iOS UI アプリ作成チュートリアル

1.0.0 版
2014 年 9 月 24 日

更新履歴

更新日付	更新内容	更新担当者
2014/09/24	新規作成	福井重和

目次

はじめに

本ドキュメントは、Device Connect と連携する UI アプリの作成について網羅するチュートリアルである。読者が iOS そして Objective-C に関する一般的な知識を保持している事を前提に解説していく。

1. Device Connect について

Device Connect 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 Device Connect におけるデバイスプラグインとは、Device Connect 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 Device Connect 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. 用語定義

本ドキュメントで扱う用語について記載する。

用語	意味
UI アプリ	Device Connect が提供する機能を、UI を使ってユーザーに提供する事を目的として作られたアプリケーションの総称。
標準プロファイル	Device Connect において機能別に定義された API 群のカテゴリー。例えば下記の様な標準プロファイルが存在する: • Network Service Discovery: 稼働している Device Connect デバイスプラグインによって検知された、デバイスプラグイン対応周辺機器の一覧を取得する。 • Vibration:デバイスプラグイン対応周辺機器の振動機能を提供する。

3. 事前準備

まず事前に iOS 版 Device Connect 一式をダウンロードすること。
Device Connect 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の通り。

1. dConnectSDK/dConnectSDKForIOS
2. dConnectDevicePlugin
3. NativeAppTutorial

1 つめの dConnectSDKForIOS には、デバイスプラグインや UI 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 NativeAppTutorial には次節「Get Started」で使う UI アプリ Xcode プロジェクトのサンプルが入っている。

そしてチュートリアルの最後でカバーする、デバイスプラグインを用いて iOS デバイスの加速度情報を取得するパートでは iOS 実機が必要となる。iOS 実機へのアプリ転送は Apple の開発者登録が必要となる点に注意する事。

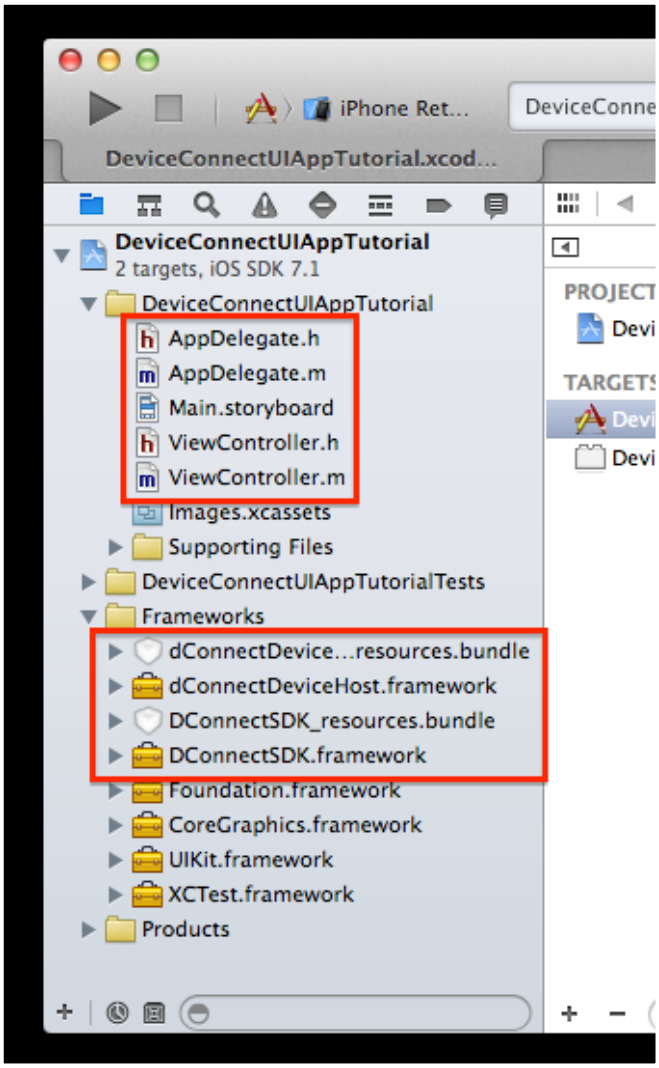
このチュートリアルを終える頃には簡単な Device Connect 対応 UI アプリの作成方法に関する知識が身に付くはずである。もし手元に dConnectDevicePlugin 以下にあるデバイスプラグインの対応デバイスがある場合、Device Connect 対応 UI アプリで対応デバイスに様々な処理を行わせてみると楽しいかもしれない。

4. Get Started

Device Connect 対応 UI 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いく。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把握できる。

1.1. ワークスペースのチェック

では NativeAppTutorial の Xcode プロジェクトに何が含まれているのかチェックしていく。Xcode の File メニューから Open を選択して、「NativeAppTutorial」プロジェクトを読み込む。



プロジェクトナビゲーター

まず、プロジェクトナビゲーターの DeviceConnectUIAppTutorial グループの下にはソースコード「AppDelegate」および「ViewController」、Storyboard「Main」が含まれています。これらソースが本チュートリアルで扱う UI アプリの UI やロジックを定義することとなる。
次に「Frameworks」グループの下を見てみる。iOS でおなじみのフレームワークなどが有る中、DConnectSDK.framework や DConnectSDK_resources.bundle、そして dConnectDeviceHost.framework や dConnectDeviceHost_resources.bundle というフレームワークやバンドルがある。これらが Device Connect SDK そして iOS Host デバイスプラグインである。Device Connect SDK は Device Connect の中核をなす存在で、デバイスプラグインの検知、Local OAuth 認証処理、Device Connect の RESTful および iOS Message API のリクエスト・レスポンス配送処理などを行う。iOS Host デバイスプラグインは iOS デバイスの機能をデバイスプラグインとして提供しており、Device Connect のほぼ全ての共通プロファイルを実装している。

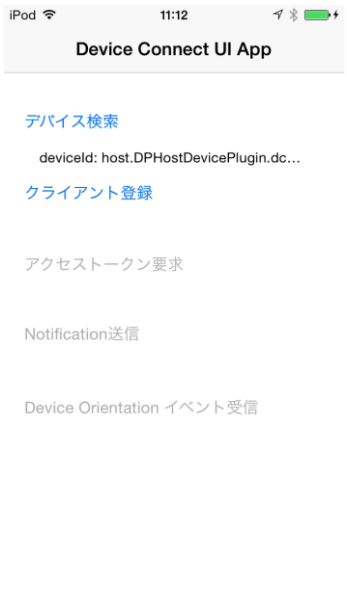
1.2. ビルドからアプリ起動まで

ビルドや起動の手順について説明するが、本チュートリアルの UI アプリをビルドして起動するまでに特別な作業は必要ではない。
ただし一点、この Device Connect UI アプリは最後に iOS Host デバイスプラグインを使って iOS 実機の加速度を表示する処理を行うが、これは加速度センサーをそもそも有していない iOS シミュレーターで起動させた場合には正常に動作しない。iOS シミュレーターで UI アプリを起動する際は加速度を取得する箇所でエラーが出るという事に留意する事。

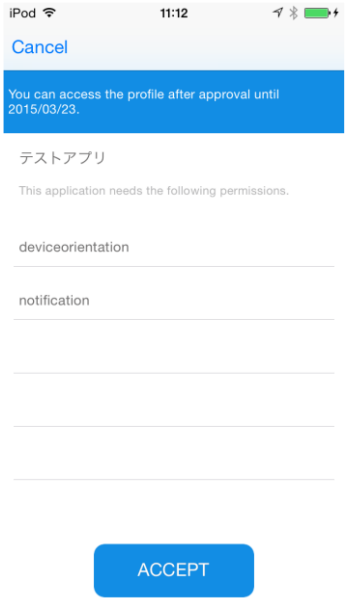
それでは、iOS 実機もしくは iOS シミュレーターでの UI アプリのビルドおよび起動を実際に行ってみる。Xcode の Run ボタンを押してビルドおよび起動を行う。順を追って内容を説明する。

1. まず「デバイス検索」ボタンを押して共通プロファイル Network Service Discovery を用いたデバイス検索を行い、UI アプリがインストールされたデバイス（iOS 実機や iOS シミュレーターなど）を検出する。

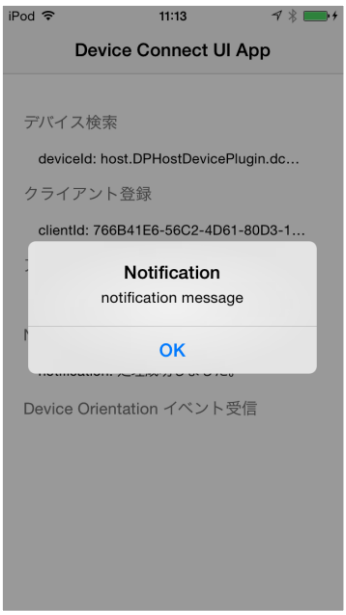
2. Local OAuth という認証機構によって Device Connect のセキュリティが担保されているが、「クライアント登録」ボタンを押して Local OAuth に対して UI アプリを登録する。



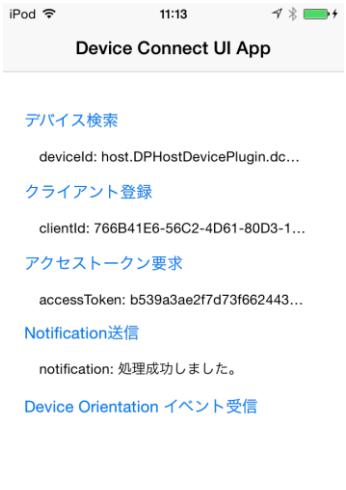
3. UI アプリの認可と登録が済んだら、次はアクセストークンの取得を行う。その際、UI アプリが用いようとしているプロファイルをユーザーが認可する為の画面が開くので認可する事。本 UI アプリは通知を行う Notification と、加速度などの取得を行う Device Orientation の 2 プロファイルの認可を求めてくる。認可後に取得できるアクセストークンは UI アプリの身元を証明する情報であり、UI アプリから Notification や Device Orientation プロファイルへのアクセスを可能にする情報となる。

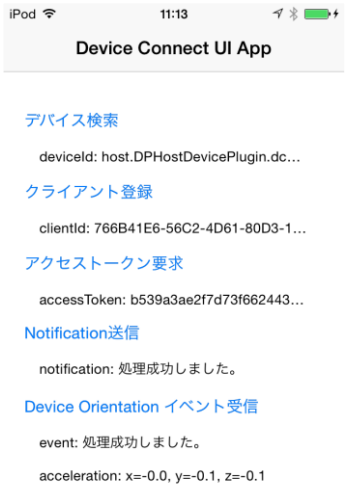


4. アクセストークンを添えて iOS Host デバイスプラグインの Notification プロファイルにアクセスし、通知ダイアログの表示要求を行う。この際、iOS Host デバイスプラグインが用いるプロファイルをユーザーが認可する為の画面が開くので認可する。認可をすると iOS Host デバイスプラグイの Notification プロファイルへ通知ダイアログの表示要求が届き、実際に通知ダイアログが表示される。



5. 最後に、アクセストークンを添えて iOS Host デバイスプラグインの Device Orientation プロファイルにアクセスし、加速度センサーやジャイロセンサーからの情報をイベントとして継続的に受け取る要求をします。受け取りに成功すると、「acceleration」の箇所に iOS デバイスの加速度がリアルタイムで表示される筈である。





上記の 5 つのプロセスがこの UI アプリの動作となる。ユーザーに Device Connect 対応 UI アプリを提供する際は、なるべくこの 5 つのプロセスを経る形で Device Connect へのアクセスを促す。

では次の節で先述の 5 つプロセスを実行するにあたり、UI アプリ側で行っていた処理の解説に移る。

5. UI アプリ側が行うべき処理

UI アプリが行う事は主に、Device Connect SDK が提供する「Device Connect マネージャー」、Device Connect の中核として処理・管理を行うクラスのインスタンスに対してのリクエスト送信・レスポンス受信、そしてイベント受領用コールバックの指定である。

では、4 節で触れた 5 プロセスのうち第 4 のプロセスで行った Notification プロファイルへのアクセスに関するリクエスト送信・レスポンス受信のコードを見てみる。プロジェクトの ViewController.m の関数「-(BOOL)callNotificationAPI:(NSString *)deviceId」のリクエスト生成部分が以下のコードになる。

```
DConnectRequestMessage *request = [[DConnectRequestMessage alloc]init];
[request setAction: DConnectMessageActionTypePost];
[request setApi: @"gotapi"];
[request setProfile: DConnectNotificationProfileName];
[request setAttribute: DConnectNotificationProfileAttrNotify];
[request setDeviceId: deviceId];
[request setAccessToken: _accessToken];

[request setInteger: DConnectNotificationProfileNotificationTypeMail forKey:
DConnectNotificationProfileParamType];
[request setString: @"notification message" forKey:
DConnectNotificationProfileParamBody];
```

「DConnectRequestMessage」は Device Connect iOS Message API でリクエストを送信する際に使われるクラスである。基本的に、

1. HTTP の GET・POST・PUT・DELETE のどれを用いるか（setAction）
2. 基本 API は何か（setApi、標準では”gotapi”を指定する）
3. アクセスするプロファイル・インターフェース・アトリビュートは何か（setProfile, setInterface, setAttribute）
4. アクセスするデバイスの ID は何か（setDeviceId）
5. アクセストークンは何か（setAccessToken）

を指定する。その上で各種プロファイルのドキュメントで記述されている通りにパラメータを設定する事でリクエストが完成する（setInteger や setString など）。
そして最後にリクエストを Device Connect マネージャーに送信し、レスポンスを受信する。

```
DConnectManager *mgr = [DConnectManager sharedManager];
[mgr sendRequest: request callback:^(DConnectResponseMessage *response) {
    NSLog(@"mgr request(callback)");
    if (response != nil) {
        NSLog(@" - response is not null");
        NSLog(@" - response - result: %d", [response result]);
        if ([response result] == DConnectMessageResultTypeOk) {
            NSString *notificationId = [response stringForKey:
DConnectNotificationProfileParamNotificationId];
            NSLog(@" - response - OK notificationId: %@", notificationId);
            result = YES;
        } else {
            NSLog(@" - response - errorCode: %d", [response errorCode]);
        }
    }

    /* Wait 解除 */
    dispatch_semaphore_signal(semaphore);
}];
```

Device Connect マネージャー（DConnectManager）はシングルトンで、それのインスタンスに対してリクエストを渡し、そのリクエストに対するレスポンスを block で受け取っている。
iOS ネイティブアプリからリクエストを送信しパラメータを受信するプロセスはこの様になるので、リクエストを送信したいプロファイルの API に応じて DConnectRequestMessage に設定するパラメータを適切に変更する事。

では次に、4 節で触れた 5 プロセスのうち第 5 のプロセスで行った DeviceOrientation プロファイルからリクエスト送信・レスポンス受信のコードを見てみる。プロジェクトの ViewController.m の関数「`- (IBAction) onDeviceOrientaionEvent: (id)sender`」のイベント登録およびイベントメッセージ受領コールバックの設定が以下のコードになる。

```
[[DConnectEventHelper sharedHelper]registerEventWithRequest: request
                responseHandler: ^(DConnectResponseMessage *response)
{
    dispatch_async(dispatch_get_main_queue(), ^{
        NSString *message = nil;
        if ([response result] == DConnectMessageResultTypeOk) {
            message = @"処理成功しました。";
        } else {
            message = @"処理失敗しました。";
        }
        [deviceOrientationInfo   setText:   [NSString   stringWithFormat:   @"event:   %@",
message]];
    });
}
                 messageHandler:^(DConnectMessage *message)
{
    dispatch_async(dispatch_get_main_queue(), ^{

        DConnectMessage        *orientation        =        [message        messageForKey:
DConnectDeviceOrientationProfileParamOrientation];

        DConnectMessage     *acceleration     =     [orientation           messageForKey:
DConnectDeviceOrientationProfileParamAcceleration];



        double x = [acceleration doubleForKey: DConnectDeviceOrientationProfileParamX];
        double y = [acceleration doubleForKey: DConnectDeviceOrientationProfileParamY];
        double z = [acceleration doubleForKey: DConnectDeviceOrientationProfileParamZ];

        [deviceOrientationEventInfo setText: [NSString stringWithFormat: @"acceleration:
x=%.1lf, y=%.1lf, z=%.1lf", x, y, z]];

    });
}];
```

DConnectEventHelper の関数「-registerEventWithRequest:
responseHandler:messageHandler:」がイベント周りの処理を引き受けています。特に、
messageHandler:の引数にする block は、非同期で送信されてくるイベントメッセージの受け口になるので、イベントメッセージを受け取った際の挙動はその block に記述する事になる。

6. UI アプリを自作するにあたって

本チュートリアルでは要点だけをかいつまんで説明した。そして Device Connect には多くのプロファイルが用意されており、また様々なデバイスプラグインも存在している。
Device Connect はさまざまなデバイスを共通のプロトコルでアクセスできる様にする規格である。ユーザエクスペリエンスの向上の為にも、開発者の皆様がプロファイル毎の機能やリクエストやレスポンスの詳細に関し、RESTful/iOS Message/Android API などのドキュメントに目を通してある程度把握することをお勧めする。そして UI アプリの対象とするデバイスの機能に応じ、UI アプリのデザインを決定する事にも心を砕く様にする事。

Appendix.A Sphero を UI アプリで使用する場合

dConnectDeviceSphero をアプリ側で使用する場合は、BuildSettings で以下の項目を設定する必要がある。

- Architectures
  - $(ARCHS_STANDARD_32_BIT)
- iOS Deployment Target
  - iOS 6.0
- Other Linker Flags
  - -ObjC
  - -lsqlite3
  - -all_load
  - -lstdc++

そして、「Supporting Files」フォルダにある「{{アプリ名}}-Info.plist」ファイルに以下のように「Supported external accessory protocols」に「com.orbotix.robotprotocol」属性を追加しなければいけない。

Sphero は、Sphero のライブラリの他に iOS の以下のライブラリを追加する必要がある。

- ExteranalAccessory.framework
- CoreMotion.framework

Appendix.B DICE+を UI アプリで使用する場合

dConnectDeviceDice+をアプリ側で使用する場合は、BuildSettings で以下の項目を設定する必要がある。

- Other Linker Flags
  - -ObjC
  - -all_load

DICE+は、DICE+のライブラリの他に iOS の以下のライブラリを追加する必要がある。

- CoreBluetooth.framework

Appendix.C Pebble を UI アプリで使用する場合

dConnectDevicePebble をアプリ側で使用する場合は、BuildSettings で以下の項目を設定する必要がある。

- Other Linker Flags
  - -ObjC

そして、「Supporting Files」フォルダにある「{{アプリ名}}-Info.plist」ファイルに以下のように「Supported external accessory protocols」に
「com.getpebble.public」属性を追加しなければいけない。

Sphero は、Sphero のライブラリの他に iOS の以下のライブラリを追加する必要がある。

- ExteranalAccessory.framework
- CoreMotion.framework
- CoreBluetooth.framework
- Libz.dylib
- CFNetwork.framework
- MessageUI.framework