InnoQuant
big data analytics

MOCA SDK for iOS 7

Proximity Experience and Analytics API.

Technical overview.

V1.1.0

InnoQuant S.L.

MOCA SDK for iOS 7

Contents

- Technical overview
- Framework Architecture
- Getting started
  - Installing SDK
  - Adding SDK to your project
  - Setting up the SDK
- MOCA APIs
  - Instance API
  - User API
  - Events API
  - Push API
  - Proximity API
- Design considerations
  - Location service authorization
  - How event tracking works
  - Background modes
  - Offline behavior
  - iOS iBeacon issues

MOCA SDK
for iOS 7

Technical Overview

- The **MOCA SDK** lets you effortlessly add iBeacon proximity experiences and analytics to your iOS app.

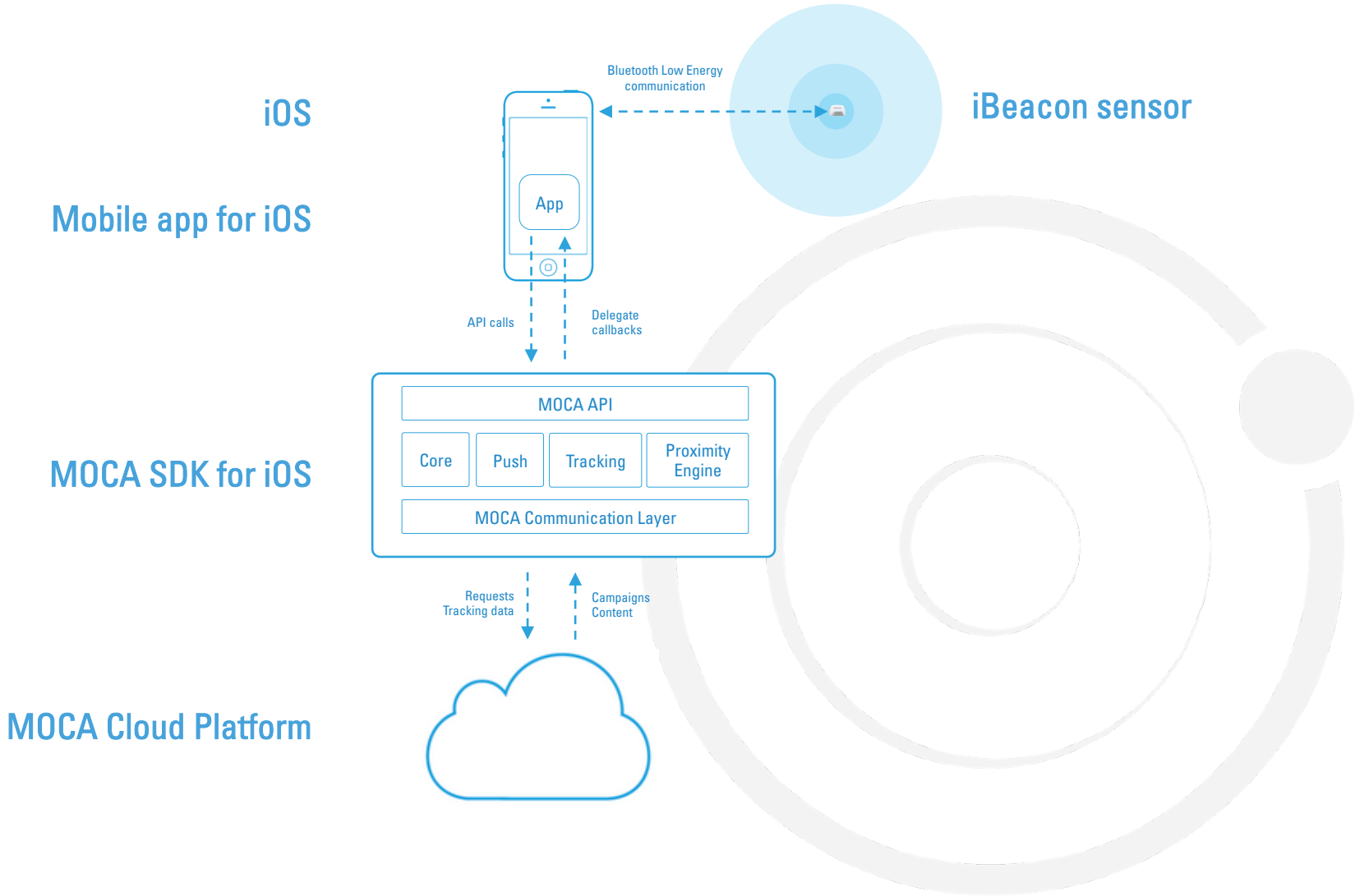
- It enables you to quickly connect to **MOCA platform** account, deploy beacon experiences, and track your users.

Technical Overview

- The **MOCA SDK** adds the following key features to your app:
  - **Proximity**
    - Automatically detect all beacon sensors as defined in your cloud account.
    - Automatically fetch and deploy proximity experiences from the cloud
    - Enrich user experience by delivering proximity actions (Video/Image/Notification/Sound/Custom) when proximity conditions are met (Enter/Exit place/zone/beacon).
  - **Analytics**
    - Automatically track the device properties, app usage, and iBeacon-related events
    - Track and store any custom, in-app events from your app
    - Store all tracked events locally, and transmit them to the cloud when Internet connectivity (Edge, 3G, 4G, Wifi) is available.

MOCA SDK
for iOS 7

MOCA Framework Architecture



Installing SDK

1. Download latest stable version of MOCA SDK archive.
2. Xcode with the iOS development kit is required to build an iOS app using MOCA SDK.
3. The SDK requires iOS 7.0 or later.
4. Unzip the archive.

Adding SDK to your app project

Once downloaded the SDK, you'll need to add all necessary frameworks to your project.

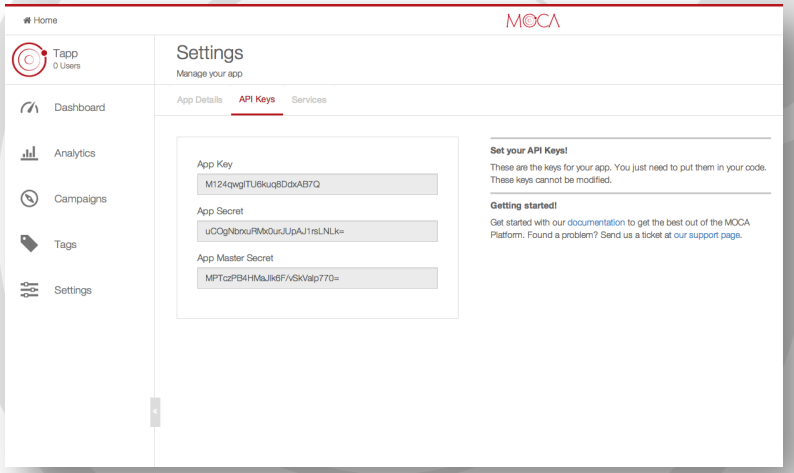
1. Open your project in Xcode.
2. Navigate to where you downloaded the SDK and drag the **MOCA.framework folder** into your project in Xcode.
3. Make sure Copy items into destination group's folder is selected.
4. Press the Finish button.
5. Ensure that you have AudioToolbox.framework, CoreTelephony.framework, MobileCoreServices.framework, SystemConfiguration.framework, CoreLocation.framework, SystemConfiguration.framework added to your project.

   To do this, select your project file in the file explorer, select your target, and select the Build Phases sub-tab. Under Link Binary with Libraries, press the + button, to select and add all required frameworks.

Setting up the SDK

To start using MOCA SDK in your app, you'll need to configure it first.

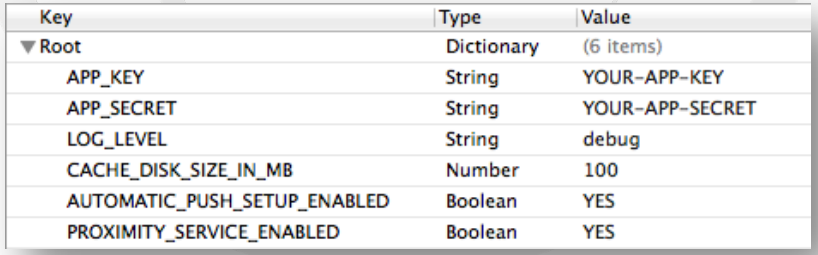
1. Goto manage.mocaplatform.com and sign in to your MOCA account.
2. Select *Apps* item at left sidebar, and then click *+ New App* in the content panel. Fill in the form and complete the app creation.
3. Open newly created app and navigate to *Settings* item. Select *API keys* tab.
4. Get **App Key** and **App Secret**.



Setting up the SDK

Now, you'll need to prepare SDK configuration file:

1. Go back to your Xcode project.
2. Add *New file / Property List* resource to your project. Save it as **MOCAConfig.plist** file.
3. Add configuration settings as shown below.
4. Be sure to replace APP_KEY and APP_SECRET values with the real values for your app which you found in the MOCA console.
5. Your may also download sample MOCAConfig.plist file.



Key	Type	Value
▼ Root	Dictionary	(6 items)
APP_KEY	String	YOUR-APP-KEY
APP_SECRET	String	YOUR-APP-SECRET
LOG_LEVEL	String	debug
CACHE_DISK_SIZE_IN_MB	Number	100
AUTOMATIC_PUSH_SETUP_ENABLED	Boolean	YES
PROXIMITY_SERVICE_ENABLED	Boolean	YES

Setting up the SDK

Finally, to start using SDK, you'll need to initialize MOCA framework.

1. Import <MOCA/MOCA.h> header file into your app's delegate implementation file.
2. A good place to initialize MOCA SDK is in you app's delegate –`application:didFinishLaunchingWithOptions:` method.

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

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Initialize MOCA SDK.
    BOOL mocaReady = [MOCA initializeSDK];
    if (!mocaReady)
    {
        NSLog(@"MOCA SDK initialization failed.");
        return NO;
    }
    return YES;
}
```

3. You must initialize the SDK before calling any other method.
4. On initialization, MOCA SDK will load configuration from MOCAConfig.plist file and perform all necessary framework setup.
5. The `[MOCA initializeSDK]` method call returns YES on success, and NO on error.

MOCA APIs

The **MOCA shared object** is a main entry point to MOCA SDK APIs.

```
@interface MOCA : NSObject

// Gets the version of the MOCA library.
+ (NSString*) version;

// Initializes the library with the configuration from MOCAConfig.plist resource file
+ (BOOL) initializeSDK:(MOCAConfig *)config;

// Gets library configuration.
+ (MOCAConfig*) config;

// Gets the application key once successfully initialized.
+ (NSString*) appKey;

// Gets the application secret once successfully initialized.
+ (NSString*) appSecret;

// Returns `YES` if the MOCA library has been initialized and is ready for use.
+ (BOOL) initialized;

// Gets the current MOCA app instance object.
+ (MOCAInstance*) currentInstance;

// Gets the current proximity service object.
+ (MOCAProximityService*) proximityService;

// Gets the current log level of MOCA library.
+ (MOCALogLevel) logLevel;

// Sets the log level.
+ (void) setLogLevel:(MOCALogLevel)logLevel;

// Tells MOCA that it can begin a cloud fetch operation if it has data to download.
+(BOOL)performFetchWithCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler;

// Updates the push device token and registers the token with MOCA cloud.
+(void)registerDeviceToken:(NSData*)deviceToken;

// Tells MOCA that a push notification has been received by the application.
+(void)handleNotification:(NSDictionary *)userInfo applicationState:(UIApplicationState)applicationState;
```

Instance API

- The app **Instance object** is a local representation of an app instance downloaded and installed in a user device.
- The instance object is **automatically managed** by MOCA.

```
MOCAInstance * theInstance = [MOCA currentInstance];
```

- The instance object holds a **collection of properties**. Property-value pairs can be set and retrieved.

```
/**
 * Sets the value for the given property name.
 *
 * @param value Value to be set. It must belong to one of the accepted classes.
 * @param prop Property name.
 */
- (void) setValue:(id)value forProperty:(NSString*)prop;

/**
 * Gets the value for the given property.
 *
 * @param prop Property name.
 * @return Value associated with the property or <code>nil</code> if none.
 */
- (id) valueForProperty:(NSString*)prop;

/**
 * Returns a new array containing all existing property names.
 *
 * @return A new array containing all existing property names.
 */
- (NSArray*) allProperties;
```

Instance API

The instance object holds predefined properties collected automatically:

- ***_apns_token*** – the push notification device token (optional)
- ***_moca_version*** – the version of MOCA SDK
- ***_device_model*** – the device model (i.e. iPhone 4S, iPhone 5C, iPhone 5S)
- ***_device_type*** – the type of device (i.e. iPhone, iPad, Simulator)
- ***_device_name*** – the user provided device name
- ***_local_timezone*** – the local time zone abbreviated name (i.e. EDT, GMT, PST)
- ***_local_timezone_name*** – the geopolitical region ID that identifies local time zone
- ***_country*** – the country code as defined by the user in Settings\General\International\Regional Format (NSLocaleCountryCode).
- ***_lang*** – the user preferred language code as defined in Settings\General\International\Language (NSLocaleLanguageCode).
- ***_system_name*** – the OS name (i.e. "iOS")
- ***_system_version*** – the OS version (i.e. "7.0")
- ***_carrier_name*** – the name of the user's home cellular service provider (CTCarrier carrierName)
- ***_carrier_country*** – the ISO country code for the user's cellular service provider.
- **_vendor_id** – the unique identifier for app vendor (i.e. UUID string)

Instance API

- The object may also store any **custom property pairs**.
- Each instance object is **persisted both locally and in the cloud**.
- It is recommended to perform as many sets as desired and then just invoke a single save operation to persist them to the MOCA cloud.

```
#import <AdSupport/ASIdentifierManager.h>


MOCAInstance * theInstance = [MOCA currentInstance];
if (theInstance)
{
    // Only track Advertising Identifier, if you app uses explicit Ad banners.
    NSString * adId = [[[ASIdentifierManager sharedManager] advertisingIdentifier] UUIDString];
    [theInstance setValue:adId forProperty:@"_ad_id"];
    [theInstance setValue:@"red" forProperty:@"favorite-color"];
    [theInstance setValue:@"women-cloth" forKey:@"last-search"];
    // Asynchronously save the instance to the cloud.
    [theInstance saveWithBlock:^(MOCAInstance *instance, NSError *error)
        {
          if (error) NSLog(@"Save instance failed: %@", error);
        }
    ];
}
```

User API

- The app User object manages information about current application user.
- This object is optional, and it is used when the app needs to authentify the user.
- The authentication provides data about user's identity such as *e-mail address*, *unique user ID* or similar.
- The user login and logout calls are managed by the app.

User API

To access existing **User object**, use *currentUser* property:

```
// Access authenticated user object
MOCAUser * theUser = [[MOCA currentInstance] currentUser];
if (theUser)
{
    // ...
}
```

To **authenticate** a new user, use *login:id* call:

```
// Authenticate a user by e-mail address
MOCAUser * currentUser = [[MOCA currentInstance] login:@"john@mycompany.com"];
if (currentUser)
{
   // ...
}
```

To **logout** a user, call:

```
[theUser logout];
```

User API

- The user object holds a collection of properties. Property-value pairs can be set and retrieved.
- Each user object is persisted both locally and in the cloud.
- It is recommended to perform as many sets as desired and then just invoke a single save operation to persist them to the MOCA cloud.

```
MOCAUser * theUser = [[MOCA currentInstance] currentUser];
if (theUser)
{
    [theUser setValue:@"male" forProperty:@"gender"];
    [theUser setValue:[NSNumber numberWithInt:1975] forProperty:@"birth_year"];
    // Asynchronously save the user object to the cloud.
    [theUser saveWithBlock:^(MOCAUser *user, NSError *error)
        {
          if (error) NSLog(@"Save user failed: %@", error);
        }
    ];
}
```

Event API

- The **Event object** provides access to the MOCA Analytics API.
- The Analytics API **captures and transmits** to the cloud the **flow of arbitrary events** associated with the app.
- A single event object represents the **record** that indicate that **something has happened**
  - Such as App started, Push notification received, Beacon detected, etc.

Event API

A single event object record contains:

– Event data:

- Identifier – unique event ID (UUID)
- Timestamp – GMT timestamp when event occurred
- Local timestamp – timestamp with local timezone
- Event type – type of event
- Action verb – string that described performed action (i.e. Buy, View, Search, Enter, etc.)
- Item – item associated with the event (i.e. Product ID)
- Item Category – category the item belongs to (Product Category)
- Value – value associated with the event (E.g. purchase amount)

– Context data:

- Instance ID – the instance object ID that generated the event
- User ID – the logged-in user that generated the event (optional)
- Device model/type
- OS name/version
- Country
- Language

MOCA SDK
for iOS 7

Event API

By default, SDK automatically tracks predefined events:

- _is_active – indicates if the app is active
- _geolocation – track last coordinates of the current location *
- _new_instance – when the app is executed for the first time
- _new_session – when the new app session was started
- _exit_app – when app was exited
- _app_leaves_fore – when app leaves foreground mode
- _app_goes_fore – when app enters foreground mode
- _launching_push – when app was started by a push notification
- _push – when push notification has been received while the app was running
- _beacon_proximity – when the device entered/exited a beacon region *
- _zone_proximity – when the device entered/exited a proximity zone *
- _place_proximity – when the device entered/exited a proximity place *
- _fire_action – when proximity experience action fired
- _user_login – when user logged in
- _user_logout – when user logged out

* Used when available and allowed by user privacy settings.

Event API

To track custom events in your app, use the following API:

```
/**
 * Tracks an event represented by an action verb
 *
 * @param verb Action verb. (E.g. Buy, View, Search, etc.).
 *
 * @return <code>YES</code> in case of success, <code>NO</code> in case of error.
 */
+ (BOOL) track:(NSString*)verb;

/**
 * Track event helper methods.
 */
+ (BOOL) track:(NSString*)verb withValue:(NSNumber*)value;
+ (BOOL) track:(NSString*)verb forItem:(NSString*)item;
+ (BOOL) track:(NSString*)verb forItem:(NSString*)item withValue:(NSNumber*)value;
+ (BOOL) track:(NSString*)verb forItem:(NSString*)item belongingTo:(NSString*)category;
+ (BOOL) track:(NSString*)verb forItem:(NSString*)item belongingTo:(NSString*)category withValue:(NSNumber*)value;
+ (BOOL) track:(NSString*)verb forItem:(NSString*)item belongingTo:(NSString*)category withIntValue:(int)value;
```

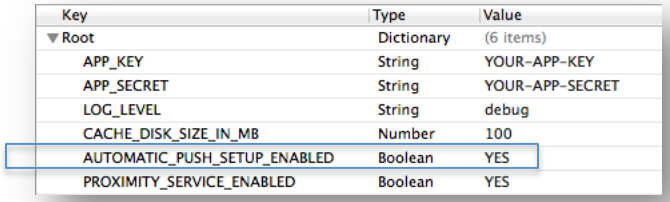
Custom tracking example:

```
-  (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if ([[segue identifier] isEqualToString:@"ViewProduct"])
    {
        NSIndexPath *indexPath = [self.tableView indexPathForSelectedRow];
        Product *product = self.products [indexPath];
        // Track view product event
        [MOCAEvent track:@"view" forItem:product.id];
    }
}
```

Push API

To start using push notifications in your app with MOCA, you'll need to enable push service.

1. Ensure automatic push setup is enabled in MOCAConfig.plist file.



Key	Type	Value
▼ Root	Dictionary	(6 items)
APP_KEY	String	YOUR-APP-KEY
APP_SECRET	String	YOUR-APP-SECRET
LOG_LEVEL	String	debug
CACHE_DISK_SIZE_IN_MB	Number	100
AUTOMATIC_PUSH_SETUP_ENABLED	Boolean	YES
PROXIMITY_SERVICE_ENABLED	Boolean	YES

2. The `[MOCA initializeSDK]` call, will register for remote push notification by means of `[[UIApplication sharedApplication] registerForRemoteNotificationTypes:]` call for you.

3. Next, implement `[didRegisterForRemoteNotificationsWithDeviceToken:]` app delegate callback:

Push API

3. Next, implement `[didRegisterForRemoteNotificationsWithDeviceToken:]` app delegate callback:

```
- (void)application:(UIApplication*)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData*)deviceToken
{
    NSLog(@"APNS token: %@", deviceToken);
    // Register push token in MOCA
    [MOCA registerDeviceToken:deviceToken];
    [_instance saveWithBlock:^(MOCAInstance *instance, NSError *error)
    {
        if (error) NSLog(@"Save APNS token failed: %@", error);
    }];
}
```

4. Complete, the process by implementing the app delegate callbacks:

```
-  (void)application:(UIApplication*)application didFailToRegisterForRemoteNotificationsWithError:(NSError*)error
{
   // Handle error
}

- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo
{
    // Notify MOCA that push notification has been received.
   [MOCA handleNotification:userInfo applicationState:application.applicationState];
}
```

MOCA SDK
for iOS 7

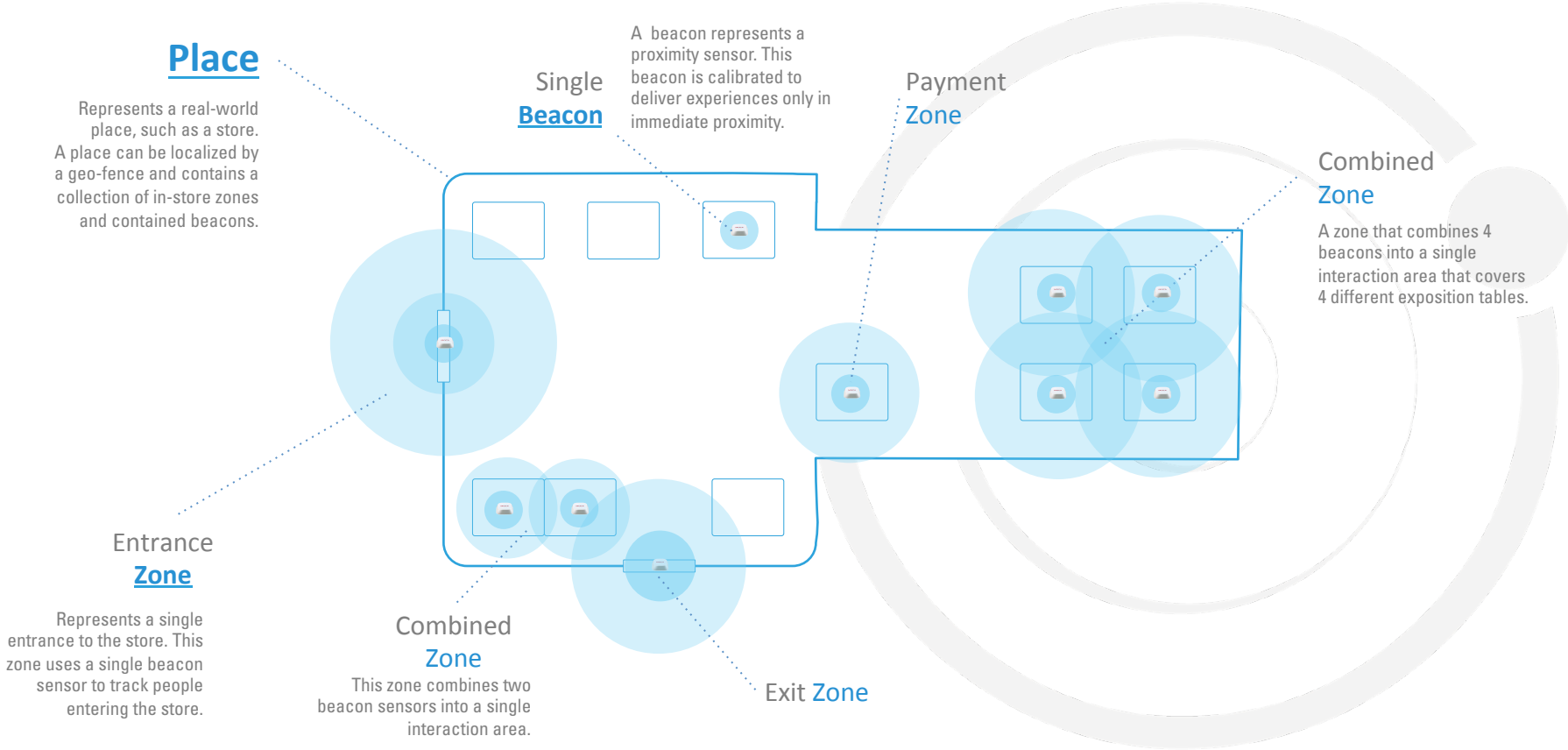
Proximity API

- The **MOCA SDK** lets you effortlessly add iBeacon proximity experiences to your iOS app.
- The SDK comes with powerful **proximity engine** based on Complex Event Processing logic (see CEP).
- MOCA defines some fundamental concepts:
  - Place, Zone, and Beacon
  - Trigger and Action

MOCA SDK
for iOS 7

Proximity API

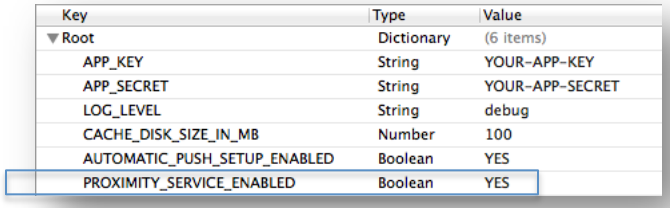
To map real-world context, MOCA introduces some fundamental concepts:



Proximity API

To start using **iBeacon-based proximity experiences** in your app, you'll need to enable proximity service.

1. Ensure proximity service is enabled in MOCAConfig.plist file.



Key	Type	Value
▼ Root	Dictionary	(6 items)
APP_KEY	String	YOUR-APP-KEY
APP_SECRET	String	YOUR-APP-SECRET
LOG_LEVEL	String	debug
CACHE_DISK_SIZE_IN_MB	Number	100
AUTOMATIC_PUSH_SETUP_ENABLED	Boolean	YES
PROXIMITY_SERVICE_ENABLED	Boolean	YES

2. The `[MOCA initializeSDK]` call, will automatically start MOCA proximity engine and synchronize proximity campaigns defined in your MOCA cloud account.

Proximity API

- **Places** – represents a real-world locations, such as a store, venue, museum, building, office and so on.
  - A place can be localized by a geo-fence (GPS location and radius in meters)
  - A place contains a collection of *zones*, where each zone defines semantics for an indoor location, such as *Kids Clothes, Women Clothes, Entrance, Exit, Books, Videogames, etc.*
- **Zone** – represents a indoor area or micro-location within the place with user-defined semantics.
  - Examples include *ds Clothes, Women Clothes, Entrance, Exit, Books, Videogames, etc.*
  - A zone is defined by a range of one or more *beacon* sensors.
- **Beacon** – is a BLE, iBeacon-compatible proximity sensor.
  - Each beacon is identified by { Proximity UUID, Major and Minor } tuple.
  - MOCA SDK works with any iBeacon-compatible devices.
    - This includes MOCA Beacons
    - And beacons provided by other manufacturers such as Kontakt.io, Estimote, etc.
  - Beacons are assigned to Zones that define their micro-location semantics.
  - Each beacon must be registered in MOCA Management Console.

Proximity API

- **Campaigns** – are collections of proximity experiences that are delivered to a user within a specified time frame.
- **Experience** – defines a proximity interaction with a user.
  - An experience is composed of trigger condition and action.
  - When proximity trigger conditions are met, the corresponding action is fired.
- **Trigger**– is a condition or a collection of conditions that must be met to fire an action associated with an experience. For example:
  - Enter place,
  - Exit place,
  - Enter zone,
  - Exit zone
  - Enter beacon range with given proximity,
  - Exit beacon range
  - Custom trigger (implemented as app-provided delegate callback)

Proximity API

- **Action** – describes an activity to be performed when experience trigger conditions are met. SDK supports the following actions:
  - Show local push notification with a specific message
  - Play sound with a specific file name (included in app bundle)
  - Play video from URL
  - Show image from URL
  - Show HTML content from URL
  - Custom action (implemented as app-provided delegate callback)

The SDK provides delegates that enable full customization of all performed actions. See `MOCAProximityActionsDelegate` in ***MOCAProximityDelegate.h*** file for **details.**

MOCA SDK for iOS 7

Proximity API

You implement MOCAProximityEventsDelegate to listen to proximity events:

```
@interface AppDelegate : UIResponder <UIApplicationDelegate, MOCAProximityEventsDelegate>

…

MOCAProximityService * proxService = [MOCA proximityService];
if (proxService)
{
    // Notify me when beacon-related events are fired.
    proxService.eventsDelegate = self;
}

/**
 * Method invoked when a proximity service loaded or updated a registry of beacons
 * from MOCA cloud.
 *
 * @param service proximity service
 * @param beacons current collection of registered beacons
 *
 * @return YES if the custom trigger fired, or NO otherwise.
 */
-(void)proximityService:(MOCAProximityService*)service
   didLoadedBeaconsData:(NSArray*)beacons
{
    NSLog(@"Current beacon registry:");
    for (MOCABeacon * beacon in beacons)
    {
        NSLog(@"\tBeacon name %@, UUID %@, Major %@, Minor %@",
           beacon.name, [beacon.proximityUUID UUIDString], beacon.major, beacon.minor);
    }
}
```

MOCA SDK for iOS 7

Proximity API

The MOCAProximityEventsDelegate exposes the following events:

```
@protocol MOCAProximityEventsDelegate <NSObject>

@optional

-(void)proximityService:(MOCAProximityService*)service didEnterRange:(MOCABeacon *)beacon withProximity:(CLProximity)proximity;
-(void)proximityService:(MOCAProximityService*)service didExitRange:(MOCABeacon *)beacon;
-(void)proximityService:(MOCAProximityService*)service didBeaconProximityChange:(MOCABeacon*)beacon
               fromProximity:(CLProximity)prevProximity toProximity:(CLProximity)curProximity;

-(void)proximityService:(MOCAProximityService*)service didEnterPlace:(MOCAPlace *)place;
-(void)proximityService:(MOCAProximityService*)service didExitPlace:(MOCAPlace *)place;

-(void)proximityService:(MOCAProximityService*)service didEnterZone:(MOCAZone *)zone;
-(void)proximityService:(MOCAProximityService*)service didExitZone:(MOCAZone *)zone;

-(BOOL)proximityService:(MOCAProximityService*)service handleCustomTrigger:(NSString*)customAttribute;

-(void)proximityService:(MOCAProximityService*)service didLoadedBeaconsData:(NSArray*)beacons;

@end
```

Proximity API

You may also customize actions using MOCAProximityActionsDelegate:

```
@protocol MOCAProximityActionsDelegate <NSObject>

@optional

-(void)displayNotificationAlert:(NSString *)alertMessage;
-(void)openUrl:(NSURL*)url;
-(void)playVideoFromUrl:(NSURL*)url;
-(void)displayImageFromUrl:(NSURL*)url;
-(void)addTag:(NSString*)tagName withValue:(NSString*)value;
-(void)playNotificationSound:(NSString *)soundFilename;
-(void)performCustomAction:(NSString*)customAttribute;

@end
```

Design considerations

Location service authorization

- By default, MOCA tracks user location (GPS) and proximity interactions.
- The geo-location is tracked with 100-meter precision.
- This is performed by means of iOS CLLocationManager
- The user must allow the location tracking for the app.
- When the permission is not granted, MOCA does not track user location.

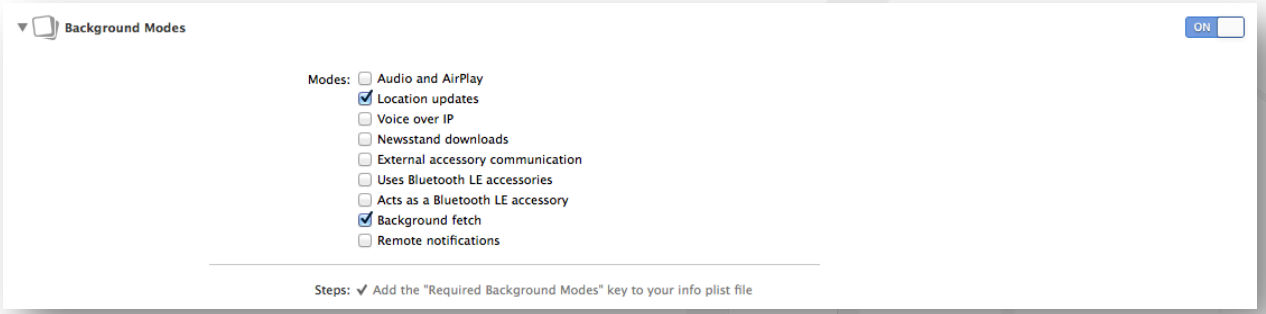
How event tracking works

- MOCA SDK uses SQLite database stored in local file on your phone to store all locally tracked events.
- Periodically, when your device is connected to Internet (via Edge, 3G, LTE or Wifi), the SDK incrementally transfers the packs of events to the cloud.
- The local database is freed only when the transfer is successful (*Store-And-Forward)*
- By default, the event transfer is performed every 60 seconds.

Design considerations

Background modes

- MOCA SDK requires you to declare *Background Modes* for your app.
- Open *Capacities* tab in your App project build settings and configure the following options:



- The **Location Updates** option is required if you wish to track user geo-location even if the app is in background.
- The **Background Fetch** option is required to periodically fetch the proximity campaign data from the cloud.

Design considerations

Offline behavior

- MOCA SDK downloads proximity campaigns data from the cloud asynchronously using *background fetch* mechanism.
- The frequency is controlled by iOS and depends on the app usage pattern.
- The downloaded data is persisted locally and servers as a cache.
- If the connection to Internet is not available, the cached data is used to run the campaigns.
- This enables you to offer proximity experiences even if the Internet connection is saturated or not available.

Design considerations

iOS iBeacon issues

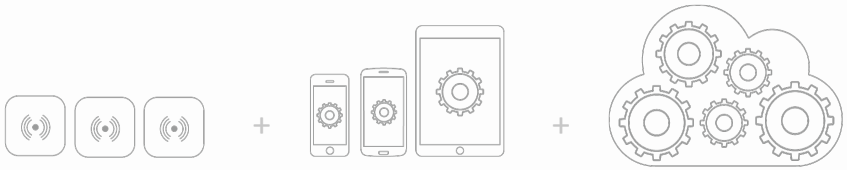
- CLProximity Toggling - iOS 7.0 and 7.1 has a known issue related to beacon unexpected proximity changes. It may happen that your have your app and a beacon is detected at a certain proximity distance, for example "Near". Both beacon and your device do not move. And suddenly the beacon ranging callback from iOS notifies you that beacon proximity changed to "Immediate", and then it switched back to "Near" again. This may happen several times.
- The reason for this behavior resides both in BLE signal interference and in iOS logic for averaging the RSSI signal strength and determining CLProximity range. The effect is that you observe sudden rapid toggles between proximity regions.
- We are working on a next version of SDK that will filter received signals, remove outliers and lower the sensitivity on region changes in order to provide smoother user experience.

MOCA Proximity Experience

The complete iBeacons platform for your app.

- ✓ Create iBeacons-aware apps.
- ✓ Deploy personalized proximity campaigns from cloud.
- ✓ Use the power of Big Data Analytics to better understand your users.

Buy your kit today!



3x iBeacon sensors + Proximity & Analytics SDK + BigData Cloud Backend Platform

Visit us at innoquant.com