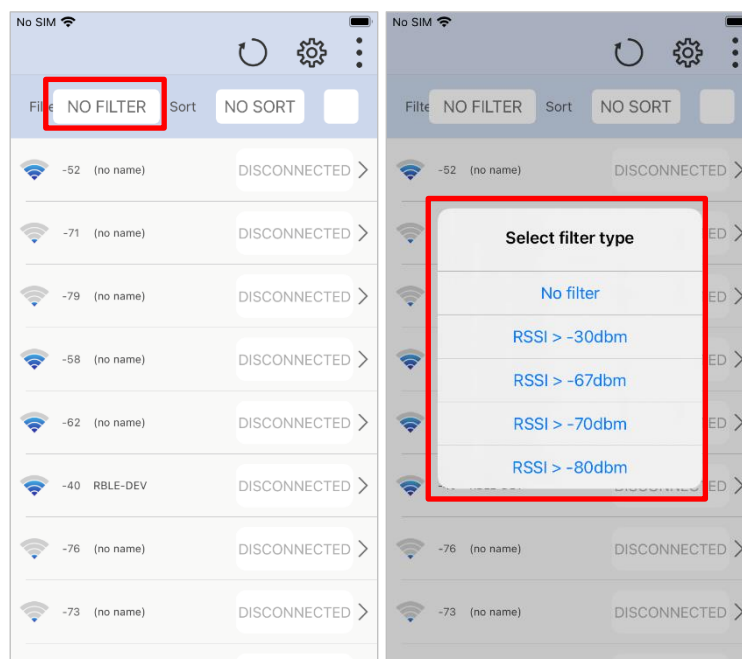


図 4.2 デバイス一覧画面(2)

"Sort"を選択するとデバイスのソート順を指定できます。

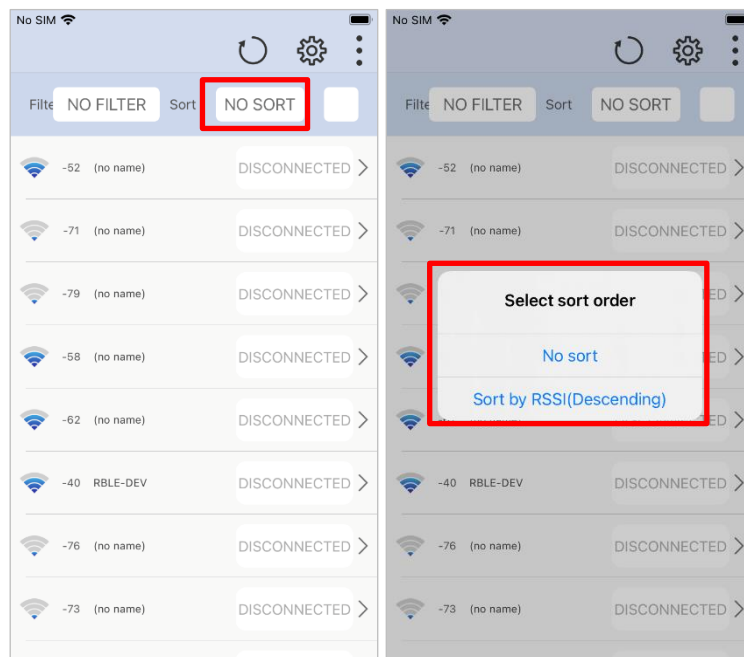


図 4.3 デバイス一覧画面(3)

リロードボタンをタップするとデバイス一覧を再読み込みできます。

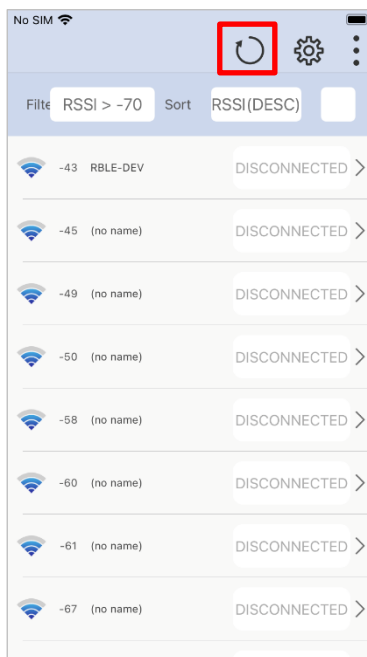


図 4.4 デバイス一覧画面(4)



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

"Register UUID name"を選択すると、GATT プロファイルのサービス名とその UUID を登録できます。設定されたサービス名は接続デバイス詳細画面の詳細情報で表示されます。なお本設定で登録できる UUID は 1 つのみです。表示例は図 4.11 を参照してください。

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

00001800-0000-1000-8000-00805f9b34fb: GAP Service

00001801-0000-1000-8000-00805f9b34fb: GATT Service

58831926-5f05-4267-ab01-b4968e8efce0: LED Switch Service

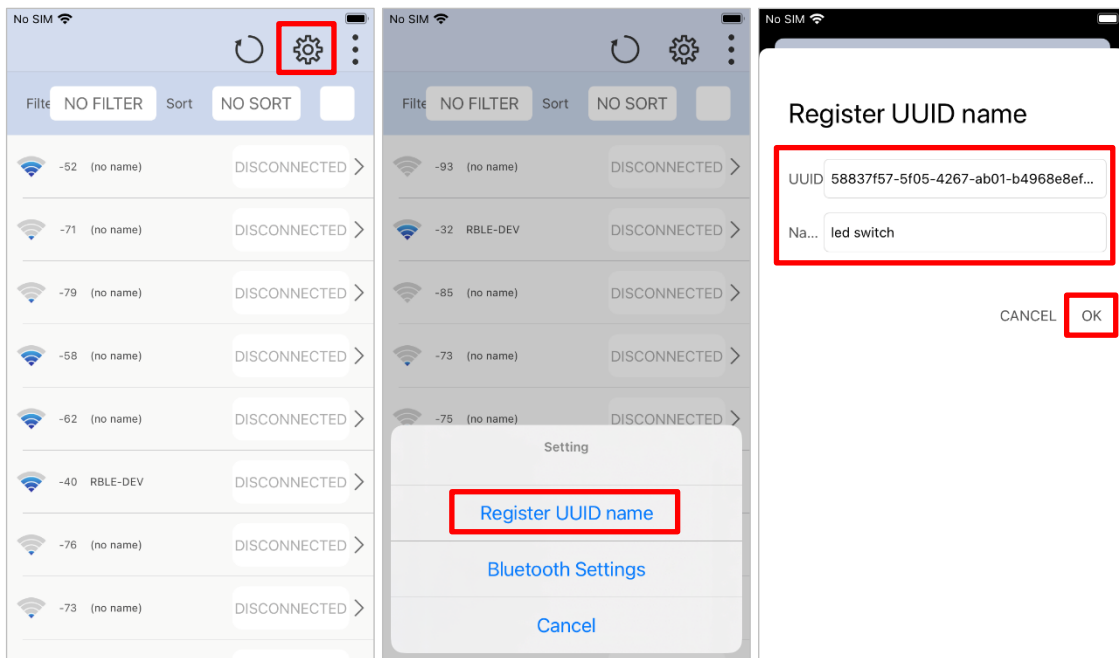


図 4.5 デバイス一覧画面(5)

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

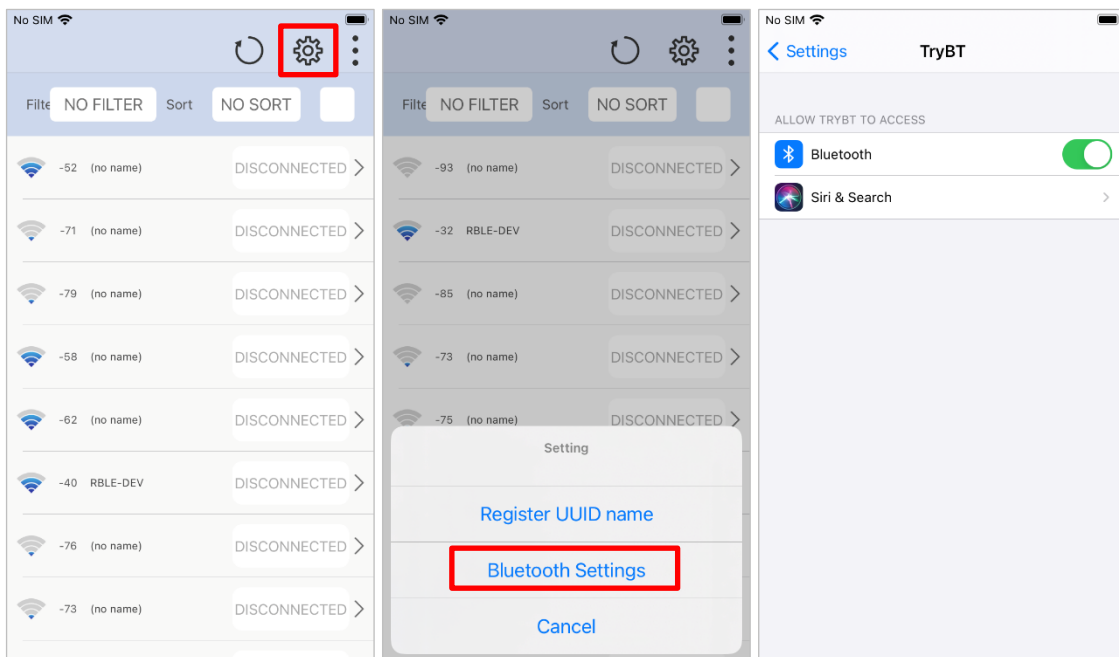


図 4.6 デバイス一覧画面(6)

## 4.2 接続デバイス詳細画面

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

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

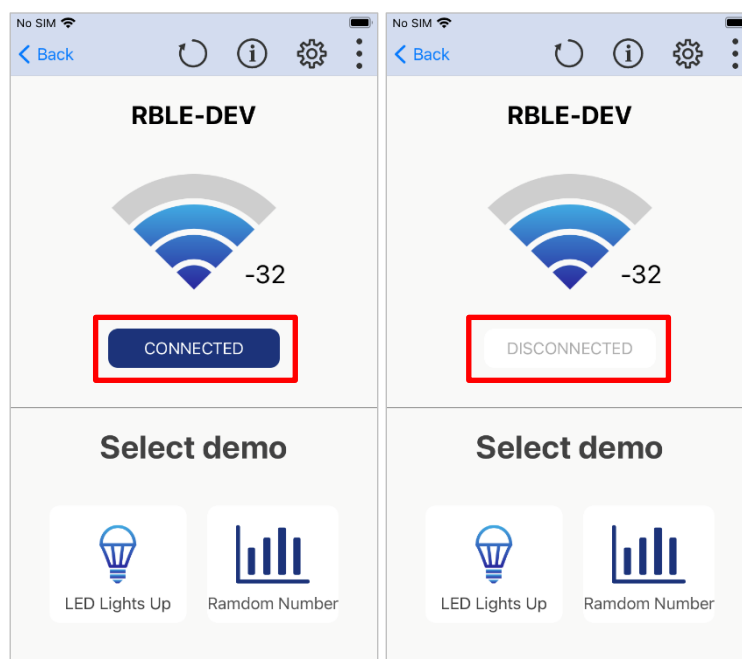


図 4.7 接続デバイス詳細画面(1)

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

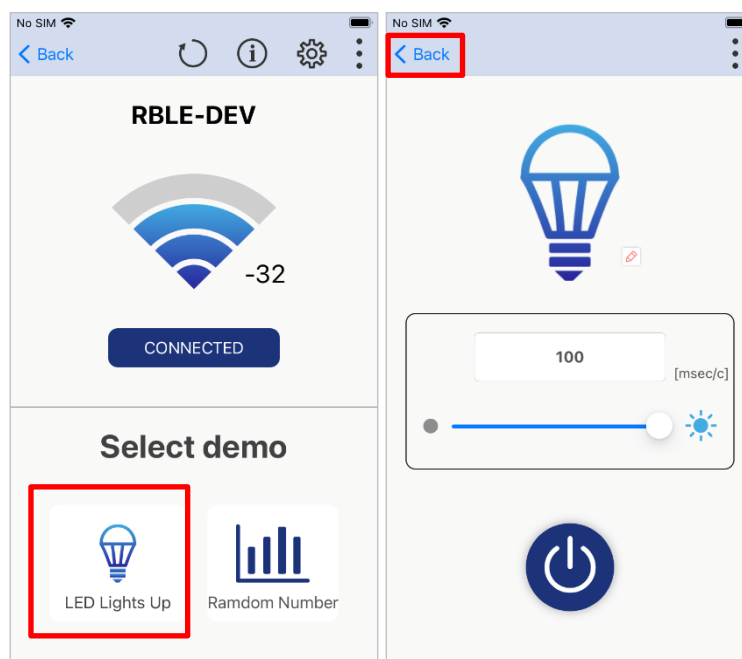


図 4.8 接続デバイス詳細画面(2)

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

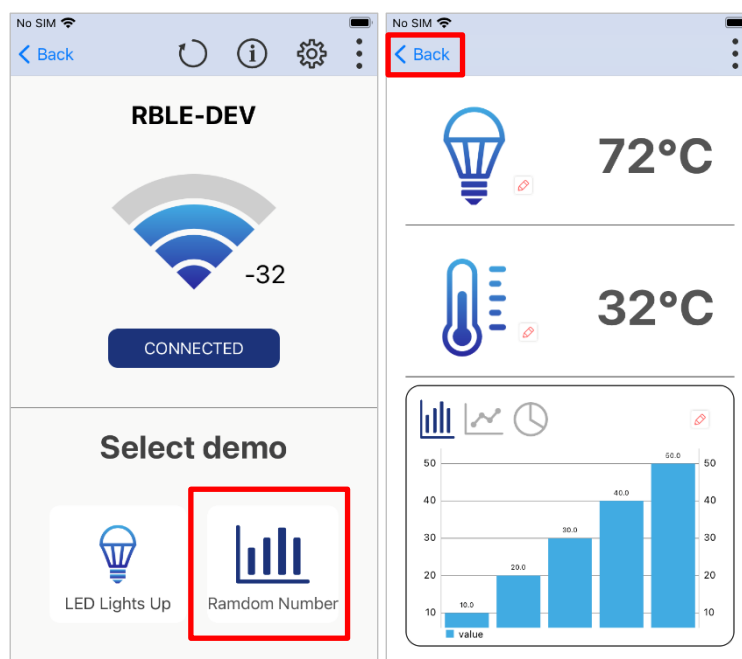


図 4.9 接続デバイス詳細画面(3)

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

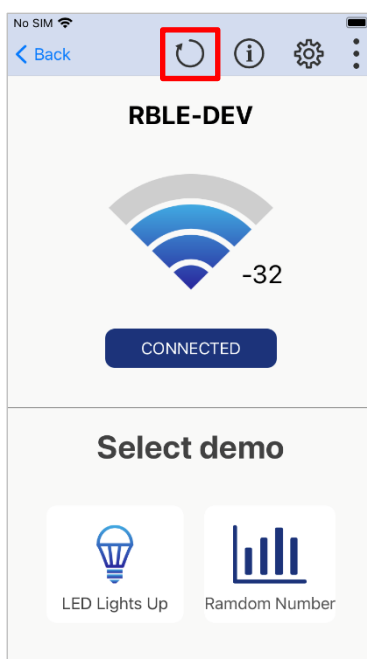


図 4.10 接続デバイス詳細画面(4)

Info ボタンを選択すると評価ボードの詳細情報を表示します。

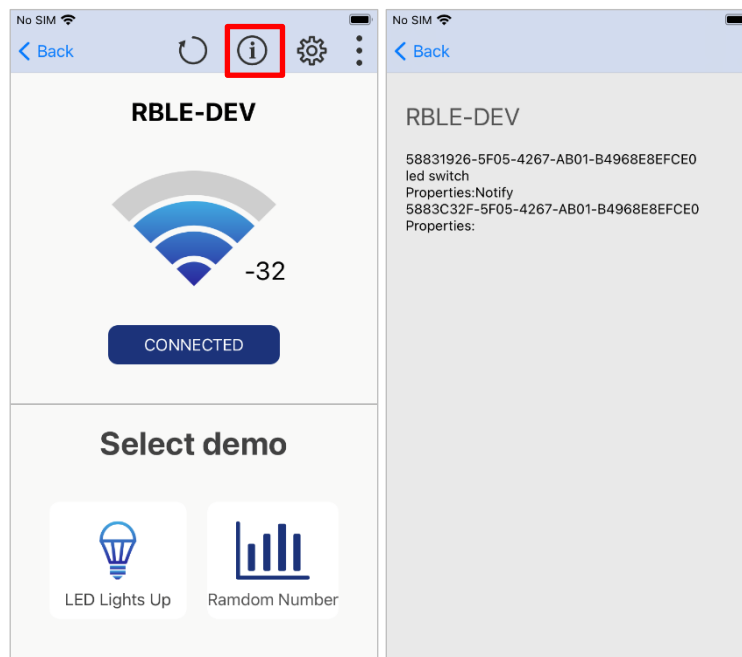


図 4.11 接続デバイス詳細画面(5)

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

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

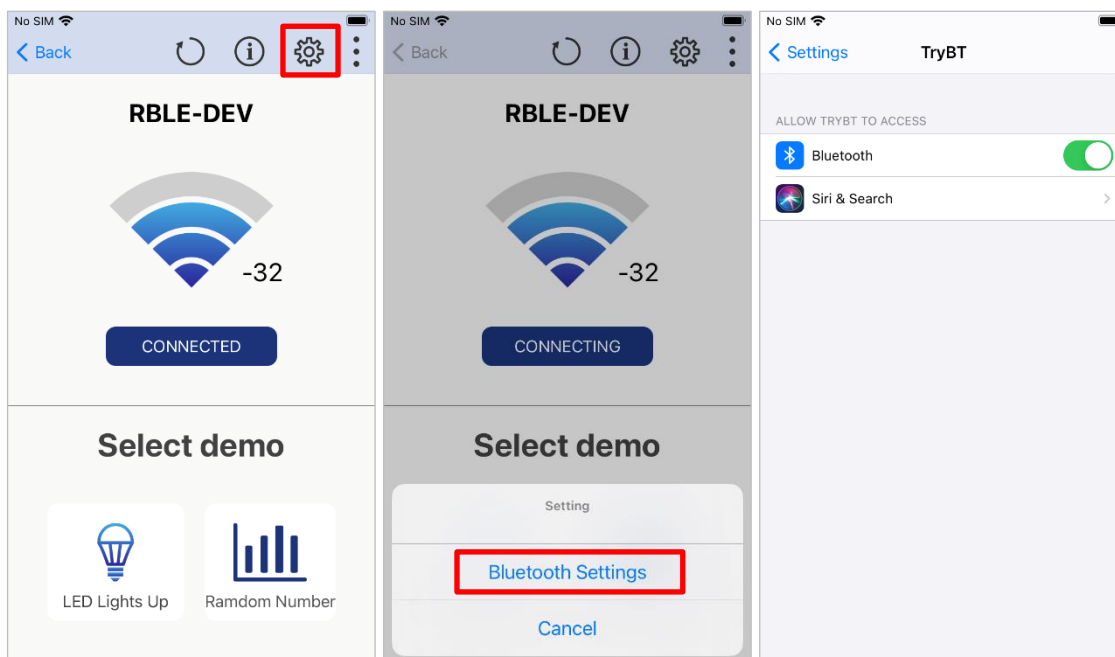


図 4.12 接続デバイス詳細画面(6)

### 4.3 ライトデモ画面

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

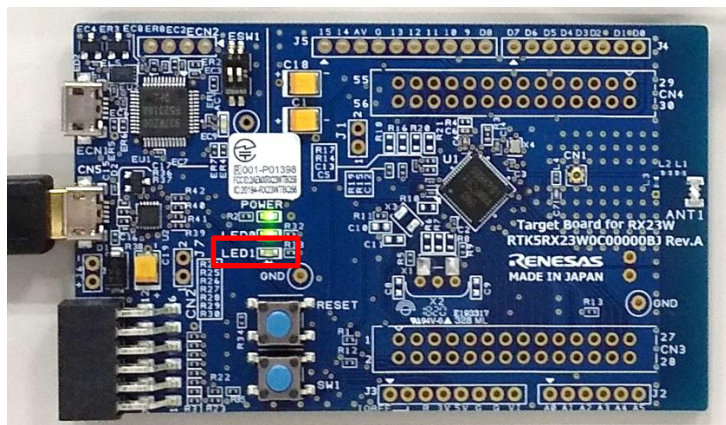


図 4.13 Target Board for RX23W のユーザ LED

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

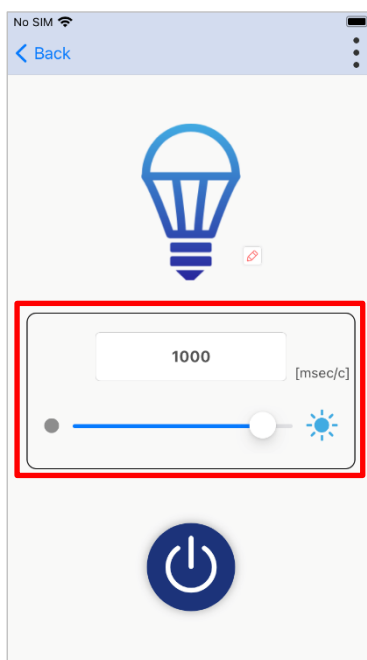


図 4.14 ライトデモ画面(1)

パワーボタンでLED点滅の停止と再開を操作します。

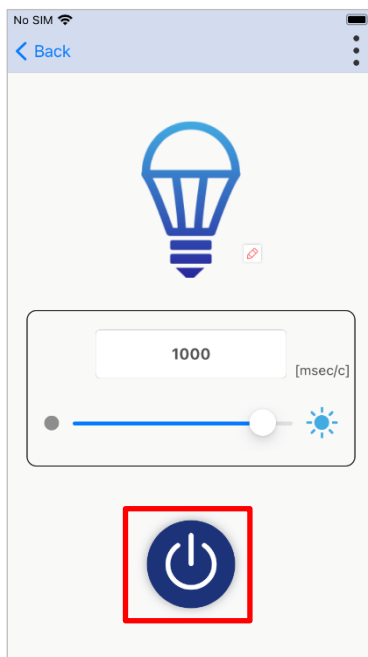


図 4.15 ライトデモ画面(2)

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

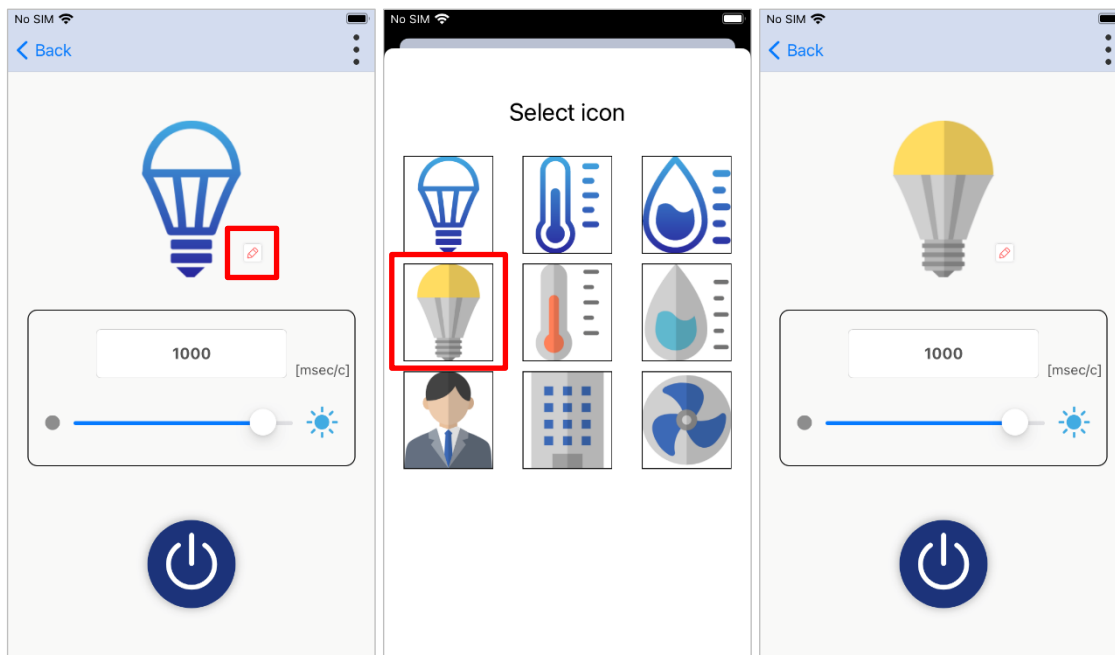


図 4.16 ライトデモ画面(3)

#### 4.4 データデモ画面

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

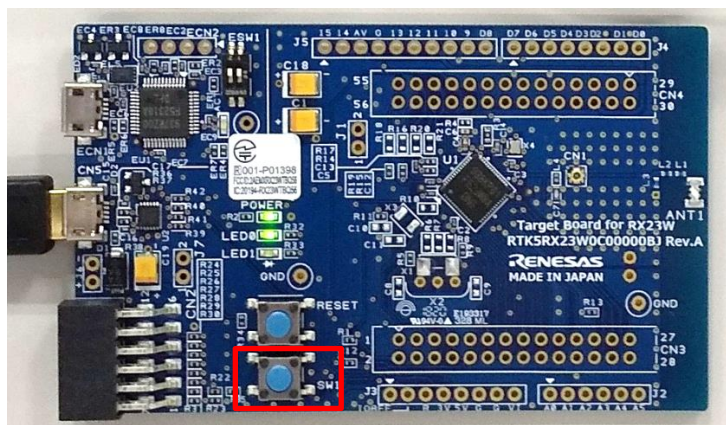


図 4.17 Target Board for RX23W のユーザスイッチ

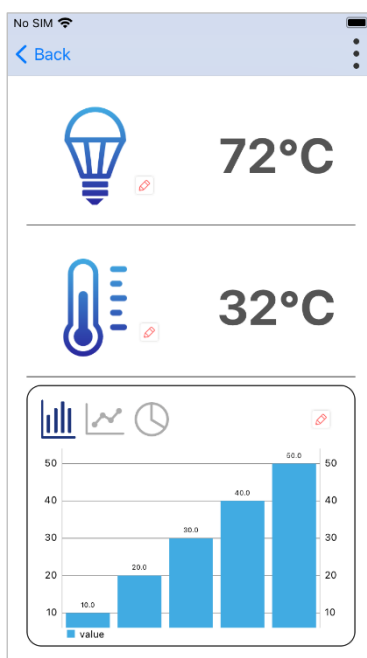


図 4.18 データデモ画面(1)

グラフ表示は棒グラフ、折れ線グラフ、円グラフを切り替えできます。

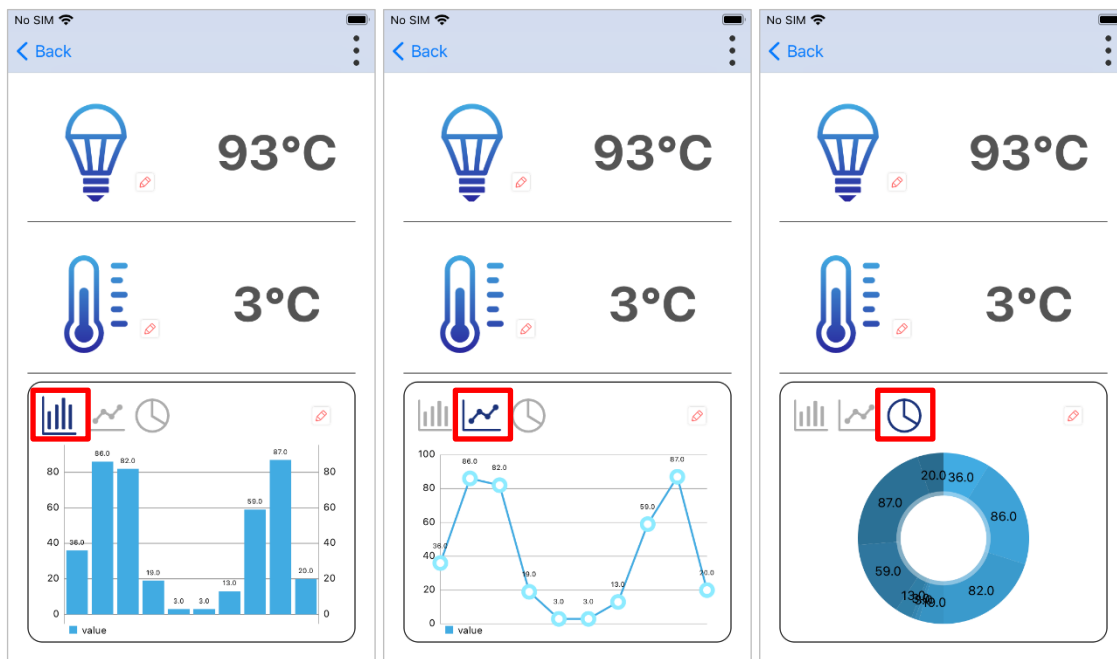


図 4.19 データデモ画面(2)

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



図 4.20 データデモ画面(3)



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

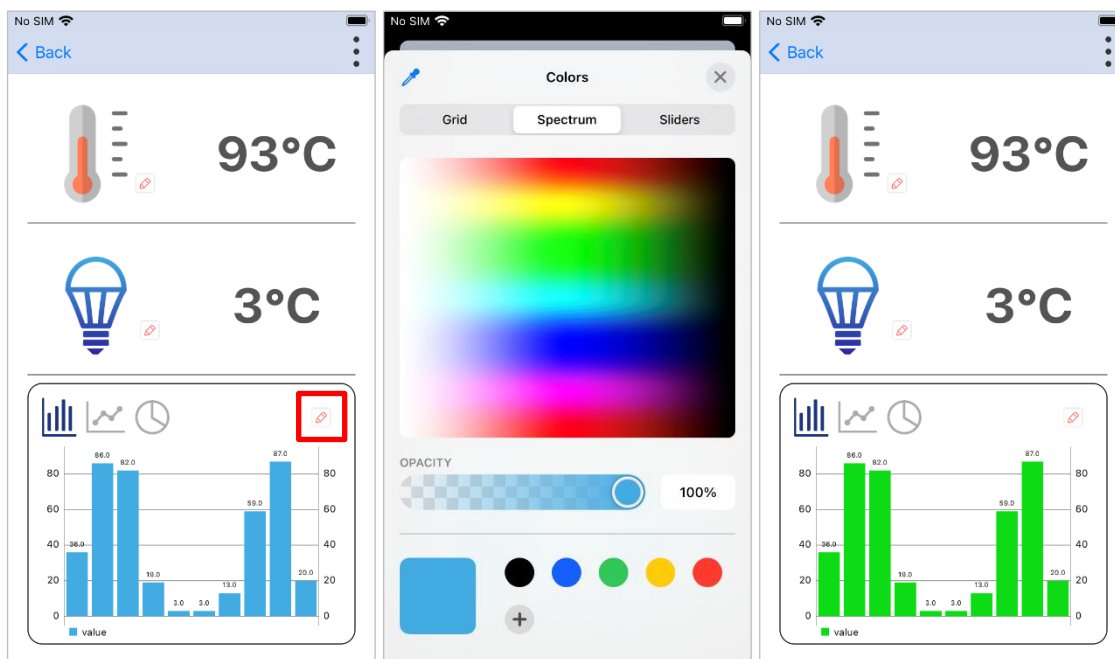


図 4.21 データデモ画面(4)

## 5. TryBT のカスタマイズ

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

### 5.1 アプリタイトルのカスタマイズ

本アプリのタイトルを変更できます。

1. Xcode で TryBT のプロジェクトを開きます。[Info]→[Bundle Name]でアプリタイトルを変更します。

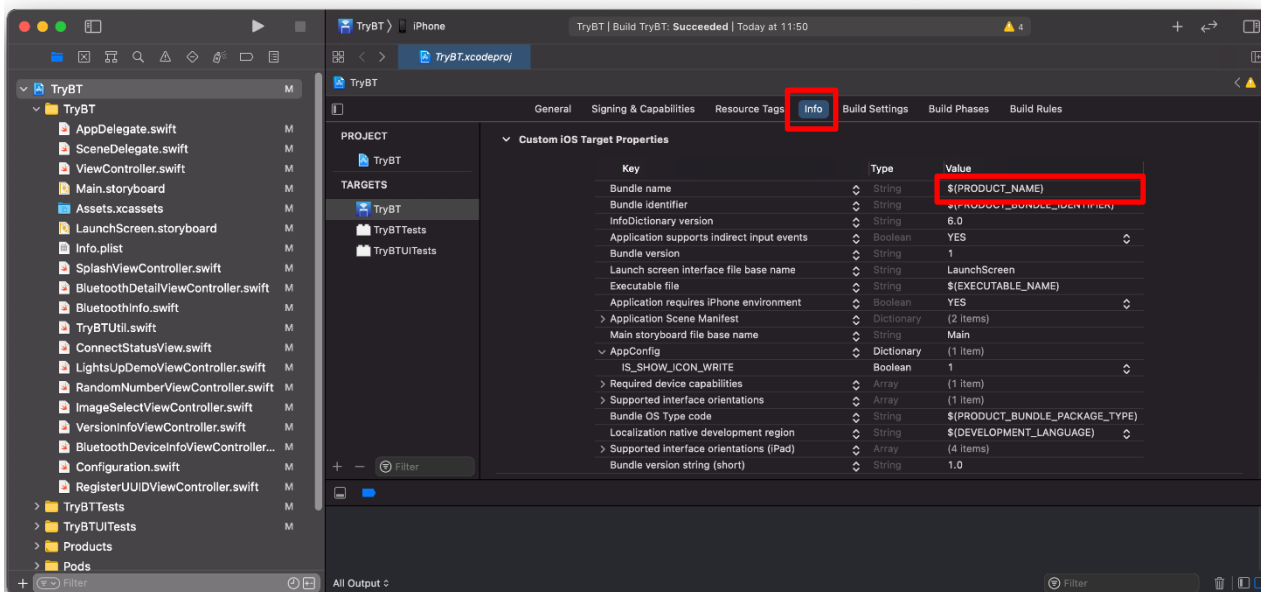


図 5.1 アプリタイトルのカスタマイズ

### 5.2 アプリアイコンのカスタマイズ

本アプリのアイコンを変更できます。

1. Xcode で TryBT のプロジェクトを開きます。[Assets.xcassets]→[AppIcon]でアイコンを変更します。  
※全ての解像度のアイコンを変更してください。

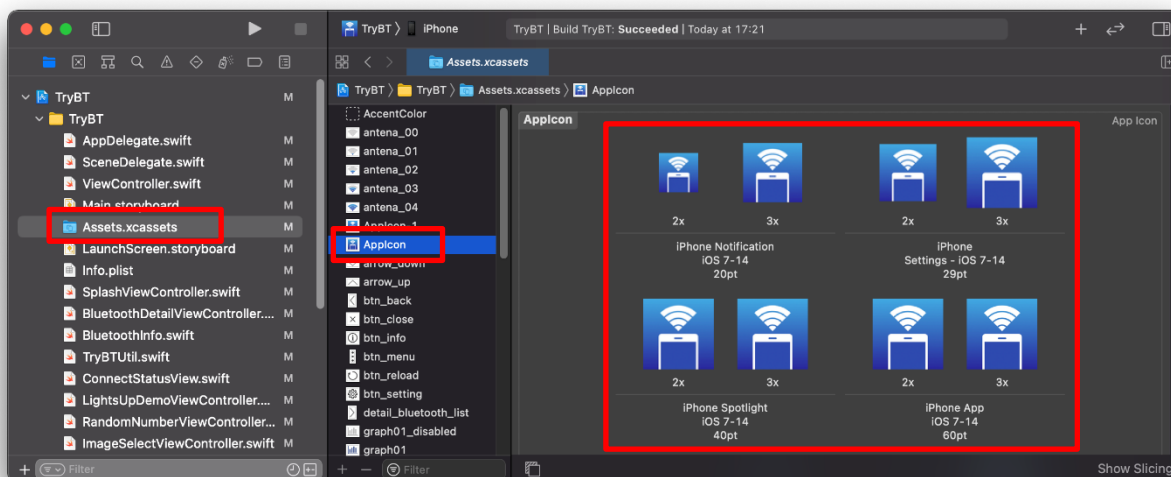


図 5.2 アプリアイコンのカスタマイズ

### 5.3 スプラッシュ画面のカスタマイズ

本アプリ起動時のスプラッシュ画面を変更できます。

1. Xcode で TryBT のプロジェクトを開きます。[Assets.xcassets] で"splash\_background"と"splash\_logo"を変更します。

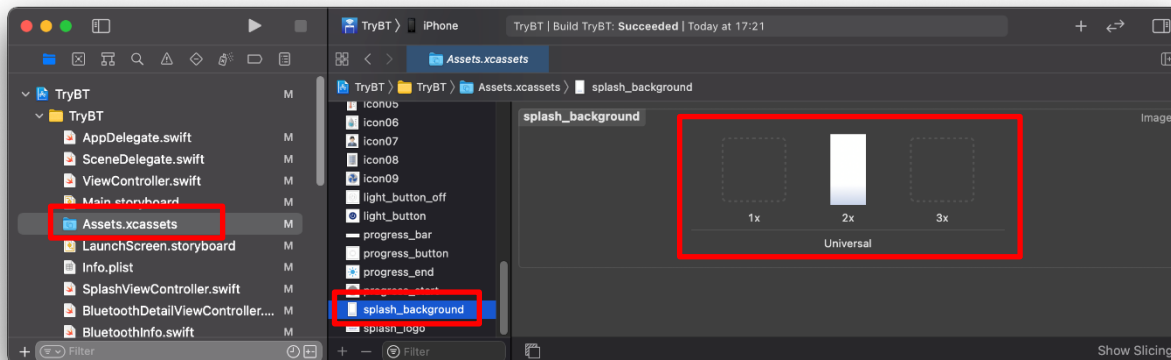


図 5.3 スプラッシュ画面のカスタマイズ(splash\_background)

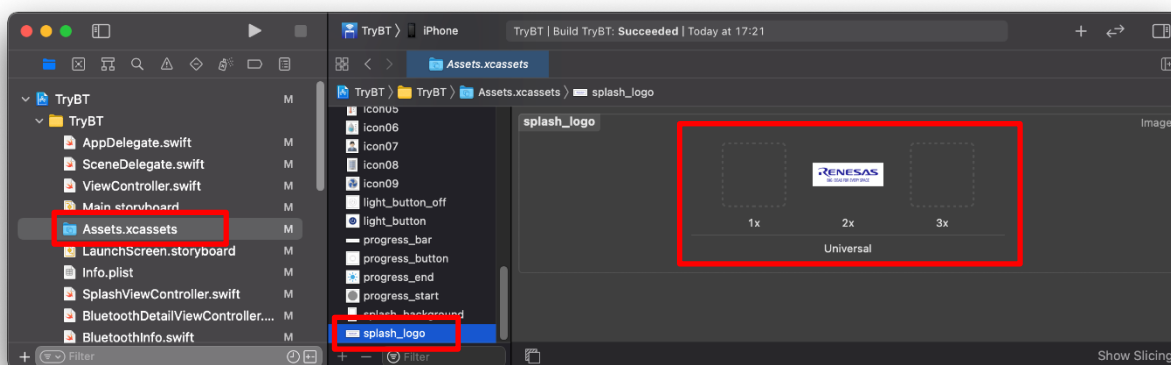


図 5.4 スプラッシュ画面のカスタマイズ(splash\_logo)

### 5.4 アイコンデータのカスタマイズ

ライトデモ画面とデータデモ画面でカスタマイズ可能なアイコンを変更できます。

※アイコンは最大9個までです。

※アイコンは png フォーマットで 200×200 の解像度で用意してください。

1. Xcode で TryBT プロジェクトを開きます。[Assets.xcassets]の icon01～icon09 を変更します。

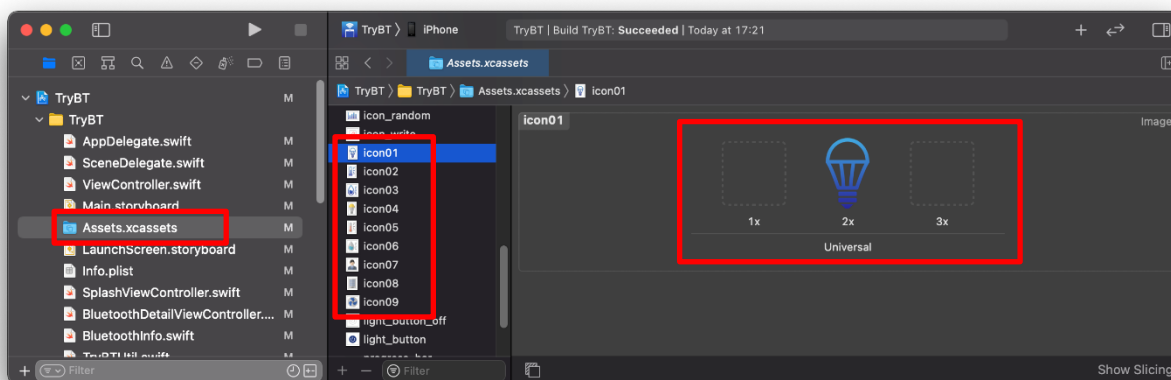


図 5.5 デモ画面のアイコンデータのカスタマイズ

## 5.5 カスタマイズモードの有効化・無効化

本アプリは実行中にアイコンなどを変更できるカスタマイズモードを持ちます。

1. Xcode で TryBT のプロジェクトを開きます。Info.plist 内の[AppConfig]→[IS\_SHOW\_ICON\_WRITE]を 0 または 1 に変更します。

1: カスタマイズモードを有効化

0: カスタマイズモードを無効化

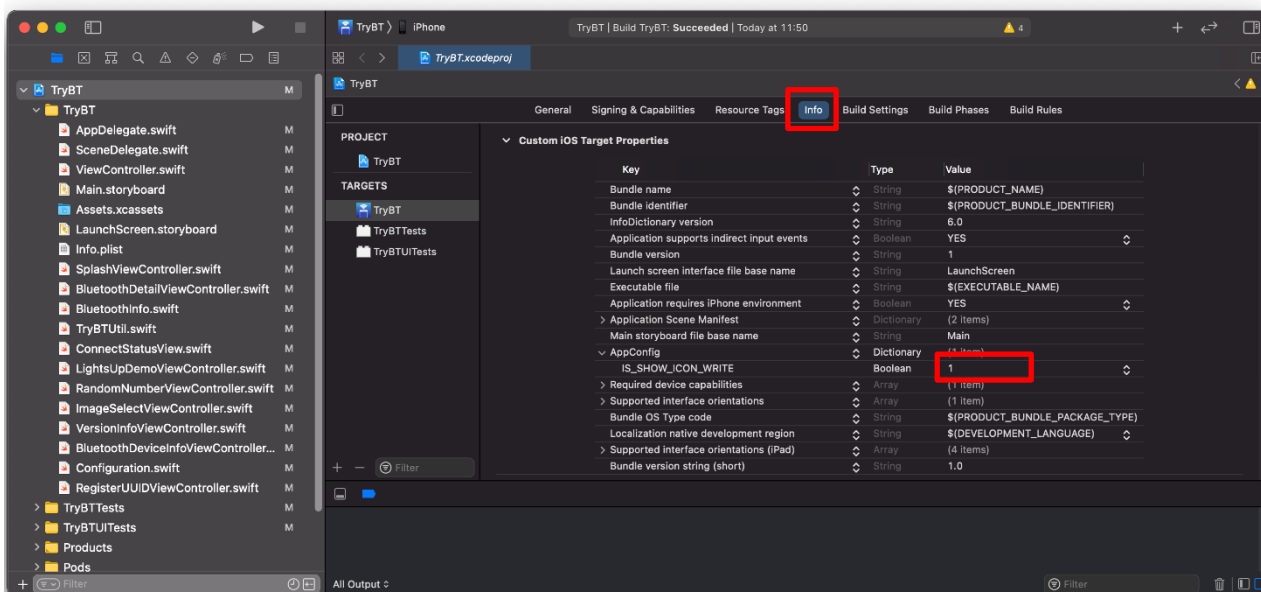


図 5.6 カスタマイズモードのオンオフ

## 6. TryBT のファイル構成

TryBT のファイル構成図を示します。本節では、Info.plist、 swift ファイルについて解説します。



図 6.1 TryBT のフォルダ構成

## 6.1 Info.plist について

Info.plist は TryBT のアプリの設定を行うためのファイルです。Info.plist ファイルは、アプリインストール時のアイコンやバージョン、アプリ表示名を設定します。主な設定値を示します。詳細は公式サイトをご覧ください。([https://developer.apple.com/documentation/bundleresources/information\\_property\\_list](https://developer.apple.com/documentation/bundleresources/information_property_list))

表 6-1 Info.plist

属性名	概要
Bundle version string (short)	本アプリのバージョン（文字列）を指定します。 例：1.0.0
Bundle version	ビルドを通し番号で表現します。
Bundle name	本アプリの表示名を指定します。 例：TryBT

## 6.2 .swift

.swift の拡張子の各ファイルは TryBT の画面、ロジックを構成するファイルになります。ここでは各ファイルの概要について記載します。

表 6-2 .swift に定義されるクラスの概要

クラス名	概要
AppDelegate	アプリケーション全体のライフサイクルを管理するクラスです。TryBT では定数の定義なども行っています。
BluetoothDetailViewController	Bluetooth 詳細画面を定義しています。
BluetoothDeviceInfoViewController	接続しているデバイスの情報を表示する画面を定義しています。
BluetoothInfo	Bluetooth 一台分の接続情報を定義する構造体です。
Configuration	Info.plist からデータを取得するためのユーティリティクラスです。
ConnectStatusView	CONNECTED と DISCONNECTED の表示を切り替えるカスタムビュークラスです。
ImageSelectViewController	アイコンを選択する画面を定義しています。
LightsUpDemoViewController	LED デモを行う画面を定義しています。
RandomNumberViewController	グラフデモを行う画面を定義しています。
RegisterUUIDViewController	UUID 登録画面を定義しています。
SplashViewController	スプラッシュ画面です。
TryBTUtil	ユーティリティ系のメソッドを定義するクラスです。
VersionInfoViewController	バージョン情報を表示する画面を定義しています。
ViewController	初期表示の Bluetooth 一覧画面を定義しています。

## 7. TryBT の画面遷移

TryBT の画面とクラス遷移を以下に記載します。

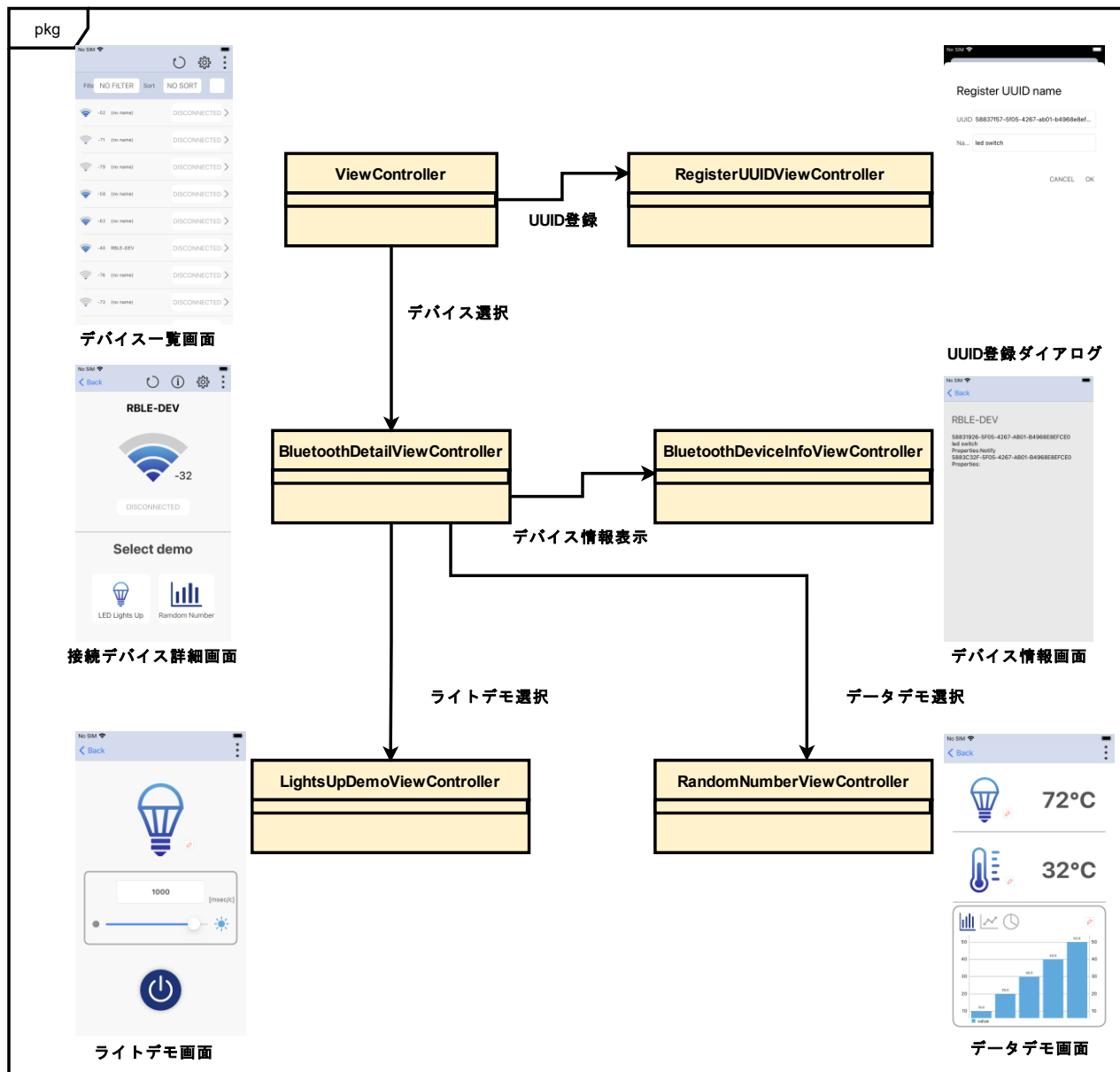


図 7.1 TryBT の画面とクラス遷移

## 8. TryBT の Bluetooth 通信

本章では TryBT での Bluetooth の使用方法について解説します。iOS の Bluetooth に関しては公式サイト (<https://developer.apple.com/documentation/corebluetooth>) も合わせてご覧ください。

### 8.1 CentralManager によるスキャンの実行

CBCentralManager クラスを利用して Peripherals デバイスをスキャンします。

```
override func viewDidLoad() {
    super.viewDidLoad()
    ...
    centralManager = CBCentralManager(delegate: self, queue: nil, options: nil)
    ...
}
```

図 8.1 CBCentralManager の初期化 (ViewController.swift line 167)

```
// begin to scan
func startScan(){
    print("begin to scan ...")
    allBluetooth = Array<BluetoothInfo>()

    centralManager.delegate = self
    centralManager.scanForPeripherals(withServices: nil)
}
```

図 8.2 スキャンの実行(ViewController.swift, line 208)

デバイスのスキャン結果を保持して表示用リストに格納します。



```
func centralManager(_ central: CBCentralManager,
                    didDiscover peripheral: CBPeripheral,
                    advertisementData: [String: Any],
                    rssi RSSI: NSNumber) {

    print("peripheral.name: ¥(String(describing: peripheral.name))")
    print("advertisementData:¥(advertisementData)")
    print("RSSI: ¥(RSSI)")
    print("peripheral.identifier.uuidString: ¥(peripheral.identifier.uuidString)¥n")

    let bluetoothInfo =
        BluetoothInfo(peripheral: peripheral, advertisementData: advertisementData, rssi: RSSI)

    var isAddBluetooth = true
    for addedBluetoothInfo in allBluetooth{
        if(peripheral.identifier.uuidString == addedBluetoothInfo.peripheral.identifier.uuidString){
            isAddBluetooth = false
            break
        }
    }
    if(isAddBluetooth){
        allBluetooth.append(bluetoothInfo)
        self.refreshTable()
    }
}
```

図 8.3 スキャン結果の保持 (ViewController.swift, line 313)

## 8.2 Peripheral デバイスへの接続

指定した Peripheral デバイスに接続します。centralManager.connect の第一引数には、CBPeripheral クラスを渡します。

```
func connect(){
    self.centralManager.delegate = self
    if(self.peripheral != nil){
        self.centralManager.cancelPeripheralConnection(self.peripheral!)
        self.peripheral = nil
    }
    self.centralManager.connect(bluetoothInfo!.peripheral, options: nil)
    // Do any additional setup after loading the view.

    statusLabel.status = ConnectStatusView.ConnectStatus.disconnected

    self.lightsUpDemoButton.isEnabled = false
    self.randomNumberButton.isEnabled = false
}
```

図 8.4 Peripheral への接続 (BluetoothDetailViewController.swift, line 168)

接続後のイベントは、centralManager の didConnect の Delegate に通知されます。TryBT では、接続された Peripheral デバイスを保持しています。

```
//call when connect peripheral success
func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {
    print("didConnect")
    statusLabel.status = ConnectStatusView.ConnectStatus.connecting

    UserDefaults.standard.set(peripheral.identifier.uuidString,
                               forKey: fromAppDelegate.LAST_CONNECTED_UUID)

    // set delegate
    peripheral.delegate = self
    self.peripheral = peripheral

    self.lightsUpDemoButton.isEnabled = true
    self.randomNumberButton.isEnabled = true
}
```

図 8.5 接続イベント (BluetoothDetailViewController.swift, line 229)

### 8.3 Service Discovery の実施

TryBT では、接続デバイス詳細画面の"LED Lights Up"ボタンもしくは、"Random Number"ボタンがタップされると、その Service の UUID を指定してサービス検索を行います。

```
@IBAction func lightsUpDemoPressed(_ sender: Any) {

    if(peripheral != nil){
        selectedDemo = .lightsUpDem
        // service-id
        let UUID = CBUUID(string: fromAppDelegate.demo_uuid)
        // 4-1. 利用可能 Service の探索開始
        peripheral!.discoverServices([UUID])
    }else{
        self.alertDisconnectMessage()
    }
}
```

図 8.6 Lights Up Demo のサービス検索 (BluetoothDetailViewController.swift, line 41)

サービス検索の完了後、キャラクターリスティックを検索します。

```
func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
    if (error != nil) {
        print("error: ¥(String(describing: error))")
        return
    }

    for service in peripheral.services!
    {
        // search Characteristic
        switch selectedDemo{
        case .lightsUpDem:
            let serviceUUIDArray =
                NSArray(object: CBUUID(string: fromAppDelegate.LIGHT_UP_SERVICE_UUID))
            peripheral.discoverCharacteristics(serviceUUIDArray as? [CBUUID], for:service as CBService)
        case .randomNumber:
            let serviceUUIDArray =
                NSArray(object: CBUUID(string: fromAppDelegate.RANDOM_NUMBER_SERVICE_UUID))
            peripheral.discoverCharacteristics(serviceUUIDArray as? [CBUUID], for:service as CBService)
        }
    }
}
```

図 8.7 キャラクターリスティックの検索 (BluetoothDetailViewController.swift, line 261)

キャラクターリスティックが検索出来たら、各画面に遷移します。

```
func peripheral(_ peripheral: CBPeripheral,
  didDiscoverCharacteristicsFor service: CBService, error: Error?) {

  if (error != nil) {
    print("error: ¥(String(describing: error))")
    return
  }
  if(service.characteristics!.isEmpty){
    //error
  }else{

    self.characteristics = service.characteristics![0]
    _ = self.characteristics?.uuid

    switch selectedDemo{
    case .lightsUpDem:
      performSegue(withIdentifier: "toLightsUpDemo",sender: nil)
    case .randomNumber:
      performSegue(withIdentifier: "toRamdomNumber",sender: nil)

    }
  }
}
```

図 8.8 キャラクターリスティックの検索結果の通知 (BluetoothDetailViewController.swift, line 283)

#### 8.4 ライトデモ画面での値の書き込み

ライトデモ画面では、スライダー、テキストエディットの変更時に peripheral に値を書き込みます。データとキャラクターリスティック、書き込み方法を指定します。

```
func writeValue(value:Int){
  var value:UIInt = UInt(value)
  let data: NSData = NSData(bytes: &value, length: 1)

  peripheral!.writeValue(data as Data,
    for: self.characteristics!,
    type: .withResponse)
}
```

図 8.9 キャラクターリスティックへの書き込み (LightsUpDemoViewConroller.swift, line 142)

#### 8.5 データデモ画面での通知待受け

評価ボードのボタンクリックイベントを受け取るための通知待受けを設定します。Peripheral デバイスに対して、Notify の有効/無効を設定します。

```
override func viewDidLoad() {
    super.viewDidLoad()
    let menuBarButtonItem = UIBarButtonItem(
        image: UIImage(named: "btn_menu")?.withRenderingMode(.alwaysOriginal),
        style: .plain,
        target: self,
        action: #selector(menuButtonPressed))

    navigationItem.rightBarButtonItem = [menuBarButtonItem]

    peripheral?.delegate = self
    peripheral?.setNotifyValue(true, for: self.characteristics!)
    ...
}
```

図 8.10 Notify の有効化 (RandomNumberViewController.swift, line 51)

評価ボードからの Notify は、以下の didUpdateValueFor の Delegate に通知されます。

```
func peripheral(_ peripheral: CBPeripheral,
               didUpdateValueFor characteristic: CBCharacteristic, error: Error?) {

    if (error != nil) {
        print("Write...error: ¥(String(describing: error))")
        return
    }

    let newValue1 = Int.random(in: 0...100)
    let newValue2 = Int.random(in: 0...100)
    let newGraphValue = Int.random(in: 0...100)
    sampleData.append(Double(newGraphValue))
    if(sampleData.count > 20){
        sampleData.remove(at: 0)
    }

    UserDefaults.standard.set(newValue1, forKey: fromAppDelegate.RANDOM_VALUE_1)
    UserDefaults.standard.set(newValue2, forKey: fromAppDelegate.RANDOM_VALUE_2)
    UserDefaults.standard.set(sampleData, forKey: fromAppDelegate.RANDOM_GRAPH_VALUE)
    refreshRandomValue()
}
```

図 8.11 Notify の取得 (RandomNumberViewController.swift, line 319)

## 改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2021.10.15	-	初版発行

## 製品ご使用上の注意事項

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

### 1. 静電気対策

CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。

### 2. 電源投入時の処置

電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

### 3. 電源オフ時における入力信号

当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れしないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。

### 4. 未使用端子の処理

未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。

### 5. クロックについて

リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後、リセットを解除してください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

### 6. 入力端子の印加波形

入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、 $V_{IL}$  (Max.) から  $V_{IH}$  (Min.) までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、 $V_{IL}$  (Max.) から  $V_{IH}$  (Min.) までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。

### 7. リザーブアドレス（予約領域）のアクセス禁止

リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

### 8. 製品間の相違について

型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違っていると、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

## ご注意書き

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

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

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

当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。

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

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

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

(Rev.5.0-1 2020.10)

## 本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレストシア）

[www.renesas.com](http://www.renesas.com)

## 商標について

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

## お問合せ窓口

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

[www.renesas.com/contact/](http://www.renesas.com/contact/)