iOS – Using the SDK

PRODUCT MANUAL | 10/05/2021

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.

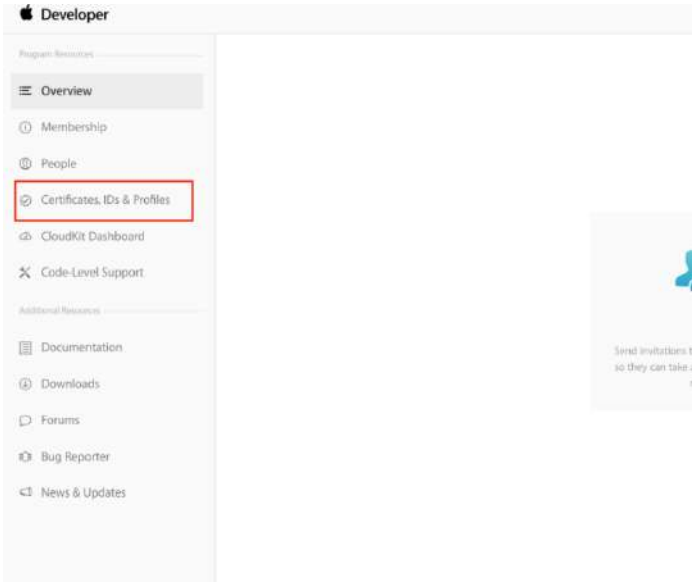
- for more detailed technical reference of the sdk please refer to **iOS - MobileSDK Reference.pdf** document
- for an example of implementation check the **SMSDKTemplate** Xcode project

Important Remark: Core version of the SDK is fully compliant to AppTrackingTransparency framework (starting from iOS and iPadOS 14.5) as we never make any usage of data used to track the device (except if you are using the geolocation version of the sdk). The device id is for example a generated id provided by selligent platform to the sdk at launch of the app.
About the Plot Project Gelolocation version of the sdk please refer to Plot project faqs for more information

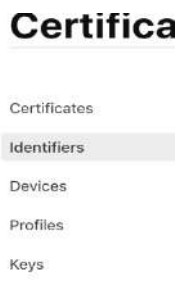
4 Create an APNS (Apple Push Notification Service) Key

4.1 Enable push notifications

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



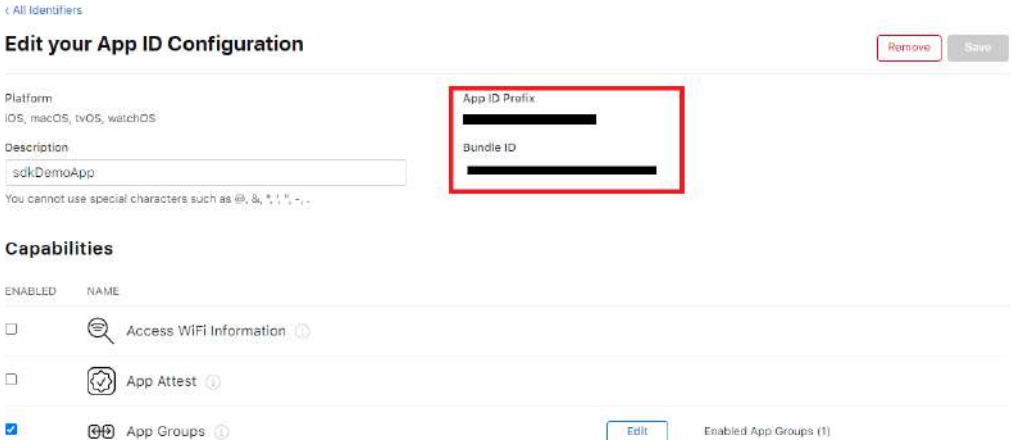
In the **Certificates, Identifiers & Profiles** page go to **Identifiers**.



In the list of your app IDs select the app you want to edit your App ID Configuration and enable the push notifications.

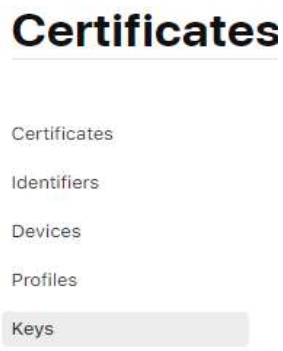
In the list of capabilities enable Push Notifications

In this screen you can also note the Bundle ID and Team ID associated to your app as you will need to provide them to Selligent



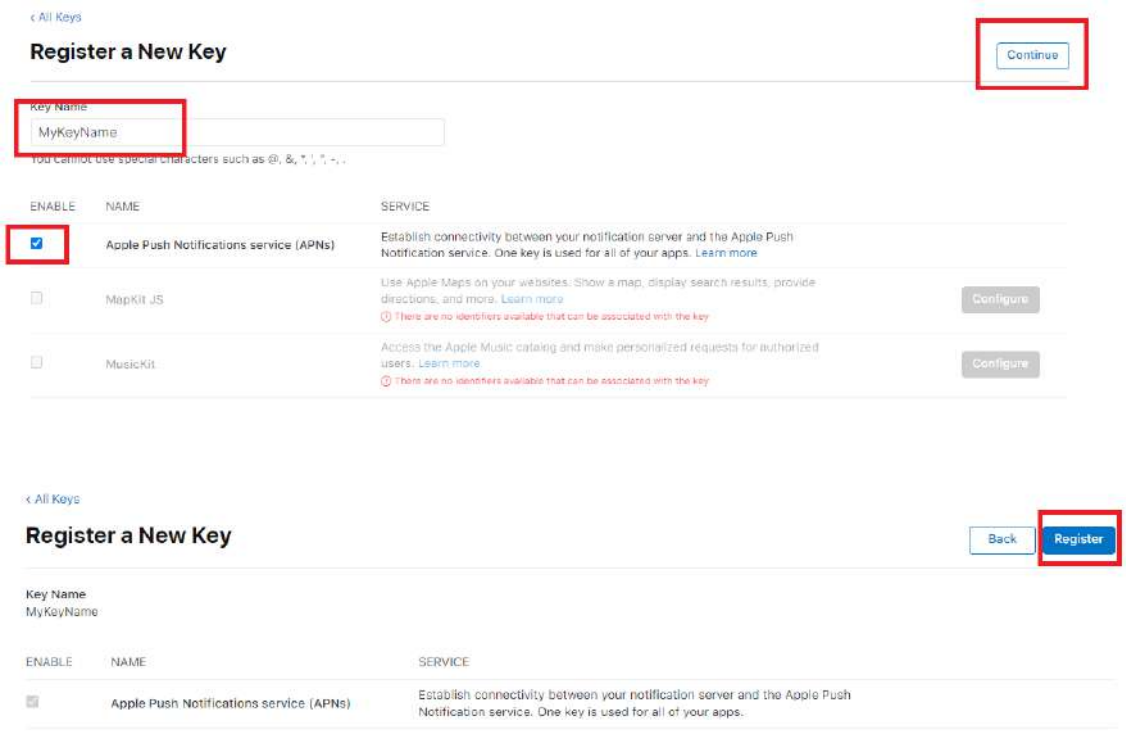
4.2 Create a Key

Back to the **Certificates, Identifiers & Profiles** page go to **Keys**



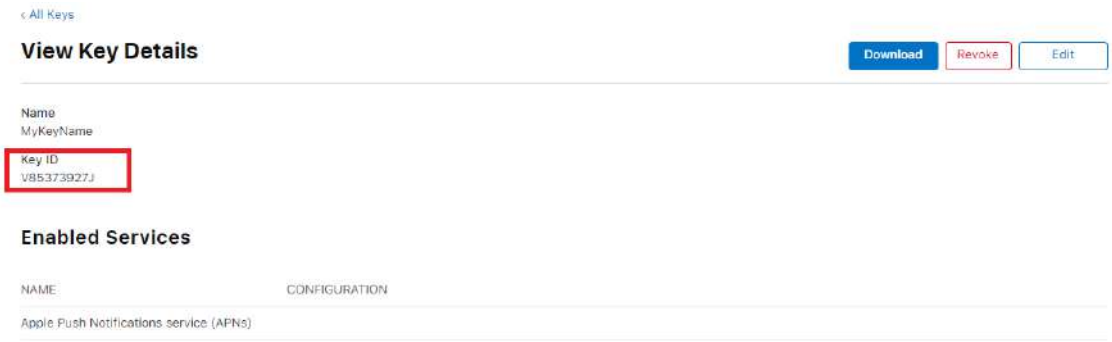
Click on the + button to add a new key

Give a name to your key, Enable Apple Push Notifications service (APNs) and click on “continue” and then “register” in next screen.



In the next screen, carefully read the warning and when you are ready click on download or done.

Back on the Keys menu, whenever you click on your newly created key you will be able to download it if you haven’t done it yet (you can download the key only once!), retrieve the **KeyID**, revoke or edit the Key.



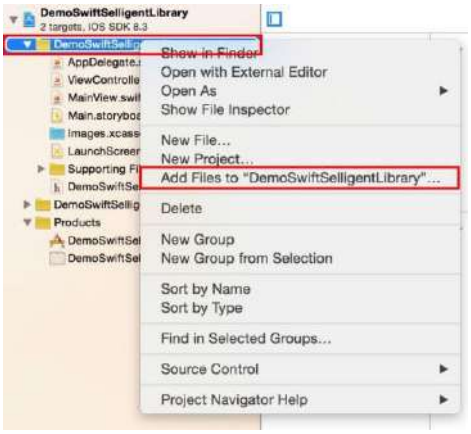
To use Selligent push services you will need to provide to us:

- the downloaded **.p8** file
- the **Key ID**
- your **Team ID**
- the **Bundle ID** of the app

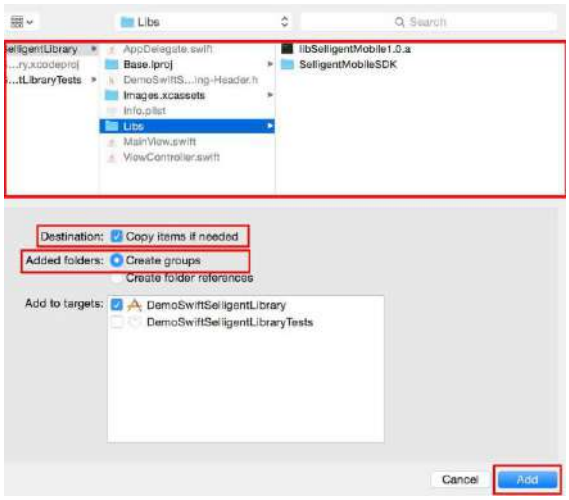
5 Include SDK in your target

5.1 Import the library

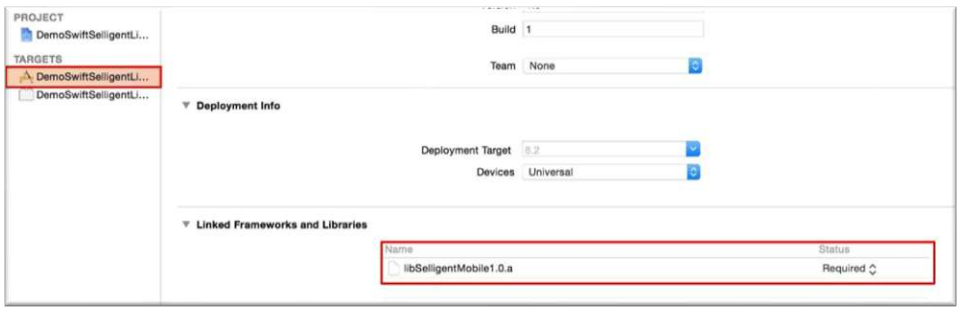
Right click (CTRL + Left click) on your app target and select **Add Files to "YOURTARGET".**



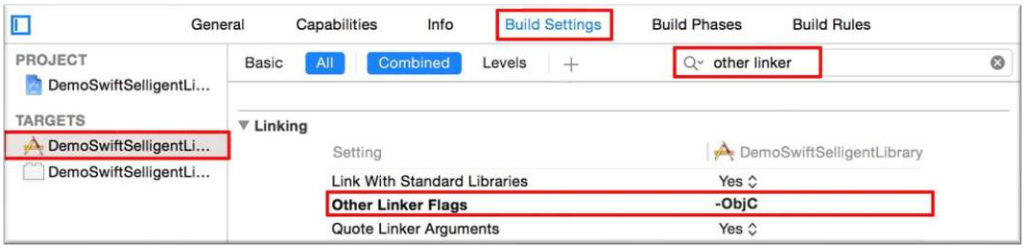
Select the lib folder (the main folder containing the header and the lib files). Depending on your project check the option **Copy item if needed** and select the **Create groups option**.



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



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



5.2 Install the SDK as framework

Since version 2.6 the sdk is also available on github as a **.xcframework**.

You can either install it via cocoapod (refer to the github Readme for the exact procedure) or as stand alone by downloading and drag and dropping the .xcframework package to your project (don't forget to add it to the app and notification extensions in which you will use the framework).

In the case you are using the framework, the import will be:

```
1.  #import <SelligentMobileSDK/SelligentMobileSDK.h>
```

5.3 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 cases, you will need to import the **SMHelper.h** (or **<SelligentMobileSDK/SelligentMobileSDK.h>** in case you are using the framework) to expose those files to Swift. Do it by adding this line:

```
//Static Library
#import "SMHelper.h"

//Framework
#import <SelligentMobileSDK/SelligentMobileSDK.h>
```

More information about this configuration in [apple documentation.]

5.4 Add entries to your app .plist file

5.4.1 Deep Linking

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 in openURL.

```
Swift:

//when sdk calls for example:
let url= URL(string: "yourscheme://anypage")
UIApplication.shared.canOpenURL(url)
//or when:
UIApplication.shared.openURL(url)

//What you need to implement in your AppDelegate:
func application(_ application: UIApplication, open url: URL, sourceApplication: String?,
annotation: Any) -> Bool {
       //here you will be able to parse your url
     print(url.absoluteString)
     return true
}
//or
optional func application(_ app: UIApplication, open url: URL, options:
[UIApplication.OpenURLOptionsKey : Any] = [:]) -> Bool {
       //here you will be able to parse your url
     print(url.absoluteString)
     return true
 }
```

```
Objective-C:

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

//What you need to implement in your AppDelegate:
-(BOOL)application:(UIApplication*) application openURL:(NSURL*) url
sourceApplication:(NSString*) sourceApplication annotation:(id) annotation
{
  //here you will be able to parse your url
  NSLog(@"%@", [url absoluteString]);
  return YES;
}
//or
-(BOOL)application:(UIApplication *)app openURL:(NSURL* )url
options:(NSDictionary<UIApplicationOpenURLOptionsKey,id> *)options
{
  //here you will be able to parse your url
  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 will trigger the 'appscheme://anypage'.

5.4.2 Permission for geolocation

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.

5.5 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 (since sdk v2.1 minimum supported version of plot projects is v3.2.0).

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

  The following must be done in

```
Swift:

application(_:didFinishLaunchingWithOptions:)
```

```
Objective-C:

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)
    - **shouldAddInAppMessageFromPushToInAppMessageList** if you want to add the inapp message associated to a remote notification to the inapp message list. Default value is **false.** (for more information please refer to Display IAM section and more precisely to With your own layout)
    - **clearCacheIntervalValue**: define the interval value for clear of the sdk internal cache (This setting is only used for messages that do not have an expiration date – otherwise expiration date of the message will be taken in count for lifetime of the message in cache)
    - **appGroupId**: provide to the sdk the app groupid (**group.<your group name>**) that you will have previously configured in your apple developer portal. This is mandatory when you use one Notification extension (Service and/or Content) please refer to Notification extensions for more detail on how to setup.
  - Optionally initialise and configure In App Message
  - Optionally initialise and configure In App Content
  - Optionally configure location service (May not be available depending on your sdk version)

Swift:

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

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

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

//Optional - Default value is kSMClearCache_Auto
setting.clearCacheIntervalValue = kSMClearCache_Auto
//Only mandatory when you want to use a Notification extension (Service or/and Content)
setting.appGroupId = "group.<your group name>"

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

//Optional - Initialize location services
setting.configureLocationService()
```

Objective-C:

```objc
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;
setting.shouldAddInAppMessageFromPushToInAppMessageList = FALSE;

//Optional - Default value is kSMClearCache_Auto
setting.clearCacheIntervalValue = kSMClearCache_Auto;
 //Only mandatory when you want to use a Notification extension (Services or/and Content)
setting.appGroupId = @"group.<your group name>";

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

//Initialize 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];
```

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

```
1.  Swift:
2.
3.  //Start the SDK
4.  SMManager.sharedInstance().start(launchOptions:launchOptions, setting: setting)
```

```
1.  Objective-C:
2.
3.  //Starting the library
4.  [[SMManager sharedInstance] startLaunchOptions:launchOptions Setting:setting];
```

6.2 Push notifications

6.2.1 Register for push notification

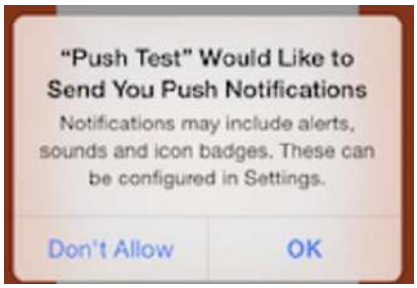
Starting the library will not register for remote notification. You will need to call:

```
1.  Swift:
2.
3.  SMManager.sharedInstance().registerForRemoteNotification()
```

```
1.  Objective-C:
2.
3.  [[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 Provide Device token to Selligent platform

If user has accepted to receive remote notification, the device will be given by apple a device token that you must provide to platform. To do that you must implement:

```
1.  Swift:
2.
3.  func application(_ application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
    deviceToken: Data) {
4.    SMManager.sharedInstance().didRegisterForRemoteNotifications(withDeviceToken:deviceToken)
5.  }
```

```
1.  Objective-C:
2.
3.  -(void)application:(UIApplication*)application
    didRegisterForRemoteNotificationsWithDeviceToken:(NSData*)deviceToken {
4.     [[SMManager sharedInstance] didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];
5.  }
```

6.2.3 Provide information to the sdk about remote notification status on the device

To do this you can implement (optionally – sdk is able to get this information by itself)

```
Swift:

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

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

```
Objective-C:

-(void)application:(UIApplication*)application
didRegisterUserNotificationSettings:(UIUserNotificationSettings *)notificationSettings {

  [[SMManager sharedInstance] didRegisterUserNotificationSettings:notificationSettings];
}

- (void)application:(UIApplication *)application
didFailToRegisterForRemoteNotificationsWithError:(NSError *)error {

  [[SMManager sharedInstance] didFailToRegisterForRemoteNotificationsWithError:error];
}
```

Since iOS 10 didRegisterUserNotificationSettings has been deprecated

6.2.4 Disable Selligent push notifications

By default, Selligent Push are enabled and if the device is correctly registered to iOS than you don't have anything to do.

But if you want to disable only Selligent pushes (Provided that device, accept push notification, that means that the device will still be able to receive third parties pushes but not one sent with Selligent Marketing Cloud), you have the possibility to do so by using:

```
- (void) enableRemoteNotification;
- (void) disableRemoteNotification;
```

Keep in mind that if at device level push notifications are not registered and enabled the enableRemoteNotification will not be enough to display Selligent pushes.

6.2.5 Listening and displaying the push notifications

6.2.5.1 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**:

```
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
}
```

```
Objective-C:

#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))completionHandler
{

            [[SMManager sharedInstance] willPresentNotification:notification];
   completionHandler(UNNotificationPresentationOptionAlert);

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

   //OR in this case the sdk will be in charge to call completionHandler:
   //[[SMManager sharedInstance] didReceiveNotificationResponse:response
withCompletionHandler:completionHandler];
}
```

when you use geolocation version of the sdk and plotprojects framework you will mandatorily have, in

application(_:didFinishLaunchingWithOptions:)

to assign the delegate :

let center = UNUserNotificationCenter current();
center.delegate = self;

before calling to

SMManager.sharedInstance().start(launchOptions:launchOptions, setting:setting)

6.2.5.2 App that does not build against iOS + 10

Implement methods described in **[SMManager(RemoteNotification)]** in your UIApplication's delegate.

```
Swift:

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

```
Objective-C:

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

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

6.2.6 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.

```
Swift:

SMManager.sharedInstance().displayNotificationID("idNotification")
```

```
Objective-C:

[[SMManager sharedInstance] displayNotificationID:@"idNotification"];
```

- Display last received remote push notification.

```
Swift:

SMManager.sharedInstance().displayLastReceivedPushNotification()
```

```
Objective-C:

[[SMManager sharedInstance] displayLastReceivedRemotePushNotification];
```

- Retrieve last push notification (return dictionary with id and title of the notification)

```
Swift:

SMManager.sharedInstance().retrieveLastRemotePushNotification()
```

```
Objective-C:

[[SMManager sharedInstance] retrieveLastRemotePushNotification];
```

- Display and process the InApp message on app side – In this case the sdk will not be in charge of the display of the InApp message content – you can refer to Display IAM linked to a remote notification on app side for how to do this

6.2.7 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 InApp 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:

```
Swift:

SMManager.sharedInstance().enable(inAppMessage: true)
```

```
Objective-C:

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

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

6.3.2 Display IAM

To retrieve your InAppMessage you must listen to **kSMNotification_Event_DidReceiveInAppMessage** (see Broadcasts).

6.3.2.1 With SDK helper functions

This will provide you an **NSArray** of **NSDictionary** containing 2 properties: **id** and **title** for each InAppMessage available for the device.

```
Swift:

NotificationCenter.default.addObserver(self, selector: #selector( didReceiveInAppMessage(_:)),
name: NSNotification.Name(rawValue: kSMNotification_Event_DidReceiveInAppMessage), object: nil)

@objc func didReceiveInAppMessage(_ notif : Notification){
  //example to display first received in app message
  let dictIAM = notif.userInfo
  let array : Array =  dictIAM![kSMNotification_Data_InAppMessage] as! Array<AnyObject>
}
```

```
Objective-C:

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


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

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

```
Swift:

SMManager.sharedInstance().displayNotificationID("idNotification")
```

```
Objective-C:

[[SMManager sharedInstance] displayNotificationID:@"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)

6.3.2.2 With your own layout

Since **sdk v 2.5**, once notified that InApp messages are available, you can retrieve them and store them on app side. This will give you more flexibility when displaying them. To do that you will have to call first:

```
- (NSArray*) getInAppMessages;
```

This method will provide you an array of **SMInAppMessage.**

When an InApp message has been displayed to the user you will have to do a **mandatory** call to:

```
- (NSArray*) setInAppMessageAsSeen:(SMInAppMessage*) message;
```

This will permit to the selligent platform to set the adequate (open) event to the In App message

If you wish to remove the flag of a message as seen you can do so by calling (this will not change the event state at platform level):

```
- (NSArray*) setInAppMessageAsUnseen:(SMInAppMessage*) message;
```

You also have a helper method that will allow you to set a message as deleted (in this case the message will not be provided to you anymore)

```
- (NSArray*) setInAppMessageAsDeleted:(SMInAppMessage*) message;
```

If an InApp message has links associated to it, you can retrieve them with arrayIAMLinks on the **SMInAppMessage** object, you can then display the link to the user the way you prefer, and once user has triggered the link you will have to do a **mandatory** call to:

```
- (NSArray*) executeLinkAction:(SMLink*)link Type:(SMInAppMessage*)message;
```

```
Swift:

 //retrieve InApp messages and display them or store them for future usage and custom display
 let inAppMessages : [Any] = SMManager.sharedInstance().getInAppMessages()
        for item in inAppMessages {
            let message = item as! SMInAppMessage
            let title = message.title
//refer to sdk reference document for more available properties (SMBaseMessage and SMInAppMessage class)
//... here for example, we will suppose that the message has been displayed  to the user
//If the user has seen the message you will have to (and this MANDATORY to inform the services)
            SMManager.sharedInstance().setInAppMessageAsSeen(message)
//If the user has triggered a link you will have to (and this MANDATORY to inform the services)
//get the triggered link (in this case the first one)
            let link = message.arrayIAMLinks[0]
            SMManager.sharedInstance().executeLinkAction(link as? SMLink, inAppMessage: message)
        }
```

```
Objective-C:

//retrieve InApp messages and display them or store them for future usage and custom display
    NSArray *inAppMessages = [[SMManager sharedInstance] getInAppMessages];
    for (SMInAppMessage *message in inAppMessages) {
        NSString * title = message.title;
//refer to sdk reference document for more available properties (SMBaseMessage and SMInAppMessage class)
//... here for example, we will suppose that the message has been displayed  to the user
//If the user has seen the message you will have to (and this MANDATORY to inform the services)
        [[SMManager sharedInstance] setInAppMessageAsSeen:message];
//If the user has triggered a link you will have to (and this MANDATORY to inform the services)
//get the triggered link (in this case the first one)
        SMLink *link = message.arrayIAMLinks[0];
        [[SMManager sharedInstance] executeLinkAction:link InAppMessage:message];
    }
```

6.3.3 Implement WKNavigationDelegate methods on app side

When an InApp Message of type url is displayed by the sdk in a web view, if you wish for example to process the clicked link inside the web view on the app side, the sdk gives you the ability to do so by having a class in your app that implement the **WKNavigationDelegate** and providing an instance of this class to the sdk.

To achieve this, you can call the method **inAppMessageWKNavigationDelegate:** and give it as parameter, the instance of the class that will be in charge of implementing the **WKNavigationDelegate** methods.

You will also need to call **removeViewController** whenever you wish to close the web view controller displayed by the sdk.

```
- (void) inAppMessageWKNavigationDelegate:(id <WKNavigationDelegate>) delegate;
- (void) removeViewController;
```

Swift:

```swift
//provide an instance of a class implementing WKNavigatonDelegate to the sdk.(you can do that for example at launch time)

SMManager.sharedInstance().inAppMessageWKNavigationDelegate(AppWKNavigationDelegateExample())

//your class will look like . in this case with decidePolicy
import Foundation
class AppWKNavigationDelegateExample: NSObject, WKNavigationDelegate{
 func webView(_ webView: WKWebView, decidePolicyFor navigationAction: WKNavigationAction, decisionHandler: @escaping (WKNavigationActionPolicy) -> Void) {
        if navigationAction.navigationType == .linkActivated  {
            if let url = navigationAction.request.url,
                let host = url.host, !host.hasPrefix("www.google.com"),
                UIApplication.shared.canOpenURL(url) {
                UIApplication.shared.open(url)
                print(url)
                print("Redirected to browser. No need to open it locally")
                decisionHandler(.cancel)
            } else {
                print("Open it locally")
                decisionHandler(.allow)
            }
        } else {
            print("not a user click")
            decisionHandler(.allow)
        }
    }
}
```

Objective-C:

```objc
//provide an instance of a class implementing WKNavigatonDelegate to the sdk.(you can do that for example at launch time)
 [[SMManager sharedInstance] inAppMessageWKNavigationDelegate:[AppWKNavigationDelegateExample new]];

//your class will look like
#import <WebKit/WebKit.h>
#import <Foundation/Foundation.h>
@interface AppWKNavigationDelegateExample: NSObject <WKNavigationDelegate>
@end

@implementation AppWKNavigationDelegateExample
- (void)webView:(WKWebView *)webView decidePolicyForNavigationAction:(nonnull WKNavigationAction *)navigationAction decisionHandler:(nonnull void (^)(WKNavigationActionPolicy))decisionHandler
{
    if (navigationAction.navigationType == WKNavigationTypeLinkActivated) {
        if (navigationAction.request.URL) {
            NSLog(@"%@", navigationAction.request.URL.host);
            if (![navigationAction.request.URL.resourceSpecifier containsString:@"anypath"]) {
                if ([[UIApplication sharedApplication] canOpenURL:navigationAction.request.URL]) {
                    [[UIApplication sharedApplication] openURL:navigationAction.request.URL];
                    decisionHandler(WKNavigationActionPolicyCancel);
                }
            } else {
                decisionHandler(WKNavigationActionPolicyAllow);
            }
        }
    } else {
        decisionHandler(WKNavigationActionPolicyAllow);
    }
}
@end
```

6.3.4 Display IAM linked to a remote notification on app side

When an InApp Message is available with the payload of a remote notification you can also decide to process content and display it on app side

To achieve this, you can call the method **inAppMessageDelegate:** and give it as parameter, the instance of the class that will be in charge of implementing the **SMManagerInAppMessageDelegate** methods.

```
- (void) inAppMessageDelegate:(id <SMManagerInAppMessageDelegate>) delegate;
```

In the class conforming to **SMManagerInAppMessageDelegate** protocol you will need to implement

```
Swift:

//provide an instance of a class implementing SMManagerInAppMessageDelegate to the sdk.(you can do that for example at launch time)
SMManager.sharedInstance().inAppMessageDelegate(AppInAppMessageDelegateExample())

//your class will look like . in this case with decidePolicy
import Foundation
class AppInAppMessageDelegateExample: NSObject, SMManagerInAppMessageDelegate{
func display(_ inAppMessage: SMInAppMessage) {
   print(inAppMessage.title!)
   print(inAppMessage.body!)
   print(inAppMessage.arrayIAMLinks!)
}
```

```
Objective-C:

//provide an instance of a class implementing WKNavigatonDelegate to the sdk.(you can do that for example at launch time)
 [[SMManager sharedInstance] inAppMessageDelegate:[AppInAppMessageDelegateExample new]];

//your class will look like
#import <Foundation/Foundation.h>
@interface AppInAppMessageDelegateExample: NSObject <SMManagerInAppMessageDelegate>
@end

@implementation AppInAppMessageDelegateExample
-(void)displayInAppMessage:(SMInAppMessage *)inAppMessage
{
    NSLog(@"title : %@", inAppMessage.title);
    NSLog(@"content : %@", inAppMessage.body);
    NSLog(@"links : %@", inAppMessage.arrayIAMLinks);
}
@end
```

6.3.5 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 **NSArray** instance with an **NSDictionary** of **SMNotificationMessage** with title and id as properties.

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**)

```
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];
}
```

```
Objective-C:

[[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];
}
```

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:

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

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

```
+(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):

```
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)
  }
}
```

```
Objective-C:

//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];
}
```

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:**

```
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)
```

```
Objective-C:

//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];
```

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:**

```
Swift:

@IBOutlet weak var yourImageContainer: UIView!

func prepare(for segue: UIStoryboardSegue?, sender identifier: Any?, isEqualToString : ) {
 if (segue.identifier == "iacSegue") {
    yourImageContainer = segue.destination
    yourImageContainer.category = "news"
 }
}
```

```
Objective-C:

@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:

```
- (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.

```
1.  - (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**

```
1.  - (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]**

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

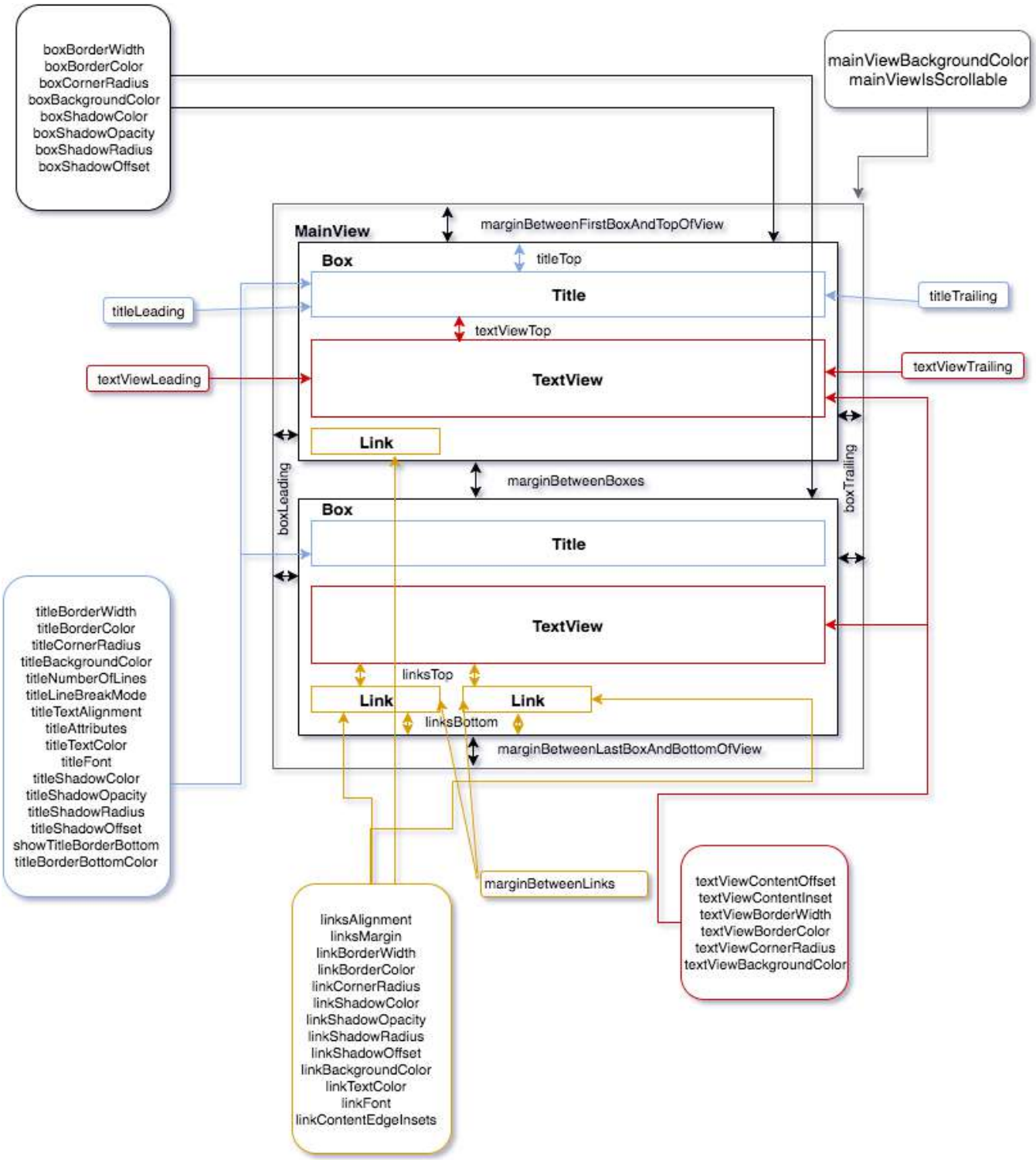
or pass it as a parameter to your **SMInAppContentViewController** constructor:

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

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

```
1.  @property (nonatomic) UIActivityIndicatorViewStyle activityIndicatorStyle;
2.  @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:

```
1.  Swift:
2.
3.  UITextView.appearanceWhenContained(in: SMInAppContentHTMLViewController.self, nil).font =
    UIFont(name: "Marker Felt", size: 10)
4.  UITextView.appearanceWhenContained(in: SMInAppContentHTMLViewController.self, nil).textColor =
    UIColor.red
```

```
Objective-C:

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

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 download a **specific version of the sdk**, contact selligent support for more information and embed **plotprojects.framework** in your app (since sdk v2.1 minimum supported version of plot projects is v3.2.0).

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:

```
Swift:

SMManager.sharedInstance().enableGeoLocation()
```

```
Objective-C:

[[SMManager sharedInstance] enableGeoLocation];
```

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

```
Swift:

SMManager.sharedInstance().disableGeoLocation()
```

```
Objective-C:

[[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**

```
Swift:

SMManager.sharedInstance().requestLocationAuthorisation(kSMLocationAuthorisationType_Always)
```

```
Objective-C:

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

For more information on **plotconfig.json** check PlotProjects documentation.

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

6.6 Events

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

  This method takes in parameter a **SMDeviceInfos** object. This object contains for the moment one unique property **externalId**:

```
Swift:

let deviceInfos = SMDeviceInfos(externalID: "12345")
SMManager.sharedInstance().sendDeviceInfo(deviceInfos)
```

```
Objective-C:

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

Three possible constructors:

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

```
Swift:

let event = SMEventUserRegistration.event(withEmail: "usermail@mail.com", dictionary: [
    "key": "value"
]) //with alternate key/value example: SMEventUserRegistration(["userID": "1234"])
//Optional
event.shouldCache = true //not necessary as it is the default value
event.applyBlockSuccess({ (success) -> Void in
            print("success")
        }){(failure) -> Void in
                print("failure")
        }
SMManager.sharedInstance().send(event)
```

```
Objective-C:

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

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

```
Swift:

let event = SMEventUserUnregistration.event(withEmail: "usermail@mail.com", dictionary: [
    "key": "value"
]) //with alternate key/value example: SMEventUserUnregistration( ["userID": "1234"])
//Optional
event.shouldCache = true //not necessary as it is the default value
event.applyBlockSuccess({ (success) -> Void in
            print("success")
        }){(failure) -> Void in
                print("failure")
        }
SMManager.sharedInstance().send(event)
```

```
Objective-C:

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

```
1.  + (instancetype)eventWithEmail:(NSString *)mail
2.  + (instancetype)eventWithEmail:(NSString *)mail Dictionary:(NSDictionary<NSString*,NSString*>
    *)dict;
3.  +(instancetype)eventWithDictionary:(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

```
1.  Swift:
2.
3.  let event = SMEventUserLogin.event(withEmail: "usermail@mail.com", dictionary: [
4.      "key": "value"
5.  ])
6.  //with alternate key/value example: SMEventUserLogin(["userID": "1234"])
7.  //Optional
8.  event.shouldCache = true //not necessary as it is the default value
9.  event.applyBlockSuccess({ (success) -> Void in
10.             print("success")
11.         }){(failure) -> Void in
12.                 print("failure")
13.         }
14. SMManager.sharedInstance().send(event)
15.
```

```
1.  Objective-C:
2.
3.  SMEventUserLogin *event = [SMEventUserLogin eventWithEmail:@"usermail@mail.com" Dictionary:
    @{@"key": @"value"}];
4.  //with alternate key/value example: [SMEventUserLogin eventWithDictionary: @{@"userID":
    @"1234"}];
5.  //Optional
6.  event.shouldCache = TRUE; //not necessary as it is the default value
7.  [event applyBlockSuccess:^(SMSuccess *success) {
8.    NSLog(@"success");
9.  } BlockFailure:^(SMFailure *failure) {
10.   NSLog(@"failure");
11. }];
12. //Send
13. [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

```
Swift:

let event = SMEventUserLogout.event(withEmail: "usermail@mail.com", dictionary: [
    "key": "value"
])
//with alternate key/value example: SMEventUserLogout(["userID": "1234"])
//Optional
event.shouldCache = true //not necessary as it is the default value
event.applyBlockSuccess({ (success) -> Void in
            print("success")
        }){(failure) -> Void in
                print("failure")
        }
SMManager.sharedInstance().send(event)
```

```
Objective-C:

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

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.

```
Swift:

let event = SMEvent.init(dictionary: ["key": "value"])
//Optional
event.shouldCache = true //not necessary as it is the default value
event.applyBlockSuccess({ (success) -> Void in
            print("success")
            }) {(failure) -> Void in
                print("failure")
            }
SMManager.sharedInstance().send(event)
```

```
Objective-C:

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];
```

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

```
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){}
```

```
Objective-C:

//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{
}
```

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.

```
1.  Swift:
2.
3.  let smSettings = SMManagerSetting(url: currentUrl, clientID: clientID, privateKey: privateKey)
4.  SMManager.sharedInstance().reload(smSettings)
```

```
1.  Objective-C:
2.
3.  SMManagerSetting *smSettings = [SMManagerSetting settingWithUrl:currentUrl ClientID:clientID
    PrivateKey:privateKey];
4.  [[SMManager sharedInstance] reloadSetting:smSettings];
```

6.8.2 LogLevel

```
1.  - (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!!!

```
1.  Swift:
2.
3.  SMManager.sharedInstance().applyLoglevel(.All)
```

```
1.  Objective-C:
2.
3.  [[SMManager sharedInstance] applyLoglevel:kSMLogLevel_All];
```

6.8.3 Retrieve Device id

In case you need the device id you can use this helper method :

```
1.  - (NSString*_Nullable)deviceID;
```

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

To be notified if device is has changed, the application must register to correct notification **kSMNotification_Event_DidReceiveInDeviceId**.

The Notification will provide the app with the new device id (key **kSMNotification_Data_DeviceId**)

```
Swift:

//retrieve device id
SMManager.sharedInstance().deviceID()

//listen to broadcast in case device id has changed and retrieve new device id
NotificationCenter.default.addObserver(self, selector: #selector( didReceiveDeviceId (_:)),
name: NSNotification.Name(rawValue: kSMNotification_Event_ DidReceiveDeviceId), object: nil)

@objc func didReceiveDeviceId (_ notif : Notification){
  let dict = notif.userInfo
  let deviceid : String =  dict![kSMNotification_Data_ DeviceId]
}
```

```
Objective-C:

//retrieve device id
[[SMManager sharedInstance] deviceID];

//listen to broadcast in case device id has changed and retrieve new device id
[[NSNotificationCenter defaultCenter] addObserver: self selector:
@selector(didReceiveDeviceId:) name: kSMNotification_Event_DidReceiveDeviceId object: nil];


-(void)didReceiveDeviceId:(NSNotification*)notif{
  NSDictionary *dict = [notif userInfo];
  NSString *deviceID = [dict  objectForKey:kSMNotification_Data_DeviceId];
}
```

Note: Do not forget to check IOS - MobileSDK Reference.pdf for more detailed information about:

- background mode
- all possible values for Constant References

6.9 Notification Extensions

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

Those functionalities available through Selligent Marketing Cloud are:

- Push action buttons - buttons to be displayed inside the notification center.
- Decrypt an encrypted payload.
- Rich push content.

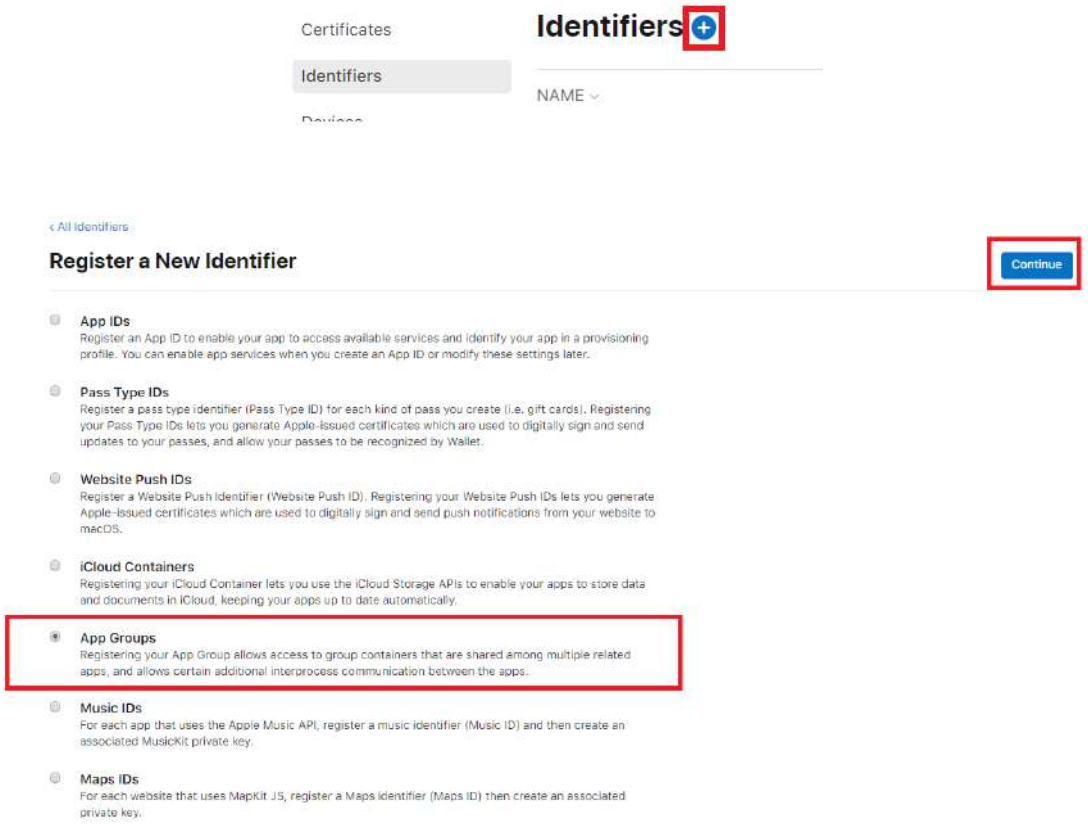
The features you will find under this section are only available for iOS 10 and later devices.

They are also only configurable in Selligent Marketing Cloud.

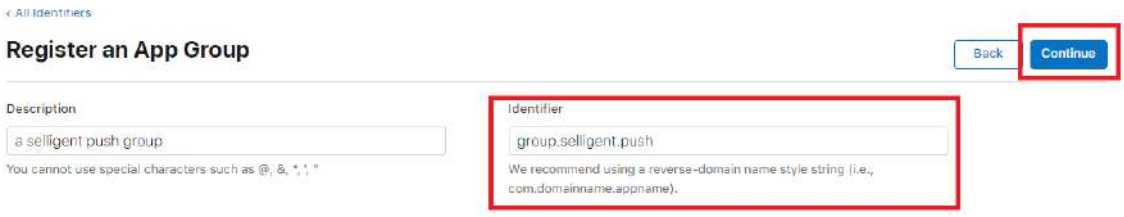
6.9.1 General set up

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

Connect to your account and go to Identifiers to create a new App Group

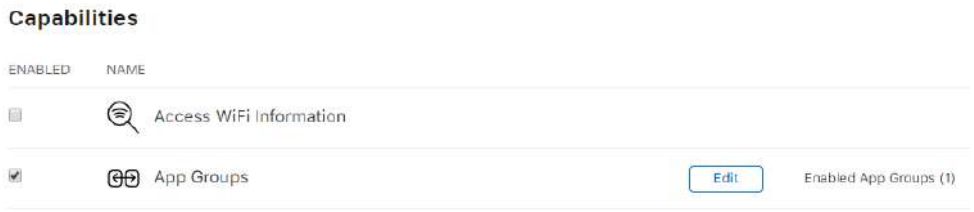


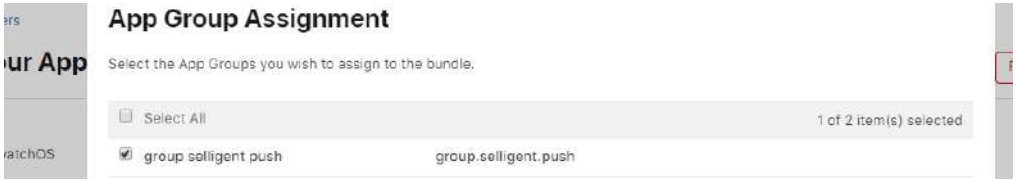
Create an AppGroup named **group.<your group name>**



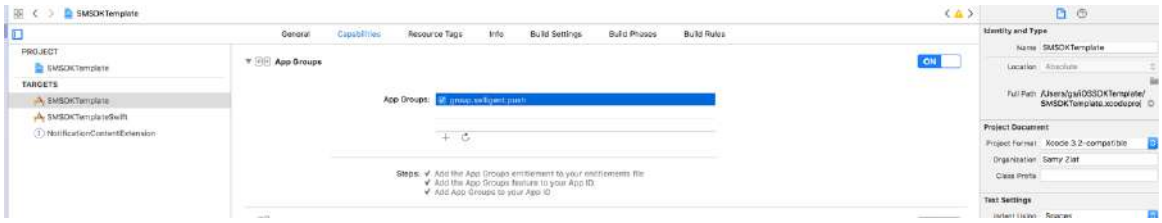
When this is done, you will need to enable App group capabilities and check **group.<your group name>** in the Capabilities tab of your main app target.

You can do this either in your apple developer account





or directly in your project in Xcode:



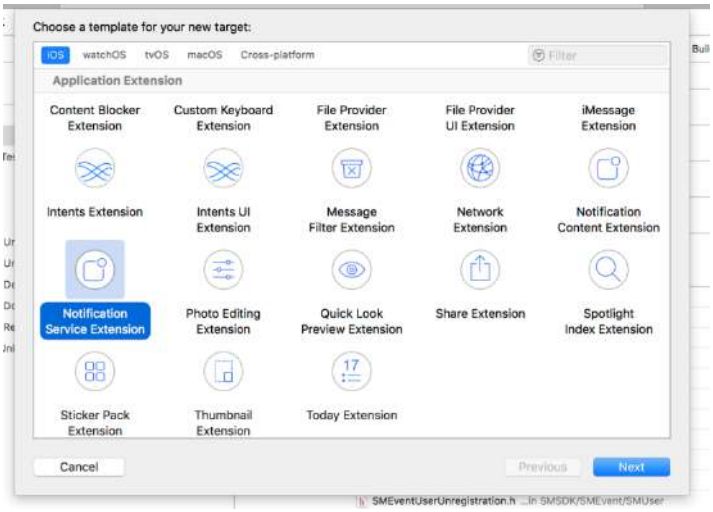
6.9.2 UNNotificationServiceExtension – Notification Service Extension

This extension will allow:

- To decrypt a payload before it is provided in the push banner.
- To download the media of the rich push payload

6.9.2.1 Configuration

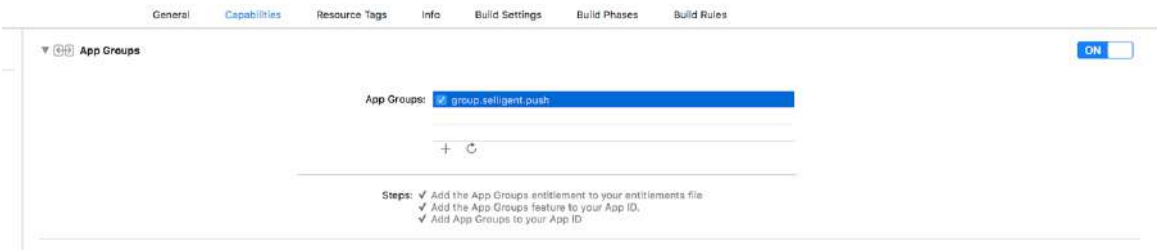
To get started you will need to add a new target **Notification Service Extension** to your project:



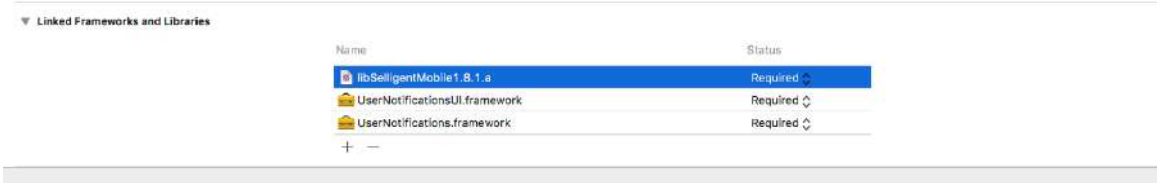
Note the creation of the files (in Swift in this example):

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

In the capabilities of your notification service extension target, enable App groups and check **group.<your group name>**



And finally link Selligent Sdk to the Content Extension target:



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

6.9.2.2 Start sdk from inside extension

1. In an Objective C project, import **SMHelper.h** (or **<SelligentMobileSDK/SelligentMobileSDK.h>** in case you are using the framework) in NotificationService.m

   In a Swift project, you just need to import **SMHelper.h** (or **<SelligentMobileSDK/SelligentMobileSDK.h>** in case you are using the framework) in your bridging header file.
2. To start the sdk, please follow the steps below.

   All the following must be done inside:

```
Swift:

didReceive(_:withContentHandler:)
```

```
Objective-C:

didReceiveNotificationRequest:withContentHandler:
```

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

   b. Set the property **appGroupId** with **group.<your group name>** this property is mandatory and should also be provided in AppDelegate file (refer to Starting sdk chapter)

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

```
Swift:

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

//Create the SMManagerSetting instance
let setting: SMManagerSetting= SMManagerSetting(url: url, clientID: clientID, privateKey: privateKey)
 //Provide the app group id to the sdk
setting.appGroupId = "group.<your group name>"

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

```
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];
//Provide app group id to the sdk
Setting.appGroupId = @"group.<your group name>";

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

6.9.2.3 Push notification content modification before displayed to user

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

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

You have the possibility to choose between two methods: either you want to manage the call to the block, which is executed with the modified notification content, by yourself or you want to let the library manage this for you.

Under you will find a complete example of implementation for both cases and in both objective c and swift

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

```
1.  Swift:
2.
3.  // Storage for the completion handler and content.
4.  var contentHandler: ((UNNotificationContent) -> Void)?
5.  var bestAttemptContent: UNMutableNotificationContent?
6.
7.  // Modify the payload contents.
8.  override func didReceive(_ request: UNNotificationRequest, withContentHandler contentHandler:
    @escaping (UNNotificationContent) -> Void) {
9.    self.contentHandler = contentHandler
10.             self.bestAttemptContent = (request.content.mutableCopy() as?
    UNMutableNotificationContent)
11.
12. // Init and start the sdk.
13. let url = "URL"
14. let clientID = "ClientID"
15. let privateKey = "privateKey"
16. //Create the SMManagerSetting instance
17. let setting: SMManagerSetting= SMManagerSetting(url: url, clientID: clientID, privateKey:
    privateKey)
18. //Provide the app group id to the sdk
19. setting.appGroupId = "group.<your group name>"
20.
21. //Start the sdk
22. SMManager.sharedInstance().startExtension(with: setting)
23.
24. //Provide the request with the original notification content to the sdk and return the updated
    notification content
25. bestAttemptContent = SMManager.sharedInstance().didReceive(request)
26.
27.   //call the completion handler when done.
28.             contentHandler(bestAttemptContent)
29. }
30. // Return something before time expires.
31. override func serviceExtensionTimeWillExpire() {
32.    if let contentHandler = contentHandler, let bestAttemptContent = bestAttemptContent {
33.             // Mark the message as still encrypted.
34.             bestAttemptContent.subtitle = "(Encrypted)"
35.             bestAttemptContent.body = ""
36.             contentHandler(bestAttemptContent)
37.    }
38. }
```

Objective-C:

```
#import "SMHelper.h"

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

@end

@implementation NotificationService

- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentHandler:(void (^)(UNNotificationContent * _Nonnull))contentHandler {
self.contentHandler = contentHandler;

NSString *url       = @"YourProvidedURL";
NSString *clientID  = @"YourClientID";
NSString *privatKey = @"YourPrivateKey";
//Create the SMManagerSetting instance
SMManagerSetting *setting = [SMManagerSetting settingWithUrl:url ClientID:clientID PrivateKey:privatKey];
//Provide app group id to the sdk
Setting.appGroupId = @"group.<your group name>";

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

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

// call the completion handler when done
contentHandler(_bestAttemptContent);
}

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

- In the second case sdk will manage all for you:

```
Swift:

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

//Create the SMManagerSetting instance
let setting: SMManagerSetting= SMManagerSetting(url: url, clientID: clientID, privateKey: privateKey) as! SMManagerSetting
//Provide the app group id to the sdk
setting.appGroupId = "group.<your group name>"

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

                    //Provide the request with the original notification content to the sdk and the contentHandler
                    SMManager.sharedInstance().didReceive(request, withContentHandler: contentHandler)
}

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

```
Objective-C:

#import "SMHelper.h"
@implementation NotificationService

- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentHandler:(void (^)(UNNotificationContent * _Nonnull))contentHandler {
                    NSString *url       = @"YourProvidedURL";
NSString *clientID  = @"YourClientID";
NSString *privatKey = @"YourPrivateKey";
//Create the SMManagerSetting instance
SMManagerSetting *setting = [SMManagerSetting settingWithUrl:url ClientID:clientID PrivateKey:privatKey];
 //Provide app group id to the sdk
Setting.appGroupId = @"group.<your group name>";

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

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


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

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

If you don't choose to use encryption feature do not call sdk serviceExtensionTimeWillExpire and let the original push payload to be used

For more information on Notification service extension you can also refer to apple documentation

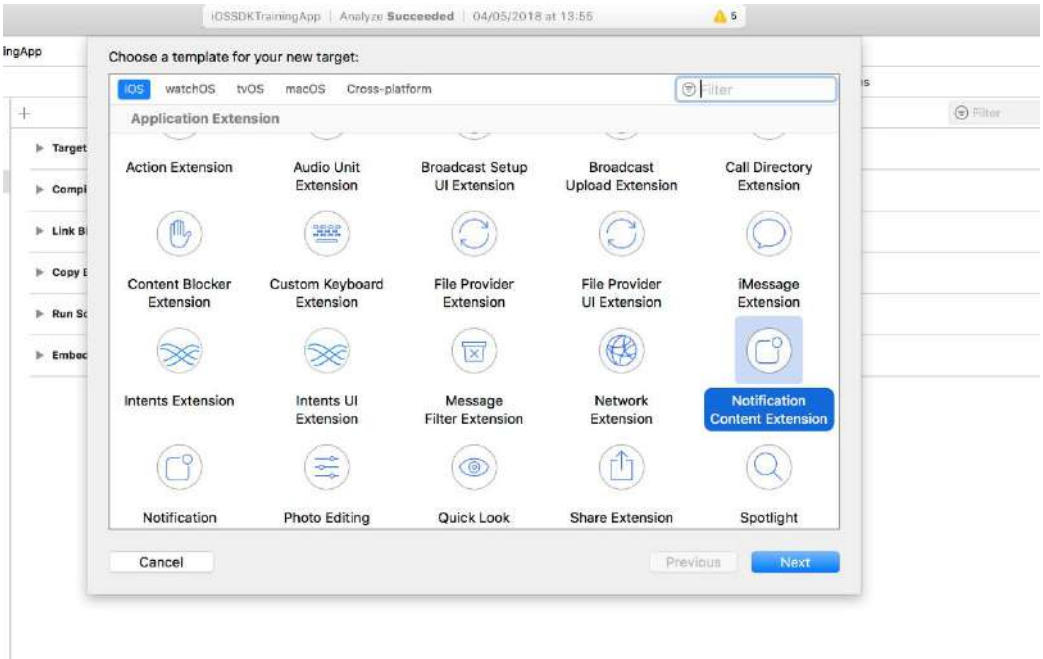
6.9.3 UNNotificationContentExtension - Notification Content Extension

This extension will allow:

- To provide to you the content of the attachment of a rich push media (if the extension is not present and depending on the type of the media, iOS will use a default template to display the message but in this case the action buttons will not be displayed if they are present)
- To display action buttons of a push

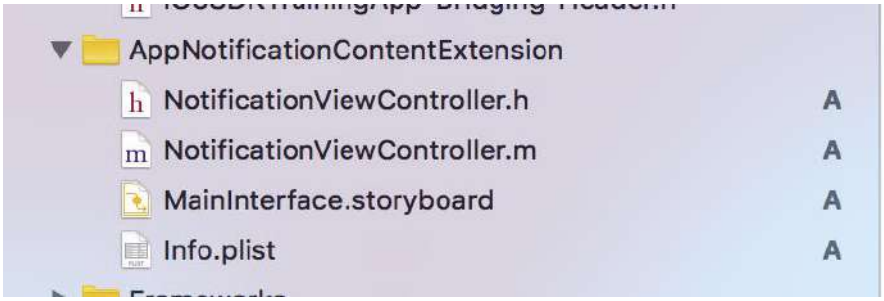
6.9.3.1 Configure notification content extension to your project for Selligent category

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

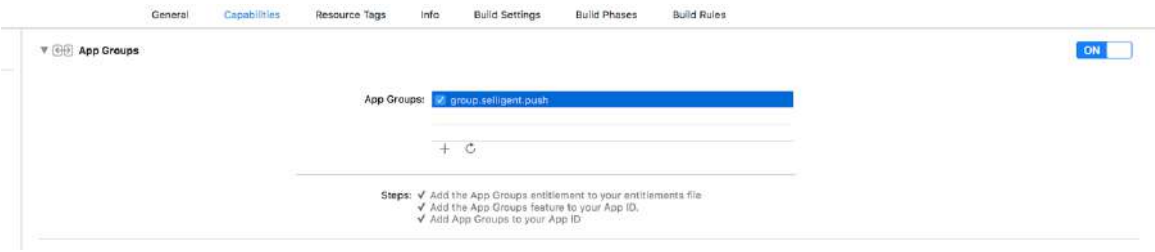


You will notice the creation of those files (in Objective C in this example):

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



In the capabilities of your target enable App groups and check **group.<your group name>**



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

A category is a property inside the payload that will inform your app which extension should manage the content of the payload.

It is mandatory to have one once a button is present in the payload and should be displayed in the notification center (cf. 6.9.3.3 Push action buttons)

By default, the Selligent category will be **SELLIGENT_BUTTON** If you plan to send also rich push, Selligent provide you the ability to have a specific extension to manage the way you want to display it in the UI. The category will be named in this case **SELLIGENT_IMAGE.**

To set the category on your content extension just open the Info.plist of the extension, find the **NSExtensionAttributes** dictionary and set the value of the **UNNotificationExtensionCategory** key to **SELLIGENT_BUTTON** or **SELLIGENT_IMAGE**.

You can also have both category in one extension as **UNNotificationExtensionCategory** can be an array type.

And finally link Selligent Sdk to the Content Extension target:

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

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

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

For more info on Notification Content Extension please relate to apple documentation

6.9.3.2 Start sdk from inside extension

You can refer and follow all steps describe in notification service extension start

The only difference concerns the point 2

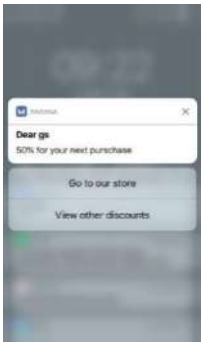
All the steps must be done inside

```
1.  Swift:
2.
3.  didReceive()
```

```
1.  Objective-C:
2.
3.  didReceiveNotification:
```

6.9.3.3 Push action buttons

If you have correctly added a Selligent Notification Content Extension target to your project you will be able to display action buttons directly to your push:



Now in your NotificationViewController file (.m or .swift) just call

```
- (void)didReceiveNotification:(UNNotification*)notification
```

```
Swift:

class NotificationViewController: UIViewController, UNNotificationContentExtension {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any required interface initialization here.
    }

    func didReceive(_ notification: UNNotification) {
        self.label?.text = notification.request.content.body

        let url = "URL"
        let clientID = "ClientID"
        let privateKey = "privateKey"
        //  Create the SMManagerSetting instance
        let setting: SMManagerSetting = SMManagerSetting(url: url, clientID: clientID, privateKey: privateKey)
        //Only mandatory when you want to use a Notification extension (Service or/and Content)
        setting.appGroupId = "group.<your group name>"
        //Start sdk from extension
        SMManager.sharedInstance().startExtension(with: setting)
        //sdk api to add buttons from banner
        SMManager.sharedInstance().didReceive(notification)
    }
}
```

```
Objective-C:

#import SMHelper.h

@implementation NotificationViewController

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

- (void)didReceiveNotification:(UNNotification *)notification {
                    self.label.text = notification.request.content.body;
NSString *url       = @"YourProvidedURL";
NSString *clientID  = @"YourClientID";
NSString *privatKey = @"YourPrivateKey";
//Create the SMManagerSetting instance
SMManagerSetting *setting = [SMManagerSetting settingWithUrl:url ClientID:clientID PrivateKey:privatKey];
//Provide app group id to the sdk
Setting.appGroupId = @"group.<your group name>";

//Starting the library
[[SMManager sharedInstance] startExtensionWithSetting:setting];
                     [[SMManager sharedInstance] didReceiveNotification:notification];
}
```

If you want those buttons to be processed without the need of opening the app you will also need to call in NotificationViewController file:

```
- (void)didReceiveNotificationResponse:(UNNotificationResponse*)response
```

```
Swift:

func didReceive(_ response: UNNotificationResponse, completionHandler completion: @escaping
(UNNotificationContentExtensionResponseOption) -> Void) {
        SMManager.sharedInstance().didReceive(response)
        completion(.dismiss)
    }
```

```
Objective-C:

- (void)didReceiveNotificationResponse:(UNNotificationResponse *)response
                    completionHandler:(void (^)(UNNotificationContentExtensionResponseOption
option))completion {
   [[SMManager sharedInstance] didReceiveNotificationResponse:response];
            completion(UNNotificationContentExtensionResponseOptionDismiss);
}
```

Do not forget to call the completionHandler with one of the response options UNNotificationContentExtensionResponseOption

6.9.3.4 Rich push image type

A rich push of type image will be displayed by default like this if you haven't implemented a notification content extension with the SELLIGENT_IMAGE category.

<IMAGE>

Keep in mind that if you plan to add push action buttons to your rich push (or also if you want to manage the way the UI display the content) the content extension with the category **SELLIGENT_IMAGE** is mandatory

In this case you will have to manage the display of the provided image

The service extension will then be in charge to download the media and you will be able to retrieve it and process it in the content extension.

Under you will have an example of how you can do that in:

```
Swift:

didReceive()
```

```
Objective-C:

didReceiveNotification:
```

```
Swift:

class NotificationViewController: UIViewController, UNNotificationContentExtension {

    @IBOutlet var label: UILabel?
    @IBOutlet weak var imageView: UIImageView!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any required interface initialization here.
    }

    func didReceive(_ notification: UNNotification) {
        self.label?.text = notification.request.content.body
        if(notification.request.content.attachments.count > 0)
        {
            let attachment:UNNotificationAttachment =
notification.request.content.attachments[0];
            if (attachment.url.startAccessingSecurityScopedResource())
            {
                let imageData = NSData(contentsOf: attachment.url)
                let image = UIImage(data: imageData! as Data)
                self.imageView.image = image;
                attachment.url.stopAccessingSecurityScopedResource()
               }
        }
        let url = "URL"
        let clientID = "ClientID"
        let privateKey = "privateKey"
        //  Create the SMManagerSetting instance
        let setting: SMManagerSetting = SMManagerSetting(url: url, clientID: clientID,
privateKey: privateKey)
        //Only mandatory when you want to use a Notification extension (Service or/and Content)
        setting.appGroupId = "group.<your group name>"
        //Start sdk from extension
        SMManager.sharedInstance().startExtension(with: setting)
        //sdk api to add buttons from banner
        SMManager.sharedInstance().didReceive(notification)
    }
[.......]
}
```

```
Objective-C:

@interface NotificationViewController () <UNNotificationContentExtension>
@property (weak, nonatomic) IBOutlet UILabel *label;
@property (weak, nonatomic) IBOutlet UIImageView *imageView;
@end

@implementation NotificationViewController
- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any required interface initialization here.
}
- (void)didReceiveNotification:(UNNotification *)notification {
  self.label.text = notification.request.content.body;
  if(notification.request.content.attachments.count > 0)
            {
                            UNNotificationAttachment* attachment =
notification.request.content.attachments[0];
                            if (attachment.URL.startAccessingSecurityScopedResource())
                            {
                                  NSData* imageData = [NSData
dataWithContentsOfURL:attachment.URL];
                                  UIImage* image = [UIImage imageWithData: imageData];
                                  self.imageView.image = image;
                                  [attachment.URL stopAccessingSecurityScopedResource];
                             }
            }
  NSString *url       = @"YourProvidedURL";
  NSString *clientID  = @"YourClientID";
  NSString *privatKey = @"YourPrivateKey";
  //Create the SMManagerSetting instance
  SMManagerSetting *setting = [SMManagerSetting settingWithUrl:url ClientID:clientID
PrivateKey:privatKey];
  //Provide the app group id to the sdk
  setting.appGroupId = "group.<your group name>"
  //Starting the library
  [[SMManager sharedInstance] startExtensionWithSetting:setting];
  [[SMManager sharedInstance] didReceiveNotification:notification];
}
```

6.10 Changelog

- **SDK 2.6.1**
  - Provide device id to main app
  - Provide delegate to allow app to process links contained in an inapp message of type url
  - Add Push InApp Message payload to InApp Message cache
  - Provide a delegate to allow app to manage display on app side of InApp Message linked to a remote notification

- **SDK 2.6**
  - Provide the sdk as a .xcframework
  - Remove geolocalisation api references from core sdk

- **SDK 2.5.2**
  - Correct bug 136398 conflict with SDWebImagePDFCoder that avoid displays of images

- **SDK 2.5.1**
  - Provide helper method to set in app message as deleted
  - Provide helper method to set in app message as unseen
  - Correct bug 124219 send pushopened event when push is viewed only with notification content extension
  - Correct bug 128940 getinappmessage not providing inapp even if they are already broadcasted
  - Correct bug 128941 nil in body property when json in content of inappmessage
  - Correct bug 124144 warning "nil host used in call to .." is displayed when generating inapp icon buttons

- **SDK 2.5**
  - Provide flexibility to In App Message by giving access to public methods and properties that will enable developer to process content

- **SDK 2.4.1**
  - Support of iOS 14
  - Correct bug 109032 property contentCreation (creation date) of In AppContent public
  - documentation update to avoid call to serviceExtensionTimeWillExpire resulting in (Encrypted) displayed even when encryption not applied
  - Correct bug 113738 Events are sent twice

- **SDK 2.4**
  - Support of rich push of type image
  - Correct bug 104272 provide an override method to send SMEventUserLogin and avoiding login with providing empty mail
  - Remove code that forces having to ask for camera permissions in plist file

- **SDK 2.3.1**
  - Correct bug 102254 set way to provide app group id to the sdk

- **SDK 2.3**
  - Support call to didReceiveNotificationResponse: to allow trigger action of notification content extension without opening app
  - Correct bug 102401 inappmessage flag was not set correctly

- **SDK 2.2.1**
  - Correct bug 100688 manage useragent without usage of WKWebView

- **SDK 2.2**
  - Correct bug 96667 UIWebView deprecation - use WKWebView instead
  - Correct bug 96827 Buttons inside inapp messages (url/html/image) do not work in SwiftUI
  - Correct bug 90501 add robustness to check if device was already registered
  - Add robustness when parsing plotID

- **SDK 2.1**

- Support of plot projects v3.2.0

- **SDK 2.0.3**
  - Format devicetoken without using of NSData description property for better support of Xcode11 and iOS 13

- **SDK 2.0.2.1**
  - Correct bug 86049 Optout is forced to true when app is reinstalled with v2.0.2

- **SDK 2.0.2**
  - Correct bug 77531 device id is nil when performing background fetch and app is killed
  - Correct bug 74907 IAC URL type stuck forever with incorrect URL
  - Correct fetch in app with correct last date fetch when app was killed
  - Correct bug 75572 decryption issues in a swift project
  - Correct mainaction value of open browser type is not decrypted in a simple push
  - Correct bug 73088 inapp message controller not dismissed when clicking on deeplink button
  - Correct bug 77309 conflict with anvato sdk due to common crypto library
  - Correct bug 74898 inapp message of type map is not displayed
  - Correct bug 70428 shouldDisplayRemoteNotification impeach send pushopened event
  - Correct bug 69351 unregister notification functionality in not working correctly
  - Adapt method signature to avoid swift warnings
  - Update documentation

- **SDK 2.0.1**
  - Correct bug 64260 iOS fetching in app message with date - URL is not valid and header validation is not successful
  - Correct bug 64246 encoding of URL is causing issue when there is a #
  - Improve way optout is retrieved from iOS

- **SDK 2.0**
  - Support decryption of remote notification
  - Changed the way the sdk is initialized from inside a notification extension

- **SDK 1.9**
  - Support action buttons in push notification center

- **SDK 1.8**
  - Support push only without in app message

- **SDK 1.7.1**
  - Correction on duplicate symbol due to integrated library

- **SDK 1.7**
  - Added geolocation functionality
  - Misc. Bugs correction
  - Add http header to inform which version of the platform is supported
  - Support of bitcode

- **SDK 1.6**
  - iOS 11 support
  - Misc. bug corrections
  - consolidate received event
  - adapt user-agent of request

- **SDK 1.5.2**
  - correct bug for in app content that must be displayed only once

- **SDK 1.5.1**

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

- **SDK 1.5**
  - sendDeviceInfo deprecated method replaced with **sendDeviceInfo:(SMDeviceInfos*)deviceInfos** method
  - New SMManager category for DataTransaction with backend
  - public SMDeviceInfos object
  - iOS 10 support of UserNotifications framework
  - stop supporting of iOS 7
  - cache on last sent UserCustomEvent
  - Update deviceID with the one received from platform

- **SDK 1.4.5**
  - Store last sent user custom event and check if a modification has been done before sending next one

- **SDK 1.4.4**
  - compare device token based on string instead of NSData (bug swift 3 and Data class)

- **SDK 1.4.3**
  - correction for max number of InApp Content crash when max > number of messages received
  - creation date of in app content
  - dismiss when no button in notification payload

- **SDK 1.4.2**
  - correction on unregisterForRemoteNotification method
  - issue with the notification when the application was killed

- **SDK 1.4.1**
  - bug corrections

- **SDK 1.4**
  - enum SMInAppMessageRefreshType has been renamed in SMInAppRefreshType (this enum is used both for InApp Message and for InApp Content) possible values are :
    - kSMIA_RefreshType_None
    - kSMIA_RefreshType_Hourly
    - kSMIA_RefreshType_Daily

- **SDK 1.3**
  - To easily access 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).