mfms

Библиотека PushLib Lite v2.4.5

Интеграция с использованием Notification Service Extension

Оглавление

Термины и сокращения

Термин	Полная форма	Описание
appPackage		Уникальный код приложения заказчика в магазинах приложений Google Play и App Store. Используется при регистрации приложения на Push-сервере.
deviceAddress		Уникальный адрес мобильного приложения, установленного на конкретном устройстве. Формируется по заданному алгоритму при каждом запуске приложения на стороне Push-сервера. Может меняться при изменении Push-адреса приложения в PNS. Это зависит от настроек, заданных для приложения на Push-сервере
deviceToken		32-байтовый уникальный номер, который однозначно определяет устройство
deviceUid		Уникальный идентификатор мобильного приложения, установленного на конкретном устройстве. Формируется по заданному алгоритму при каждом запуске приложения на стороне приложения
Remote Push-уведомление		Удаленное Push-уведомление
PROVIDER_UID		Уникальный идентификатор, используемый при взаимодействии приложения и платформы. Генерируется автоматически.
PNS	Push Notification Services	Провайдеры Push-уведомлений. APNS (Apple Push Notification Service) и GCM (Google Cloud Message), обеспечивают доставку Push-уведомлений в приложение на устройстве
Push-адрес		Уникальный адрес приложения в PNS, может изменять значение по инициативе PNS
Push-сервер		Программно-аппаратный комплекс, который реализует сервис доставки Push-сообщений. В частности, механизм регистрации приложений на Push-сервере, отправку Push-уведомлений и доставку содержимого Push-сообщений в приложение на устройстве, резервирование доставки с помощью sms-сообщений, хранение статусов доставки Push-сообщений
Push-сообщение		Сообщение от заказчика, включающее Push-уведомление и содержимое сообщения в текстовом формате. В качестве содержимого заказчик может передавать: тексты, а также бинарные данные в формате Base64

Термин	Полная форма	Описание
Push-уведомление		Короткое уведомление, которое Push-сервер отправляет в приложение посредством сервисов PNS. Push-уведомления, доставленное в приложение, инициирует процедуру получения содержимого Push-сообщения с Push-сервера
Библиотека	Push-библиотека	Компонент мобильного приложения, участвует в интеграции с приложением. Обеспечивает регистрацию приложения на Push-сервере, привязку дополнительных идентификаторов, доставку Push-уведомлений и содержимого Push-сообщений в приложение
Приложение	Мобильное приложение	Программное обеспечение заказчика, установленное на мобильном устройстве клиента, в которое интегрирована Push-библиотека
Расширение	Notification Service Extension	Часть мобильного приложения предназначенное для модификации PUSH-уведомлений и подтверждения доставки.

Введение

Библиотека PushLib Lite предназначена для:

- обработки Push-уведомлений;
- получения контента с сервера MFMS;
- передачи статуса доставки сообщения.

Для повышения качества доставки Push-уведомлений Библиотека использует расширение Notification Service Extension. Расширение – фактически отдельное приложение iOS, которое запускается при получении Push-уведомления без запуска основного приложения. Расширение используется для модификации полученного Push-уведомления, например, для загрузки картинки и последующего показа Push-уведомления с изображением.

Библиотека разработана на языке Swift, расширение Notification Service Extension – на языке Objective C.

Интеграция Библиотеки и Расширения

Создание App Group

Для корректной работы сервиса доставки Push-уведомлений требуется использовать в Библиотеке и Расширении актуальное значение `deviceAddress`, совпадающее со значением этого параметра на Push-сервере.

Прежде чем приступить к интеграции Библиотеки и Расширения, нужно выбрать способ, которым будет передаваться это значение с Push-сервера в Библиотеку.

Есть несколько вариантов передачи значения `deviceAddress` с Push-сервера в Библиотеку:

- через `App Groups` с использованием `User Defaults (Shared);`
- через `Keychain`;
- другой вариант передачи значения `deviceAddress` (через любое удобное хранилище, к которому имеет доступ Приложение, Библиотека и Расширение).

Рекомендуется использовать метод, работающий через `App Groups` с использованием `UserDefaults`. Для использования этого способа нужно создать группу в консоли управления сертификатами и приложениями (см. Рисунок 1) на сайте developer.apple.com по адресу:

`https://developer.apple.com/account/ios/identifier/applicationGroup`

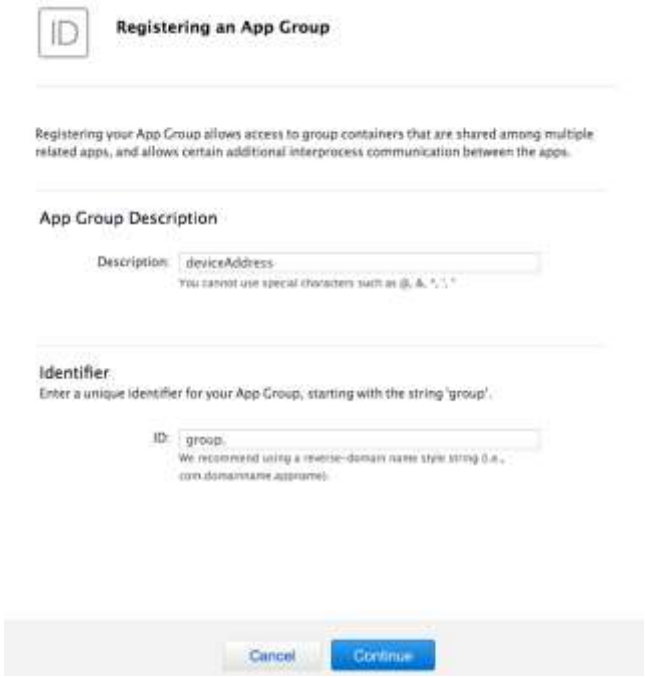

Рисунок 1 - Создание новой группы

Для удобства в качестве ID группы рекомендуется использовать Bundle ID приложения, в которое интегрируется Библиотека и Расширение.

В дальнейшем ID группы будет использоваться в настройках проекта и расширения. Также его нужно указывать в коде при обработке событий, связанных с изменением `deviceAddress`.

Подключение Библиотеки к проекту

Подключение Библиотеки и настройка проекта

Подключение библиотеки к проекту можно производить через `CocoaPods`, добавив зависимость:

```
«pod 'MFMSPushLite', :podspec => 'https://maven-pub.mfms.ru/repository/maven-public/com/mfms/ios/Push-lite/2.4.5/MFMSPushLite.podspec'»
```

Примечание. При подключении через `CocoaPods` будет использована Universal-сборка (iOS + Simulator).

Либо можно подключить библиотеку вручную. Для этого нужно распаковать файлы библиотеки в любой каталог проекта. А затем добавить библиотеку в `Embedded Binaries` на вкладке `General`.

Рисунок 2 - Подключение библиотеки к проекту

Также при ручном добавлении библиотеки на вкладке `General` требуется добавить в список `Linked Frameworks and Libraries` следующие фреймворки:

- `UserNotifications.framework`,
- `UIKit.framework`,
- `Foundation.framework`.

Рисунок 3 - Добавление фреймворка в проект

На вкладке `Capabilities` следует включить сервис Push-нотификаций.

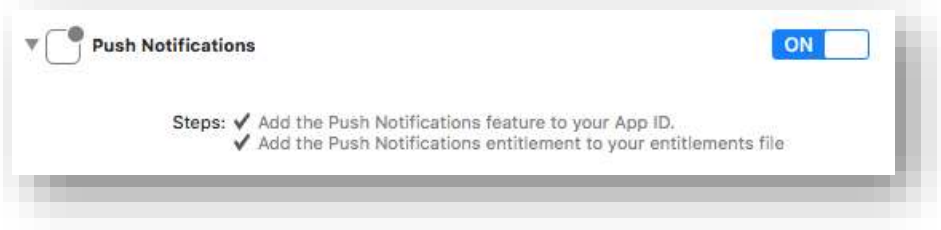

Рисунок 4 - Включение сервиса Push-нотификаций

Также на вкладке `Capabilities` следует включить параметр `App Groups` и выбрать ранее созданную группу.

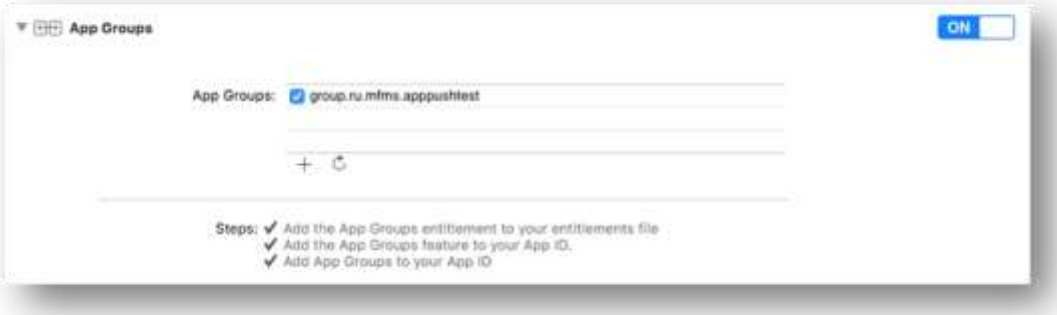

Рисунок 5 - Включение параметра App Groups

В разделе `Background Modes` включить режимы (установить флажки) `Background fetch` и `Remote notifications`.

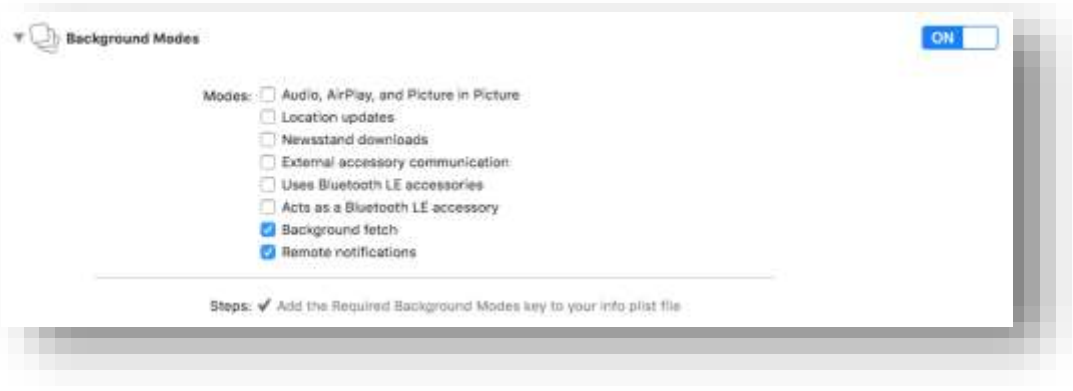

Рисунок 6 - Включение режимов Background Modes

На вкладке `Build Settings` в разделе `Build Options` установить для параметра `Always Embed Swift Standard Libraries` значение `Yes`.

Рисунок 7 – Включение параметра Always Embed Swift Standard Libraries

Конфигурация Библиотеки в коде проекта

В рамках данного документа описан способ конфигурации в упрощенном режиме с настройками «По умолчанию», для этого в Библиотеке реализован класс-фасад `MFMSPushLite` с переменной `_PushLite`. Также сохраняется возможность использования собственных настроек. Описание методов и примеры приведены в Таблице 1.

В файле **AppDelegate.m** импортируем библиотеку и регистрируем устройство для получения уведомлений**:**

AppDelegate.m

```
#import "AppDelegate.h"
#import <MFMSPushLite/MFMSPushLite-Swift.h>
@interface AppDelegate () <MFMSPushLiteDelegate>

@end

@implementation AppDelegate{
   MFMSPushLite * _PushLite;
}
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

     [self registerForRemoteNotification];
   _PushLite = [[MFMSPushLite alloc] initWithDelegate:self];
   _PushLite.appGroup = @"group.ru.mfms.appPushtest";
   _PushLite.autoRegisterForNotification = false;
   [_PushLite start];
   return YES;
}
```

После чего требуется реализовать следующие методы:

AppDelegate.m

```
#pragma mark – MFMSPushLiteDelegate
// Определяет платформу: YES - боевой сервер, NO - тестовый сервер.
- (BOOL)isProductionWithPushApi:(MFMSPushLite * _Nonnull)PushApi {
#ifdef DEBUG
```

```
    return NO;
#else
    return YES;
#endif
}
-(void)onPushMessagesReceivedWithPushApi:(MFMSPushLite *)PushApi messages:(NSArray<PushNotificationMessage *> *)messages{
//метод обработки события получения Push
}
// Логирование ошибок
-(void)onErrorWithPushApi:(MFMSPushLite *)PushApi error:(NSString *)error{
    NSLog(@"MFMSPushLite: %@", error);
}

#pragma mark -- remote notifications

- (void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken{
    [_PushLite.appDelegate didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];
}
- (void)application:(UIApplication *)application didFailToRegisterForRemoteNotificationsWithError:(NSError *)error{
    [_PushLite.appDelegate didFailToRegisterForRemoteNotificationsWithError:error];
}
- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo fetchCompletionHandler:(void
(^)(UIBackgroundFetchResult))completionHandler{
    [_PushLite.appDelegate didReceiveRemoteNotification:userInfo fetchCompletionHandler:completionHandler];
}
- (void)application:(UIApplication *)application didRegisterUserNotificationSettings:(UIUserNotificationSettings *)notificationSettings{
    [_PushLite.appDelegate didRegisterUserNotificationSettings:notificationSettings];
}
```

Подключение Расширения к проекту

Перед началом использования Расширения необходимо подключить его к проекту и настроить.

Подключение Расширения

В Xcode-проекте необходимо добавить новый Target:

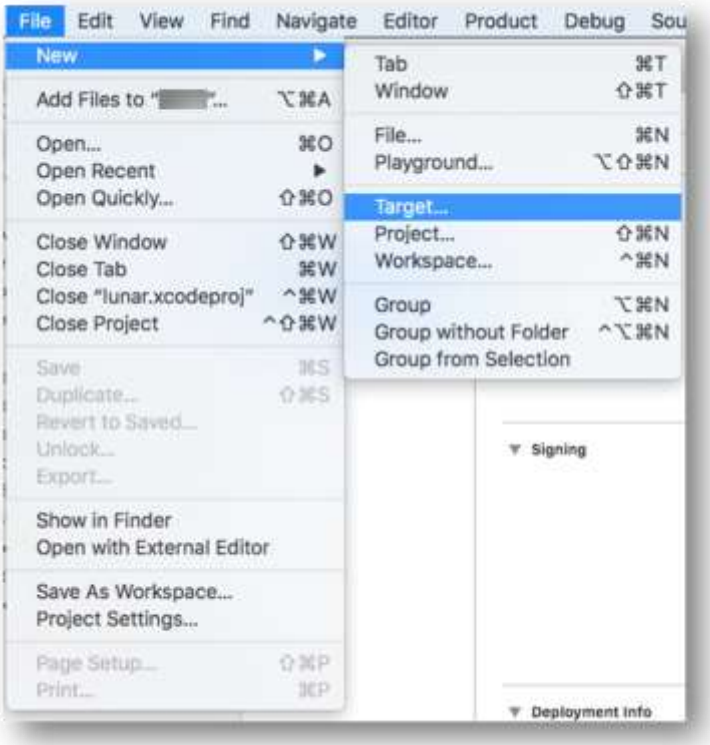

Рисунок 8 - Добавление Target к проекту

Далее в окне Choose a template for your new target следует выбрать тип расширения Notification Service Extension.

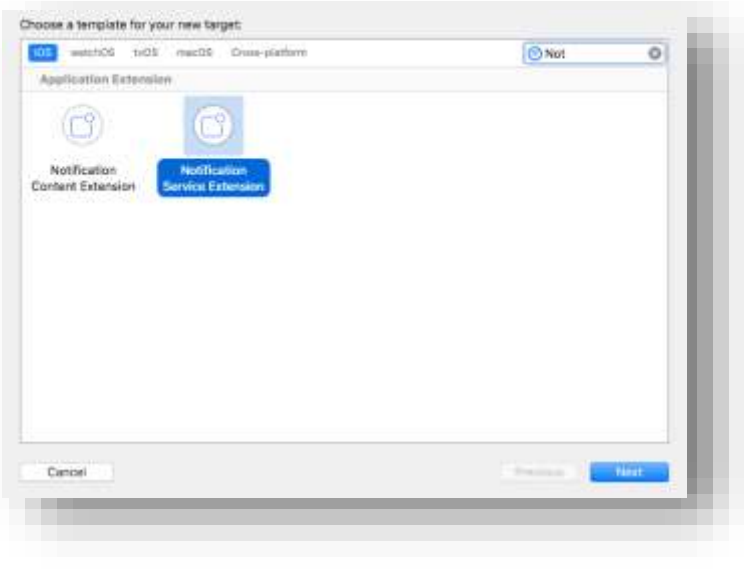

Рисунок 9 – Выбор типа расширения

В окне Choose options for your new target необходимо в качестве значения Product Name ввести .notificationService. Если указать другое название, то это не гарантирует работу расширения.

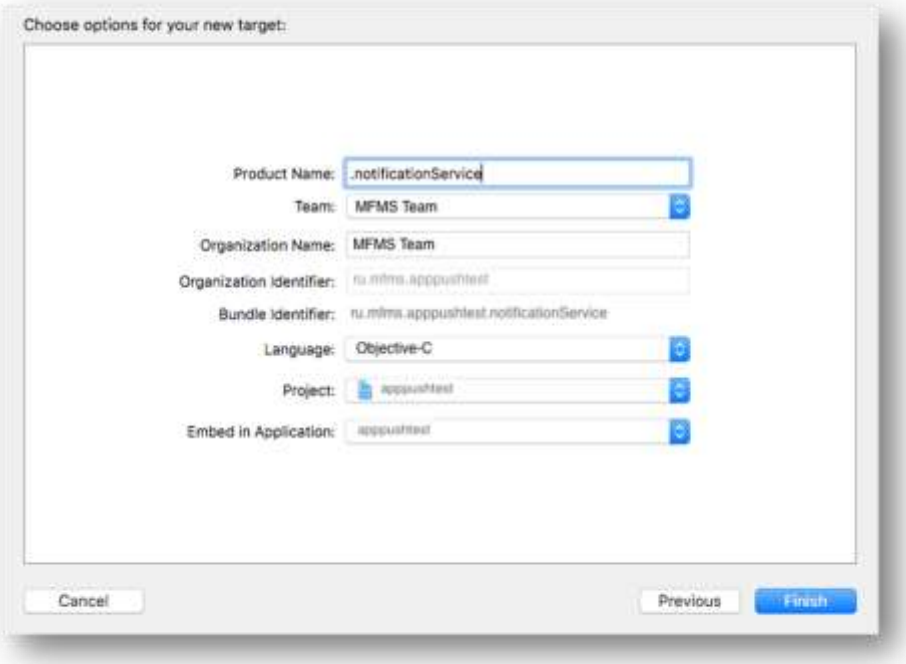

Рисунок 10 – Определение параметров для Target

Далее следует активировать схему, для этого нажать кнопку Activate в окне Activate “notificationService” scheme.

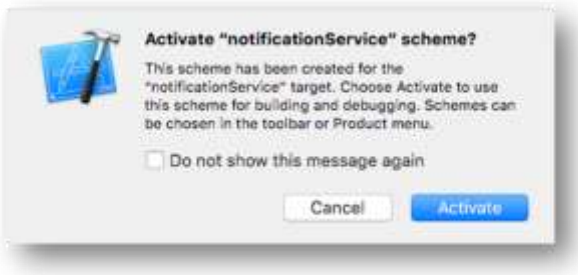

Рисунок 11 – Активация схемы

Настройка Расширения

Подключить Расширение к проекту можно через CocoaPods, добавив зависимость:

«pod 'MFMSPushNE', :podspec => 'https://maven-pub.mfms.ru/repository/maven-public/com/mfms/ios/Push-ne/2.4.5/MFMSPushNE.podspec'»

Примечание. При подключении через CocoaPods будет использована Universal-сборка (iOS + Simulator).

Либо можно подключить Расширение вручную. Для этого нужно распаковать файлы библиотеки в любой каталог проекта. На вкладке `General` в подразделе `Linked Frameworks and Libraries` необходимо подключить библиотеку `MFMSPushNE.framework`.

Рисунок 12 - Подключение библиотеки к проекту

Также необходимо изменить значение `Bundle Identifier` в подразделе `Identity`. Необходимо добавить «.ne» к текущему значению `BundleId` (см. Рисунок 13). Это нужно для того, чтобы данному расширению можно было обращаться к App Groups и получать значение `deviceAddress`.

Рисунок 13 - Изменение BundleId

На вкладке `Capabilities` следует включить сервис Push-нотификаций.

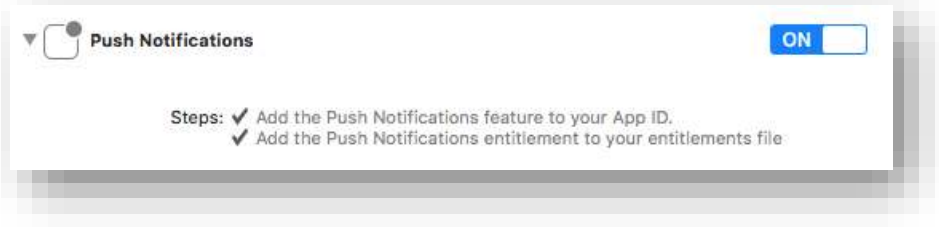

Рисунок 14 - Включение сервиса Push-нотификаций

Также на вкладке `Capabilities` следует включить параметр `App Groups` и указать ранее созданную группу.

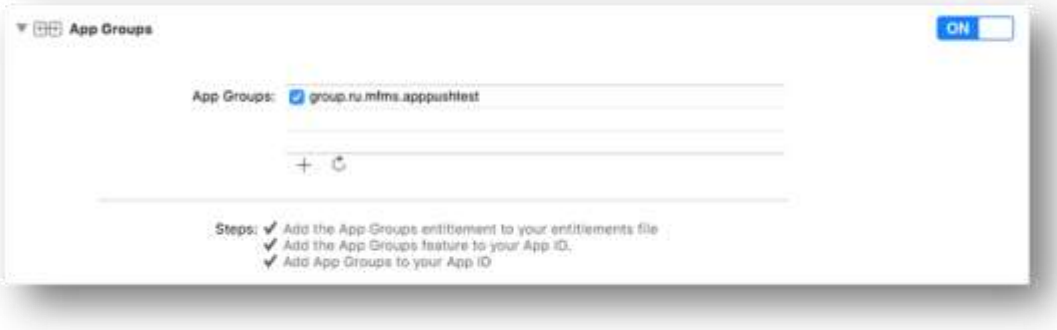

Рисунок 15 - Включение параметра App Groups

На вкладке `Build Settings` в разделе `Build Options` необходимо установить значения параметра:

`Require Only App-Extension-Safe API` – значение NO.

Рисунок 16 - Выключение параметра Require Only App-Extension-Safe API

Конфигурация в коде Расширения

Для работы библиотеки необходимо добавить в код Расширения – в файл **NotificationService.m** - следующие строки.

Добавить протокол `MFMSPushNEHelperDelegate`:

NotificationService.m

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

@interface NotificationService () <MFMSPushNEHelperDelegate>

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

@end
```

Реализовать следующие методы:

NotificationService.m

```
@implementation NotificationService

@implementation NotificationService

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

  self.contentHandler = contentHandler;
  self.bestAttemptContent = [request.content mutableCopy];

  NSDictionary * userInfo = self.bestAttemptContent.userInfo;
  MFMSPushNEHelper * PushHelper = [[MFMSPushNEHelper alloc] initWithDelegate:self];
  PushHelper.appGroup = @"group.ru.mfms.appPushtest";

  __weak typeof(self) wself = self;
  [PushHelper confirmPushDeliveryWithUserInfo:userInfo completionHandler:^{
     [wself finishExtension];
  } errorHandler:^(NSError * _Nullable error) {
```

```
        [wself finishExtension];
    }];
    }

-(void) finishExtension{
    self.contentHandler(self.bestAttemptContent);
}
- (void)serviceExtensionTimeWillExpire {
    [self finishExtension];
}

#pragma mark -- MFMSPushNEHelperDelegate

-(BOOL)isProductionModeForHelper:(MFMSPushNEHelper *)helper{
#ifdef DEBUG
    return NO;
#else
    return YES;
#endif
}
@end
```

Описание методов Библиотеки.

Основной класс (singleton) реализации – `PushServerAPI`. Приложение в режиме runtime перехватывает вызовы методов в классе `AppDelegate` и передает возвращаемые данные в библиотеку.

Таблица 1. Описание класса PushServerAPI.

№	Метод	Описание
1.	didRegisterForRemoteNotificationsWithDeviceToken:(NSData * _Nonnull)deviceToken	Метод регистрирует устройство на Push-платформе. [[PushServerAPI default] didRegisterForRemoteNotificationsWithDeviceToken: deviceToken];
2.	didFailToRegisterForRemoteNotificationsWithError:(NSError * _Nonnull)error	Метод возвращает библиотеке ошибку при неуспешной регистрации приложения на сервере APNS. [[PushServerAPI default] didFailToRegisterForRemoteNotificationsWithError: error];
3.	didReceiveRemoteNotification:(NSDictionary * _Nullable)userInfo;	Метод получения Push-сообщения. [[PushServerAPI default] didReceiveRemoteNotification: userInfo]
4.	didReceiveRemoteNotification:(NSDictionary * _Nullable)userInfo fetchCompletionHandler:(void (^ _Nonnull)(UIBackgroundFetchResult))fetchCompletionHandler	Метод получения данных: сообщает приложению о получении remote Push-уведомления, указывающего на наличие данных, которые необходимо получить. [[PushServerAPI default] didReceiveRemoteNotification:userInfo fetchCompletionHandler:^(UIBackgroundFetchResult result) {completionHandler(result);}];
5.	didRegisterUserNotificationSettings(_ settings: UIUserNotificationSettings)	Метод изменения настроек подписки, устаревший, для версий ОС до 10. [[PushServerAPI default] didRegisterUserNotificationSettings: notificationSettings];
	didUpdateNotificationSettings	Метод изменения настроек подписки, для версий ОС 10+ [[PushServerAPI default] didUpdateNotificationSettings];
6.	handleActionWithIdentifier:(NSString * _Nullable)identifier userInfo:(NSDictionary * _Nonnull)userInfo responseInfo:(NSDictionary * _Nonnull)responseInfo completion:(void (^ _Nonnull)(void))completion	Метод вызывается, когда приложение было активировано клиентом по выбору действия из remote Push-уведомления. [[PushServerAPI default] handleActionWithIdentifier: identifier userInfo: userInfo responseInfo: responseInfo completion: completionHandler];
7.	Default	Переменная инициализации Библиотеки.

8.	showNetworkActivity:(BOOL)show	Метод отображает индикатор загрузки данных из интернета в6 статус баре ОС. Значение по умолчанию – NO. [PushServerAPI showNetworkActivity: YES];
9.	onDeviceAddressChangedWithPushServerAPI:(PushServerAPI * _Nonnull)PushServerAPI deviceAddress:(NSString * _Nonnull)deviceAddress	Метод-обработчик изменения deviceAddress приложения. NSUserDefaults *defaults = [[NSUserDefaults alloc] initWithSuiteName:@"group.ru.mfms.appPushtest"]; [defaults setValue:deviceAddress forKey:@"deviceAddress"];

10.	onPushMessagesReceivedWithPushServerAPI:(PushServerAPI * _Nonnull)PushServerAPI messages:(NSArray<PushNotificationMessage *> * _Nonnull)messages	Метод-обработчик получения новых Push-сообщений.
		
		(void)onPushMessagesReceivedWithPushServerAPI:(PushServerAPI *)PushServerAPI messages:(NSArray<PushNotificationMessage *> *)messages{
		}

11.	void (^ _Nullable onPushMessagesWereRead)(NSArray<NSString *> * _Nonnull)	Метод-обработчик информации о Push-сообщениях, прочитанных в другом приложении. [[PushServerAPI default] OnPushMessagesWereRead:^(NSArray<NSString *> * _Nonnull messages) }];
12.	@property (nonatomic, readonly, copy) NSString * _Nullable deviceAddress;	Возврат актуального значения deviceAddress приложения.
13.	@property (nonatomic, readonly, copy) NSString * _Nullable deviceUID;	Возврат актуального значения deviceUID приложения.
	needAutomaticallyGenerateDeviceUid	Автоматическая генерация DeviceUid, если значение YES. Если выставлено значение NO, тогда deviceUid необходимо указать до вызова setEnabledPushNotifications. config.needAutomaticallyGenerateDeviceUid = YES;
14.	isSubscriptionEnabled	Метод проверки включения Push-уведомлений. PushServerAPI считает Push-уведомления доступными для отправки в приложение при условии: setSubscriptionEnabled (true …); - установлено разрешение на отправку Push-уведомлений у приложения.
15.	setSubscriptionEnabled:(BOOL)enabled	Метод для включения и выключения Push-уведомлений на Push-сервере: - если передан параметр «YES», начинается регистрация приложения на Push-сервере (после завершения вызывается complete-замыкание);

		- если передан параметр «NO», приложение отказывается от получения Push-уведомлений от Push-сервера (после завершения вызывается complete-замыкание).
16.	forceSync:(void (^ _Nullable)(PushServerAPIResponse * _Nullable))complete	Метод принудительной синхронизации. Запрос выполняется при наличии прочитанных сообщений в одном приложении и неполученных вторым приложением. Неполученные сообщения отработают соответствующие callback: - didPushMessagesReceived; - didPushMessagesWereRead.

17.	markMessagesAsRead:(NSArray<NSString *> * _Nonnull)messagesIds complete:(void (^ _Nullable)(PushServerAPIResponse * _Nullable))complete	Метод помечает полученные сообщения как прочитанные.
		
		[[PushServerAPI default] markMessagesAsRead:@[message.serverMessageId] complete:^(PushServerAPIResponse * _Nullable response) {
		}
		];

18.	resetNewMessageCounter:(void (^ _Nullable)(PushServerAPIResponse * _Nullable))complete	Метод для сброса счетчика непрочитанных сообщений на Push-платформе. [[PushServerAPI default] resetNewMessageCounter:^(PushServerAPIResponse * _Nullable error) { }];

19.	@property (nonatomic, readonly, copy) NSDictionary<NSString *, NSString *> * _Nonnull urls;	Метод реализует возможность указать адрес сервера, отличного от дефолтного, на который будет обращаться устройство за контентом.
		
		config.urls = @{
		@"1" : @"url = https://адрес веб-сервера",
		@"2" : @"url = https://адрес веб-сервера",
		};

20.	@property (nonatomic, readonly, copy) NSString * _Nullable primaryUrlCode;	Метод указывает, какой из добавленных вручную серверов является основным. config.primaryUrlCode = @"1";
21.	(BOOL)isProductionWithPushApi:(MFMSPushLite * _Nonnull)PushApi { }	Метод определяет окружение: YES - боевой сервер, NO - тестовый сервер. По умолчанию используется боевой сервер. - (BOOL)isProductionWithPushApi:(MFMSPushLite * _Nonnull)PushApi { return NO; return YES;

		}
22.	autoRegisterForNotification	Автоматическая регистрация для получения Push-уведомлений. _PushLite.autoRegisterForNotification = true;

Описание методов Расширения.

Основной класс реализации – MFMSPushNEHelper. Приложение перехватывает вызовы методов в классе NotificationService и передает возвращаемые данные в расширение.

Таблица 2. Описание методов Расширения.

№	Метод	Описание

1	(MFMSPushConfig*)
	configForHelper:(MFMSPushNEHelper*)helper;
		Метод возвращает конфигурацию, прописанную вручную и отличную от дефолтной.
		(MFMSPushConfig *)configForHelper:(MFMSPushNEHelper *)helper{
		MFMSPushConfig*config = [[MFMSPushConfig alloc] init];
		…
		return config;
		}

2	addServerWithCode:(NSString* _Nonnull)code url:(NSString* _Nonnull)url;	Метод реализует возможность указать адрес сервера, отличного от дефолтного, на который будет обращаться устройство за контентом. Реализуется в configForHelper. [config addServerWithCode:@"номер" url:@"https://адрес веб-сервера"];
3	addServerWithCode:(NSString* _Nonnull)code url:(NSString* _Nonnull)url isPrimary:(BOOL)isPrimary;	Метод указывает, какой из добавленных вручную серверов является основным. [config addServerWithCode:@"номер" url:@"http://адрес веб-сервера" isPrimary:YES];

4	(NSString*)
	deviceAddressForHelper:(MFMSPushNEHelper*)helper;
		Передает Расширению адрес устройства, полученный ранее от пуш-платформы в основном приложении.
		- (NSString * _Nullable)deviceAddressForHelper:(MFMSPushNEHelper *)helper {
		return [self readDeviceAddress];
		}

5	(BOOL) isProductionModeForHelper:(MFMSPushNEHelper*)helper;	Метод определяет окружение: YES - боевой сервер, NO - тестовый сервер. По умолчанию используется боевой сервер. -(BOOL)isProductionModeForHelper:(MFMSPushNEHelper *)helper{ #ifdef DEBUG return NO; #else

		return YES; #endif }

6	(void) logWithInfo:(NSString*)info forHelper:(MFMSPushNEHelper*)helper;	Метод включает подробное логирование.
		
		-(void) logWithInfo:(NSString*)info forHelper:(MFMSPushNEHelper*)helper
		{
		NSLog(@"%@", info);
		}

Структура сообщения PushNotificationMessage

Структура сообщения PushNotificationMessage представлена в Таблица 2.

Таблица 2. Структура сообщения PushNotificationMessage.

Имя	Тип	Значение
messageId	String	Идентификатор сообщения
sentAt	Int64	Время отправки (поступления на платформу)
secured	Bool	Сообщение было отправлено в безопасном режиме
shortMessage	String	Заголовок сообщения, отправляемый PNS
fullMessage	String	Полный текст сообщения
markMessageAsRead	Boolean	Сообщить на Push-платформу о прочтении сообщения. Значение read выставляется в true и отправляется на Push-платформу
hashValue	Int	Hash-значение объекта
description	String	Стандартное описание объекта, наследника от NSObject

Коды ошибок

В метод, обрабатывающий ошибки взаимодействия с Push-сервером, передаются коды ошибок. Описание кодов ошибок представлено в Таблица 3.

Таблица 3. Список кодов ошибок.

№ п/п	Код ошибки	Описание
1.	DEVICE_ADDRESS_INVALID	Указано некорректное значение идентификатора deviceAddress
2.	ACCESS_DENIED	Недостаточно прав для выполнения операции
3.	INTERNAL_SERVER_ERROR	Внутренняя ошибка сервера: требуется повторить запрос позже
4.	BAD_PARAMETERS	Указаны неверные параметры запроса. Данная ошибка возникает в случае неверного значения deviceAddress или его отсутствия на Push-сервере

№ п/п	Код ошибки	Описание
5.	IO_ERROR	Сетевая ошибка
9.	DEVICE_ADDRESS_TEMPORARY_BLOCKED	Превышено количество попыток регистрации приложения
12.	APP_PACKAGE_NOT_FOUND	На Push-сервере отсутствуют настройки для приложения
13.	PROVIDER_UID_NOT_FOUND	На Push-сервере отсутствует параметр доступа библиотеки «PROVIDER_UID»
14.	PROVIDER_UID_LOCKED	Параметр доступа библиотеки заблокирован на Push-сервере
15.	PUSH_ADDRESS_INVALID	Недоступна процедура отправки Push-сообщения на указанный deviceAddress. Возможные причины: - сертификат Push-сообщений в выбранном provision profile не соответствует сертификату, загруженному на Push-сервере

Примечание. При возникновении ошибок сетевого взаимодействия или некорректной передаче параметров методам errorCode > 0. Если ошибок нет, то errorCode = nil, либо errorCode = 0: errorCode:Int32 = 0.

Особенности публикации в AppStore

Библиотека поставляется в двух вариантах архитектурs: `ios_only` и `universal`.

- Сборки `ios_only` подходят для размещения в AppStore, но не подходят для тестирования на симуляторе.
- Сборки `universal` будут работать и на устройстве, и на симуляторе, но при размещении в AppStore появится ошибка про лишние архитектуры, сборку с которыми нельзя отправлять в AppStore.

Для решения этой проблемы необходимо для тестирования использовать `universal`-сборку, а для отправки в AppStore - сборку `ios_only`.