
RECO SDK for iOS Documentation

Release 0.1.1

2014 Perples Inc.

May 21, 2014

CONTENTS

1	Pre-requisites	3
1.1	iOS SDK Version	3
2	Getting Started	5
2.1	Configuration for the SDK	5
2.2	RECO SDK	5
2.3	Monitoring RECOBeaconRegion	7
2.4	How to use SDK	9
2.5	Behaviors of monitoring	10

This document describes how to use the RECO SDK of the Perples Inc. to develop ios application.

RECO SDK includes the followings:

1. RECO SDK Framework

`Reco.framework`

2. Sample code

`reco-client-ios-sample`

Note: This document is based on the Xcode 5.1.1

PRE-REQUISITES

The followings are prerequisites to use RECO SDK framework for iOS.

1.1 iOS SDK Version

RECO SDK supports the following iOS SDK version or higher.

- iOS Deployment Target : iOS 7.0 or higher
- CoreLocation.framework (required)

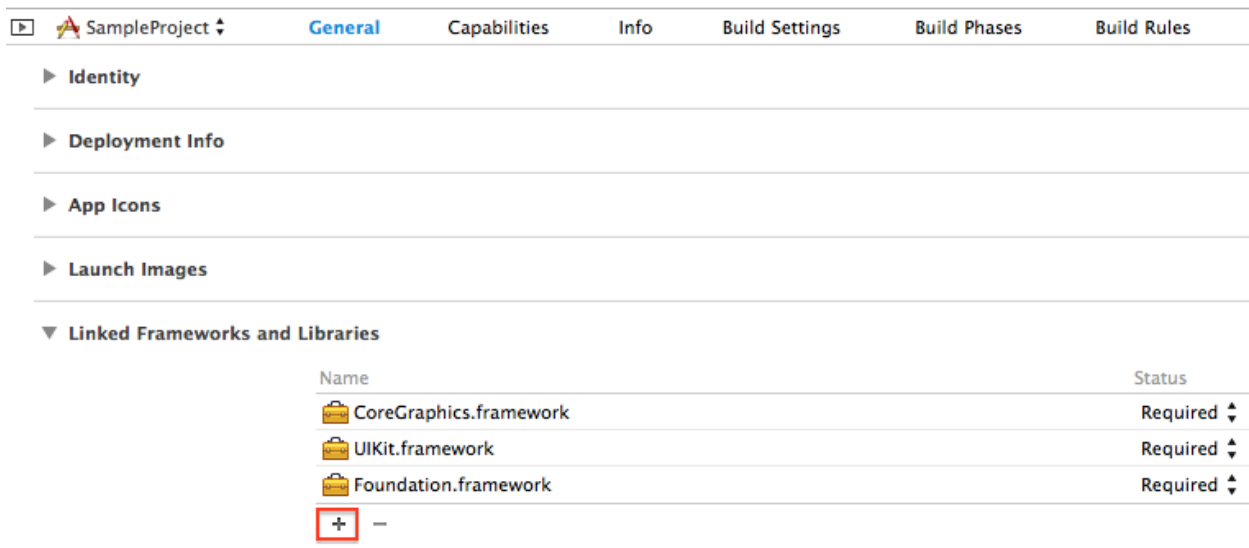
GETTING STARTED

2.1 Configuration for the SDK

Note: This document is based on the Xcode 5.1.1

The RECO SDK for iOS is provided in the form of Framework.

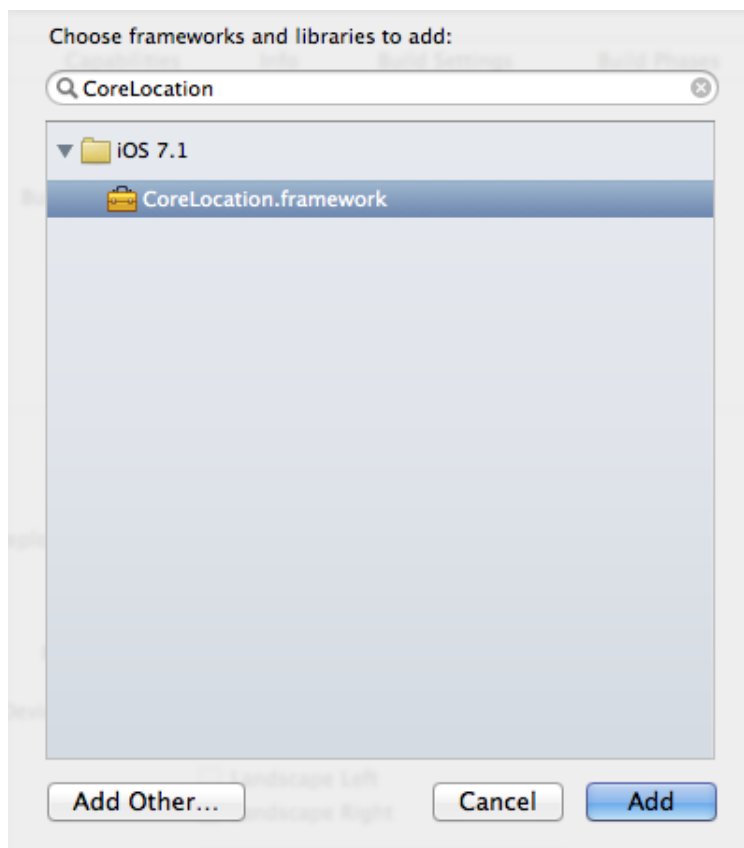
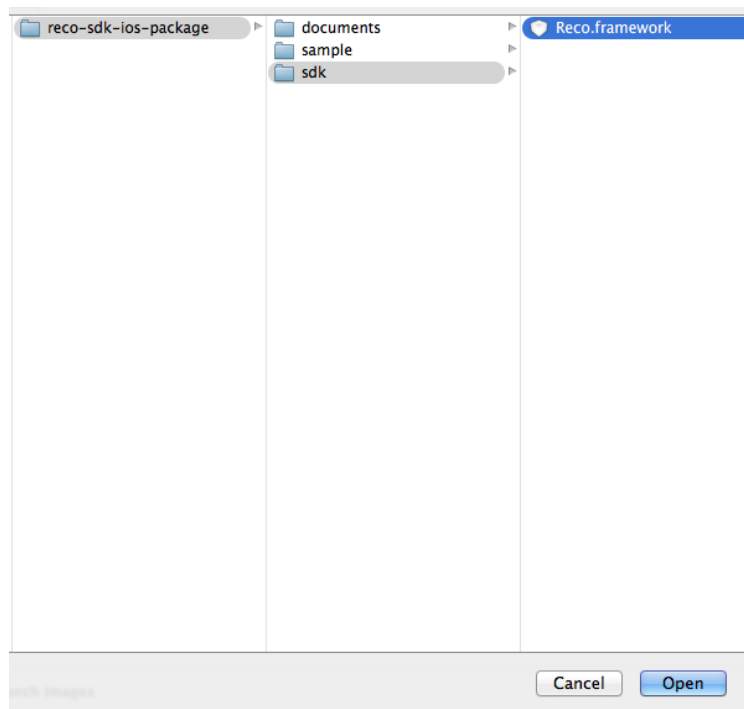
1. Copy Reco.framework directory in the SoundTag SDK Package to your project.
2. Open your project in Xcode, and select “General” by selecting the project.
3. Click “+” button in “Linked Frameworks and Libraries” in “General” tab to choose frameworks and libraries.



4. Choose the SDK copied above by selecting “Add Other...”.
5. Add other required libraries.
 - CoreLocation.framework (Required)

2.2 RECO SDK

RECO SDK is based on CoreLocation.framework provided by iOS and conforms to the iBeacon standard. Region monitoring and region ranging are the two basic functions provided by this SDK. A beacon region is an area defined



by the device's proximity to Bluetooth low-energy beacons. The only role of a RECOBeacon is to broadcast the bluetooth signal containing data which is complying with the iBeacon standard. In iOS, regions associated with your app are tracked at all times, including when the app isn't running. If a region boundary is crossed while an app isn't running, that app is relaunched into the background to handle the event.

2.3 Monitoring RECOBeaconRegion

Beacon region monitoring uses an iOS device's onboard radio to detect when the user is in the vicinity of Bluetooth low-energy devices that are advertising iBeacon information. You can use this capability to generate alerts or to provide other relevant information when the user enters or exits a beacon region. A beacon region is identified by the device's proximity to Bluetooth low-energy beacons that advertise a combination of the following values:

- Proximity UUID (universally unique identifier) which is a 128-bit value that uniquely identifies one or more beacons as a certain type or from a certain organization
- Major : 16-bit unsigned integer that can be used to group related beacons that have the same proximity UUID
- Minor : 16-bit unsigned integer that differentiates beacons with the same proximity UUID and major value

Because a single beacon region can represent multiple beacons, beacon region monitoring supports several interesting use cases. For example, an app dedicated to enhancing the experience of customers at a particular department store can use the same proximity UUID to monitor all stores in the department store chain. When the user approaches a store, the app detects the store's beacons and uses the major and minor values of those beacons to determine additional information, such as which specific store was encountered or which section of the store the user is in. (Note that although every beacon must advertise a proximity UUID, major and minor values are optional.)

2.3.1 Defining a Beacon Region to Be Monitored

To begin monitoring a beacon region, define the region and register it with the system. You define a beacon region with the appropriate initialization method of the RECOBeaconRegion class. When you create a RECOBeaconRegion object, you specify the proximityUUID, major, and minor properties of the beacons you want to monitor (the proximity UUID is required; the major and minor values are optional). You must also supply a string that uniquely identifies the region so that you can refer to it in your code.

Note: A region's identifier is unrelated to the identifying information that a beacon advertises.

To register a beacon region, call the `startMonitoringForRegion:` method of your RECOBeaconManager object.

```
- (void)registerBeaconRegionWithUUID:(NSUUID *)proximityUUID
    andIdentifier:(NSString*)identifier {

    // Create the beacon region to be monitored.
    RECOBeaconRegion *recoRegion = [[RECOBeaconRegion alloc]
        initWithProximityUUID:proximityUUID
        identifier:identifier];

    // Register the beacon region with the RECOBeaconManager
    [self.recoManager startMonitoringForRegion:recoRegion];
}
```

Monitoring of a beacon region begins immediately after registration for authorized apps. When a user's device detects a beacon that is advertising the identifying information defined by the registered beacon region (proximity UUID, major value, and minor value), the system generates an appropriate region event for your app.

Note: It's not unusual to configure a beacon region using only a UUID value. Doing so yields a region that is activated

when the device comes within range of any beacon with the specified UUID. Upon entering the beacon region, you would then begin ranging for beacons to obtain detailed information about the specific beacons that are nearby. If this value is nil, UUID will be set to the one provided by Perples Inc. by default.

2.3.2 Handling Boundary-Crossing Events for a Beacon Region

When a user enters the registered beacon region, the RECOBeaconManager calls the `recoManager:didEnterRegion:` of its delegate object. Similarly, when a user is no longer in range of any beacons in the registered beacon region, the RECOBeaconManager calls the `recoManager:didExitRegion:` of its delegate object.

You can specify which boundary-crossing events should notify your app by `setNotifyOnEntry` and `setNotifyOnExit` method of RECOBeaconRegion class.

Note: If the device is already inside of the region When the moment of calling `startMonitoringForRegion:` method, RECOBeaconManager does not call the `recoManager:didEnterRegion:` of its delegate object. There must be status change to call an appropriate delegate method.

2.3.3 Determining the Proximity of a Beacon Using Ranging

While a user's device is inside a registered beacon region, apps can use the `startRangingBeaconsInRegion:` method of the RECOBeaconManager class to determine the relative proximity of one or more beacons in the region and to be notified when that distance changes.

Note: Always call the `isRangingAvailable` class method of the RECOBeaconManager class before attempting to range beacons in a beacon region.

Knowing the relative distance to a beacon can be useful for many apps. For example, imagine a museum that places a beacon at each exhibit. A museum-specific app could use a particular exhibit's proximity as a cue to provide information about that exhibit rather than another.

The RECOBeaconManager calls the `locationManager:didRangeBeacons:inRegion:` of its delegate object whenever beacons in the specified beacon region come within range, go out of range, or their proximity changes. This delegate method provides an array of RECOBeacon objects that represent the beacons currently in range. The array of beacons is ordered by approximate distance from the device, with the closest beacon at the beginning of the array. You can use the information in these objects to determine the proximity of the user to each beacon. The value in the proximity property of the RECOBeacon object gives a general sense of the relative distance to a beacon.

Note: Beacon ranging depends on detecting the strength of Bluetooth low-energy radio signals, and the accuracy of those signals is attenuated (or lessened) by walls, doors, and other physical objects. The signals are also affected by water, which means the human body itself will affect the signals. It is important to be aware of these factors when planning your iBeacon deployment because they will impact the proximity value reported by each beacon. If needed, use the accuracy and rssi values reported by each RECOBeacon object to adjust the placement of your beacons during deployment.

```
// delegate method of RECOBeaconManager class
- (void)recoManager:(RECOBeaconManager*)manager
    didRangeBeacons:(NSArray *)beacons
    inRegion:(RECOBeaconRegion*)region {

    if ([beacons count] > 0) {
        RECOBeacon *nearestExhibit = [beacons firstObject];
    }
}
```

```

        // Present the exhibit-specific UI only when
        // the user is relatively close to the exhibit.
        if (RECOProximityNear == nearestExhibit.proximity) {
            [self presentExhibitInfoWithMajorValue:nearestExhibit.major.integerValue];
        } else {
            [self dismissExhibitInfo];
        }
    }
}

```

Note: If multiple beacon devices are advertising the same combination of proximity UUID, major values, and minor values, they may be reported by the `recoManager:didRangeBeacons:inRegion:` method as having different proximities and accuracies. It's recommended that each beacon device be uniquely identified.

2.4 How to use SDK

The following describes how to use RECO SDK.

1. Import SDK

```
#import <Reco/Reco.h>
```

2. Implement delegate in accordance with protocol

```

- (void) recoManager:(RECOBeaconManager *)manager didEnterRegion:(RECOBeaconRegion *)region {
    NSLog(@"Background: didEnterRegion %@", region.identifier);
    // Notification when a user enters the region
}

- (void) recoManager:(RECOBeaconManager *)manager didExitRegion:(RECOBeaconRegion *)region {
    NSLog(@"Background: didExitRegion %@", region.identifier);
    // Notification when a user exits the region
}

- (void) recoManager:(RECOBeaconManager *)manager didRangeBeacons:(NSArray *)beacons
    inRegion:(RECOBeaconRegion *)region {
    NSLog(@"didRangeBeaconsInRegion: %@", ranged %lu beacons",
        region.identifier, (unsigned long)[beacons count]);
    // The information of beacons in the region is passed through (NSArray *)beacons
    // in the form of RECOBeacon objects.
}
// etc.

```

3. Make an instance of RECOBeaconManager class.

```
RECOBeaconManager *recoManager = [[RECOBeaconManager alloc] init];
```

4. Define a RECOBeaconRegion

```

NSUUID *proximityUUID = [[NSUUID alloc] initWithUUIDString:@""];
NSString *identifier = @"myRecoRegion";
RECOBeaconRegion *recoRegion = [[RECOBeaconRegion alloc] initWithProximityUUID:proximityUUID
    identifier:identifier];

// Notify on both when user enters and exits the region.
[recoRegion setNotifyOnEntry:YES];
[recoRegion setNotifyOnExit:YES];

```

5. Get notified with the corresponding delegate method when monitoring and ranging.

```
// For monitoring
[recoManager startMonitoringForRegion:recoRegion];

// For ranging
[recoManager startRangingBeaconsInRegion:recoRegion];
```

6. Stop monitoring and ranging.

```
[recoManager stopMonitoringForRegion:recoRegion];

[recoManager stopRangingBeaconsInRegion:recoRegion];
```

please refer to the sample project for the details.

2.5 Behaviors of monitoring

There are two events for monitoring as a user enters or exits the region. Users location is determined by iOS itself, there might be a time lag as following.

- When monitoring regions and an application is in background, the time lag for an event to occur might be maximum 15 minutes.
- In most cases, the time lag is much shorter than 15 minutes. When we, Perples Inc, tested our SDK, the region enter event usually occurred within 30 seconds, and the region exit event occurred with in 1 minutes.

Note: `recoManager:didRangeBeacons:inRegion:` method gets called every second when ranging and ranging is NOT AVAILABLE in background.
