Device Connect 1.0

iOS デバイスプラグイン開発チュートリアル

1.0.0 版
2014 年 9 月 21 日

更新履歴

更新日	更新内容	更新担当者
2013/9/21	初版。	福井 重和、星 孝之

はじめに

本ドキュメントは、DeviceConnect API 1.0 を用いて iOS デバイスプラグインを開発する方法について解説している。iOS そして Objective-C に関する一般的な知識を保持しているのを前提に解説していく。

1. DeviceConnect について

DeviceConnect とは、RESTful コンセプトを用いて通信先への命令送信や、通信元と通信先の間でのデータ送受信を可能とするプロトコルである。各処理には Profile という形で URI が割り振られ、その URI に対し HTTP でアクセスしたり、Android や iOS などのプラットフォーム毎に定められたメッセージ通信プロトコルでアクセスしたりする事で処理を行う。

また DeviceConnect におけるデバイスプラグインとは、DeviceConnect プロトコルに従って作られた処理のリクエストメッセージを受け取って解釈し、実デバイスに対し適切な処理を行わせ、得られた結果を DeviceConnect に従ったレスポンスメッセージとして返却する様な、デバイスとのやり取りを実質的に担うモジュールとなる。

2. 前準備

まず事前に iOS 版 DeviceConnect 一式をダウンロードする。
DeviceConnect 一式が入ったディレクトリで、このチュートリアルで特に取り扱う物は以下になる。

1. dConnectSDK/dConnectSDKForIOS
2. dConnectDevicePlugin
3. DeviceConnectTutorial

1 つめの dConnectSDKForIOS には、デバイスプラグインやアプリで必要となる dConnectSDK のフレームワーク（.framework 拡張子ファイル）や、最終的にアプリでリソースとして同梱する必要のある dConnectSDK のバンドル（.bundle 拡張子ファイル）をビルドする為の Xcode プロジェクトが入っている。
2 つめの dConnectDevicePlugin には各種デバイスに対応したデバイスプラグインの Xcode プロジェクトが入っている。
そして 3 つめの dConnectTutorial には次節「Get Started」で使う開発用ワークスペースのサンプルが入っている。

このチュートリアルを終える頃にはデバイスプラグイン作成、そして簡単な DeviceConnect 対応アプリの作成方法に関する知識が身に付くはずである。もし手元に dConnectDevicePlugin 以下にあるデバイスプラグインの対応デバイスがあるなら、DeviceConnect 対応アプリでデバイスに様々な処理を行わせてみるのも楽しいかもしれない。
そして DeviceConnect プロトコルのコンセプトを理解し、様々なデバイスとプログラムスキルに恵まれた方々は是非、新たな DeviceConnect 対応アプリ、そして DeviceConnect デバイスプラグインの開発に挑戦していただきたい。

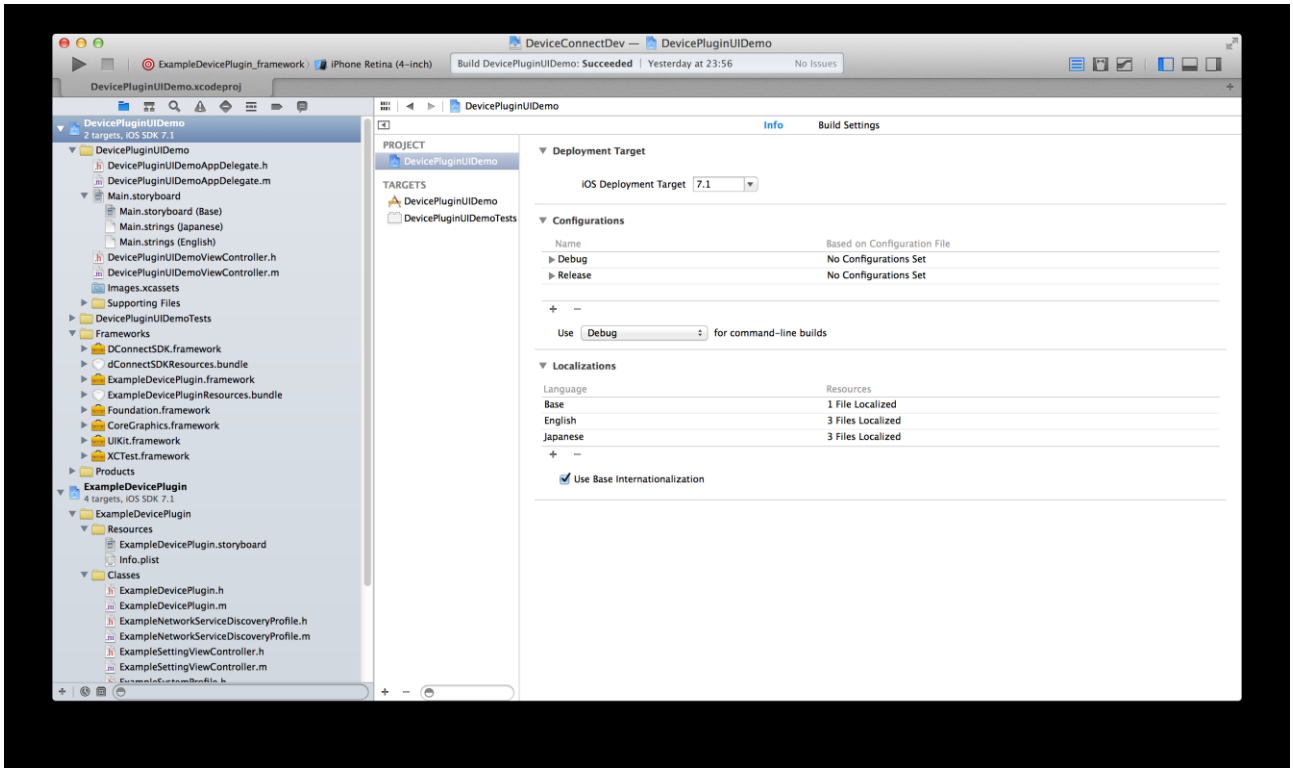
3. Get Started

デバイスプラグイン、そして簡単な DeviceConnect 対応アプリを作成・開発するための環境を 1 から整えるのは簡単な作業ではない。よって、まずは既に出来上がった作成・開発環境の見本から触れてみる。そうすることでデバイスプラグイン開発における最終的な終着点、作成・開発を行う為に必要な物事とは何かを掴めるはずである。

1.1. 開発用ワークスペースのチェック

第 2 節で挙げた DeviceConnectTutorial には開発用ワークスペースのサンプルが入っている。まず Xcode ワークスペースである DeviceConnectDev.xcworkspace を開く。開くと以下の様に ExampleDevicePlugin そして DevicePluginUIDemo プロジェクトが既にプロジェクトナビゲーターに追加されているのが確認できる。

ExampleDevicePlugin はこのチュートリアルで取り扱うことになるデバイスプラグイン見本であり、DevicePluginUIDemo はそのデバイスプラグイン見本がどう機能するのかチェックする為の簡単なアプリである。



本チュートリアルで用いる開発用ワークスペース見本

1.2. ビルドからデモアプリ起動まで

DConnectSDK のフレームワークやバンドルは既に同梱してあり、各プロジェクトで既に参照済みとなっている。ここでやる事は ExampleDevicePlugin そして DevicePluginUIDemo プロジェクトの必要な各ビルドターゲットのビルドである。ビルドに際して各ビルドターゲットの依存関係を解決

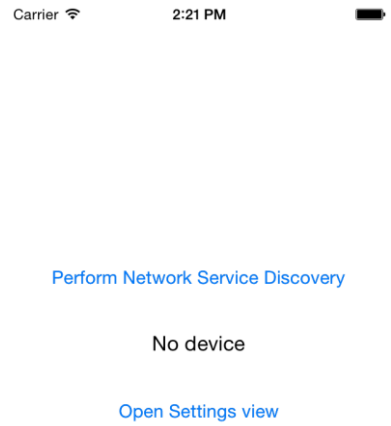
する必要があるが、既にその対処は済ませてある。そして依存関係を解決する順番でのビルドは以下の様になる。

1. ExampleDevicePlugin_framework
2. ExampleDevicePluginResources
3. DevicePluginUIDemo

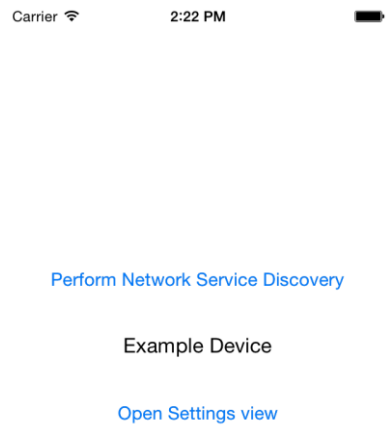
ExampleDevicePlugin はデバイスと連携させることなく、System や Network Service Discovery を含む各種 DeviceConnect プロファイルのモックアップとして準備されている。デバイスプラグインの開発を行う際はこれらモックアップ実装を足がかりにすることをお勧めする。

それでは上記の順番でビルドターゲットをビルドし、最後に DevicePluginUIDemo をシミュレーターで実行してみる。

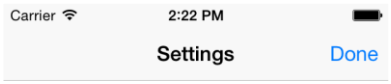
アプリを起動すると以下のような画面が表示される。



次に「Perform Network Service Discovery」を押すことによってデバイスプラグインの検索を行う。このデモアプリでは ExampleDevicePlugin が見つかるはずなので、以下のように「No device」の部分に ExampleDevicePlugin のデバイス名が表示される。



この状態で「Open Settings view」を押すとデバイスプラグインで作成した設定画面が起動する。設定画面では、デバイスプラグインとスマートフォンの接続設定を記述する画面となる。ユーザがシームレスに DeviceConnect を使ってデバイスプラグインを利用できる様な、デバイスとの接続方法や接続処理などを正しく、解りやすく行える内容を記述する事。

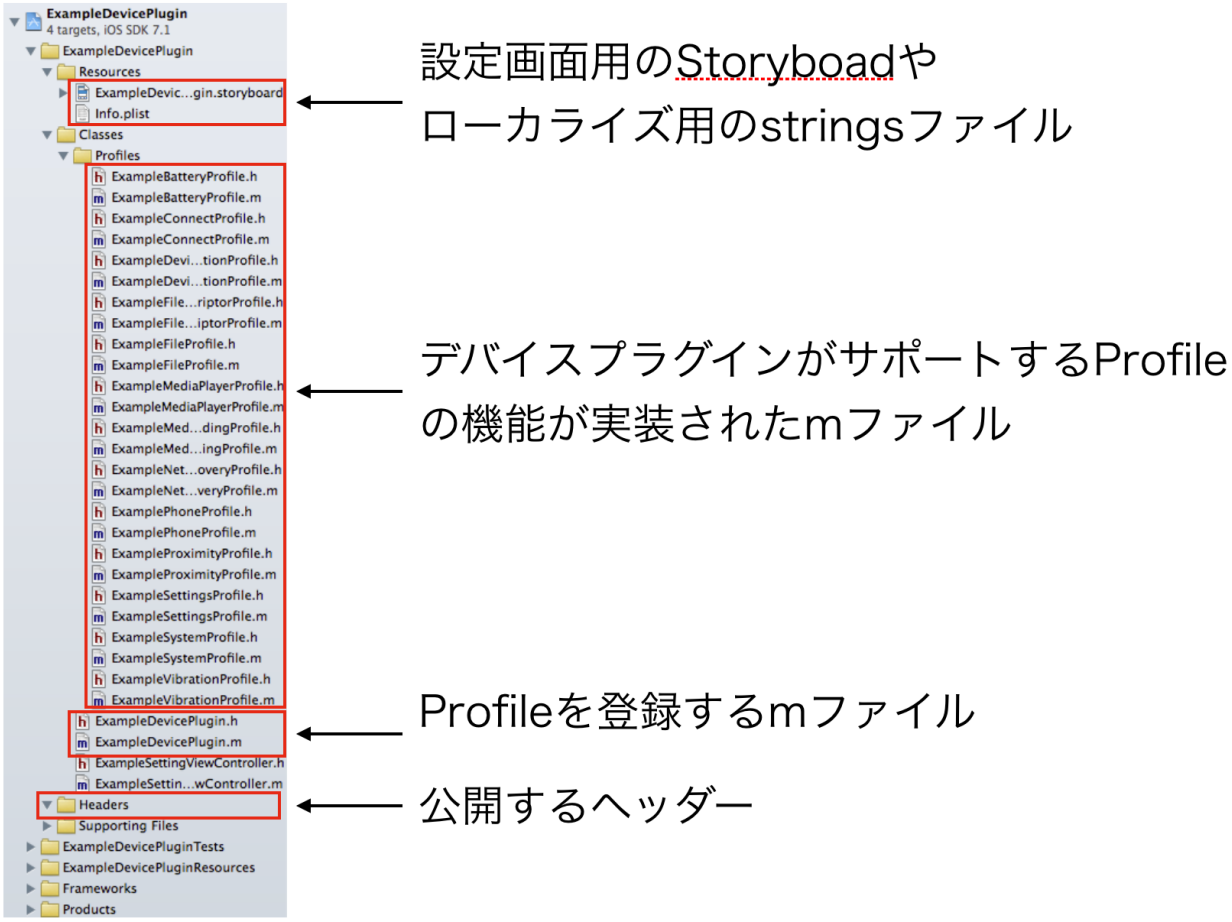


このアプリでは DeviceConnect のセキュリティ機構である Local OAuth 認証を必要とする場面、例えば Vibration でデバイスを振動させたり、Settings で日時変更をしたりなどのユーザ認証が必要な場面を用意していない。更に詳細なアプリ作成、Local OAuth 認証の使い方を知りたい場合は、HTML5/Android/iOS アプリチュートリアルを参照の事。

1.3. ExampleDevicePlugin の説明

ここでは、ExampleDevicePlugin の構成について簡単に説明する。

3.3.1. ディレクトリ構造



基本、デバイスプラグインは framework として作成されるため、「Resources」と「Classes」と「Headers」フォルダを作成する。詳しい説明は、iOS における Framework の作成の仕方について調べる事。

「公開するヘッダー」は、デバイスプラグインが拡張プロファイルという独自に定義したプロファイルを公開する場合にヘッダーファイルを配置する。DeviceConnect では、「DConnectProfile」を継承することによって独自のプロファイルを作成することが出来る。この資料では標準で用意されているプロファイルではない、独自の「拡張プロファイル」の作り方については割愛する。

3.3.2. Resources ディレクトリ

Resources ディレクトリには、設定画面の UI を作成するための Storyboard や、ローカライズ用に使用する.strings ファイルを配置する。

3.3.3. Classes ディレクトリ

Classes ディレクトリには、作成するデバイスプラグインのソースコードがおかれている。例えば、デバイスプラグインが実現する機能である各種プロファイルの m ファイル、それらプロファイルを登録するデバイスプラグインの m ファイル、そして設定画面 UI やその他の画面 UI を制御する機能を持つ m ファイルを配置する。

基本的に、Classes ディレクトリ内ではディレクトリの制限は特にないので各自わかりやすいように配置する事。

3.3.3.1. ExampleBatteryProfile

ここで、バッテリーの機能を提供するプロファイル実装である ExampleBatteryProfile の仕組みを見てみる。基本的な仕組みは Android などプラットフォームを問わず同じで、プラグインを継承し用意されているメソッドをオーバライドすることによってデバイスプラグインの機能を実現する。DeviceConnect の標準プロファイルである Battery や DeviceOrientation などのモック実装が用意されているので、必要なプロファイルがあれば各自そのプロファイルを実装したクラスを再利用して役立てる事。

```
#import "ExampleBatteryProfile.h"
@implementation ExampleBatteryProfile

- (instancetype)init
{
    self = [super init];
    if (self) {
        self.delegate = self;
    }
    return self;
}

#pragma mark - DConnectBatteryProfileDelegate

- (BOOL) profile:(DConnectBatteryProfile *)profile didReceiveGetAllRequest:(DConnectRequestMessage *)request
        response:(DConnectResponseMessage *)response deviceId:(NSString *)deviceId
{
    [response setResult:DConnectMessageResultTypeOk];
......
    return YES;
}

- (BOOL) profile:(DConnectBatteryProfile *)profile didReceiveGetLevelRequest:(DConnectRequestMessage *)request
        response:(DConnectResponseMessage *)response deviceId:(NSString *)deviceId
{
    [response setResult:DConnectMessageResultTypeOk];
......
    return YES;
}
......
@end
```

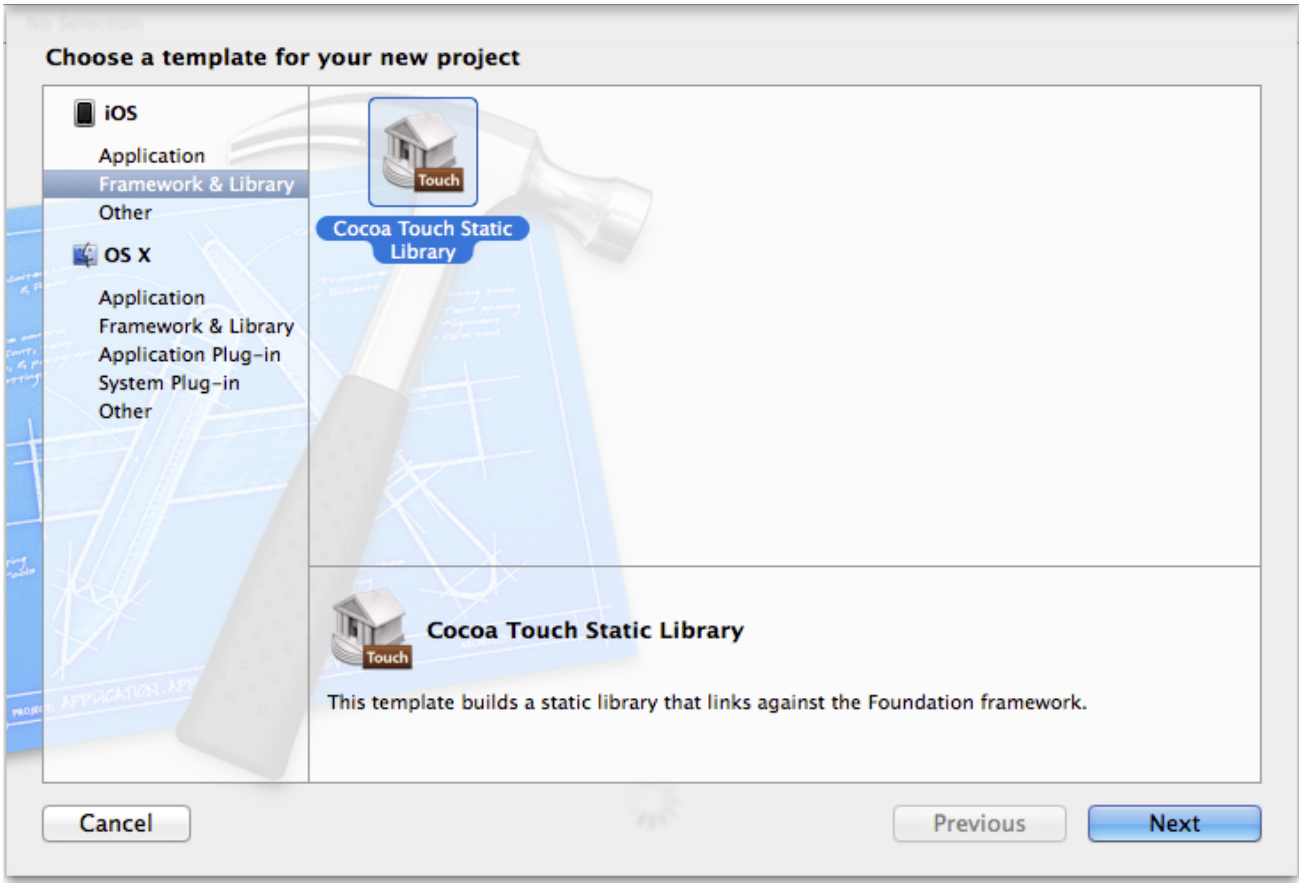
3.3.3.2. ExampleDevicePlugin

ExampleDevicePlugin は、DConnectDevicePlugin を継承しており、その m ファイルの中の init メソッドの中で以下のように addProfile することによって必要なプロファイルを追加することができる。プロファイルを実装した際、addProfile メソッドで登録を行わない限り DeviceConnect でそのプロファイルが認識されないので気をつける事。

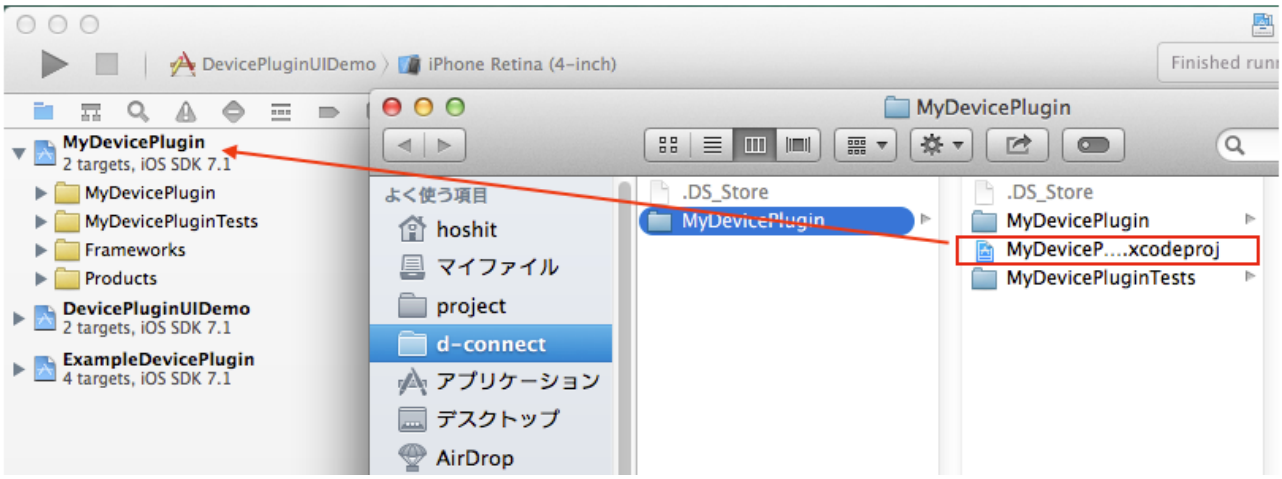
```
.....

@implementation ExampleDevicePlugin

- (instancetype) init {
    self = [super init];

    if (self) {
        self.mDeviceId = @"example_device_id";
.....
        [self addProfile:[ExampleSystemProfile new]];
        [self addProfile:[ExampleNetworkServiceDiscoveryProfile new]];

        [self addProfile:[ExampleBatteryProfile new]];
        [self addProfile:[ExampleConnectProfile new]];
        [self addProfile:[ExampleDeviceOrientationProfile new]];
        [self addProfile:[ExampleFileDescriptorProfile new]];
        [self addProfile:[ExampleFileProfile new]];
        [self addProfile:[ExampleMediaPlayerProfile new]];
        [self addProfile:[ExampleMediaStreamRecordingProfile new]];
        [self addProfile:[ExamplePhoneProfile new]];
        [self addProfile:[ExampleProximityProfile new]];
        [self addProfile:[ExampleSettingsProfile new]];
        [self addProfile:[ExampleVibrationProfile new]];
    }

    return self;
}
.....
```

4. デバイスプラグイン開発用プロジェクトの作り方

デバイスプラグインは、Framework(Cocoa Touch Static Library)として作成する。



開発を行う上で、Workspace に追加する場合は、プロジェクトの新規作成時のオプションでプロジェクトの追加先 Workspace を選択するか、xcodeproj ファイルを以下の用に Workspace にドラッグアンドドロップする。余談だが、Xcode プロジェクトは同時に 2 つのプロジェクトやワークスペースから変更できないようになっているので、Workspace 側で作業する前に Workspace に追加されているプロジェクトを別ウィンドウで開いている際は、それら別ウィンドウを全て閉じる事。



Resources や Classes といったディレクトリの作成の仕方などは、Framework としての作成の仕方と同じなので、このマニュアルでは解説を割愛する。
デバイスプラグインは、Framework として作成するが、設定画面などの Resources ディレクトリにあるファイルに関しては Bundle ファイルとして UI アプリに同梱する。Bundle に関しても iOS の基本的な知識になるので、このマニュアルでは解説を割愛する。

5. 多言語化

デバイスプラグイン側でプラグイン名などの文字列や UI を多言語化する場合は、Resources ファイルに各言語の文字列を定義した Localizable.strings ファイルを作成したり、各言語用にローカライズされた設定画面用 UI の Storyboard を作成したりして、それらを同梱した Bundle ファイルを用意、アプリ側で状況に応じて読み込むようにする事。
デバイスプラグインで用意した特定言語のリソースが利用されるには、UI アプリ側も同じ言語で多言語化されていなければならないので注意する事。