

iOS - Using the SDK

1 Foreword

Copyright

The contents of this manual cover material copyrighted by Selligent. Selligent reserves all intellectual property rights on the manual, which should be treated as confidential information as defined under the agreed upon software licence/lease terms and conditions.

The use and distribution of this manual is strictly limited to authorized users of the Selligent Interactive Marketing Software (hereafter the "Software") and can only be used for the purpose of using the Software under the agreed upon software licence/lease terms and conditions. Upon termination of the right to use the Software, this manual and any copies made must either be returned to Selligent or be destroyed, at the latest two weeks after the right to use the Software has ended.

With the exception of the first sentence of the previous paragraph, no part of this manual may be reprinted or reproduced or distributed or used in any form or by any electronic, mechanical or other means, not known or hereafter invented, included photocopying and recording, or in any information storage or retrieval or distribution system, without the prior permission in writing from Selligent.

Selligent will not be responsible or liable for any accidental or inevitable damage that may result from unauthorized access or modifications.

User is aware that this manual may contain errors or inaccuracies and that it may be revised without advance notice. This manual is updated frequently.

Selligent welcomes any recommendations or suggestions regarding the manual, as it helps to continuously improve the quality of our products and manuals.

2 Table of Contents

1	Foreword	2
2	Table of Contents	3
3	Intro	5
4	Configure the APNS (Apple Push Notification Service)	6
4.1	Enable push notifications	6
4.2	Create and submit a Certificate Signing Request (CSR)	8
4.3	Install the APNS certificate and Export the .p12 file	10
4.4	Update your provisioning profiles in Xcode	11
5	Include SDK in your target	12
5.1	Import the library	12
5.2	Note for Swift project	13
5.3	Add entries to your app .plist file.....	13
5.3.1	Deep Linking.....	13
5.3.2	Permission for camera and image gallery usage.....	14
5.3.3	Permission for geo location	15
5.4	External framework.....	16
6	How to use the SDK	17
6.1	Starting sdk	17
6.2	Push notifications.....	19
6.2.1	Register for push notification	19
6.2.2	Listening and displaying the push notifications.....	19
6.2.2.1	App that does not build against iOS + 10	19
6.2.2.2	App that build against iOS + 10	20
6.2.3	Push notification helper methods.....	21
6.2.4	Broadcasts (NSNotification).....	21
6.3	In App messages	23
6.3.1	Enable IAM	23
6.3.2	Display IAM.....	23
6.3.3	Broadcasts (NSNotification).....	23
6.4	In App Content.....	25
6.4.1	Enabling IAC	25
6.4.2	Displaying IAC	25
6.4.2.1	With SDK view controllers	25
6.4.2.2	With your own view controllers	27
6.4.3	Customize IAC	28

6.4.4	Broadcasts (NSNotification)	31
6.5	Geolocation	32
6.6	Events	33
6.6.1	Registration / Unregistration	33
6.6.1.1	SMUserEventRegistration	33
6.6.1.2	SMEventUserregistration	34
6.6.2	Login/Logout	34
6.6.2.1	SMEventUserLogin	34
6.6.2.2	SMEventUserLogout	35
6.6.3	Custom	35
6.6.3.1	SMEvent	35
6.7	Broadcasts (NSNotification) summary	37
6.7.1	Push notifications – IAM – IAC event broadcasts	37
6.7.2	Data broadcasts	38
6.7.3	Examples	38
6.8	Miscellaneous	40
6.8.1	Reload	40
6.8.2	LogLevel	40
6.9	Notification Extensions	41
6.9.1	General set up	41
6.9.2	UNNotificationServiceExtension – Notification Service Extension	42
6.9.2.1	Configuration	42
6.9.2.2	Start sdk from inside extension	43
6.9.2.3	Push notification content modification before displayed to user	44
6.9.3	UNNotificationContentExtension - Notification Content Extension	47
6.9.3.1	Configure notification content extension to your project for Selligent category	47
6.9.3.2	Start sdk from inside extension	49
6.9.3.3	Push action buttons	49
6.10	Changelog	50

3 Intro

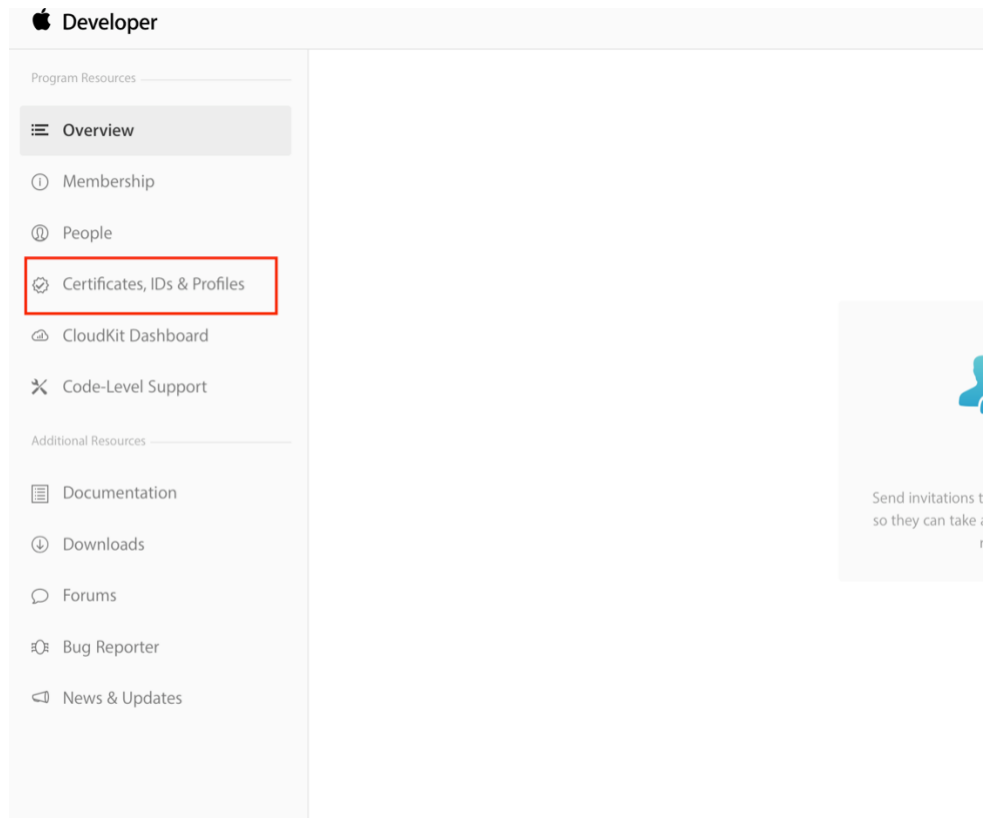
The purpose of this document is to detail how to install the SDK into your app and how to easily start using it.

- for more detailed technical reference of the sdk please refer to **IOS - MobileSDK Reference.pdf** document
- for an example of implementation check the **SMSDKTemplate** xCode project

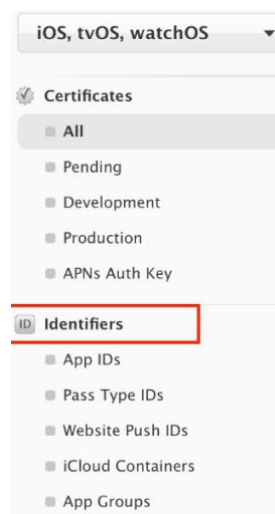
4 Configure the APNS (Apple Push Notification Service)

4.1 Enable push notifications

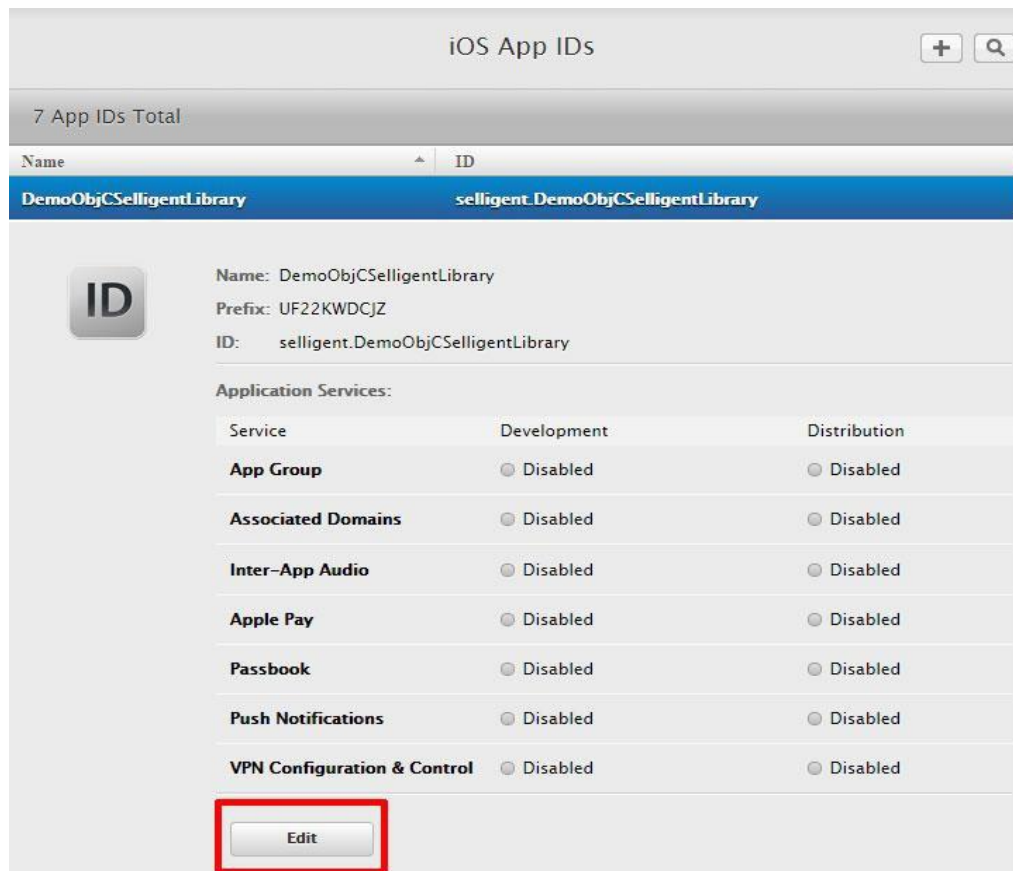
To enable push notifications, go to the [Apple Developer Portal](#) and login to the **Member centre**. When logged in, go to the **Certificates, IDs & Profiles** section to manage the certificates.



In the **Certificates, Identifiers & Profiles** page go to **Identifiers** under iOS Apps.

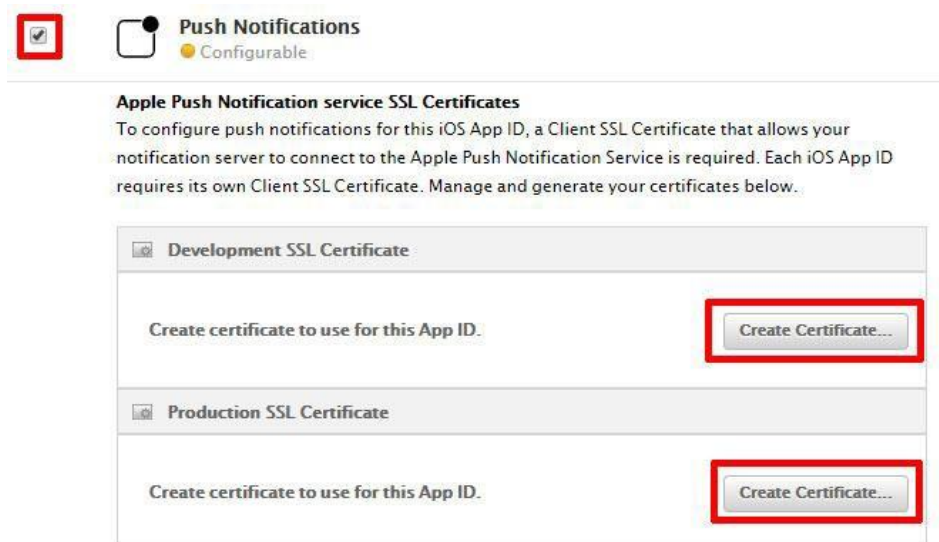


In the list of your app IDs select the app you want to enable the push notifications. Edit the application services to enable the push notification.



In the list of the application services enable Push Notifications and create an SSL Certificate for Production and/or Development.

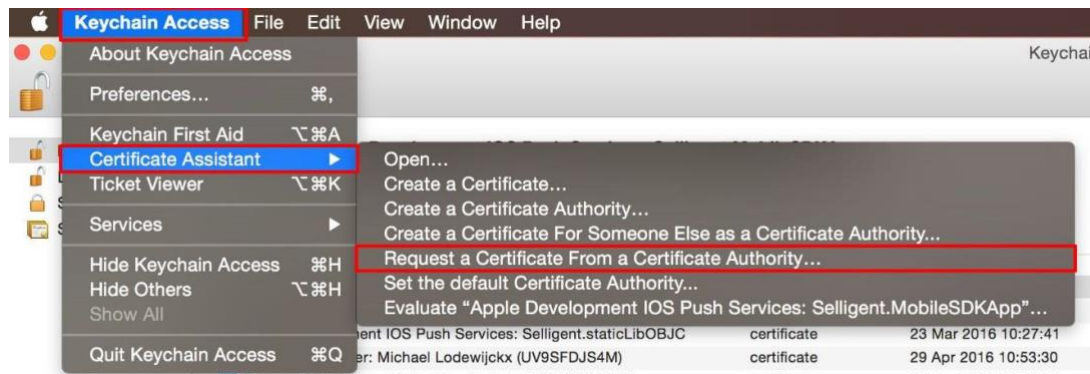
Since iOS 10 production certificate can be used for both production and development/sandbox environment.



4.2 Create and submit a Certificate Signing Request (CSR)

Before going further, we need to generate a **Certificate Signing Request (CSR)**. For this purpose, you will need the **Keychain Access** of Mac OS. Search for **Keychain Access** in Spotlight

Once **Keychain** is open, go to **Keychain Access>Certificate Assistant>Request a Certificate from a Certificate Authority**



In the opened window fill the **User Email Address** field, the **Common Name** and select the **Saved to disk** option.

Return to the **Certificate Signing Request** page, click on **Continue**, upload the **.certSigningRequest** file previously generated by the Keychain. After selecting the file, click on **Generate**.




Select Type

Request

Generate


Download

**Generate your certificate.**

With the creation of your CSR, Keychain Access simultaneously generated a public and private key pair. Your private key is stored on your Mac in the login Keychain by default and can be viewed in the Keychain Access application under the "Keys" category. Your requested certificate will be the public half of your key pair.

Upload CSR file.
Select .certSigningRequest file saved on your Mac.

Choose File...

 CertificateSigningRequest.certSigningRequest

Cancel

Back

Generate


Click on Download to get the **aps_TARGET.cer** file.

Select Type


Request

Generate

Download

**Your certificate is ready.**

Download, Install and Backup
Download your certificate to your Mac, then double click the .cer file to install in Keychain Access. Make sure to save a backup copy of your private and public keys somewhere secure.



Name: Apple Development iOS Push Services: com.selligent.demoapp

Type: APNs Development iOS

Identifier ID: SelligentDemoApp

Expires: May 11, 2016

Download

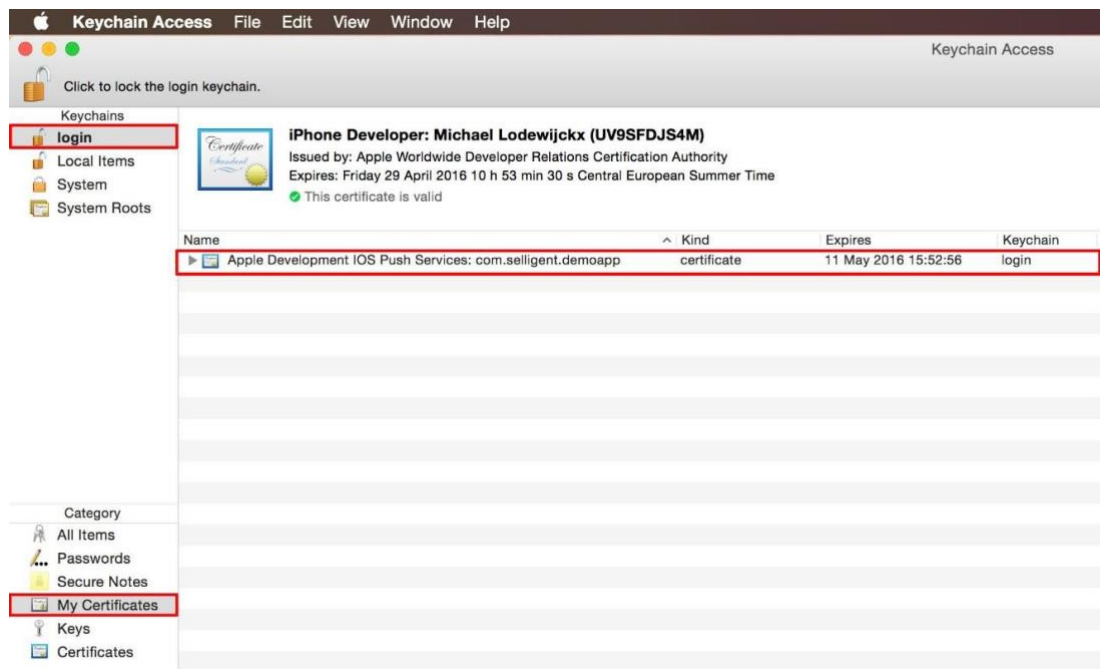
Documentation
For more information on using and managing your certificates read:
[App Distribution Guide](#)

Add Another

Done

4.3 Install the APNS certificate and Export the .p12 file

To install the generated .cer file into the Keychain Access, double click on it, it will open the Keychain Access with the installed certificate.



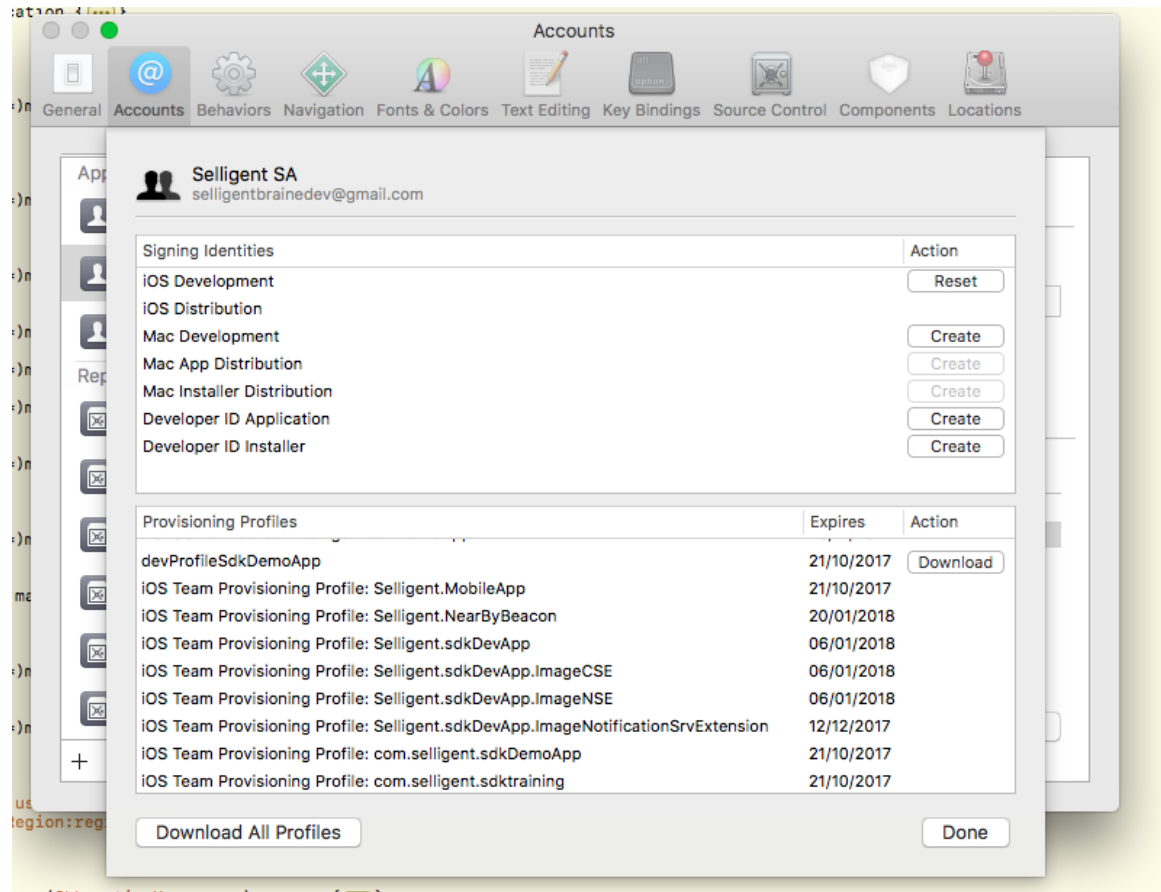
To export the .p12 file, expand the certificate, right click (or CTRL + left click) on the certificate only and select export.



It is important to expand and select only the certificate and not the private key associated with it. Otherwise the certificate may be invalid to use with selligent platform.

4.4 Update your provisioning profiles in Xcode

Don't forget to update your provisioning profiles in **Xcode/Preferences/your Apple ID** then click on **view details** button. And in the dialog, that appears you just need to click on **Download all profiles**

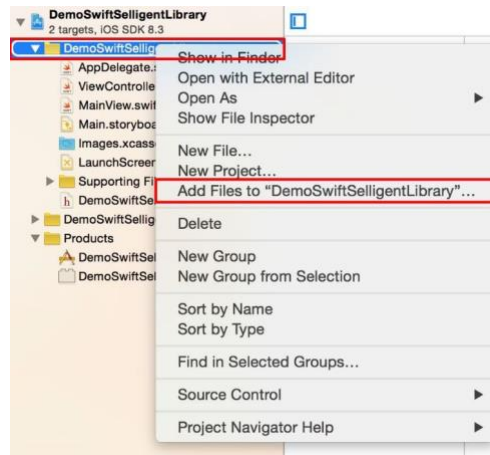


starting XCode 7 some issues have been reported to apple that the update is not always correctly refreshed and must triggered a few times to be valid

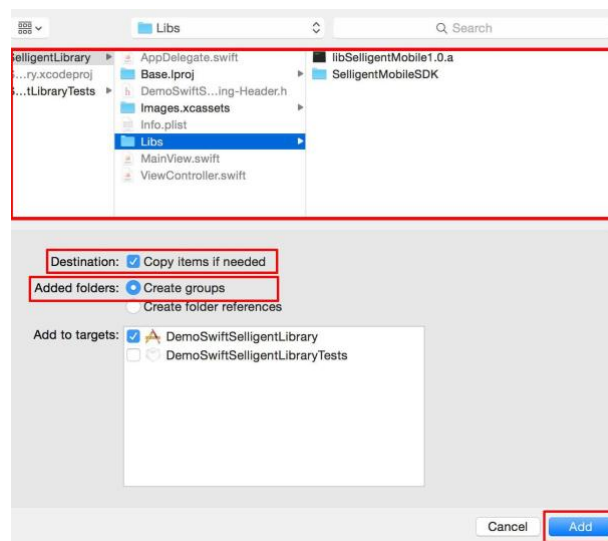
5 Include SDK in your target

5.1 Import the library

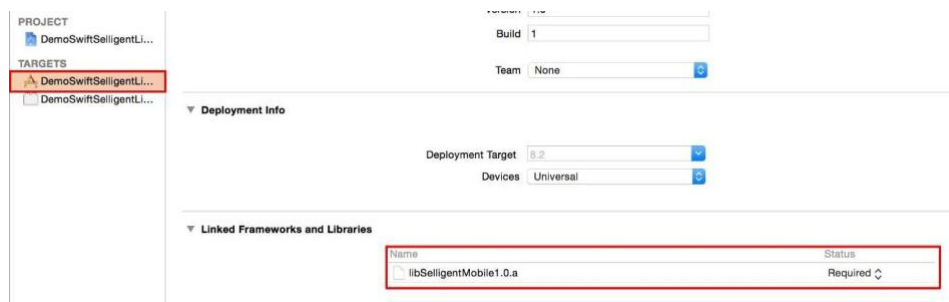
Right click (CTRL + Left click) on your app target and select **Add Files to "YOURTARGET"**



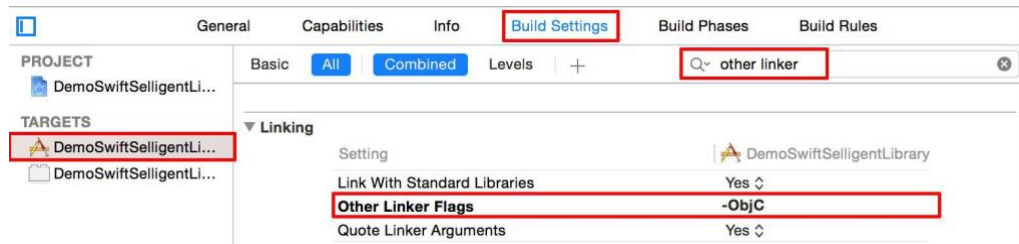
Select the lib folder (the main folder containing the header and the lib files). Depending on your project check the option **Copy item if needed** and select the **Create groups option**.



Make sure the library has been added to your target and that its status is **Required**

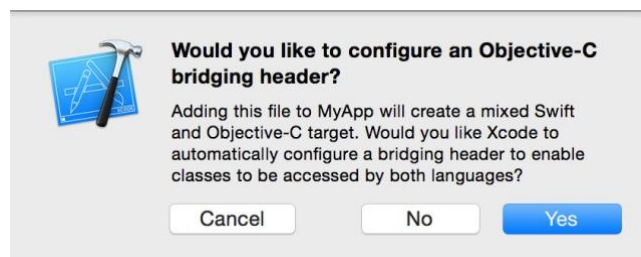


Then, go to the Build Settings of your target app, search for **Other Linker Flags** property and set the value to **-ObjC**.



5.2 Note for Swift project

For a Swift application, you need to create a Bridging-Header file. To create it automatically, add an Objective-C file to your Swift app and XCode will offer you the possibility to create this header file. If you accept, XCode creates the header file along with the file you were creating, and names it by your product module name followed by "-Bridging-Header.h".



You can also create it manually by adding a header file to your project, named **[MyProjectName]-Bridging-Header.h**. In your project build settings, find **Swift Compiler – Code Generation**, and next to **Objective-C Bridging Header** add the path to your bridging header file from the project's root folder. So, it could be **MyProject/MyProject-Bridging-Header.h** or simply **MyProject-Bridging-Header.h** if the file is in the project root folder.

In both case, you will need to import the **SMHelper.h** to expose those files to Swift. Do it by adding this line:

```
#import "SMHelper.h"
```

More information about this configuration in [apple documentation](#).

5.3 Add entries to your app .plist file

5.3.1 Deep Linking

For now, deeplinking is not possible directly when user click on push notification in notification centre. This is still done when user click on a button in the notification displayed in app.

You should configure correctly the **plist** of your app to allow this to work, by registering a custom URL scheme

```
<key>CFBundleURLTypes</key>
<array>
  <dict>
    <key>CFBundleURLName</key>
    <string>yourappbundle</string>
    <key>CFBundleURLSchemes</key>
    <array>
      <string>yourscheme</string>
    </array>
  </dict>
</array>
```

You will also have to add **LSApplicationQueriesSchemes** key with your scheme as string to allow your app to open the url:

```
<key>LSApplicationQueriesSchemes</key>
<array>
  <string> appscheme </string>
</array>
```

By doing this you will be able to parse the URL and process it like you want.

ObjectiveC:

```
NSURL *url = [NSURL URLWithString: @"yourscheme://anypage"];
[[UIApplication sharedApplication] openURL:url];
```

or when :

```
[[UIApplication sharedApplication] canOpenURL:url];
```

```
-(BOOL)application:(UIApplication*) application openURL:(NSURL*) url sourceApplication:(NSString*)
sourceApplication annotation:(id) annotation
{
    NSLog(@"%@@", [url absoluteString]);
    return YES;
}
```

If all is correctly set then when the user receives a push and click it, the app will open, an alert with the link will be displayed and the click on the link will trigger the 'appscheme://anypage'

5.3.2 Permission for camera and image gallery usage

There is a selligent push functionality – *a selligent push that will have a button that requires an answer with a picture provided by the user, the user will be able to take a photo or to pick an image from the device gallery and then send it to the platform* - that will need the usage of your camera or photo gallery access.

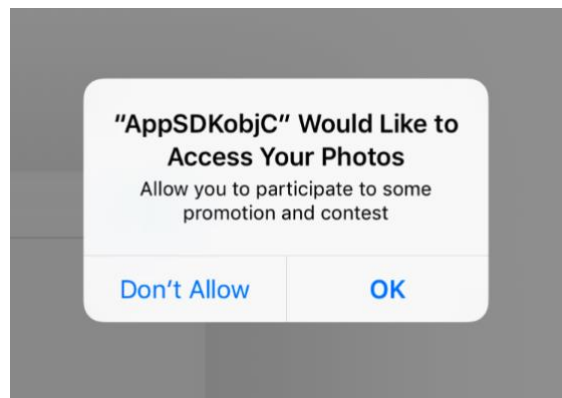
Since iOS 10 you must add these two keys in your **plist** (if not already present for your usage) to inform correctly the user of the usage of these features:

```
<key>NSCameraUsageDescription</key>
  <string>Allow you to participate to some promotion and contest</string>
<key>NSPhotoLibraryUsageDescription</key>
  <string>Allow you to participate to some promotion and contest</string>
```

You will then have those two items in your plist file (the string description is shown here just for example. It is at your convenience to describe the usage of these features the way you want)

Privacy - Camera Usage Description	String	Allow you to participate to some promotion and contest
Privacy - Photo Library Usage Description	String	Allow you to participate to some promotion and contest

And when user will access camera or gallery a message will be displayed to him:



Keep in mind that this will be display one time and only if you send a push with a button to access this feature. Otherwise the permission will be never asked to the user and the keys are just present in plist to avoid a potential reject from apple when app is submitted.

5.3.3 Permission for geo location

Add the `NSLocationWhenInUseUsageDescription` key and the `NSLocationAlwaysAndWhenInUseUsageDescription` key to your Info.plist file.

(Xcode displays these keys as "Privacy - Location When In Use Usage Description" and "Privacy - Location Always and When In Use Usage Description" in the Info.plist editor.)

If your app supports iOS 10 and earlier, add the `NSLocationAlwaysUsageDescription` key to your Info.plist file.

(Xcode displays this key as "Privacy - Location Always Usage Description" in the Info.plist editor)

Take attention to the description that you will provide to those keys as that is what will be displayed to the user when the permissions will be asked to him.

To use geo location, you will need a specific version of the sdk. Contact selligent support for more information about this.

5.4 External framework

If you consider using geofencing module of the library and you have the correct version of the selligent sdk, you will need to embed **plotproject.framework** beside the selligent library in your app. You will also need to configure it with the **plotconfig.json** file in the root folder of your project. (more info in [Geolocation](#) part of the document)

6 How to use the SDK

6.1 Starting sdk

- In an Objective C project, import **SMHelper.h** wherever you will need to access to the SDK
- In a swift project, you just need to import **SMHelper.h** in your bridging header file
- To start the library, please follow the steps below (will mainly happen in your **UIApplication's** delegate):

The following must be done in **application:didFinishLaunchingWithOptions**:

- Create an instance of **SManagerSetting** with the *URL*, *clientId* and *private key* provided by Selligent.
- Set the following optional properties according to your need:
 - **shouldClearBadge**: if you want the sdk to manage badge clearance
 - **shouldDisplayRemoteNotification**: if you want to prevent the display of push message by sdk and manage it by your app (cf. Push notification helper methods)
 - **clearCacheIntervalValue**: define the interval value for clear of the sdk internal cache
- Optionally initialise and configure In App Message
- Optionally initialise and configure In App Content
- Optionally configure location service (May not be available depending of your sdk version)

ObjectiveC

```

NSString *url      = @"YourProvidedURL";
NSString *clientId = @"YourClientID";
NSString *privatKey = @"YourPrivateKey";

//Then:
//Create the SManagerSetting instance
SManagerSetting *setting = [SManagerSetting settingWithUrl:url ClientID:clientId PrivateKey:privatKey];

//Optional - Default value is true
setting.shouldClearBadge = TRUE;
setting.shouldDisplayRemoteNotification = TRUE;

//Optional - Default value is kSMClearCache_Auto
setting.clearCacheIntervalValue = kSMClearCache_Auto;

//Initialise InApp Message settings - other constructors exist (cf. documentation)
SManagerSettingIAM *iamSetting = [SManagerSettingIAM settingWithRefreshType:kSMIA_RefreshType_Daily];
[setting configureInAppMessageServiceWithSetting:iamSetting];

//Initialise InApp Content settings - other constructors exist (cf. documentation)
SManagerSettingIAC *iacSetting = [SManagerSettingIAC settingWithRefreshType:kSMIA_RefreshType_Daily];
[setting configureInAppContentServiceWithSetting:iacSetting];

```

```
//Configure location service (may not be available depending of the sdk version you have acquired)
[setting configureLocationService];
```

Swift

```
let url = "URL"
let clientID = "ClientID"
let privateKey = "privateKey"

//Create the SMMManagerSetting instance
let setting: SMMManagerSetting= SMMManagerSetting.setting(withUrl: url, clientID: clientID, privateKey: privateKey) as! SMMManagerSetting

//Optional - Default value is true
setting.shouldClearBadge = true;
setting.shouldDisplayRemoteNotification = true;

//Optional - Default value is kSMClearCache_Auto
setting.clearCacheIntervalValue = kSMClearCache_Auto;

//Optional - Initialise InApp Message settings
let settingIAM = SMMManagerSettingIAM.setting(with:.smia_RefreshType_Daily)
setting.configureInAppMessageService(withSetting: settingIAM)

//Optional - Initialise InApp Content settings
let settingIAC = SMMManagerSettingIAC.setting(with:.smia_RefreshType_Daily)
setting.configureInAppContentService(withSetting: settingIAC)
```

- Mandatory call to **startWithLaunchOptions:Setting:** using SDK Singleton **[SMMManager sharedInstance]**

ObjectiveC

```
//Starting the library
[[SMMManager sharedInstance] startLaunchOptions:launchOptions Setting:setting];
```

Swift

```
//Start the SDK
SMMManager.sharedInstance().start(launchOptions:launchOptions, setting: setting)
```

6.2 Push notifications

6.2.1 Register for push notification

Starting the library will not register for remote notification. You will need to call:

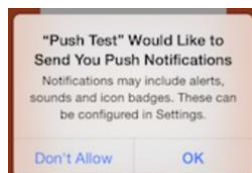
ObjectiveC:

```
[[SMMManager sharedInstance] registerForRemoteNotification];
```

Swift:

```
SMMManager.sharedInstance().registerForRemoteNotification()
```

This can be called whenever you need to do it in your app. You can then customize the way you inform the user before the display of iOS alert which will let the user to allow push messages for the app on the device (the iOS alert is displayed only once).



6.2.2 Listening and displaying the push notifications

6.2.2.1 App that does not build against iOS + 10

Implement methods described in `[SMMManager(RemoteNotification)]` in your `UIApplication`'s delegate

ObjectiveC

```
-(void)application:(UIApplication*)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData*)deviceToken {
    [[SMMManager sharedInstance] didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];
}

-(void)application:(UIApplication*)application didRegisterUserNotificationSettings:(UIUserNotificationSettings *)notificationSettings {
    [[SMMManager sharedInstance] didRegisterUserNotificationSettings:notificationSettings];
}

-(void)application:(UIApplication *)application didFailToRegisterForRemoteNotificationsWithError:(NSError *)error {
    [[SMMManager sharedInstance] didFailToRegisterForRemoteNotificationsWithError:error];
}

-(void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo{
    [[SMMManager sharedInstance] didReceiveRemoteNotification:userInfo];
}
```

Swift

```
func application(_ application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {
    SMMManager.sharedInstance().didRegisterForRemoteNotifications(withDeviceToken:deviceToken)
}

func application(_ application: UIApplication, didRegisterUserNotificationSettings notificationSettings: UIUserNotificationSettings) {
    SMMManager.sharedInstance().didRegister(notificationSettings)
}
```

```

}

func application(_ application: UIApplication, didFailToRegisterForRemoteNotificationsWithError error: Error) {
    SMMManager.sharedInstance().didFailToRegisterForRemoteNotificationsWithError(error)
}

func application(_ application: UIApplication, didReceiveRemoteNotification userInfo: [AnyHashable: Any]) {
    SMMManager.sharedInstance().didReceiveRemoteNotification(userInfo)
}

```

you can also implement specific delegates when your app supports background mode (cf. [IOS - MobileSDK Reference.pdf](#))

6.2.2.2 App that build against iOS + 10

Besides the implementation described in [App that does not build against iOS + 10](#) (in the case you need to support iOS 8 and 9), you will need to import `<UserNotifications/UserNotifications.h>` in your `AppDelegate` file and implement two methods of `UNUserNotificationCenterDelegate`:

ObjectiveC

```

#import <UserNotifications/UserNotifications.h>

@interface AppDelegate : UIResponder <UIApplicationDelegate, UNUserNotificationCenterDelegate>

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
    center.delegate = self;
    ...
}

- (void)userNotificationCenter:(UNUserNotificationCenter*)center willPresentNotification:(UNNotification*) notification
withCompletionHandler:(void (^)(UNNotificationPresentationOptions options))completionHandler{
    [[SMMManager sharedInstance] willPresentNotification:notification];
    completionHandler(UNNotificationPresentationOptionAlert);
    //OR [[SMMManager sharedInstance] willPresentNotification:notification withCompletionHandler:completionHandler];
    //in this case the sdk will be in charge to call completionHandler with UNNotificationPresentationOptionAlert as
    UNNotificationPresentationOptions
}

- (void)userNotificationCenter:(UNUserNotificationCenter*) center didReceiveNotificationResponse:(UNNotificationResponse*) response
withCompletionHandler: (void (^)(void)) completionHandler {
    [[SMMManager sharedInstance] didReceiveNotificationResponse:response];
    //OR [[SMMManager sharedInstance] didReceiveNotificationResponse:response withCompletionHandler:completionHandler];
    // in this case the sdk will be in charge to call completionHandler
}

```

Swift

```

import UserNotifications

class AppDelegate: UIResponder, UIApplicationDelegate, UNUserNotificationCenterDelegate

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]? -> Bool {
    let center = UNUserNotificationCenter.current()
    center.delegate = self
    ...
}

func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification, withCompletionHandler
completionHandler:@escaping(UNNotificationPresentationOptions) -> Void) {
    SMMManager.sharedInstance().willPresent(notification)
    completionHandler(.alert) // or any UNNotificationPresentationOptions
}

```

```
//OR SMMManager.sharedInstance().willPresent(response, withCompletionHandler:completionHandler) in this case
// the sdk will be in charge to call completionHandler with .alert as UNNotificationPresentationOptions
}

func userNotificationCenter(_ center: UNUserNotificationCenter, didReceive response:UNNotificationResponse, withCompletionHandler
completionHandler:@escaping() -> Void) {
    SMMManager.sharedInstance().didReceive(response)
    completionHandler()
}

//OR SMMManager.sharedInstance().didReceive(response, withCompletionHandler:completionHandler) in this case
// the sdk will be in charge to call completionHandler
}
```

**when you use geolocation version of the sdk and plotprojects framework
you will mandatory have in application:didFinishLaunchingWithOptions:
to assign the delegate**

```
UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
center.delegate = self;
```

before calling to

```
[[SMMManager sharedInstance] startLaunchOptions:launchOptions Setting:setting]]
```

6.2.3 Push notification helper methods

There are three useful methods which allow you to display an In-App message based on its id or to manage the way you want to display the push message when **SMMManagerSetting shouldDisplayRemoteNotification** is set to **FALSE**.

- Display notification based on its id

```
- (void)displayNotificationID:(NSString *)idNotification
```

- Display last received remote push notification

```
- (void)displayLastReceivedRemotePushNotification
```

- Retrieve last remote push notification (return a dictionary containing id and title of the notification)

```
- (NSDictionary *)retrieveLastRemotePushNotification
```

This can be for example associated with a library like **CRToast** to display your own banner in your app.

6.2.4 Broadcasts (NSNotification)

- **kSMNotification_Event_ButtonClicked**:
 - **NSString** representing a notification name you can listen to.
 - An **NSNotification** with this name is broadcasted when the user interacts with a remote-notification. Useful to retrieve user's actions on a received remote-notification, developers may listen to **kSMNotification_Event_ButtonClicked** from **NSNotificationCenter**.

- **kSMNotification_Event_WillDisplayNotification:**
 - `NSString` representing a notification name you can listen to.
 - An `NSNotification` with this name is broadcasted shortly before displaying a remote-notification. Primary-application may use this notification to pause any ongoing work before the remote-notification is displayed. This notification-name is also triggered even if you disable `shouldDisplayRemoteNotification` (see `SMMManagerSetting`).
- **kSMNotification_Event_WillDismissNotification:**
 - `NSString` representing a notification name you can listen to.
 - An `NSNotification` with this name is broadcasted shortly before dismissing the current remote-notification. Primary-application may use this notification to resume any paused work. (see `kSMNotification_Event_WillDisplayNotification`)
- **kSMNotification_Event_DidReceiveRemoteNotification:**
 - `NSString` representing a notification name you can listen to.
 - An `NSNotification` with this name is broadcasted shortly after receiving a remote-notification. Primary-application may use this notification to decide when to display any remote-notification
- **kSMNotification_Data_ButtonData:**
 - `NSString` representing a key to retrieve an object inside `NSNotification`
 - Use the key `kSMNotification_Data_ButtonData` to retrieve the object `SMNotificationButtonData` from the `NSNotification`-name `kSMNotification_Event_ButtonClicked`.
- **kSMNotification_Data_RemoteNotification:**
 - `NSString` representing a key to retrieve an object inside `NSNotification`
 - Use the key `kSMNotification_Data_RemoteNotification` to retrieve an `NSDictionary` instance with push ID and name

Example can be found in [Broadcasts \(NSNotification\)/Examples](#)

6.3 In App messages

6.3.1 Enable IAM

If In-App message (we will refer to them by IAM) are correctly configured (cf. [6.1 Starting SDK](#)), you will need to enable them once wherever you want in your app by calling:

ObjectiveC:

```
[[SMMManager sharedInstance] enableInAppMessage:TRUE];
```

Swift:

```
SMMManager.sharedInstance().enableInAppMessage(true)
```

Note: it is also possible to fetch IAM in background mode (cf. [IOS - MobileSDK Reference.pdf](#))

6.3.2 Display IAM

To retrieve your InAppMessage you must listen to `kSMNotification_Event_DidReceiveInAppMessage` (see [Broadcasts](#)). This will provide you a `NSDictionary` containing object with 2 properties: **id** and **title** for each InAppMessage available for the device.

ObjectiveC:

```
....
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(didReceiveInAppMessage:) name:kSMNotification_Event_DidReceiveInAppMessage object:nil];
....

-(void)didReceiveInAppMessage:(NSNotification*)notif{
    NSDictionary *dict = [notif userInfo];
    NSDictionary *inAppData = dict[kSMNotification_Data_InAppMessage];
}
```

Once your IAM retrieved you can for example create an Inbox (a table with each row containing title of the InApp Message) and when the user clicks on the InAppMessage a call to:

```
- (void)displayNotificationID:(NSString *)idNotification
```

with `idNotification` as the id of the InApp Message will allow you to display the complete InAppMessage. (you can refer to [Push notification helper methods](#) to display In App Messages)

6.3.3 Broadcasts (NSNotification)

- `kSMNotification_Event_DidReceiveInAppMessage`

- **NSString** representing a notification name you can listen to.
- An **NSNotification** with this name is broadcasted shortly after receiving InApp messages. Primary-application may use this notification to manage the received InApp messages
- **kSMNotification_Data_InAppMessage**
 - **NSString** representing a key to retrieve an object inside **NSNotification**
 - Use the key **kSMNotification_Data_InAppMessage** to retrieve an **NSDictionary** instance with an array of **SMNotificationMessage**

Example can be found in [Broadcasts \(NSNotification\)/Examples](#)

6.4 In App Content

6.4.1 Enabling IAC

If in App contents (we will refer to them by IAC) are correctly configured (cf. [6.1 Starting SDK](#)), IAC are then enabled by default and will be fetched each time the App becomes active (and connected), depending on the **SMInAppRefreshType** you have set.

Once new messages are received, the sdk will notify the app.

To be notified about new IAC, the application must register to correct notification **kSMNotification_Event_DidReceiveInAppContent**.

The Notification will provide the app with the number of IAC's by category (key **kSMNotification_Data_InAppContent**)

ObjectiveC:

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(anyMethodName:)
name:kSMNotification_Event_DidReceiveInAppContent object:nil];

-(void)anyMethodName:(NSNotification*)notif{
    NSDictionary *dict = [notif userInfo];
    NSArray *inAppData = dict[kSMNotification_Data_InAppContent];
}
```

Swift:

```
NSNotificationCenter.defaultCenter().addObserver(self, selector: "anyMethod:", name: kSMNotification_Event_DidReceiveInAppContent, object: nil);

func anyMethod (notif : NSNotification){
    let dict = notif.userInfo
    let inAppContentData = dict[kSMNotification_Data_InAppContent];
}
```

6.4.2 Displaying IAC

6.4.2.1 With SDK view controllers

Each IAC is from a unique type for a category

Selligent SDK can provide the app with a specific view controller for each type of IAC:

- **SMInAppContentHTMLViewController** for IAC of type **kSMInAppContentType_HTML**
- **SMInAppContentURLViewController** for IAC of type **kSMInAppContentType_Url**
- **SMInAppContentImageViewController** for IAC of type **kSMInAppContentType_Image**

They all are children of **SMInAppContentViewController**. They can all be initialized with one of these constructors:

ObjectiveC:

```
+ (instancetype) viewControllerForCategory:(NSString*)category ;
+ (instancetype) viewControllerForCategory:(NSString*)category AndOptions:(SMInAppContentStyleOptions*)options;
```

In addition, **SMInAppContentHTMLViewController** has two more constructors

ObjectiveC:

```
+ (instancetype) viewControllerForCategory:(NSString*)category InNumberOfBoxes:(int) numberOfboxes;
+ (instancetype) viewControllerForCategory:(NSString*)category InNumberOfBoxes:(int) numberOfboxes
AndOptions:(SMInAppContentStyleOptions*)options;
```

Where:

- **category** is a **NSString** with the category of the IAC that must be displayed
- **numberOfboxes** is an **int** used only for **SMInAppContentHTMLViewController**, the maximum number of html boxes that must be displayed for a category
- **options** is a **SMInAppContentStyleOptions** which will allow you to customize your IAC (cfr. [6.4.3 Customize IAC](#))

Once the sdk has provided you with the correct view controller, a bool property (**isEmpty**) informs you if the sdk has found any message for the category you asked for. If this property is false, you can then present the **SMInAppContentViewController** in full screen mode (in this case, a red cross will be displayed in top right corner to allow the dismiss of the view controller):

ObjectiveC

```
//example for an IAC Image that must be displayed when App become active
- (void)applicationDidBecomeActive:(UIApplication *)application {
    UITabBarController *tabController = (UITabBarController *)self.window.rootViewController;
    SMInAppContentImageViewController* iacVC = [SMInAppContentImageViewController viewControllerForCategory:@"anycategory"];
    if(!iacVC.isEmpty)
        [tabController presentViewController:iacVC animated:YES completion:nil];
}
```

Swift

```
//example for an IAC Image View controller
func applicationDidBecomeActive(application: UIApplication) {
    let tabController: UITabBarController = self.window!.rootViewController as! UITabBarController
    let iacVC = SMInAppContentImageViewController(forCategory:"anycategory")
    if(!iacVC.isEmpty) {
        tabController.presentViewController(iacVC, animated: true, completion: nil)
    }
}
```

Or if a `UIContainerView`, which is intended to receive the IAC View controller, is defined in your app, you can then call `showSMController:InContainerView:OfParentViewController:`

ObjectiveC

```
//example for an IAC Image View controller
@property (weak, nonatomic) IBOutlet UIView *yourImageContainer;
SMInAppContentImageViewController* vc = [SMInAppContentImageViewController viewControllerForCategory:@"yourcategory"];
[[SMMManager sharedInstance] showSMController:vc InContainerView:_yourImageContainer OfParentViewController:self];
```

Swift

```
//example for an IAC Image View controller
@IBOutlet weak var yourImageContainer: UIView!
let vc : SMInAppContentImageViewController = SMInAppContentImageViewController(forCategory: "yourcategory")
SMMManager.sharedInstance().showSMController(vc, inContainerView:self.yourImageContainer ,ofParentViewController:self)
```

But be aware that if your `UIContainerView` is defined in storyboard and that no category has been provided to it you will need to inform the SDK for which category the `SMInAppContentImageViewController` is expected. You can do so with `prepareForSegue:sender:`

ObjectiveC

```
@property (weak, nonatomic) IBOutlet UIView *yourImageContainer;
-(void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id) {
    if([segue.identifier isEqualToString:@"iacSegue"]){
        self.yourImageContainer = segue.destinationViewController;
        [self.yourImageContainer setCategory:@"news"];
    }
}
```

6.4.2.2 With your own view controllers

If you prefer to use IAC with your own UI, the sdk can provide you the necessary api accessible with the sdk singleton `[SMMManager sharedInstance]`.

In this case, you will have to call one of these two methods to get the data:

ObjectiveC

```
-(NSArray*) getAppContentsForCategory:(NSString*)category Type:(SMInAppContentType)type;
-(NSArray*) getAppContentsForCategory:(NSString*)category Type:(SMInAppContentType)type Max:(int)max;
```

You will then receive an NSArray of **SMInAppContentMessage** with all (or a certain amount if precised by the **max** parameter) IAC for a category and for a type.

categories are available when listening to NSNotification
kSMNotification_Event_DidReceiveInAppContent (cf. 6.4.1 Enabling InAppContent)

IMPORTANT: if you decide to use this way of interacting with IAC it is important that you keep in mind that you will be responsible of the display of the content and you will have to call to **setInAppContentAsSeen:(SMInAppContentMessage*)inAppContent** whenever an InAppContent is showed to the user. These methods require the shown IAC as parameter. By doing this, the sdk can process necessary consistency task and safely inform the services about the fact the IAC has been read.

ObjectiveC

```
- (void) setInAppContentAsSeen:(SMInAppContentMessage*)inAppContent;
```

In addition to this call whenever a user interacts with an action link of the in app content you will have to call **executeLinkAction:(SMLink*)link InAppContent:(SMInAppContentMessage*)inAppContent**

ObjectiveC

```
- (void) executeLinkAction:(SMLink*)link InAppContent:(SMInAppContentMessage*)inAppContent;
```

providing the **SMLink** and the **SMInAppContentMessage** to allow the sdk to safely inform the services that a specific link has been triggered by the user.

6.4.3 Customize IAC

To customize IAC, you will have to initialize an instance of **SMInAppContentStyleOptions**.

This class provides many properties which will allow you to modify UI of IAC View controllers.

Once your **SMInAppContentStyleOptions** is initialized you can either set your new options as the default one for all IAC (a reset method is also available) using the sdk singleton [**SManager sharedInstance**]

ObjectiveC

```
-(void)loadStyleOptions:(SMInAppContentStyleOptions*)options;  
-(void)resetStyleOptions;
```

or pass it as a parameter to your **SMInAppContentViewController** constructor:

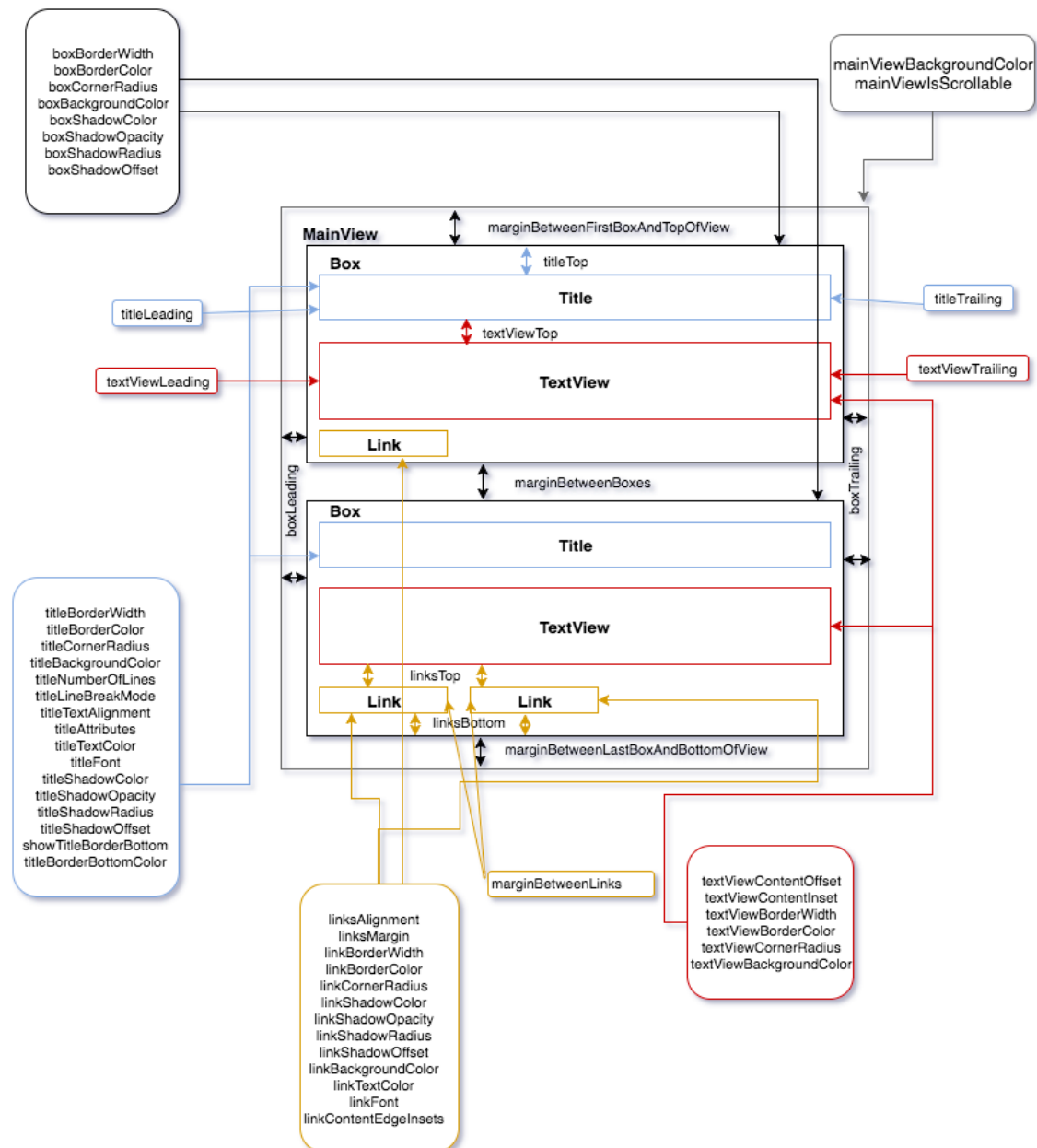
ObjectiveC

```
+ (instancetype) viewControllerForCategory:(NSString*)category AndOptions:(SMInAppContentStyleOptions*)options;
```

SMInAppContentImageViewController and **SMInAppURLViewController** have only 2 customizable properties:

```
@property (nonatomic) UIActivityIndicatorViewStyle activityIndicatorStyle;  
@property (nonatomic) bool isStatusBarHidden;
```

SMInAppContentHTMLViewController offers more possibilities, the following diagram gives an overview of the properties and their utility in the customization of the html in app content:



Besides these properties you still have the possibility to use **UIAppearance** for specific class:

ObjectiveC

```
[[UIView appearanceWhenContainedIn:[SMInAppContentHTMLViewController class], nil] setFont:[UIFont fontWithName:@"Marker Felt" size:10];  
[[UIView appearanceWhenContainedIn:[SMInAppContentHTMLViewController class], nil] setTextColor:[UIColor redColor];
```

Note: For more information on IAC cf. [IOS - MobileSDK Reference.pdf](#)

6.4.4 Broadcasts (NSNotification)

- `kSMNotification_Event_DidReceiveInAppContent`:
 - `NSString` representing a notification name you can listen to.
 - An `NSNotification` with this name is broadcasted shortly after receiving InApp content. Primary-application may use this notification to manage the received InApp contents.
- `kSMNotification_Data_InAppContent`
 - `NSString` representing a key to retrieve an object inside `NSNotification`.
 - Use the key `kSMNotification_Data_InAppContent` to retrieve an `NSDictionary` instance with an array of in-app contents categories as key and number of in-app contents for the category as value.

Example can be found in [Broadcasts \(NSNotification\)/Examples](#).

6.5 Geolocation

Geolocation is managed through a 3rd party framework: `plotprojects.framework`. To fully use this feature, you will have to ask for a **specific version of the sdk** (contact selligent for more information) and embed `plotprojects.framework` in your app.

Beside this, plot framework needs the presence of a config file (`plotconfig.json`) at the root of your project. The content of this file will look like:

```

1  {
2    "publicToken": "REPLACE_ME",
3    "enableOnFirstRun": true,
4    "maxRegionsMonitored": 10,
5    "automaticallyAskLocationPermission": true
6  }
```

Where:

- **publicToken** will be the token provided for you to be able to use plot framework
- **enableOnFirstRun** will allow you to enable plot framework automatically if value is set to true. Otherwise you will need to call:

```
[[SMMManager sharedInstance] enableGeoLocation];
```

whenever you will decide to enable plot framework. Another method exists which allow you to disable the plot framework:

```
[[SMMManager sharedInstance] disableGeoLocation];
```

- **maxRegionsMonitored** is the maximum regions monitored by Plot. The value of this property should be an integer between 5 and 20. This allows to keep some regions in case you want to monitor regions with another tool or by yourself. Keep in mind that the maximum regions that iOS allows to monitor is 20.
- **automaticallyAskLocationPermission** is a Boolean. If set to true and [plist file correctly configured](#) then iOS opt-in dialog for geo location will be displayed at app first start.

If set to false you will be able to ask user opt-in whenever you want. Try considering this [best practice](#) if you desire to do it this way.

In this case you can call **requestLocationAuthorisation:**

This method takes one **CLLocationAuthorisationType** parameter that can be **kCLLocationAuthorisationType_Always** or **kCLLocationAuthorisationType_InUse**

```
[[SMMManager sharedInstance] requestLocationAuthorisation:kCLLocationAuthorisationType_Always];
```

For more information on `plotconfig.json` check [PlotProjects documentation](#).

Once your app correctly configured, you will be able to define your campaign in plot dashboard.

6.6 Events

- Sending any set of data to the back-end can be done with `[SMManager sharedInstance] API sendSMEvent:`
- A helper method `sendDeviceInfo` allow you to send a specific set of device information

These methods take in parameter a `SMDeviceInfos` object. This object contains for the moment one unique property `externalId`:

ObjectiveC

```
SMDeviceInfos *deviceInfos = [SMDeviceInfos deviceInfosWithExternalID:@"12345"];
[[SMManager sharedInstance] sendDeviceInfo:deviceInfos];
```

- Default events are available for you to be used. They all inherit from `SMEvent` and are configurable through their constructors:
 - `SMEventUserLogin`
 - `SMEventUserLogout`
 - `SMEventUserRegistration`
 - `SMEventUserUnregistration`
- `shouldCache` property on events: If the event fails to be delivered to your backend, then by default, it is cached into an internal queue. After a while, the library will automatically try to send it again. Should you want to prevent this behaviour, feel free to set this property to `FALSE`. By default, it is set to `TRUE`
- You can also initialize a success block and/or a failure block that will be triggered after an event is sent to the services.

6.6.1 Registration / Unregistration

Two possible constructors:

```
+ (instancetype)eventWithEmail:(NSString *)mail
+ (instancetype)eventWithEmail:(NSString *)mail Dictionary:(NSDictionary<NSString*,NSString*> *)dict
```

- `mail`: the e-mail of the user
- `dict`: A Dictionary containing a string as key and a string as data

6.6.1.1 SMUserEventRegistration

This object is used to send a **register** event to the server with the e-mail of the user, potential data and a callback.

If email is not provided you can use in the dictionary an alternate key/value field to search for the user

ObjectiveC :

```
SMEventUserRegistration *event = [SMEventUserRegistration eventWithEmail:@"usermail@mail.com" Dictionary: @{@"key": @"value"}];
//with alternate key/value example: [SMEventUserRegistration eventWithEmail:@"" Dictionary: @{@"userID": @"1234"}];
//Optional
event.shouldCache = TRUE; //not necessary as it is the default value
[event applyBlockSuccess:^(SMSuccess *success) {
```

```

        NSLog(@"success");
    } BlockFailure:^(SMFailure *failure) {
        NSLog(@"failure");
    }
};
//Send
[SMMManager sharedInstance] sendSMEvent:event];

```

6.6.1.2 SMEventUserregistration

This object is used to send an **unregister** event to the server with the e-mail of the user, potential data and a callback.

If email is not provided you can use in the dictionary an alternate key/value field to search for the user

ObjectiveC :

```

SMEventUserUnregistration *event = [SMEventUserUnregistration eventWithEmail:@"usermail@mail.com" Dictionary:@{@"key": @"value"}];
//with alternate key/value example: [SMEventUserUnregistration eventWithEmail:@"" Dictionary:@{@"userID": @"1234"}];

//Optional
event.shouldCache = TRUE; //not necessary as it is the default value
[event applyBlockSuccess:^(SMSuccess *success) {
    NSLog(@"success");
} BlockFailure:^(SMFailure *failure) {
    NSLog(@"failure");
}];
//Send
[SMMManager sharedInstance] sendSMEvent:event];

```

6.6.2 Login/Logout

Two possible constructors:

```

+ (instancetype)eventWithEmail:(NSString *)mail
+ (instancetype)eventWithEmail:(NSString *)mail Dictionary:(NSDictionary<NSString*,NSString*> *)dict;

```

- **mail**: the e-mail of the user
- **dict**: A Dictionary containing a string as key and a string as data

6.6.2.1 SMEventUserLogin

This object is used to send a “login” event to the server with the e-mail of the user, potential data and a callback.

If email is not provided you can use in the dictionary an alternate key/value field to search for the user

ObjectiveC :

```

SMEventUserLogin *event = [SMEventUserLogin eventWithEmail:@"usermail@mail.com" Dictionary:@{@"key": @"value"}];
//with alternate key/value example: [SMEventUserLogin eventWithEmail:@"" Dictionary:@{@"userID": @"1234"}];

//Optional
event.shouldCache = TRUE; //not necessary as it is the default value
[event applyBlockSuccess:^(SMSuccess *success) {
    NSLog(@"success");
} BlockFailure:^(SMFailure *failure) {

```

```

        NSLog(@"failure");
    }
}
//Send
[SMMManager sharedInstance] sendSMEvent:event];

```

6.6.2.2 SMEventUserLogout

This object is used to send a **logout** event to the server with the e-mail of the user, potential data and a callback.

If email is not provided you can use in the dictionary an alternate key/value field to search for the user

ObjectiveC :

```

SMEventUserLogout *event = [SMEventUserLogout eventWithEmail:@"usermail@mail.com" Dictionary: @{@"key": @"value"}];
//with alternate key/value example: [SMEventUserLogout eventWithEmail:@"" Dictionary: @{@"userID": @"1234"}];
//Optional
event.shouldCache = TRUE; //not necessary as it is the default value
[event applyBlockSuccess:^(SMSuccess *success) {
    NSLog(@"success");
} BlockFailure:^(SMFailure *failure) {
    NSLog(@"failure");
}];
//Send
[SMMManager sharedInstance] sendSMEvent:event];

```

6.6.3 Custom

One constructor:

```

+ (instancetype)eventWithDictionary:(NSDictionary *)dict

```

- **dict:** A Dictionary containing a string as key and a string as data

6.6.3.1 SMEvent

This object is used to send a custom event to the server with some data and a callback.

ObjectiveC :

```

SMEvent *event = [SMEvent eventWithDictionary:@{@"key": @"value"}];
//Optional
event.shouldCache = TRUE; //not necessary as it is the default value
[event applyBlockSuccess:^(SMSuccess *success) {
    NSLog(@"success");
} BlockFailure:^(SMFailure *failure) {
    NSLog(@"failure");
}];
//Send
[SMMManager sharedInstance] sendSMEvent:event];

```

Swift:

```
let event = SMLocalEvent.init(dictionary: ["key": "value"]);
event.applyBlockSuccess({ (success) -> Void in
    print("success")
}) {(failure) -> Void in
    print("failure")
}
SMManager.sharedInstance().sendSMLocalEvent(event)
```

6.7 Broadcasts (NSNotification) summary

You can listen to some `NSNotification` by observing the correct notification name

6.7.1 Push notifications – IAM – IAC event broadcasts

- `kSMNotification_Event_ButtonClicked`

`NSString` representing a notification name you can listen to. An `NSNotification` with this name is broadcasted when the user interacts with a remote-notification Useful to retrieve user's actions on a received remote-notification, developers may listen to `kSMNotification_Event_ButtonClicked` from `NSNotificationCenter`.

- `kSMNotification_Event_WillDisplayNotification`

`NSString` representing a notification name you can listen to. An `NSNotification` with this name is broadcasted shortly before displaying a remote-notification Primary-application may use this notification to pause any ongoing work before the remote-notification is displayed. This notification-name is also triggered even if you disable `shouldDisplayRemoteNotification` (see `SManagerSetting`).

- `kSMNotification_Event_WillDismissNotification`

`NSString` representing a notification name you can listen to. An `NSNotification` with this name is broadcasted shortly before dismissing the current remote-notification Primary-application may use this notification to resume any paused work. (see `kSMNotification_Event_WillDisplayNotification`)

- `kSMNotification_Event_DidReceiveRemoteNotification`

`NSString` representing a notification name you can listen to. An `NSNotification` with this name is broadcasted shortly after receiving a remote-notification Primary-application may use this notification to decide when to display any remote-notification

- `kSMNotification_Event_DidReceiveInAppMessage`

`NSString` representing a notification name you can listen to. An `NSNotification` with this name is broadcasted shortly after receiving InApp messages Primary-application may use this notification to manage the received InApp messages

- `kSMNotification_Event_DidReceiveInAppContent`

`NSString` representing a notification name you can listen to. An `NSNotification` with this name is broadcasted shortly after receiving InApp content Primary-application may use this notification to manage the received InApp contents

6.7.2 Data broadcasts

- **kSMNotification_Data_ButtonData**

NSString representing a key to retrieve an object inside **NSNotification** Use the key **kSMNotification_Data_ButtonData** to retrieve the object **SMNotificationButtonData** from the **NSNotification-name** **kSMNotification_Event_ButtonClicked**.

- **kSMNotification_Data_RemoteNotification**

NSString representing a key to retrieve an object inside **NSNotification** Use the key **kSMNotification_Data_RemoteNotification** to retrieve an **NSDictionary** instance with push ID and name

- **kSMNotification_Data_InAppMessage**

NSString representing a key to retrieve an object inside **NSNotification** Use the key **kSMNotification_Data_InAppMessage** to retrieve an **NSDictionary** instance with an array of **SMNotificationMessage**

- **kSMNotification_Data_InAppContent**

NSString representing a key to retrieve an object inside **NSNotification** Use the key **kSMNotification_Data_InAppContent** to retrieve an **NSDictionary** instance with an array of in app contents categories as key and number of in app contents for the category as value

6.7.3 Examples

ObjectiveC:

```
//Listen to different broadcasting wherever you need to
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(anyMethodNameDidReceiveInAppMessage:)
name:kSMNotification_Event_DidReceiveInAppMessage object:nil];
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(anyMethodNameButtonClicked:)
name:kSMNotification_Event_ButtonClicked object:nil];
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(anyMethodNameWillDisplayNotification:)
name:kSMNotification_Event_WillDisplayNotification object:nil];
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(anyMethodNameWillDismissNotification :)
name:kSMNotification_Event_WillDismissNotification object:nil];
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(anyMethodNameDidReceiveRemoteNotification:)
name:kSMNotification_Event_DidReceiveRemoteNotification object:nil];

//Then Notifications selectors
-(void)anyMethodNameDidReceiveInAppMessage:(NSNotification*)notif{
    NSDictionary *dict = [notif userInfo];
    NSDictionary *inAppData = dict[kSMNotification_Data_InAppMessage];
}
-(void)anyMethodNameButtonClicked:(NSNotification*)notif{
    NSDictionary *dict = [notif userInfo];
    SMNotificationButtonData *btnData = dict[kSMNotification_Data_ButtonData];
}
-(void)anyMethodNameDidReceiveRemoteNotification:(NSNotification*)notif{
```

```

        NSDictionary *dict = [notif userInfo];
        NSDictionary *notifData = dict[kSMNotification_Data_RemoteNotification];
    }
    -(void)anyMethodNameWillDisplayNotification:(NSNotification*)notif{
    }
    -(void)anyMethodNameWillDismissNotification:(NSNotification*)notif{
    }

```

Swift:

```

//listen to broadcasting
NSNotificationCenter.defaultCenter().addObserver(self, selector: "anyMethod:", name: kSMNotification_Event_DidReceiveInAppMessage, object: nil);
NSNotificationCenter.defaultCenter().addObserver(self, selector: "anyMethodNameButtonClicked:", name: kSMNotification_Event_ButtonClicked, object: nil);
NSNotificationCenter.defaultCenter().addObserver(self, selector: "anyMethodNameWillDisplayNotification:", name: kSMNotification_Event_WillDisplayNotification, object: nil);
NSNotificationCenter.defaultCenter().addObserver(self, selector: "anyMethodNameWillDismissNotification:", name: kSMNotification_Event_WillDismissNotification, object: nil);
NSNotificationCenter.defaultCenter().addObserver(self, selector: "anyMethodNameDidReceiveRemoteNotification:", name: kSMNotification_Event_DidReceiveRemoteNotification, object: nil);

//Notifications selectors
func anyMethodNameDidReceiveInAppMessage(notif : NSNotification){
    let dict = notif.userInfo
    let inAppData = dict[kSMNotification_Data_InAppMessage];
}
func anyMethodNameButtonClicked(notif : NSNotification){
    let dict = notif.userInfo
    let btnData : SMNotificationButtonData = dict[kSMNotification_Data_ButtonData];
}
func anyMethodNameDidReceiveRemoteNotification(notif : NSNotification){
    let dict = notif.userInfo
    let notifData = dict[kSMNotification_Data_RemoteNotification];
}
func anyMethodNameWillDisplayNotification(notif : NSNotification){}
func anyMethodNameWillDismissNotification(notif : NSNotification){}

```

6.8 Miscellaneous

6.8.1 Reload

In case you want to change the web service URL, there is a reload method to restart the SDK.

It takes as parameter the same **SMSetting** object as the start method (all the values must be set in the object, even if they did not change).

This method is for development purpose not to be used in production.

ObjectiveC:

```
SMMManagerSetting *smSettings = [SMMManagerSetting settingWithUrl:currentUrl ClientID:clientID PrivateKey:privateKey];
[[SMMManager sharedInstance] reloadSetting:smSettings];
```

6.8.2 LogLevel

```
- (void)applyLogLevel:(SMLogLevel)logLevel
```

Will allow you to debug the library. Accepted **SMLogLevel**:

- **kSMLogLevel_None**: No log printed at all. This is the suggested log-level for release.
- **kSMLogLevel_Info**: Default log-entry. Basically, inform user when library starts / ends.
- **kSMLogLevel_Warning**: Only warning messages are printed
- **kSMLogLevel_Error**: Only Error messages are being printed
- **kSMLogLevel_HTTPCall**: Print only HTTP-requests stuff
- **kSMLogLevel_All**: Print everything. Do not use for release!!!

ObjectiveC:

```
[[SMMManager sharedInstance] applyLogLevel:kSMLogLevel_All];
```

Swift:

```
SMMManager.sharedInstance().applyLogLevel(.All)
```

Note: Don't forget to check [IOS - MobileSDK Reference.pdf](#) for more detailed information about:

- background mode
- all possible values for Constant References

6.9 Notification Extensions

Some sdk functionality are only possible with the implementation on the app side of notification extensions target.

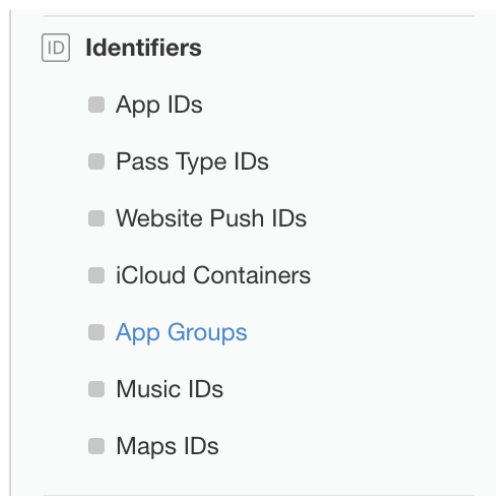
The features you will find under this section are only available for iOS 10 and later devices.

They are also only configurable in Selligent Marketing Cloud.

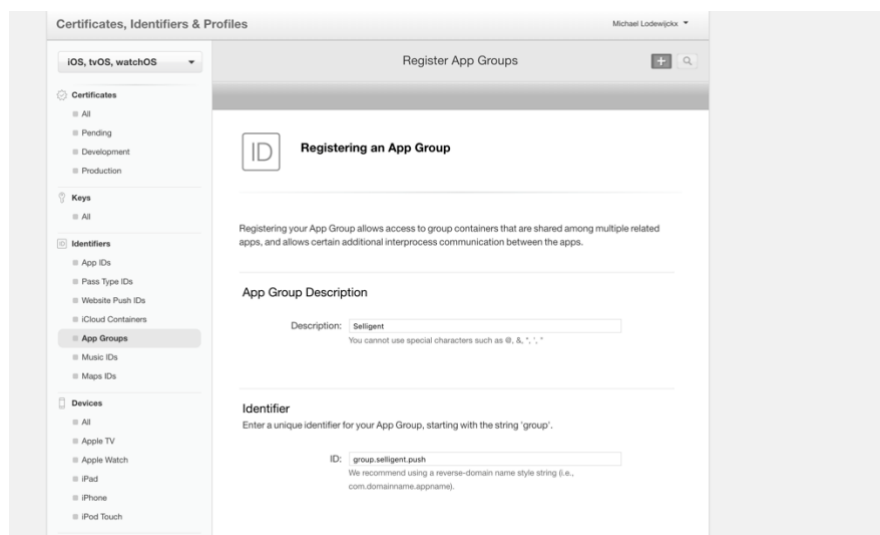
6.9.1 General set up

To use correctly those extensions a first set up must done inside your apple developer account

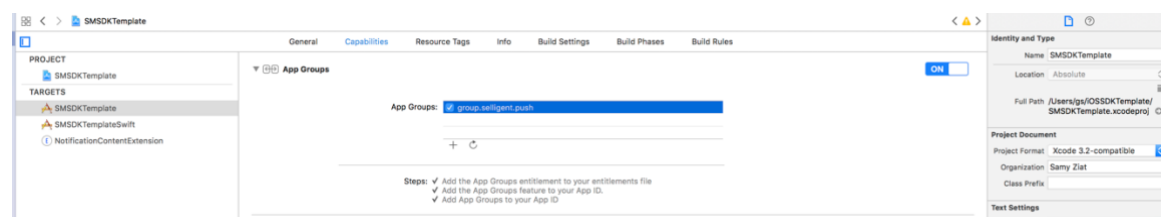
Connect to your account and go to App Groups



Create an AppGroup named **group.selligent.push**



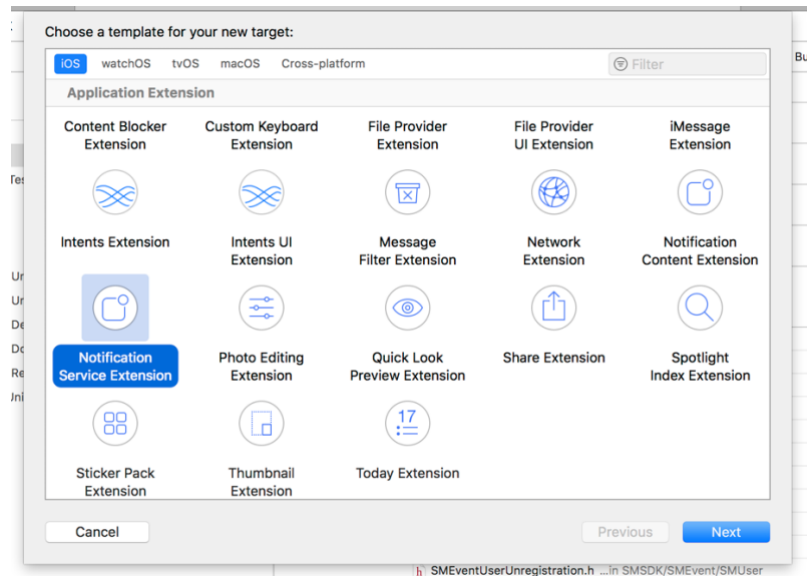
When this is done you will need to enable App group capabilities and check **group.selligent.push** in the Capabilities tab of your main app target



6.9.2 UNNotificationServiceExtension – Notification Service Extension

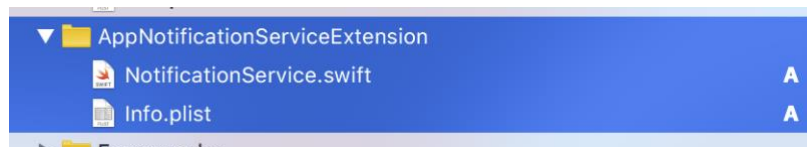
6.9.2.1 Configuration

To get started you will need to add a new target *Notification Service Extension* to your project:

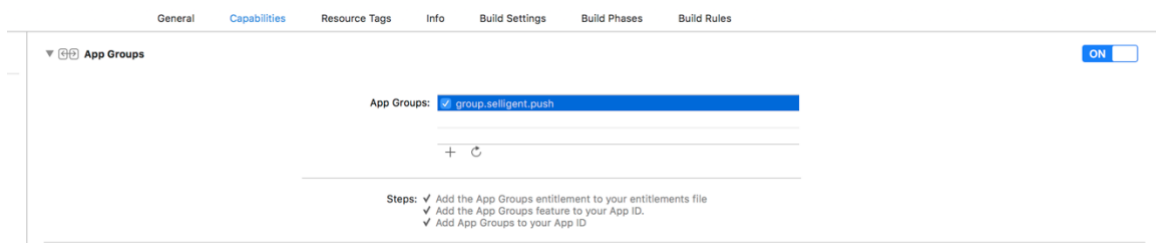


Notice the creation of the files (in Swift in this example):

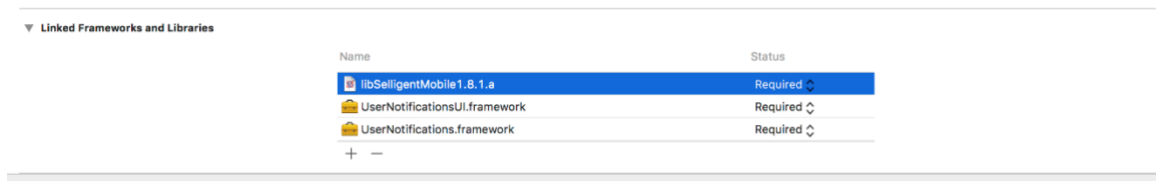
- NotificationService.swift : a subclass of UNNotificationServiceExtension
- Info.plist



In the capabilities of your target enable App groups and check **group.selligent.push**



And finally link Selligent Sdk to the Content Extension target:



Rem: if your target is in swift don't forget to set the correct path in Objective-C bridging-header property in the build settings of your target



6.9.2.2 Start sdk from inside extension

1. In an Objective C project, import **SMHelper.h** in NotificationService.m
2. In a swift project, you just need to import **SMHelper.h** in your bridging header file
3. To start the sdk, please follow the steps below. All the following must be done inside

ObjectiveC

didReceiveNotificationRequest:withContentHandler:

Swift

didReceive(_:withContentHandler:)

4. Create an instance of **SMManagerSetting** with the *URL*, *clientID* and *private key* provided by Selligent.

ObjectiveC

```
NSString *url = @"YourProvidedURL";
NSString *clientID = @"YourClientID";
NSString *privatKey = @"YourPrivateKey";

//Then:
//Create the SMManagerSetting instance
SMManagerSetting *setting = [SMManagerSetting settingWithUrl:url ClientID:clientID PrivateKey:privatKey];
```

Swift

```
let url = "URL"
let clientID = "ClientID"
let privateKey = "privateKey"

//Create the SMManagerSetting instance
let setting: SMManagerSetting = SMManagerSetting.setting(withUrl: url, clientID: clientID, privateKey: privateKey) as! SMManagerSetting
```

5. Mandatory call to **startExtensionWithSetting:** using SDK Singleton **[SMManager sharedInstance]**

ObjectiveC

```
//Starting the library
[[SMMManager sharedInstance] startExtensionWithSetting:setting];
```

Swift

```
//Start the SDK
SMMManager.sharedInstance().startExtension(with: setting)
```

6.9.2.3 Push notification content modification before displayed to user

Once your service extension correctly configured and the library is started., the extension will allow to modify the push content before displaying it to the user.

This feature is used for the moment by the sdk to **decrypt** the payload if it is flagged as encrypted.

You have possibility to choose between two methods either you want to manage the call to the block, which is executed with the modified notification content, by yourself or you want to let the library manage this for you. Under you will find a complete example of implementation for both cases and in both objective c and swift

- In first case a UNMutableNotificationContent will be returned to you:

ObjectiveC

```
#import "SMHelper.h"

@interface NotificationService ()
@property (nonatomic, strong) void (^contentHandler)(UNNotificationContent *contentToDeliver);
@property (nonatomic, strong) UNMutableNotificationContent *bestAttemptContent;
@end

@implementation NotificationService

- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentHandler:(void (^)(UNNotificationContent *_Nonnull))contentHandler {

    self.contentHandler = contentHandler;

    NSString *url = @"YourProvidedURL";
    NSString *clientId = @"YourClientId";
    NSString *privateKey = @"YourPrivateKey";
    //Create the SMMManagerSetting instance
    SMMManagerSetting *setting = [SMMManagerSetting settingWithUrl:url ClientID:clientId PrivateKey:privateKey];

    //Starting the library
    [[SMMManager sharedInstance] startExtensionWithSetting:setting];

    // Provide the request with the original notification content to the sdk and return the updated notification content
    self.bestAttemptContent = [[SMMManager sharedInstance] didReceiveNotificationRequest:request];

    // call the completion handler when done
```

```

        contentHandler(_bestAttemptContent);
    }

- (void)serviceExtensionTimeWillExpire {
    // Called just before the extension will be terminated by the system.
    // Use this as an opportunity to deliver your "best attempt" at modified content, otherwise the original push payload will be used.
    self.contentHandler(self.bestAttemptContent);
}

@end

```

Swift

```

// Storage for the completion handler and content.
var contentHandler: ((UNNotificationContent) -> Void)?
var bestAttemptContent: UNMutableNotificationContent?

// Modify the payload contents.
override func didReceive(_ request: UNNotificationRequest, withContentHandler contentHandler: @escaping (UNNotificationContent) -> Void) {
    self.contentHandler = contentHandler
    self.bestAttemptContent = (request.content.mutableCopy()
    as? UNMutableNotificationContent)

    // Init and start the sdk.
    let url = "URL"
    let clientID = "ClientID"
    let privateKey = "privateKey"

    //Create the SMManagerSetting instance
    let setting: SMManagerSetting= SMManagerSetting.setting(withUrl: url, clientID: clientID, privateKey: privateKey) as! SMManagerSetting

    //Start the sdk
    SMManager.sharedInstance().startExtension(with: setting)

    //Provide the request with the original notification content to the sdk and return the updated notification content
    bestAttemptContent = SMManager.sharedInstance().didReceive(request)

    //call the completion handler when done.
    contentHandler(bestAttemptContent)
}

// Return something before time expires.
override func serviceExtensionTimeWillExpire() {
    if let contentHandler = contentHandler,
        let bestAttemptContent = bestAttemptContent {

        // Mark the message as still encrypted.
        bestAttemptContent.subtitle = "(Encrypted)"
        bestAttemptContent.body = ""
        contentHandler(bestAttemptContent)
    }
}

```

- In second case sdk will manage all for you:

ObjectiveC

```
#import "SMHelper.h"

@implementation NotificationService

- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentHandler:(void (^)(UNNotificationContent *_Nonnull))contentHandler {

    NSString *url = @"YourProvidedURL";
    NSString *clientId = @"YourClientID";
    NSString *privateKey = @"YourPrivateKey";
    //Create the SMMManagerSetting instance
    SMMManagerSetting *setting = [SMMManagerSetting settingWithUrl:url ClientID:clientId PrivateKey:privateKey];

    //Starting the library
    [[SMMManager sharedInstance] startExtensionWithSetting:setting];

    // Provide the request with the original notification content to the sdk and the contentHandler
    [[SMMManager sharedInstance] didReceiveNotificationRequest:request WithContentHandler:contentHandler];

}

- (void)serviceExtensionTimeWillExpire {
    // Called just before the extension will be terminated by the system.
    [[SMMManager sharedInstance] serviceExtensionTimeWillExpire]
}

@end
```

Swift

```
// Modify the payload contents.
override func didReceive(_ request: UNNotificationRequest, withContentHandler contentHandler: @escaping (UNNotificationContent) -> Void) {
    // Init and start the sdk.
    let url = "URL"
    let clientId = "ClientID"
    let privateKey = "privateKey"

    //Create the SMMManagerSetting instance
    let setting: SMMManagerSetting= SMMManagerSetting.setting(withUrl: url, clientId: clientId, privateKey: privateKey) as! SMMManagerSetting

    //Start the sdk
    SMMManager.sharedInstance().startExtension(with: setting)

    //Provide the request with the original notification content to the sdk and the contentHandler
    SMMManager.sharedInstance().didReceive(request, withContentHandler: contentHandler as! (UNNotificationContent?) -> Void)
}

// Return something before time expires.
override func serviceExtensionTimeWillExpire() {
    SMMManager.sharedInstance().serviceExtensionTimeWillExpire()
}
```

```
}
```

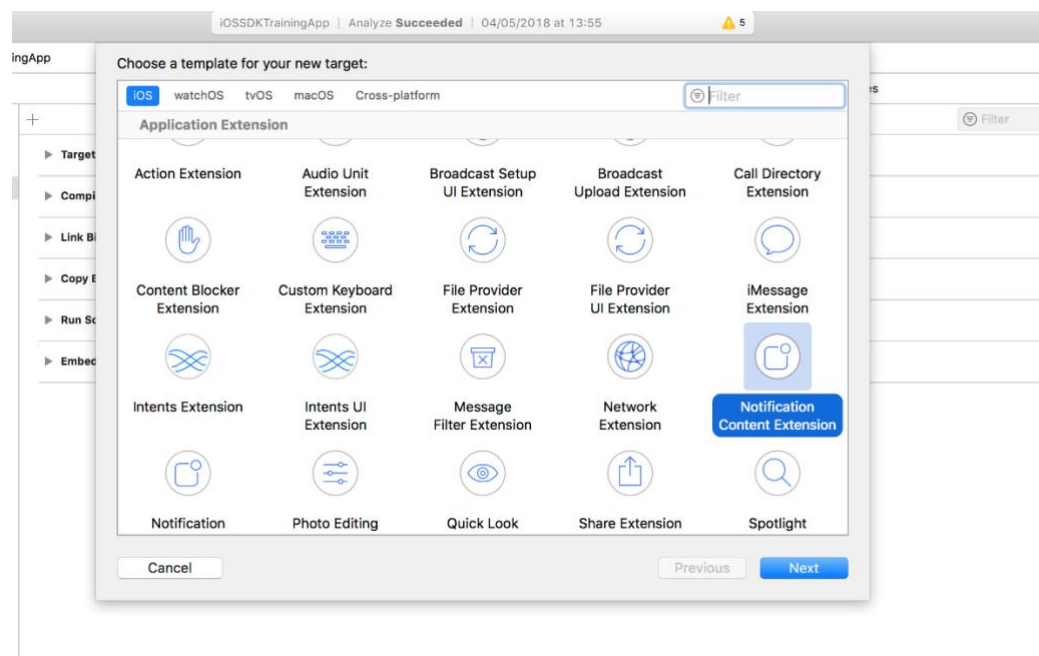
Rem: if message cannot be decrypted or if `serceExtensionTimeWillExpire` has been called before decryption is complete, "(Encrypted)" will be the values of all encrypted payload properties.

For more information on Notification service extension you can refer to [apple documentation](#)

6.9.3 UNNotificationContentExtension - Notification Content Extension

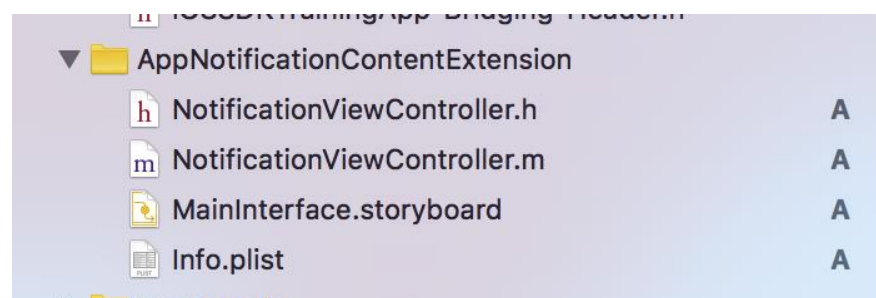
6.9.3.1 Configure notification content extension to your project for Selligent category

To get started you will need to add a new target *Notification Content Extension* to your project:

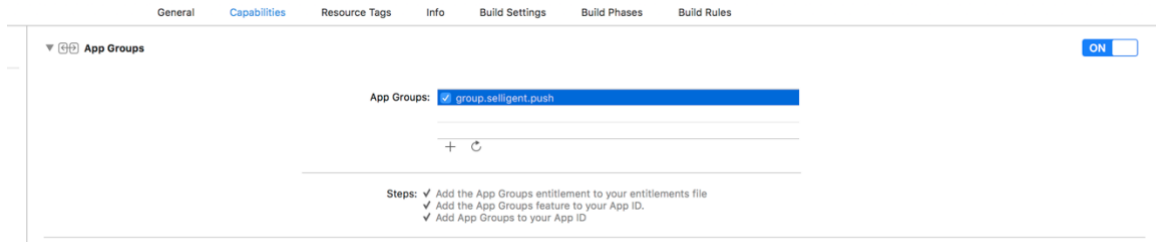


You will notice the creation of those files (in ObjectiveC in this example):

- `MainInterface.storyboard` : where you will be able to design the notification
- `NotificationViewController.m` : a `UIViewController` subclass
- `Info.plist`



In the capabilities of your target enable App groups and check **group.selligent.push**

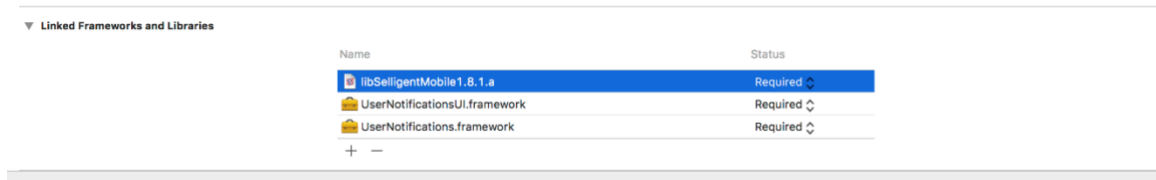


Now you will need to associate the Extension with Selligent Notification Category

Open the Info.plist of the extension, find the NSEXTENSIONATTRIBUTES dictionary and set the value of the UNNotificationExtensionCategory key to **SELLIGENT_BUTTON**

Information Property List		Dictionary	(10 items)
Localization native development region	String	\$(DEVELOPMENT_LANGUAGE)	
Bundle display name	String	AppNotificationContentExtension	
Executable file	String	\$(EXECUTABLE_NAME)	
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)	
InfoDictionary version	String	6.0	
Bundle name	String	\$(PRODUCT_NAME)	
Bundle OS Type code	String	XPC!	
Bundle versions string, short	String	1.0	
Bundle version	String	1	
NSEXTENSION	Dictionary	(3 items)	
NSEXTENSIONATTRIBUTES	Dictionary	(3 items)	
UNNotificationExtensionDefaultContentHidden	String	YES	
UNNotificationExtensionCategory	String	SELLIGENT_BUTTON	
UNNotificationExtensionInitialContentSizeRatio	Number	1	
NSEXTENSIONMAINSTORYBOARD	String	MainInterface	
NSEXTENSIONPOINTIDENTIFIER	String	com.apple.usernotifications.content-extension	

And finally link Selligent Sdk to the Content Extension target:



Rem: if your target is in swift don't forget to correct Objective-C bridging-header property in the build settings of your target



The storyboard will allow you to customise the display of the push notification.

If you want to keep default one please just hide the UIView created by default and don't set the UNNotificationExtensionDefaultContentHidden key. On the other hand set the key to YES and customise the display of the body and title of your notification.

For more info on Notification Content Extension please relate to [apple documentation](#)

6.9.3.2 Start sdk from inside extension

You can refer and follow all steps describe in [notification service extension start](#)

The only difference concerns the point 3.

All the steps must be done inside

ObjectiveC

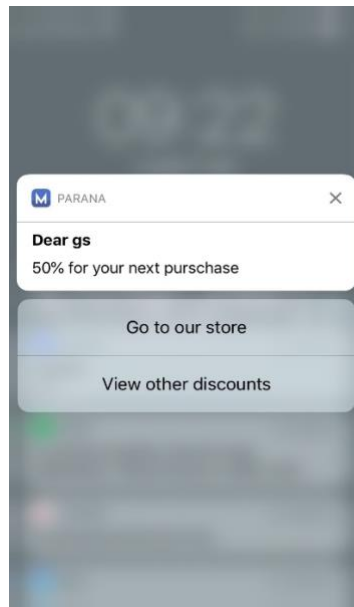
didReceiveNotification:

Swift

didReceive(_:)

6.9.3.3 Push action buttons

If you have correctly added a Selligent Notification Content Extension target to your project you will be able to display action buttons directly to your push without the need to open the app:



Now in your NotificationViewController file (.m or .swift) just call
`-(void)didReceiveNotification:(UNNotification*)notification;`

ObjectiveC:

```
#import SMHelper.h
```

```
@implementation NotificationViewController
```

```
- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any required interface initialization here.
}
```

```

- (void)didReceiveNotification:(UNNotification *)notification {
    self.label.text = notification.request.content.body;

    NSString *url      = @"YourProvidedURL";
    NSString *clientId  = @"YourClientID";
    NSString *privateKey = @"YourPrivateKey";
    //Create the SMMManagerSetting instance
    SMMManagerSetting *setting = [SMMManagerSetting settingWithUrl:url ClientID:clientId PrivateKey:privateKey];

    //Starting the library
    [[SMMManager sharedInstance] startExtensionWithSetting:setting];

    [[SMMManager sharedInstance] didReceiveNotification:notification];
}

```

Swift:

```

class NotificationViewController: UIViewController, UNNotificationContentExtension {
    @IBOutlet var label: UILabel?
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any required interface initialization here.
    }
    func didReceive(_ notification: UNNotification) {
        self.label?.text = notification.request.content.body

        // Init and start the sdk.
        let url = "URL"
        let clientId = "ClientID"
        let privateKey = "privateKey"

        //Create the SMMManagerSetting instance
        let setting: SMMManagerSetting= SMMManagerSetting.setting(withUrl: url, clientId: clientId, privateKey: privateKey) as! SMMManagerSetting

        //Start the sdk
        SMMManager.sharedInstance().startExtension(with: setting)

        SMMManager.sharedInstance().didReceive(notification)
    }
}

```

6.10 Changelog

- SDK 2.0.1

- Correct bug 64260 iOS fetching in app message with date - url is not valid and header validation is not successful
- Correct bug 64246 encoding of url is causing issue when there is a #

- Improve way optout is retrieved from iOS

- SDK 2.0

- Support decryption of remote notification
- Changed the way the sdk is initialized from inside a notification extension

- SDK 1.9

- Support action buttons in push notification center

- SDK 1.8

- Support push only without in app message

- SDK 1.7.1

- Correction on duplicate symbol due to integrated library

- SDK 1.7

- Added geolocation functionality
- Misc. Bugs correction
- Add http header to inform which version of the platform is supported
- Support of bitcode

- SDK 1.6

- iOS 11 support
- Misc. bug corrections
- consolidate received event
- adapt user-agent of request

- SDK 1.5.2

- correct bug for in app content that must be displayed only once

- SDK 1.5.1

- correct crash bug that happens when expiration or creation date for in app content is null

- SDK 1.5

- `sendDeviceInfo` deprecated method replaced with **`sendDeviceInfo:(SMDeviceInfos*)deviceInfos`** method
- New `SMMManager` category for `DataTransaction` with back-end
- public `SMDeviceInfos` object
- iOS 10 support of `UserNotifications` framework
- stop supporting of iOS 7
- cache on last sent `UserCustomEvent`
- Update `deviceId` with the one received from platform

- SDK 1.4.5

- Store last sent user custom event and check if a modification has been done before sending next one

- SDK 1.4.4

- compare device token based on string instead of `NSData` (bug swift 3 and `Data` class)

- SDK 1.4.3

- correction for max number of InApp Content crash when `max > number of messages received`
- creation date of in app content
- dismiss when no button in notification payload

- SDK 1.4.2

- correction on `unregisterForRemoteNotification` method
- issue with the notification when the application was killed

- SDK 1.4.1

- bug corrections

- SDK 1.4

- enum `SMInAppMessageRefreshType` has been renamed in `SMInAppRefreshType` (this enum is used both for InApp Message and for InApp Content) possible values are :

- `kSMIA_RefreshType_None`
- `kSMIA_RefreshType_Hourly`
- `kSMIA_RefreshType_Daily`

- SDK 1.3

- To access easily all API methods, you will need to replace `#import SMMManager.h` by `#import SMHelper.h`

- SDK 1.2

- The API `sendEvent:` has been renamed to `sendSMEvent:` (This call will prevent compilation)
- The API `registerForRemoteNotification` has been added. It allows applications to register remote-notification when they really need it. This, then, becomes a mandatory call for receiving pushes (after starting the library).