IOS – Using the SDK 1.8

PRODUCT MANUAL | April 18 2018

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 authorised 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 utilised 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 unauthorised 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

3 Intro

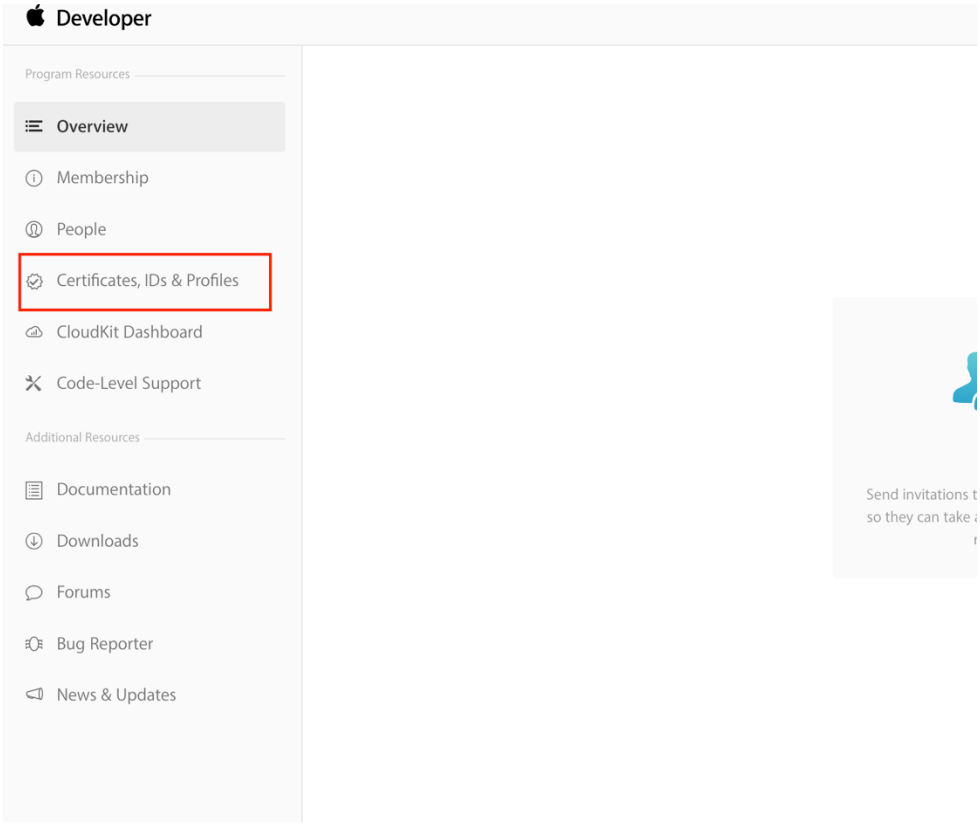
The purpose of this document is to detail how to install the SDK into your app and how to easily start using it.

- for more detailed technical reference of the sdk please refer to **IOS - MobileSDK Reference 1.8.pdf** document
- for an example of implementation check the **SMSDKTemplate** xCode project

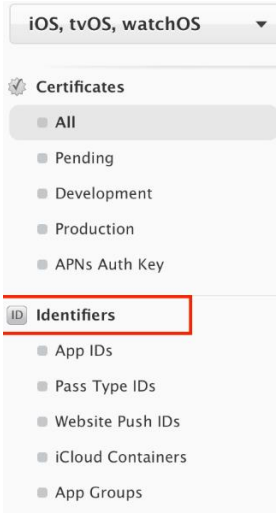
4 Configure the APNS (Apple Push Notification Service)

4.1 Enable push notifications

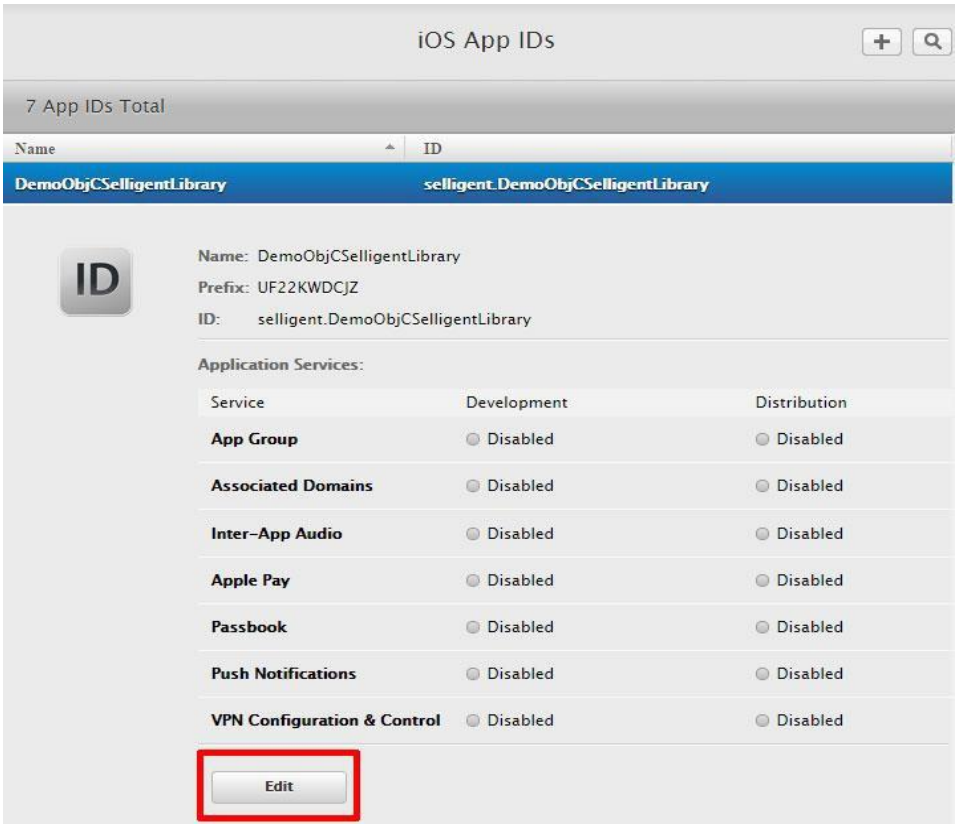
To enable push notifications, go to the Apple Developer Portal and login to the **Member centre**. When logged in, go to the **Certificates, IDs & Profiles** section to manage the certificates.



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

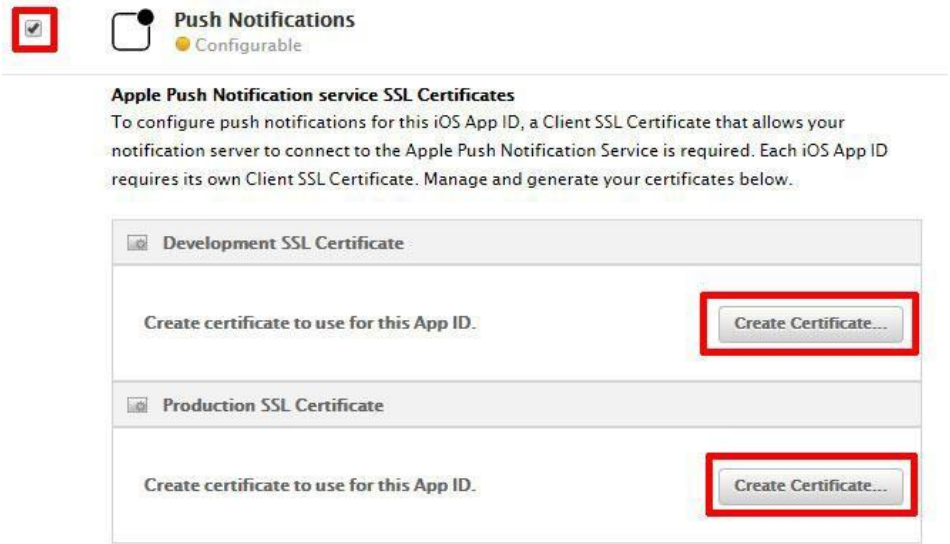


In the list of your app IDs select the app you want to enable the push notifications. Edit the application services to enable the push notification.



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

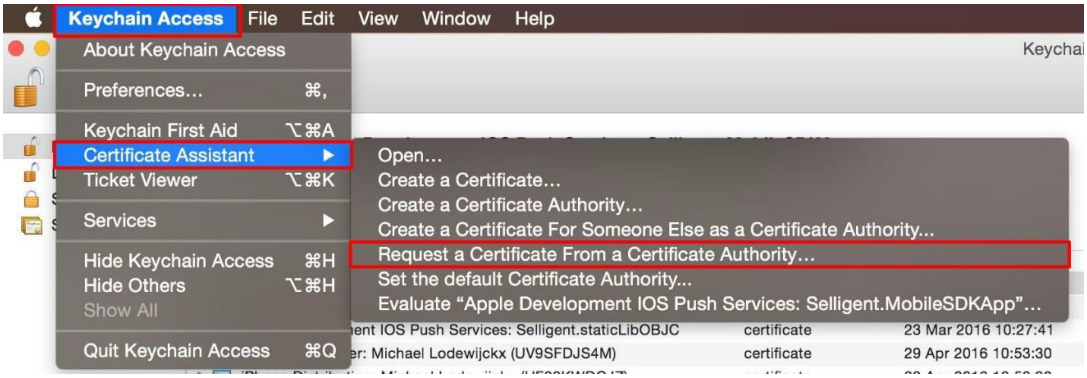
Since iOS 10 production certificate can be used for both production and development/sandbox environment.



4.2 Create and submit a Certificate Signing Request (CSR)

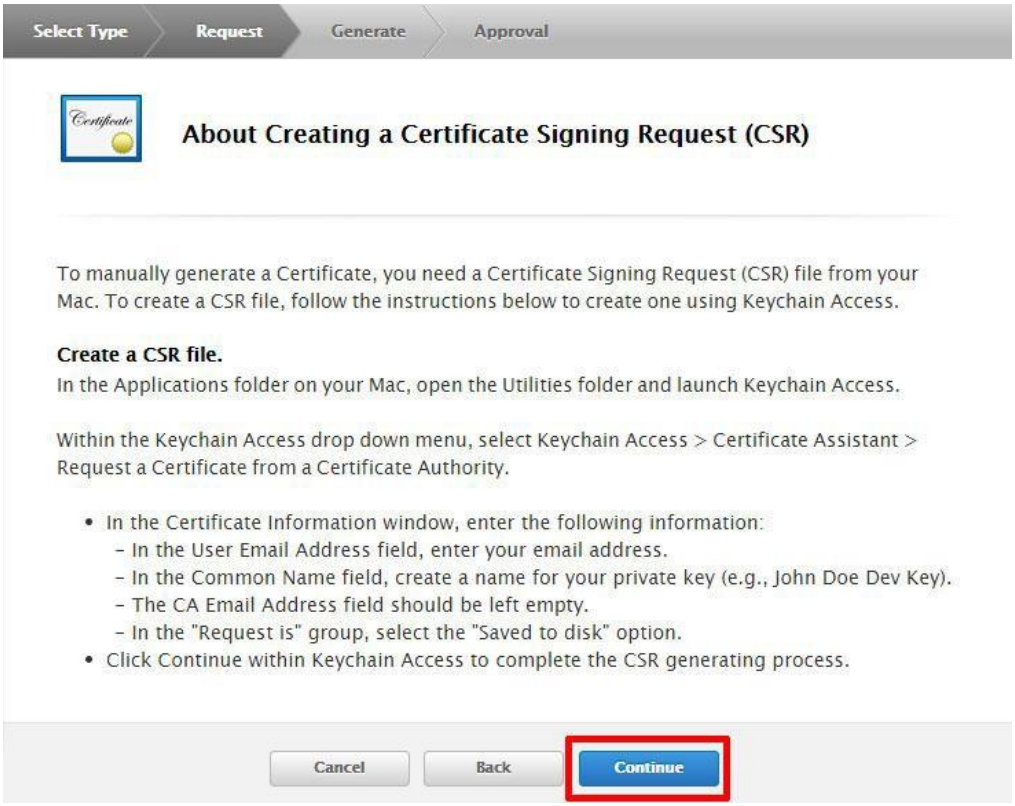
Before going further, we need to generate a **Certificate Signing Request (CSR)**. For this purpose, you will need the **Keychain Access** of Mac OS. Search for **Keychain Access** in Spotlight

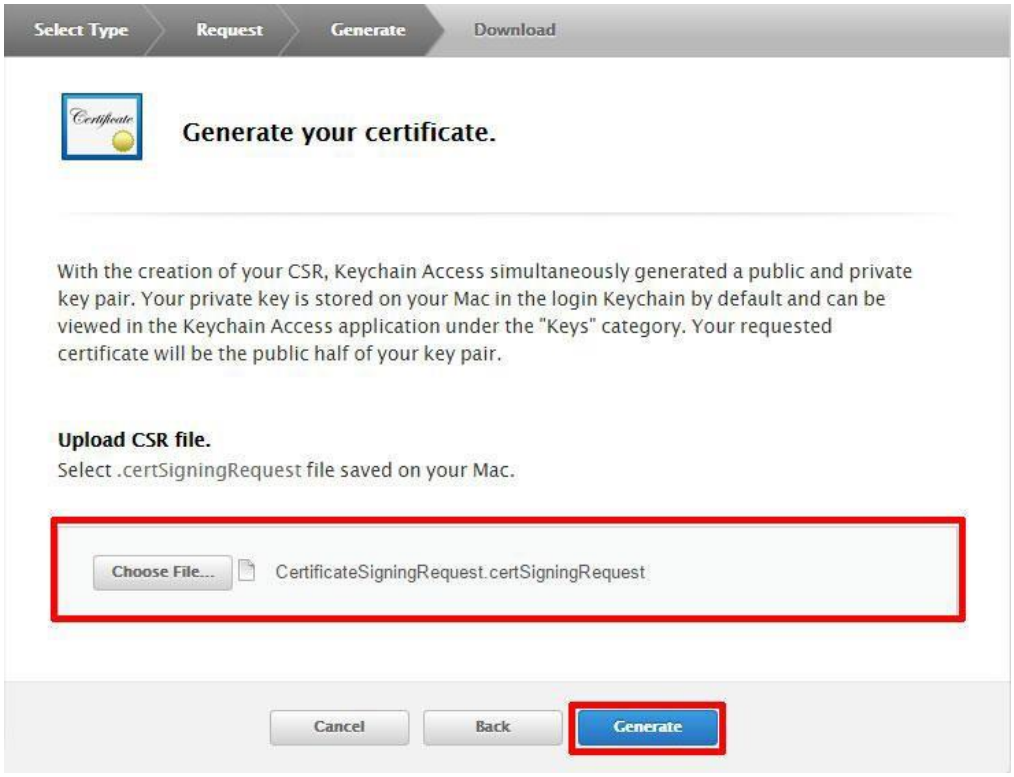
Once **Keychain** is open, go to **Keychain Access>Certificate Assistant>Request a Certificate from a Certificate Authority**



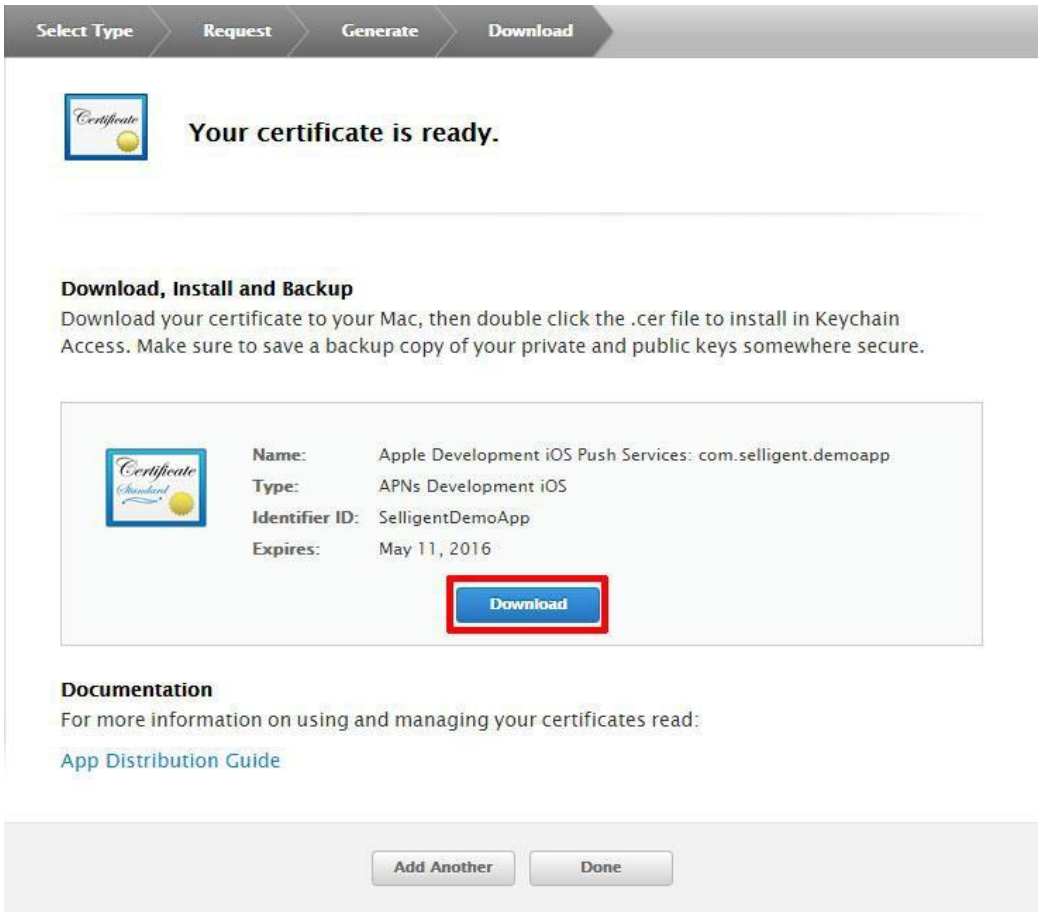
In the opened window fill the **User Email Address** field, the **Common Name** and select the **Saved to disk** option.

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.



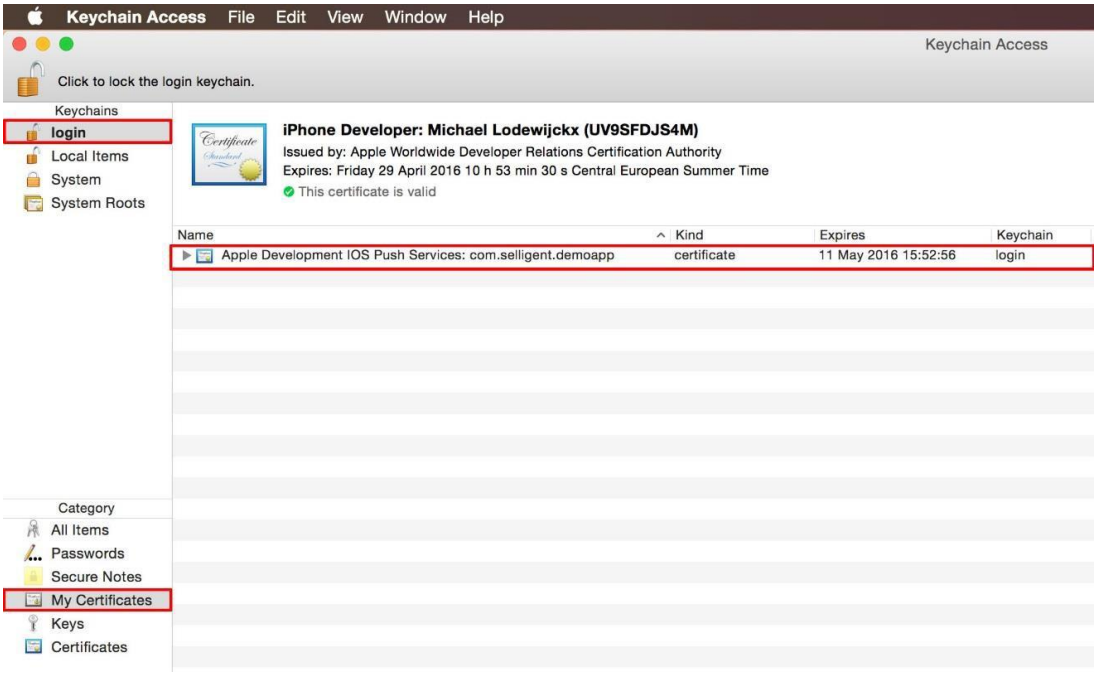


Click on Download to get the **aps_TARGET.cer** file.



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.

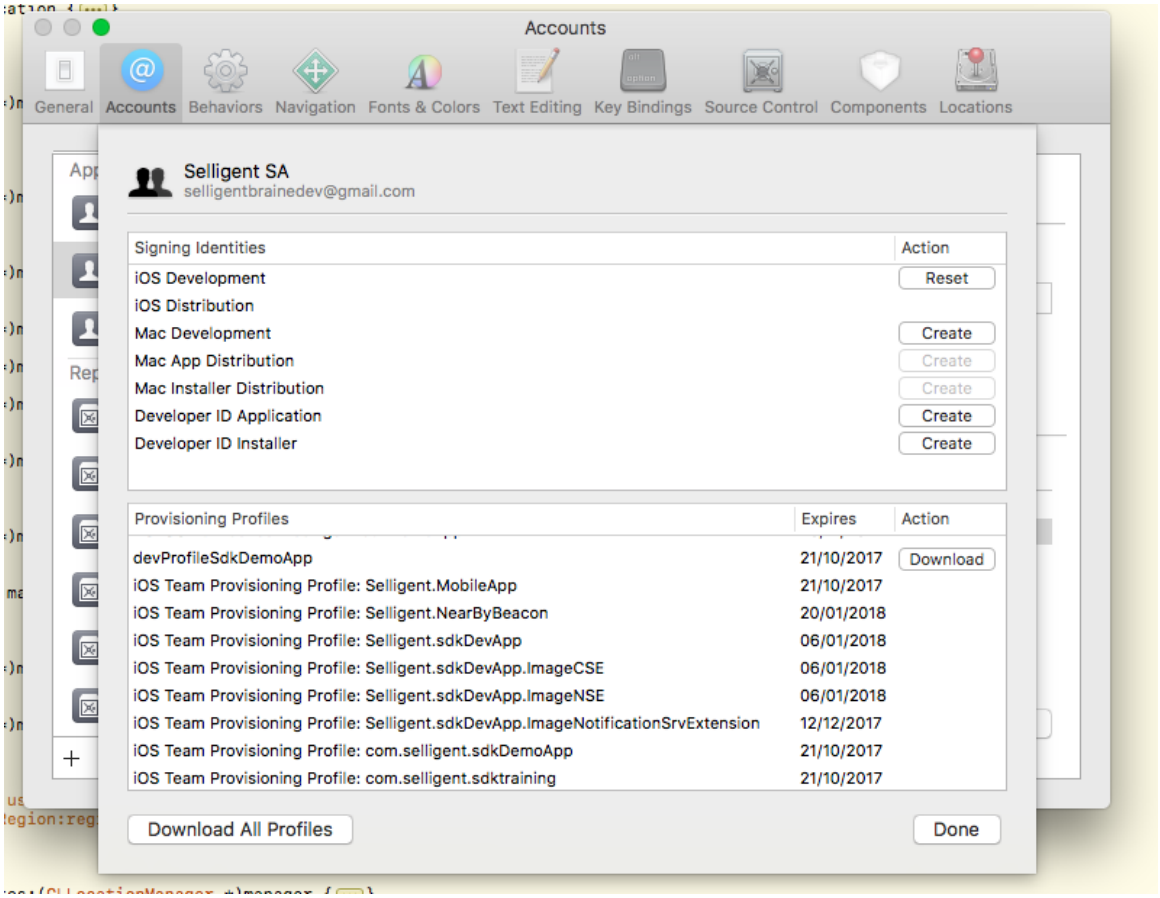


To export the **.p12** file, **expand the certificate**, right click (or CTRL + left click) on the **certificate only** and select export.

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**

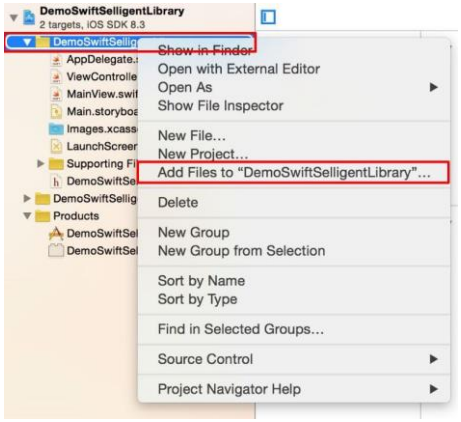


starting XCode 7 some issues have been reported to apple that the update is not always correctly refreshed and must triggered a few times to be valid

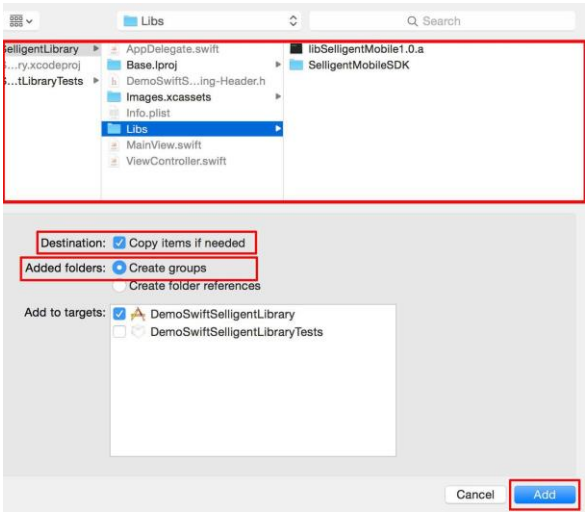
5 Include SDK in your target

5.1 Import the library

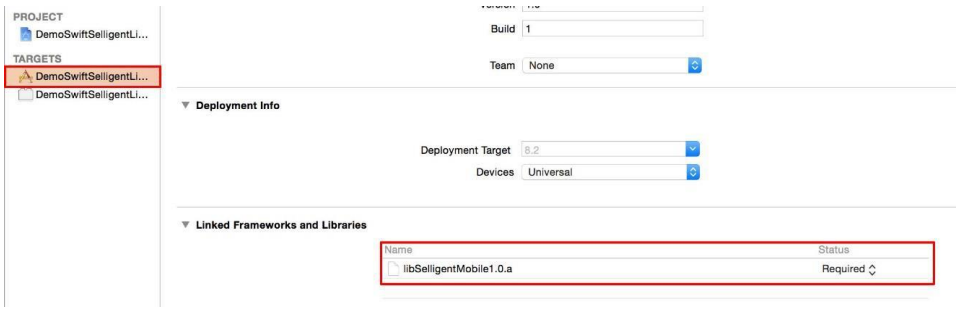
Right click (CTRL + Left click) on your app target and select **Add Files to "YOURTARGET"**



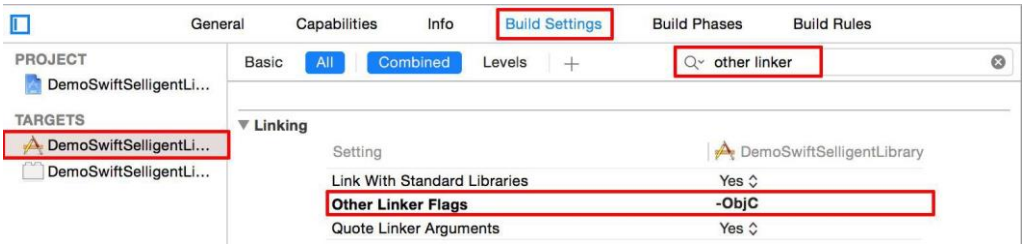
Select the lib folder (the main folder containing the header and the lib files). Depending on your project check the option **Copy item if needed** and select the **Create groups option**.



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

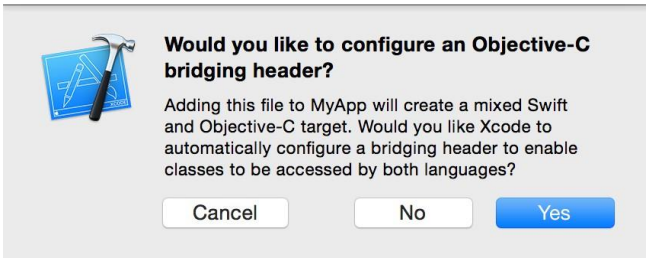


Then, go to the Build Settings of your target app, search for **Other Linker Flags** property and set the value to **–ObjC**.



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 DeepLinking

Rem for sdk before v.1.8: 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 and :

- Sdk < v.1.8 or push configured in campaign: an alert with the link will be displayed and the click on the link will trigger the 'appscheme://anypage'
- Sdk >= v1.8: the deeplinking will redirect you to the correct page in your app

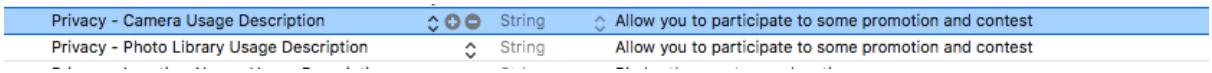
5.3.2 Permission for camera and image gallery usage

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.

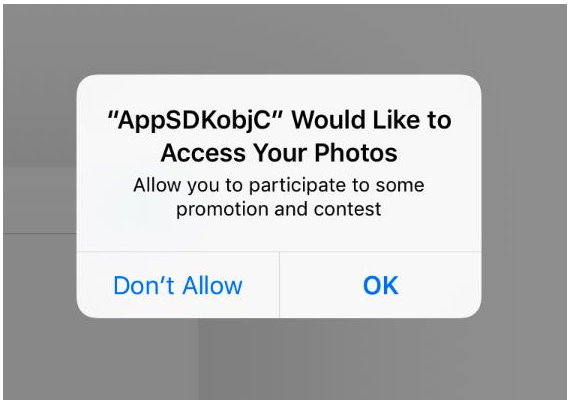
Since iOS 10 you must add these two keys in your **plist** (if not already present for your usage) to inform correctly the user of the usage of these features:

```
<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)



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 **SMManagerSetting** 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 SMManagerSetting instance
SMManagerSetting *setting = [SMManagerSetting 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)
SMManagerSettingIAM *iamSetting = [SMManagerSettingIAM settingWithRefreshType:kSMIA_RefreshType_Daily];
[setting configureInAppMessageServiceWithSetting:iamSetting];

//Initialise InApp Content settings - other constructors exist (cf. documentation)
SMManagerSettingIAC *iacSetting = [SMManagerSettingIAC 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 SMManagerSetting instance
let setting: SMManagerSetting= SMManagerSetting.setting(withUrl: url, clientID: clientID, privateKey: privateKey) as! SMManagerSetting

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

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

ObjectiveC

```
//Starting the library

[[SMManager sharedInstance] startLaunchOptions:launchOptions Setting:setting];
```

Swift

```
//Start the SDK

SMManager.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:

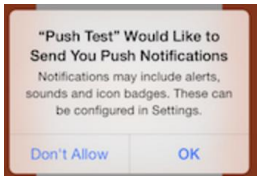
ObjectiveC:

```
[[SMManager sharedInstance] registerForRemoteNotification];
```

Swift:

```
SMManager.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 **[SMManager(RemoteNotification)]** in your UIApplication's delegate

ObjectiveC

```
-(void)application:(UIApplication*)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData*)deviceToken {
        [[SMManager sharedInstance] didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];
}
-(void)application:(UIApplication*)application didRegisterUserNotificationSettings:(UIUserNotificationSettings *)notificationSettings {
        [[SMManager sharedInstance] didRegisterUserNotificationSettings:notificationSettings];
}
- (void)application:(UIApplication *)application didFailToRegisterForRemoteNotificationsWithError:(NSError *)error {
        [[SMManager sharedInstance] didFailToRegisterForRemoteNotificationsWithError:error];
}
- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo{
        [[SMManager sharedInstance] didReceiveRemoteNotification:userInfo];
}
```

Swift

```
func application(_ application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {
        SMManager.sharedInstance().didRegisterForRemoteNotifications(withDeviceToken:deviceToken)
}
func application(_ application: UIApplication, didRegisterUserNotificationSettings notificationSettings: UIUserNotificationSettings) {
        SMManager.sharedInstance().didRegister(notificationSettings)
}
func application(_ application: UIApplication, didFailToRegisterForRemoteNotificationsWithError error: Error) {
        SMManager.sharedInstance().didFailToRegisterForRemoteNotificationsWithError(error)
}
func application(_ application: UIApplication, didReceiveRemoteNotification userInfo: [AnyHashable: Any]) {
        SMManager.sharedInstance().didReceiveRemoteNotification(userInfo)
}
```

you can also implement specific delegates when your app supports background mode (cf. IOS - MobileSDK Reference 1.8.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{

        [[SMManager sharedInstance] willPresentNotification:notification];

        completionHandler(UNNotificationPresentationOptionAlert);

//OR [[SMManager 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
(^)()) completionHandler {

        [[SMManager sharedInstance] didReceiveNotificationResponse:response];

//OR [[SMManager 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) {
        SMManager.sharedInstance().willPresent(notification)
        completionHandler(.alert) // or any UNNotificationPresentationOptions
//OR SMManager.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) {
        SMManager.sharedInstance().didReceive(response)
        completionHandler()
//OR SMManager.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
[[SMManager 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 **SMManagerSetting shouldDisplayRemoteNotification** is set to FALSE.

- Display notification based on its id (this works only for in app messages. It would not work for push with in app messages)

```
- (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 (since iOS 10 a banner can always be displayed even within the 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 **SMManagerSetting**).
- **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:

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

Swift:

```
SMManager.sharedInstance().enableInAppMessage(true)
```

Note: it is also possible to fetch IAM in background mode (cf. IOS - MobileSDK Reference 1.8.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(didReceiveInAppMesage:) name: kSMNotification_Event_DidReceiveInAppMessage object: nil];

....



-(void)didReceiveInAppMessage:(NSNotification*)notif{

        NSDictionary *dict = [notif userInfo];

          NSDictionary *inAppData = dict[kSMNotification_Data_InApMessaget];

}
```

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"];
[[SMManager 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")
SMManager.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 **SMInAppContentViewController** 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 **[SMManager sharedInstance].**

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

```
ObjectiveC

- (NSArray*) getInAppContentsForCategory:(NSString*)category Type:(SMInAppContentType)type;
- (NSArray*) getInAppContentsForCategory:(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 **[SMManager sharedInstance]**

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

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

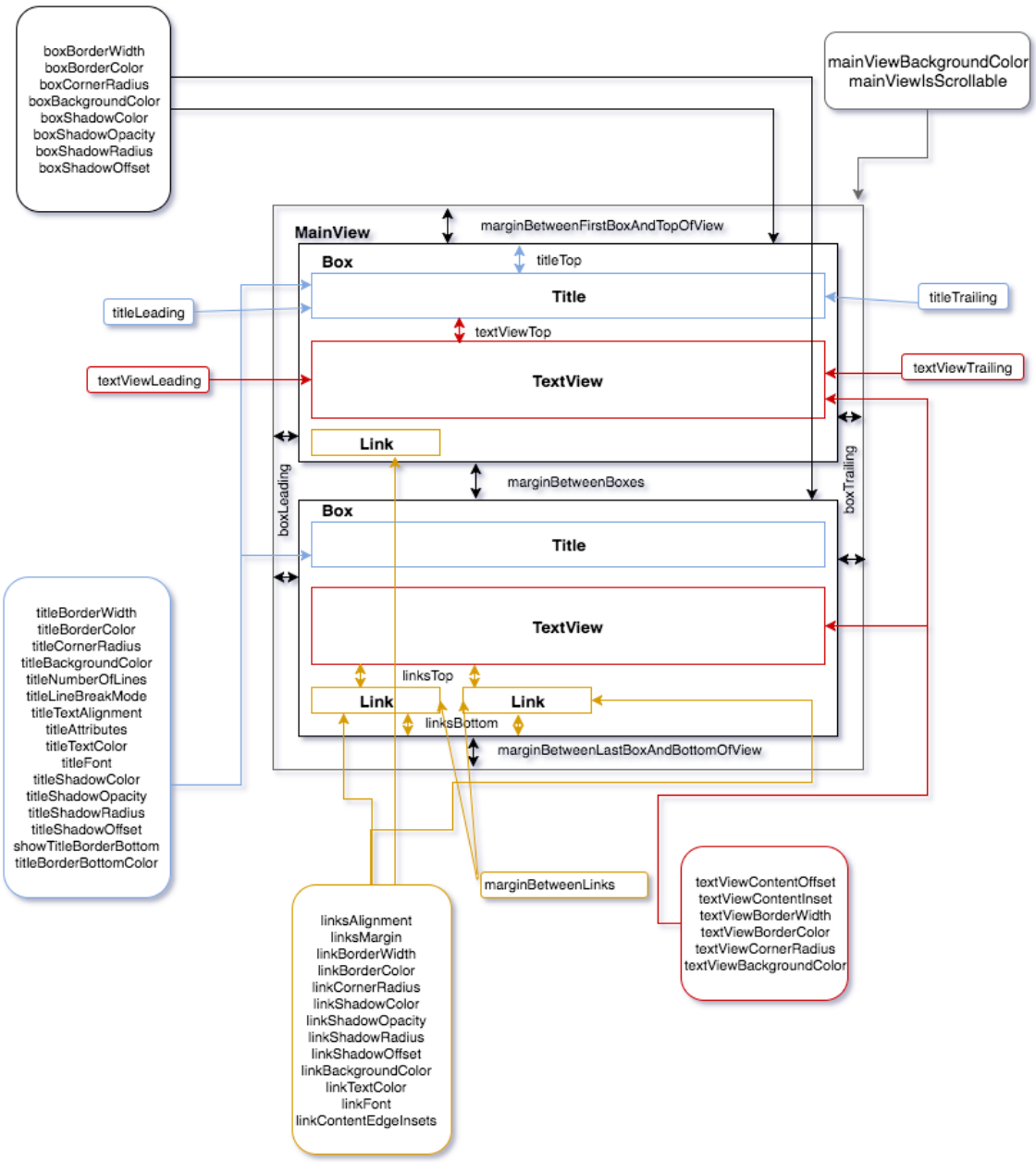
```objectivec
+ (instancetype) viewControllerForCategory:(NSString*)category AndOptions:(SMInAppContentStyleOptions*)options;
```

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

```objectivec
@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

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

Note: For more information on IAC cf. IOS - MobileSDK Reference 1.8.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:

```
{
  "publicToken": "REPLACE_ME",
  "enableOnFirstRun": true,
  "maxRegionsMonitored": 10,
  "automaticallyAskLocationPermission": true
}
```

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:

```
[[SMManager sharedInstance] enableGeoLocation];
```

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

```
[[SMManager 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 **SMLocationAuthorisationType** parameter that can be **kSMLocationAuthorisationType_Always** or **kSMLocationAuthorisationType_InUse**

```
[[SMManager sharedInstance] requestLocationAuthorisation:kSMLocationAuthorisationType_Always];
```

Once your app correctly configured, you will be able to define your campaign in plot dashboard. For more information on **plotconfig.json** check PlotProjects documentation.

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
[SMManager 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
[SMManager 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
[SMManager 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
[SMManager sharedInstance] sendSMEvent:event];
```

6.6.3 Custom event

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
[SMManager sharedInstance] sendSMEvent:event];
```

Swift:

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

- **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:

```
SMManagerSetting *smSettings = [SMManagerSetting settingWithUrl:currentUrl ClientID:clientID PrivateKey:privateKey];
[[SMManager 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:

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

Swift:

```
SMManager.sharedInstance().applyLogLevel(.All)
```

Note: Don't forget to check IOS - MobileSDK Reference 1.8.pdf for more detailed information about:

- background mode
- all possible values for Constant Referencesf

6.9 Changelog

- **SDK 1.8**
    - Support push only without In app messages
    - Support deeplinking and main action when user click on push notification

- **SDK 1.7.4**
    - Improve how In App content are managed in cache
    - Correction on dismiss of in app content viewcontroller when it is not wrapped inside other controller and when link is present

- **SDK 1.7.3**
    - Improve behavior and robustness of the way In App messaged are fetched

- **SDK 1.7.2**
    - Correction on IAC cache issue when app was killed (file was not updated)

- **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 SMManager 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_Minutely (to be used only in dev)
    - kSMIA_RefreshType_Hourly
    - kSMIA_RefreshType_Daily

- SDK 1.3

- To access easily all API methods, you will need to replace #import SMManager.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).