# iOS – Using the SDK

Current release: 01-07-15

selligent

reconnect your brand

Selligent

# Table of Contents

# 1. Intro

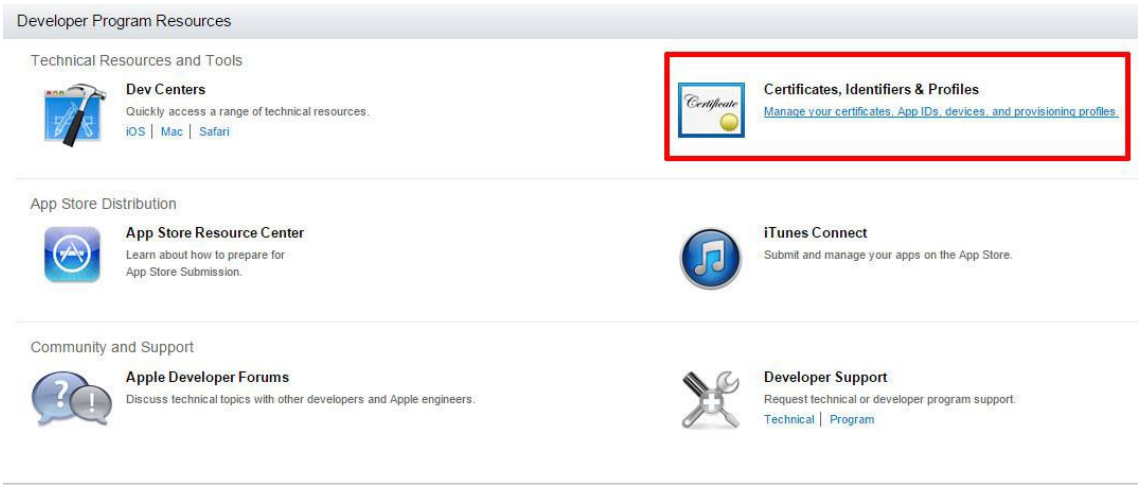
The purpose of this document is to detail how to install the SDK into your app and how to easily start using it.

- for more detailed implementation of the sdk please refer to IOS - MobileSDK Reference 1.4.pdf document
- for an example of implementation check the SMSDKTemplate project

# 2. Configure the APNS ( Apple push notification service )

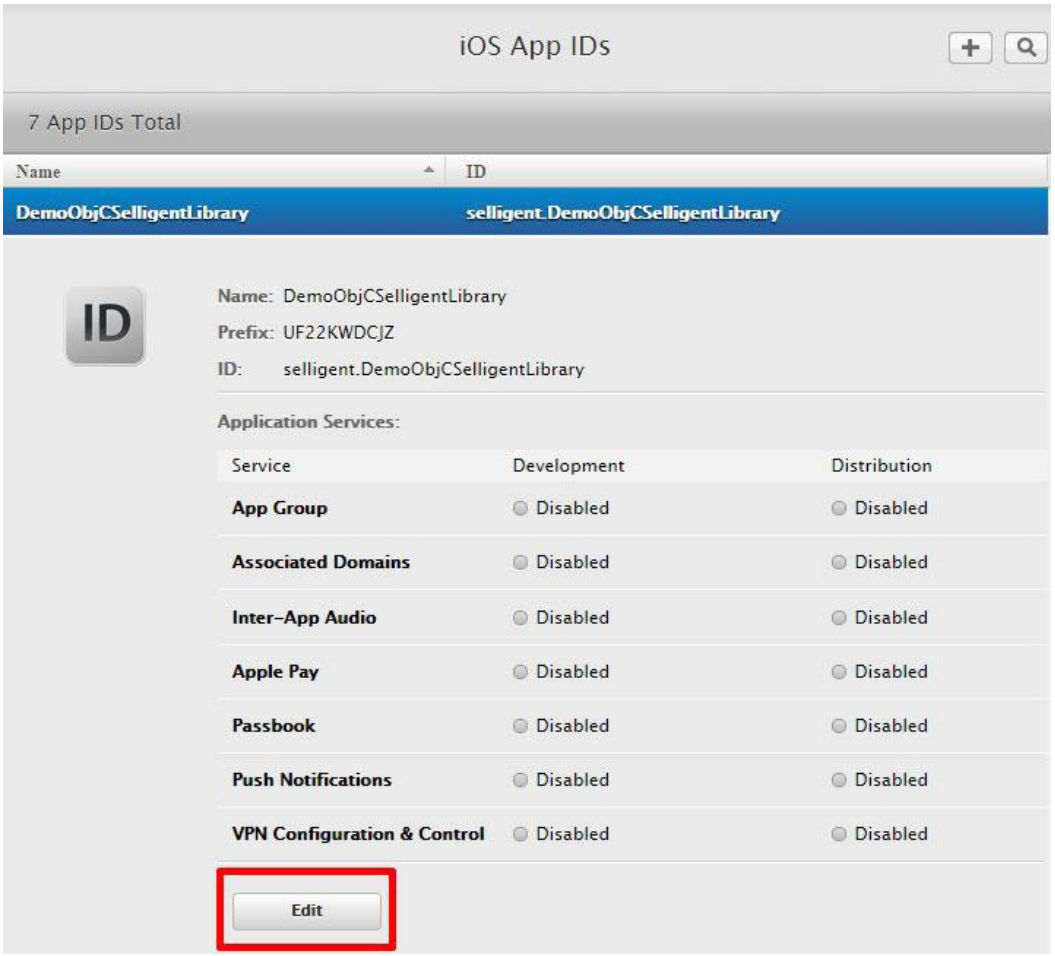
## 2.1 Enable push notifications

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

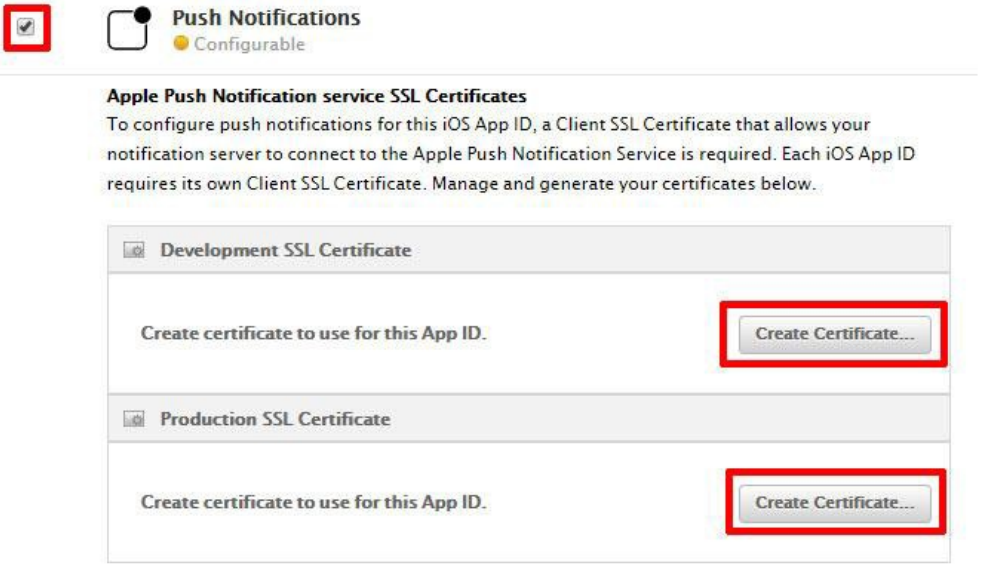



In the Certificates, Identifiers & Profiles page go to **Identifiers** under iOS Apps.

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.



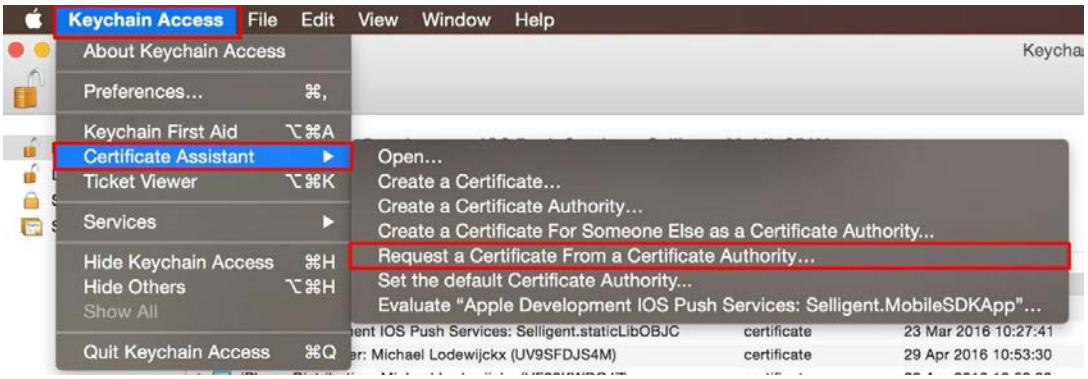

In the list of the application services enable Push Notifications and create an SSL Certificate for Development and Production.



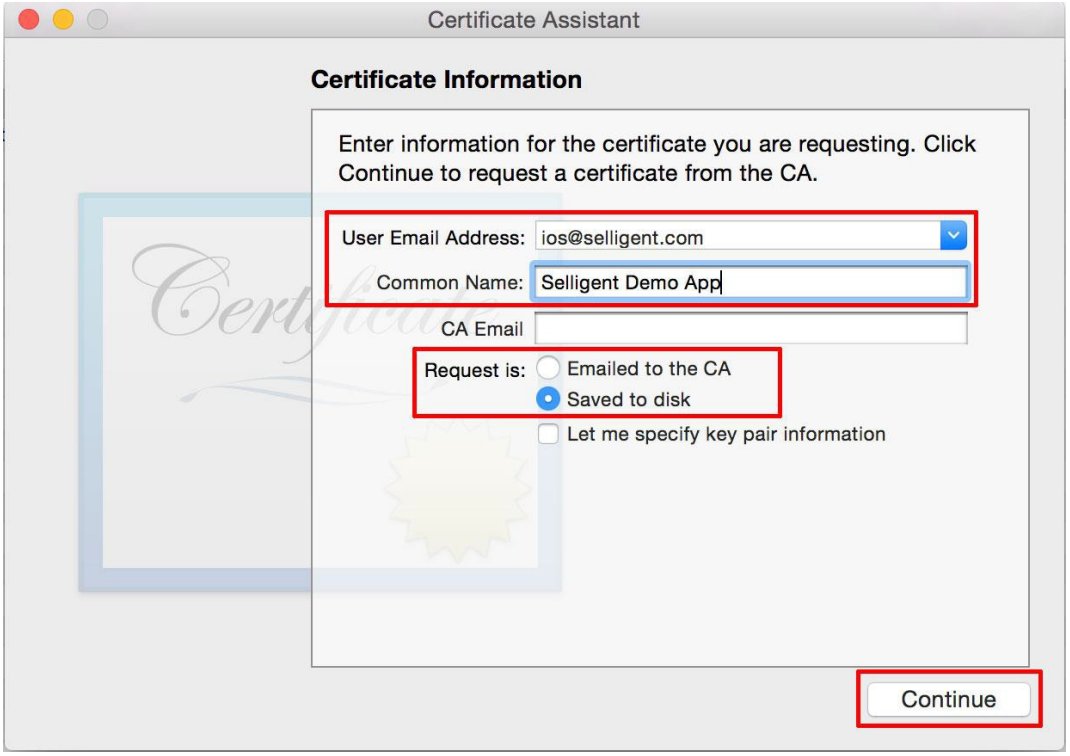
## 2.2 Create and submit a Certificate Signing Request (CSR)

Before going further, we need to generate a "Request a Certificate". For this purpose, you will need the Keychain Acces of MacOS. Search for Keychain Access in Spotlight

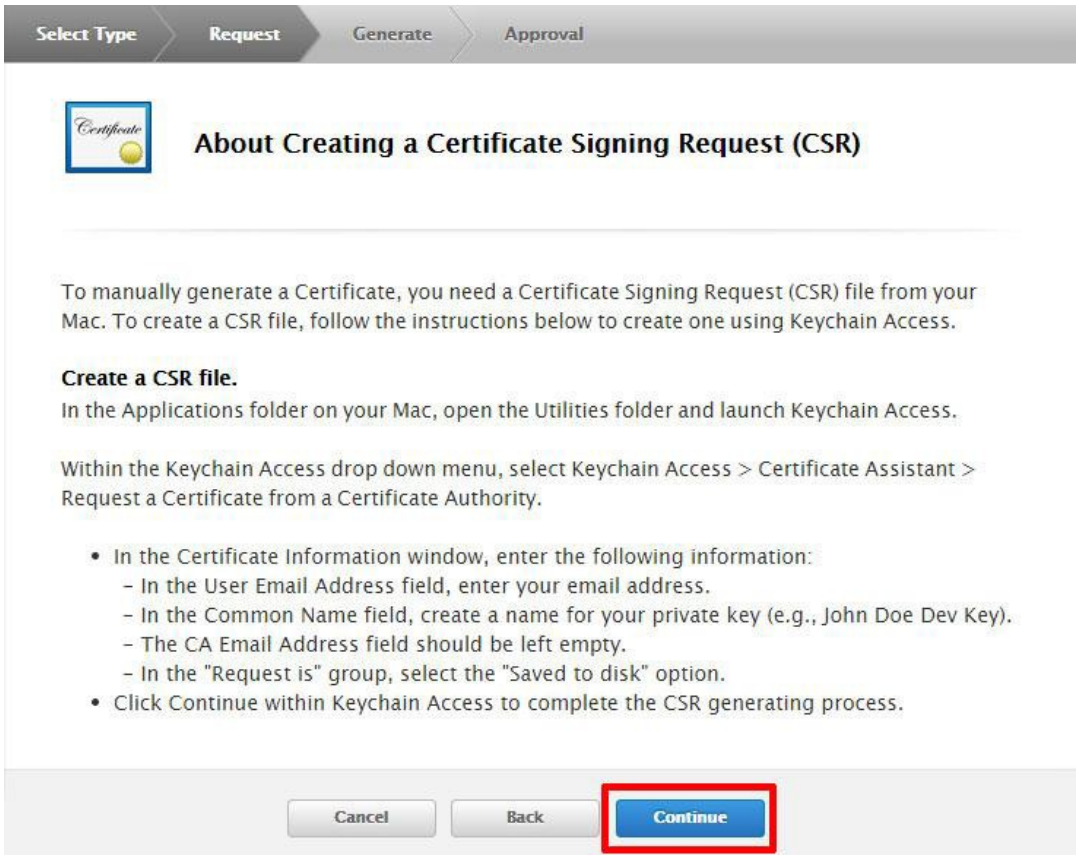
Once Keychain is open, go to **Keychain Access>Certificate Assistant>Request a Certificate From a Certificate Authority**

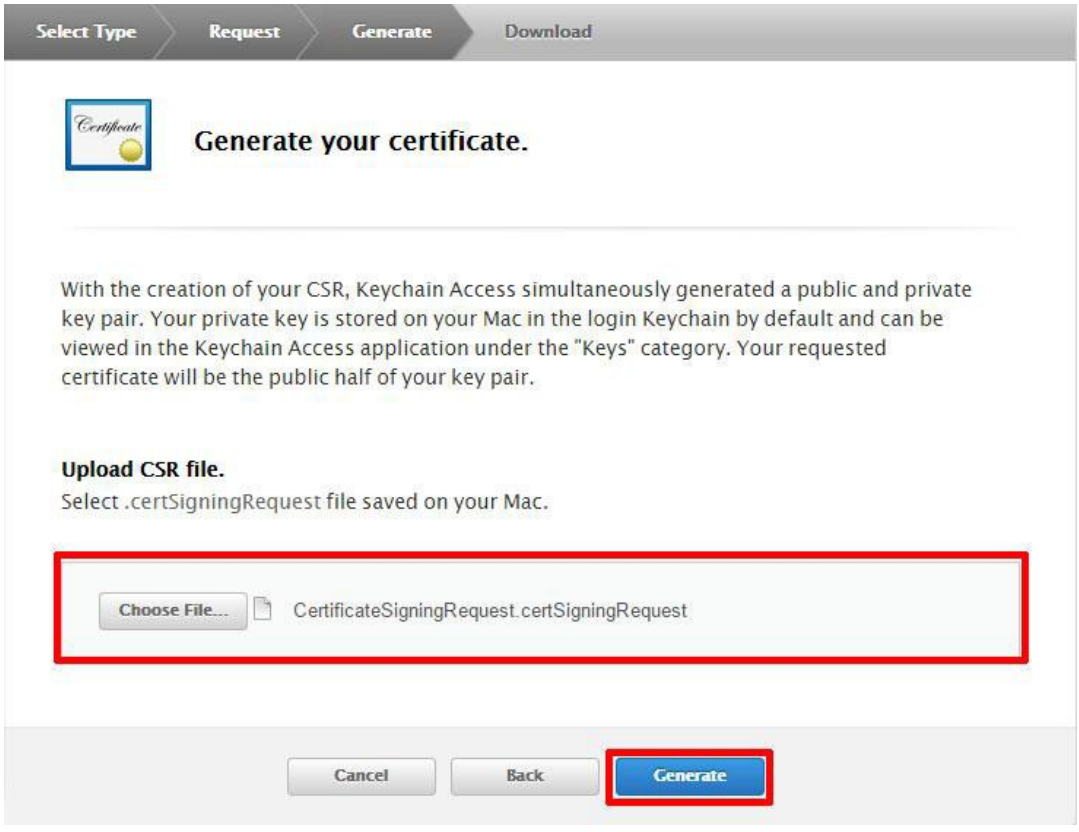



In the opened window fill the **User Email Address** field, the **Common Name** and select the **Saved to disk** option

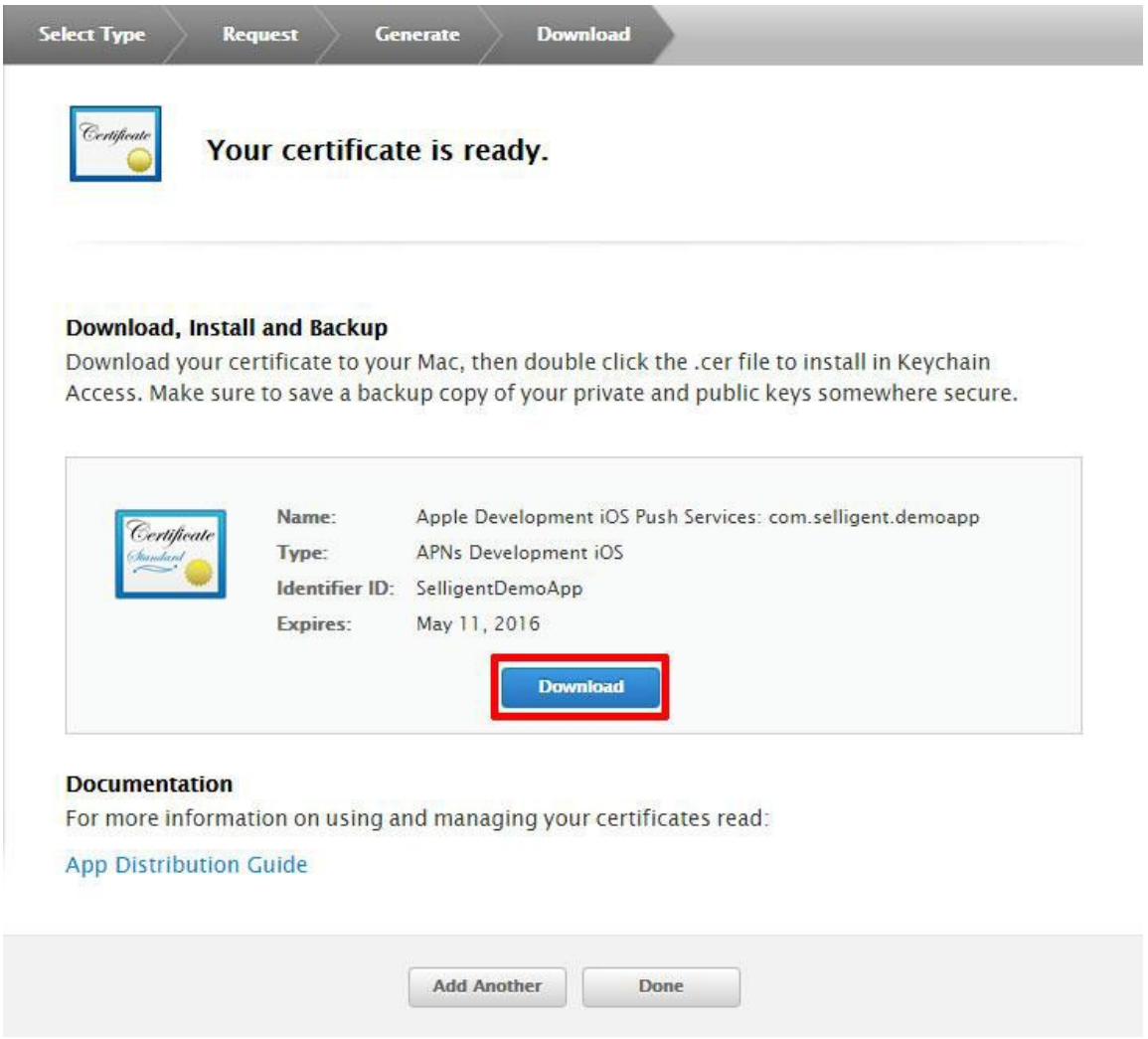


Return to the Certificate Signing Request page, click on Continue, load the .**certSigningRequest** file previously generated by the Keychain and load your .**certSigningRequest.** After selecting the file, click on Generate.



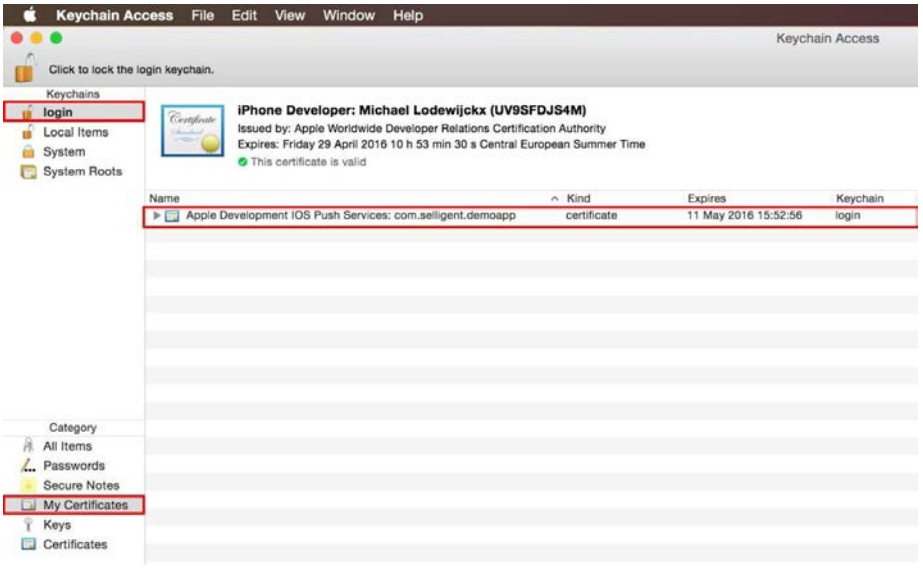



Click on Download to get the **aps_TARGET.cer** file.



## 2.3 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.

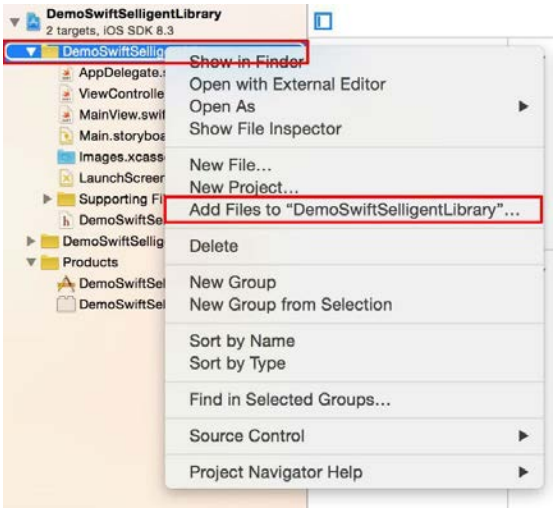



To export the .p12 file, expand the certificate, right click (or CTRL + left click) on the certificate only and select export.

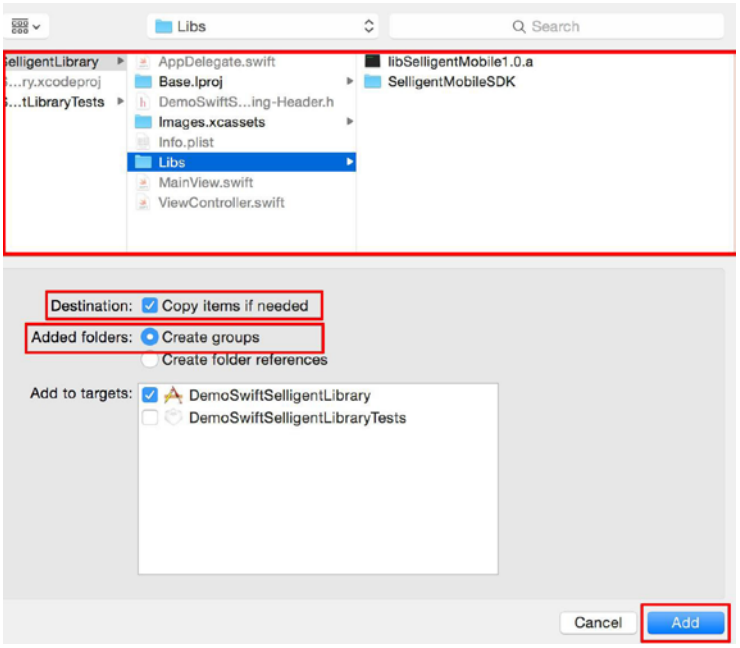

Do the same procedure for the Production certificate.

# 3. Import the library in your target

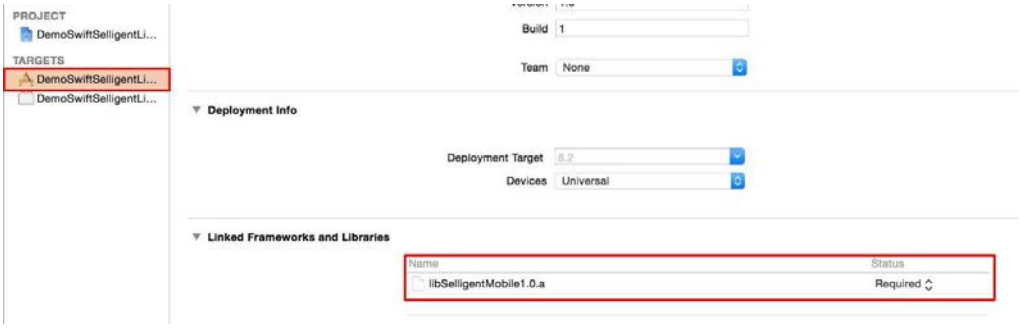
Right click ( CTRL + Left click ) on your app target and select **Add Files to "YOURTARGET"**



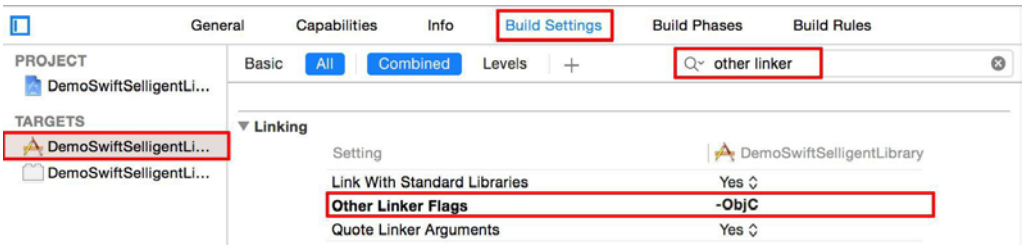

Select the lib folder (the main folder containing the header and the lib files).
Depending on your project check the option "**Copy item if needed**" and select the Create groups option.



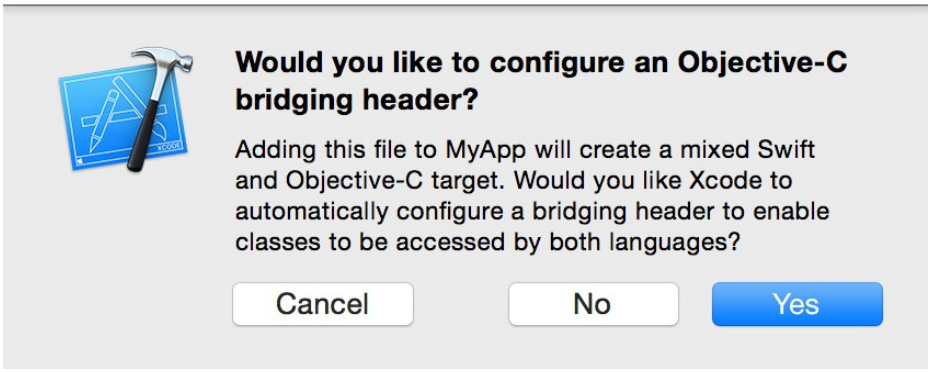

Make sure the library has been added to your target and that its status is **Required**


PROJECT
DemoSwiftSelligentLi...
TARGETS
DemoSwiftSelligentLi...
DemoSwiftSelligentLi...
Build 1
Team None
Deployment Info
Deployment Target 8.2
Devices Universal
Linked Frameworks and Libraries
Name
Status
libSelligentMobile1.0.a
Required

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




For the SWIFT applications, 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:

https://developer.apple.com/library/ios/documentation/Swift/Conceptual/BuildingCocoaApps/MixandM atch.html

# 4. Use SDK

## 4.1 Starting sdk

1. In an Objective C project, import **SMHelper.h** wherever you will need to access to the SDK
2. In a swift project you just need to import **SMHelper.h** in your bridging header file
3. In order 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. Miscellaneous)
            - **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*

**Objective C**

```
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];
```

**Swift**

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

//Create the SMManagerSetting instance
let setting: AnyObject = SMManagerSetting.settingWithUrl(url, clientID: clientID, privateKey: privateKey)


//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.settingWithRefreshType(.RefreshType_Daily)
setting.configureInAppMessageServiceWithSetting(settingIAM)
//Optional - Initialise InApp Content settings
```

```
let settingIAC = SMManagerSettingIAC.settingWithRefreshType(.RefreshType_Daily)
setting.configureInAppContentServiceWithSetting(settingIAC)
```

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

**ObjectiveC**

```
//Starting the library
[[SMManager sharedInstance] startWithLaunchOptions:launchOptions Setting:setting];
```

**Swift**

```
//Start the SDK
SMManager.sharedInstance().startWithLaunchOptions(launchOptions, setting: setting as! SMManagerSetting)
```

- 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: NSData) {
SMManager.sharedInstance().didRegisterForRemoteNotificationsWithDeviceToken(deviceToken)
}

func application(application: UIApplication, didRegisterUserNotificationSettings notificationSettings: UIUserNotificationSettings) {
SMManager.sharedInstance().didRegisterUserNotificationSettings(notificationSettings)
}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError error: NSError) {
SMManager.sharedInstance().didFailToRegisterForRemoteNotificationsWithError(error)
}

func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject : AnyObject]) {
SMManager.sharedInstance().didReceiveRemoteNotification(userInfo)
}
```

Note: you can also implement specific delegates when your app supports background mode (cf. IOS - MobileSDK Reference 1.4.pdf)

## 4.2 Register for push notifications

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

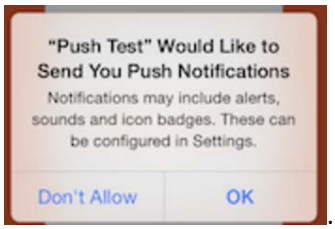
**ObjectiveC**:

```
[[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 wich will let the user to allow push messages for the app on the device (the iOS alert is displayed only once)




## 4.3 Enable In App messages

Assuming that In App message (we will refer to them by IAM) are correctly configured (cf. 4.1 Starting SDK), if you want to use them, 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 1.4.pdf)

## 4.4 In App Content

### 4.4.1 Enabling In App Content

Assuming that In App contents (we will refer to them by IAC) are correctly configured (cf. 4.1 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.
In order 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];
    SMNotificationButtonData *btnData = 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];
}
```

### 4.4.2 Displaying IAC

#### 4.4.2.1 *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. 4.4.3 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"];
        }
}
```

#### 4.4.2.2 *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.

Rem: category are available when listening to NSNotification kSMNotification_Event_DidReceiveInAppContent (cf. 4.4.1 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

**ObjectiveC**

```
- (void) 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.**

**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.**

### 4.4.3 Customize IAC

In order to customize IAC you will have to initialize an instance of **SMInAppContentStyleOptions**. This class provides a number of 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:

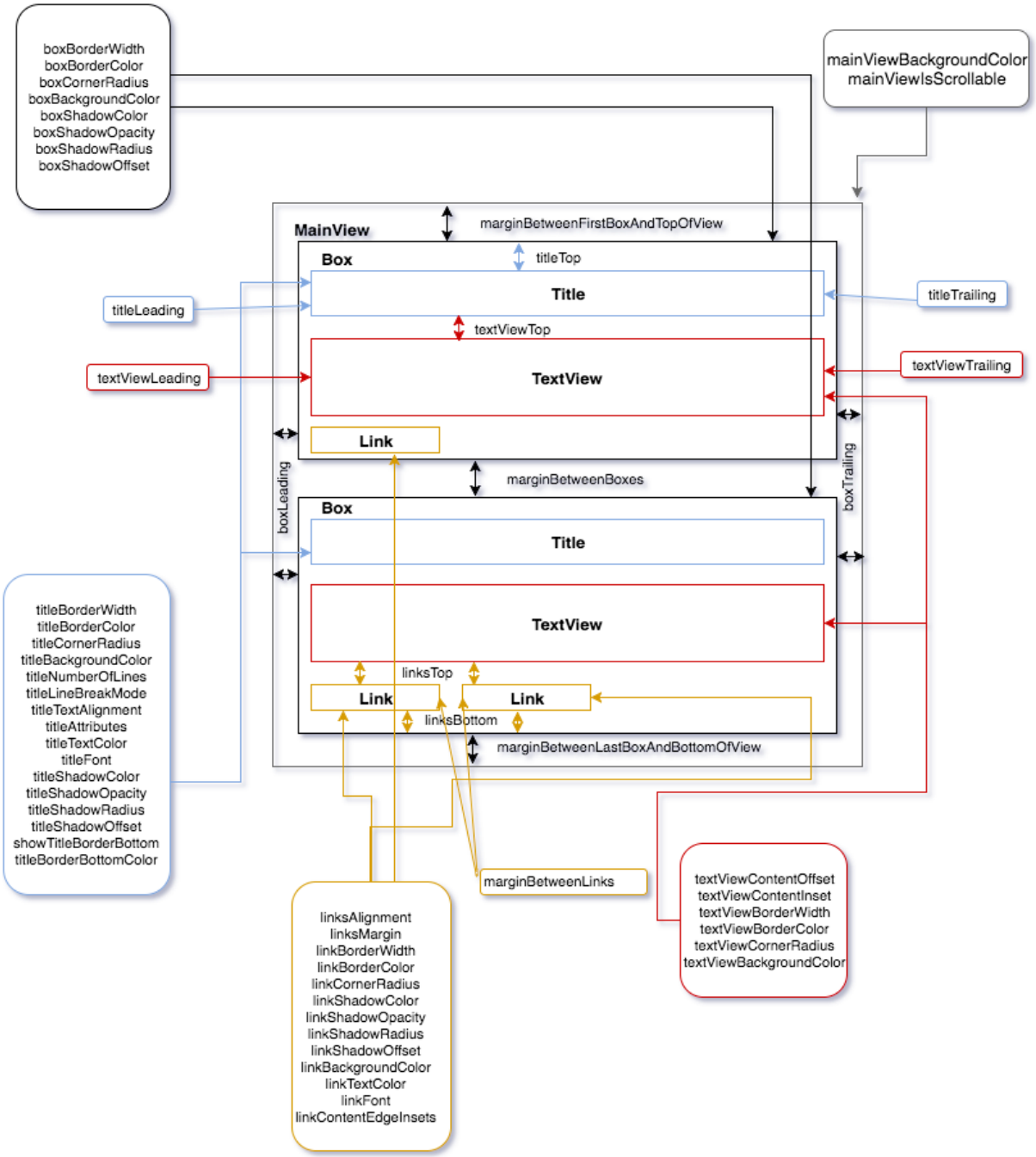
**ObjectiveC**

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

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

```
@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 1.4.pdf

## 4.5 Events

- Sending any set of data to the back-end can be done with the [SMManager sharedInstance] API sendSMEvent:
- 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 fail 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.

**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);
```

**Types of Events and their constructor:**

1. SMEvent
   `+ (instancetype)eventWithDictionary:(NSDictionary *)dict`
   `dict` : a Dictionary containing a string as key and a string as data
2. SMEventUserLogin
3. SMEventUserLogOut
4. SMEventUserRegistration
5. SMEventUserUnregistration
   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

## 4.6 Broadcasted NSNotification

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

- Push Notifications :

**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 Usefull 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

- Data :

**kSMNotification_Data_ButtonData** NSString representing a key to retrieve an object inside NSNotification Use the key kSMNotification_Data_ButtonData to retrive 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

## ObjectiveC:

```
//Listen to differents 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){
}
```

## 4.7 Miscellaneous

### 4.7.1 Remote notification

Useful method 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

```
- (void)displayNotificationID:(NSString *)idNotification
- (void)displayLastReceivedRemotePushNotification
- (NSDictionary *)retrieveLastRemotePushNotification
```

### 4.7.2 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 1.4.pdf for more detailed information about:

- background mode
- all possible values for Constant References