Общество с ограниченной ответственностью
«ОСК»
115114, г. Москва, ул. Ленинская слобода, д. 19
тел.: +7 (495) 609-60-88
www.mfms.ru

Сервис доставки push-сообщений

---

mfmsPUSH iOS SDK v.2

Руководство по интеграции push-библиотеки в приложение на базе ОС iOS

г. Москва 2017

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
clientId		Идентификационный номер клиента в системах заказчика. Является дополнительным идентификатором приложения на push-сервере
confirmationCode		Код подтверждения 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
CocoaPods		Средство по управлению зависимостями библиотек в Objective-C проектах
phoneNumber		Номер мобильного телефона клиента в формате E.164. Является дополнительным идентификатором приложения на push-сервере
Remote push-уведомление		Удаленное push-уведомление
PROVIDER_UID		Уникальный идентификатор провайдера
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

Термин	Полная форма	Описание
		текстовом формате. В качестве содержимого заказчик может передавать: тексты, а также бинарные данные в формате Base64
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
VoIP push-уведомление		Уведомления от VoIP-систем
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
Заказчик		Издатель мобильного приложения, является инициатором отправки push-сообщений
ИС		Информационные системы заказчика
Клиент		Владелец мобильного устройства с установленным мобильным приложением заказчика. Является конечным получателем push-сообщений
Подписка		Соглашение между владельцем мобильного устройства и информационной системой о пересылке нужной информации посредством push-уведомлений
Приложение	Мобильное приложение	Программное обеспечение заказчика, установленное на мобильном устройстве клиента, в которое интегрирована push-библиотека

1. ВВЕДЕНИЕ

Данный документ представляет собой руководство разработчика, описывающее порядок работы по интеграции push-библиотеки в приложение на базе ОС iOS. В документе приводится необходимая информация по интеграции, настройке и администрированию библиотеки.

Рабочий язык документа – русский, использование английских терминов и сокращений допускается.

2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

2.1. Назначение

Библиотека предназначена для решения следующих задач в рамках сервиса доставки push-сообщений:

- регистрация приложения на push-сервере;
- привязка дополнительных идентификаторов к зарегистрированным приложениям;
- доставка remote и VoIP push-уведомлений, а также push-сообщений в приложение;
- отправка файлов и сообщений из приложения;
- передача статусов доставки сообщений.

2.2. Область применения

Процедура интеграции библиотеки в приложение заказчика является часть процесса внедрения сервиса доставки push-сообщений.

3. ИНТЕГРАЦИЯ БИБЛИОТЕКИ В ПРИЛОЖЕНИЕ

3.1. Порядок настройки интеграции

Настройка интеграции библиотеки в приложение производится в следующей последовательности:

1. Подключить библиотеку вручную или посредством CocoaPods.
2. Инициализировать библиотеку.
3. Зарегистрировать устройство на push-платформе.
4. Подписаться на получение push-уведомлений.
5. Выполнить регистрацию приложения по deviceUid.
6. Выполнить привязку идентификаторов clientId и phoneNumber к зарегистрированному приложению.

3.2. Подключение библиотеки

Подключить библиотеку можно двумя способами:

1. Вручную, посредством переноса файла PushServerAPI_version*.framework в директорию мобильного приложения.
2. Посредством CocoaPods.

```
source 'https://github.com/MFMGroup/PushService_SDK_iOS.git'
...
target ...
pod "PushServerAPI"
```

3.3. Инициализация библиотеки

Для инициализации библиотеки необходимо:

1. Добавить в Info.plist ключи соответствующих типов (Рисунок 1):
   - вкладка «General»:
     - PS_API_CONFIG, тип Dictionary, значения:
       PS_PROVIDER_UID (String) – уникальный идентификатор приложения;
       PS_URLS (Dictionary):
       <код сервера> (String) – url-адрес сервера;
       PS_PRIMARY_URL (String) – код из списка PS_URLS;
       PS_NEED_AUTOMATICALLY_GENERATE_DEVICE_UID (Boolean) – флаг генерации уникального идентификатора приложения deviceUid:
       «YES» – генерация значения идентификатора deviceUid в приложении;
       «NO» – генерация значения идентификатора deviceUid библиотекой.
   - Вкладка «Capabilities»:
     - voip over IP для voip push-уведомлений;

- background fetch;
- remote notifications.

2. Включить push-нотификацию в настройках приложения.

Примечание! Все параметры конфигурации Info.plist являются обязательными. При несоблюдении требований к конфигурации, библиотека принудительно завершит работу приложения с ошибкой.

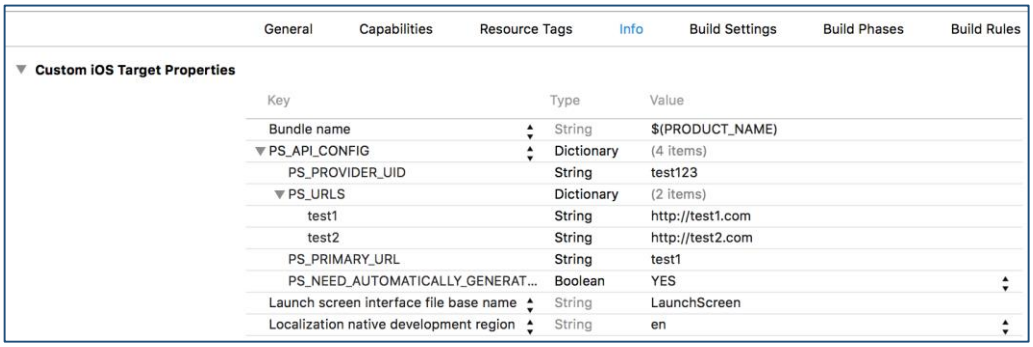

Рисунок 1. Параметры конфигурации Info.plist.

3.4. Регистрация устройства на push-платформе

Для начала необходимо зарегистрировать устройство в службе APNS. Для этого следует изменить метод didFinishLaunchingWithOptions следующим образом:

– VoIP push-уведомлений:

```
(BOOL)application:(UIApplication )application
didFinishLaunchingWithOptions:(NSDictionary )launchOptions:
   registry = [[PKPushRegistry alloc] initWithQueue: dispatch_get_main_queue()];
   registry.delegate = self;
   registry.desiredPushTypes = [NSSet setWithObject: PKPushTypeVoIP];
```

– remote push-уведомления:

```
(BOOL)application:(UIApplication )application
didFinishLaunchingWithOptions:(NSDictionary )launchOptions:
[[UIApplication sharedApplication] registerForRemoteNotifications];
```

Далее зарегистрировать устройство на push-платформе. Для этого ApplDelegate должен реализовать следующие методы у интерфейса UIApplicationDelegate:

– VoIP push-уведомления:

```
(void)pushRegistry:(PKPushRegistry )registry
didUpdatePushCredentials:(PKPushCredentials )credentials forType:(PKPushType)type
{
   [[PushServerAPI default] didRegisterForRemoteNotificationsWithDeviceToken:
credentials.token];
}
```

– remote push-уведомления:

```
(void)application:(UIApplication )application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData )deviceToken {
   [[PushServerAPI default] didRegisterForRemoteNotificationsWithDeviceToken:
deviceToken];
```

Также в обоих случаях необходимо реализовать метод изменения настроек подписки:

```
(void)application:(UIApplication )application
didRegisterUserNotificationSettings:(UIUserNotificationSettings
)notificationSettings {
   [[PushServerAPI default] didRegisterUserNotificationSettings:
notificationSettings];
}
```

3.5. Подписка приложения на получение push-уведомлений

Перед тем как подписаться на получение push-уведомлений приложение должно запрашивать разрешения клиента на отображение push-уведомлений:

```
UIUserNotificationSettings *settings = [UIUserNotificationSettings settingsForTypes:
(UIUserNotificationTypeBadge | UIUserNotificationTypeSound |
UIUserNotificationTypeAlert) categories:nil];
   [[UIApplication sharedApplication] registerUserNotificationSettings: settings];
```

Подписка на получение push-уведомлений выполняется следующим методом:

```
[[PushServerAPI default] setEnablePushNotifications: YES complete:^{
   }];
```

Отписка на получение push-уведомлений выполняется следующим методом:

```
[[PushServerAPI default] setEnablePushNotifications: NO complete:^{
   }];
```

3.6. Получение push-сообщений

Для получения push-сообщений ApplDelegate должен реализовать следующие методы у интерфейса UIApplicationDelegate:

– VoIP push-уведомления:

```
(void)pushRegistry:(PKPushRegistry )registry
didReceiveIncomingPushWithPayload:(PKPushPayload )payload forType:(PKPushType)type {
[[PushServerAPI default] didReceiveRemoteNotification: payload.dictionaryPayload
fetchCompletionHandler:^(UIBackgroundFetchResult result) {
   }];
}
```

– remote push-уведомления:

```
(void)application:(UIApplication )application
didReceiveRemoteNotification:(NSDictionary )userInfo {
   [[PushServerAPI default] didReceiveRemoteNotification: userInfo];
}
(void)application:(UIApplication )application
didReceiveRemoteNotification:(NSDictionary )userInfo fetchCompletionHandler:(void
(^)(UIBackgroundFetchResult))completionHandler {
   [[PushServerAPI default] didReceiveRemoteNotification: userInfo
fetchCompletionHandler: completionHandler];
}
```

Вызов метода onPushMessageReeived в библиотеке позволяет получить новые push-сообщения, а метод FalureBlock возвращает ошибки работы библиотеки.

3.7. Класс PushServerAPI

Основной класс (singleton) реализации – PushServerAPI. Приложение в режиме runtime перехватывает вызовы методов в классе AppDelegate и передает возвращаемые данные в библиотеку. Подробное описание методов класса «PushServerAPI» представлено в

Таблица 1.

Таблица 1. Описание класса PushServerAPI.

№ п/п	Метод	Описание
1.	didRegisterForRemoteNotificationsWithDeviceToken(_ deviceToken: Data)	Метод регистрирует устройство на push-платформе. `[[PushServerAPI default] didRegisterForRemoteNotificationsWithDeviceToken: deviceToken];`
2.	didFailToRegisterForRemoteNotificationsWithError(_ error: NSError)	Метод возвращает библиотеке ошибку при неуспешной регистрации приложения на сервере APNS. `[[PushServerAPI default] didFailToRegisterForRemoteNotificationsWithError: error];`
3.	didReceiveRemoteNotification(_ userInfo: [AnyHashable: Any]?)	Метод получения push-сообщения. `[[PushServerAPI default] didReceiveRemoteNotification: userInfo]`
4.	didReceiveRemoteNotification(_ userInfo: [AnyHashable: Any]?, fetchCompletionHandler: @escaping ((UIBackgroundFetchResult) -> ()))	Метод получения данных: сообщает приложению о получении remote push-уведомления, указывающего на наличие данных, которые необходимо получить. `[[PushServerAPI default] didReceiveRemoteNotification: userInfo fetchCompletionHandler: completionHandler];`
6.	didRegisterUserNotificationSettings(_ settings: UIUserNotificationSettings)	Метод изменения настроек подписки. `[[PushServerAPI default] didRegisterUserNotificationSettings: notificationSettings];`
7.	handleActionWithIdentifier(_ identifier: String?, userInfo: [AnyHashable: Any], responseInfo: [AnyHashable: Any], completion: @escaping () -> Void)	Метод вызывается, когда приложение было активировано клиентом по выбору действия из remote push-уведомления. `[[PushServerAPI default] handleActionWithIdentifier: identifier userInfo: userInfo responseInfo: responseInfo completion: completionHandler];`
8.	Default	Переменная инициализации библиотеки.

№ п/п	Метод	Описание
9.	static var PUSH_API_LOG_ENABLE: Bool	Переменная включения логирования в консоли вывода (по умолчанию логирование выключено).
10.	static func showNetworkActivity(_ show: Bool)	Метод отображает индикатора загрузки данных из интернета в статус баре ОС. Значение по умолчанию – NO. `[PushServerAPI showNetworkActivity: YES];`

11.	var onDeviceAddressChanged: ((String?) -> ())?	Метод-обработчик изменения deviceAddress приложения.
		`[[PushServerAPI default] setOnDeviceAddressChanged:^(NSString * _Nullable deviceAddress) {`
		`}];`

12.	onPushMessagesReceived	Метод-обработчик получения новых push-сообщений.
		`[[PushServerAPI sharedInstance] setOnPushMessagesReceived:^(NSArray<PushNotificationMessage *> * _Nonnull messages) {`
		`}];`

13.	onPushMessagesWereRead	Метод-обработчик информации о push-сообщениях, прочитанных в другом приложении.
		`[[PushServerAPI default] setOnPushMessagesWereRead:^(NSArray<NSString *> * _Nonnull messages)`
		`}];`

14.	var deviceAddress: String?	Переменная deviceAddress приложения.
15.	var deviceUid: String?	Переменная deviceUid – уникальный идентификатор приложения: - если в Info.plist флаг установлен «PS_NEED_AUTOMATICALLY_GENERATE_DEVICE_UID = YES», то deviceUid генерируется автоматически; - если в Info.plist флаг не установлен «PS_NEED_AUTOMATICALLY_GENERATE_DEVICE_UID = NO», то deviceUid необходимо выставить заранее методом setEnabledPushNotifications(true …).
16.	isPushNotificationsEnabled()	Метод проверки включения push-уведомлений. PushServerAPI считает push-уведомлений доступными для отправки в приложение при условии: - setEnabledPushNotificaions(true …); - установлено разрешение на отправку push-уведомлений у приложения.

№ п/п	Метод	Описание

17.	setEnablePushNotificaions (_:types:categories:complete:)	Метод включение и выключение push-уведомлений на push-сервере:
		- если передан параметр «true», начинается регистрация приложения на push-сервере (после завершения вызывается complete-замыкание);
		- если выставляется параметр «false», приложение отказывается от получения push-уведомлений от push-сервера (после завершения вызывается complete-замыкание).
		`(void)changedValue:(UISwitch *)sender {`
		`UIUserNotificationSettings *settings = [UIUserNotificationSettings settingsForTypes: (UIUserNotificationTypeBadge | UIUserNotificationTypeSound | UIUserNotificationTypeAlert) categories:nil];`
		`[[UIApplication sharedApplication] registerUserNotificationSettings: settings];`
		`[[PushServerAPI default] setEnablePushNotifications: sender.isOn complete:^{`
		`}];`
		`}}];`
		`}`

18.	setClientId(_:complete:)	Метод привязки clientId к зарегистрированному приложению. Ограничения для clientId: размер строки от 1 до 256 символов включительно. В замыкание приходит PushServerAPIResponse с кодами ошибок и описанием ошибок. `public func setClientId(clientId:String, complete:((error:PushServerAPIResponse?) ->())?) -> PushServerAPI`

19.	setPhoneNumber(_:complete:)	Метод привязки phoneNumber к зарегистрированному приложению.
		Ограничения к формату телефонного номера: строка, состоящая только из чисел 0123456789, длина которой должна быть не менее 2 и не более 15 символов.
		`[[PushServerAPI default] setPhoneNumber:@"79161111111" complete:^(SetPhoneNumberResponse * _Nullable response) {`
		`if (response.errorCode > 0) {`
		`//Error handling`
		`}`
		`if (response.requredConfirmation) {`
		`[[PushServerAPI default] confirmPhoneNumber:@"000" complete:^(ConfirmPhoneNumberResponse * _Nullable confirmationResponse) {`

№ п/п	Метод	Описание

		`if (response.errorCode > 0) {`
		`//Error handling`
		`}`
		`}];`
		`}`
		`}];`
		public func setPhoneNumber(phoneNumber:String, complete:((error:SetPhoneNumberResponse?) ->())?) -> PushServerAPI

20.	confirmPhoneNumber(_:complete:)	Метод подтверждения PhoneNumber.
		Ограничения к коду подтверждения: строка, состоящая только из чисел 0123456789, длина которой должна быть не менее 2 и не более 15 символов.
		`[[PushServerAPI default] setPhoneNumber:@"79161111111" complete:^(SetPhoneNumberResponse * _Nullable response) {`
		`if (response.errorCode > 0) {`
		`//Error handling`
		`}`
		`if (response.requredConfirmation) {`
		`[[PushServerAPI default] confirmPhoneNumber:@"000" complete:^(ConfirmPhoneNumberResponse * _Nullable confirmationResponse) {`
		`if (response.errorCode > 0) {`
		`//Error handling`
		`}`
		`}];`
		`}`
		`}];`
		public func confirmPhoneNumber(code:String, complete:((error:ConfirmPhoneNumberResponse?) ->())?) -> PushServerAPI

21.	forceSync(_:)	Метод принудительной синхронизации. Запрос выполняется при наличии прочитанных сообщений в одном приложении и неполученных вторым приложением. Неполученные сообщения отработают соответствующие callback: - didPushMessagesReceived; - didPushMessagesWereRead. public func forceSync(complete:((error:PushServerAPIResponse?) -> ())?) -> PushServerAPI
22.	markMessagesAsRead(_:complete:)	Метод помечает полученные сообщения как прочитанные.

№ п/п	Метод	Описание
		[[PushServerAPI sharedInstance] markMessagesAsRead:@[self.message.messageId] complete:^(PushServerAPIResponse * _Nullable error) { }]; public func markMessagesAsRead(messagesId:Array<String>?, complete:((error:PushServerAPIResponse?) -> ())?) -> PushServerAPI
24.	resetNewMessageCounter(_:)	Метод сброса счетчика непрочитанных сообщений на push-платформе. [[PushServerAPI sharedInstance] resetNewMessageCounter:^(PushServerAPIResponse * _Nullable error) { }]; public func resetNewMessageCounter(complete:((error:PushServerAPIResponse?) -> ())?) -> PushServerAPI
25.	showInputConfirmationCodeDialog (_:complete:)	Метод привязки PhoneNumber к зарегистрированному приложению. public func showInputConfirmationCodeDialog(ctrl:UIViewController, complete: (() -> ())?)
26.	messageSend(_ content: String?, systemType: Bool, complete:((_ response:SendMessageResponse?) -> ())?)	Метод отправки сообщения на push-сервер. Для работы данного метода требуется обязательная привязка clientId к зарегистрированному приложению (история сообщений хранится с привязкой к clientId). (PushServerAPI * _Nonnull)messageSend:(NSString * _Nullable)content systemType:(BOOL)systemType complete:(void (^ _Nullable)(PushServerAPIResponse * _Nullable error))complete;
27.	getMessageHistory(_ clientId:String, maxCount:Int, complete:((_ syncToken: String?, _ response:InOutMessageHistoryResponse?) -> ())?) -> PushServerAPI	Метод получения истории сообщений. Для работы данного метода требуется обязательная привязка clientId к зарегистрированному приложению (история сообщений хранится с привязкой к clientId). [[PushServerAPI default] getMessageHistory:@"sadfasd" maxCount:25 complete:^(NSString *token, InOutMessageHistoryResponse * _Nullable response) { }];
28.	getNextMessageHistory(_ clientId:String, maxCount:Int, complete:((_ syncToken: String?, _	Метод для получения следующих сообщений истории.

№ п/п	Метод	Описание
	response:InOutMessageHistoryResponse?) -> ())?) -> PushServerAPI	Для работы данного метода требуется обязательная привязка clientId к зарегистрированному приложению (история сообщений хранится с привязкой к clientId). `[[PushServerAPI default] getNextMessageHistory:@"sadfasd" maxCount:25 complete:^(NSString *token, InOutMessageHistoryResponse * _Nullable response) { }];`
29.	getMessageHistory(_ clientId:String, maxCount:Int, syncToken:String?, complete:((_ syncToken: String?, _ response:InOutMessageHistoryResponse?) -> ())?) -> PushServerAPI	Метод получения сообщений из истории по syncToken: Для работы данного метода требуется обязательная привязка clientId к зарегистрированному приложению (история сообщений хранится с привязкой к clientId). `[[PushServerAPI default] getMessageHistory:@"sadfasd" maxCount:25 syncToken: @"asdfas3bb3rg" complete:^(InOutMessageHistoryResponse * _Nullable response) { }];`
30.	setFailureBlock	Метод-обработчик возвращает ошибки работы библиотеки. `[[PushServerAPI default] setFailureBlock:^(NSString *error) { }];`

3.8. Регистрация приложения по deviceUid

Для регистрации приложения на push-сервере по deviceUid необходимо:

1. Указать в параметре PS_NEED_AUTOMATICALLY_GENERATE_DEVICE_UID значение «NO»:

```
PS_NEED_AUTOMATICALLY_GENERATE_DEVICE_UID = NO
```

2. Указать значение deviceUid:

```
[PushServerAPI sharedInstance].deviceUid = @"value_deviceUid";
```

3. Произвести регистрацию voip/remote push-уведомлений:
   - для remote push-уведомлений:

```
[[UIApplication sharedApplication] registerForRemoteNotifications];
```

   - для VoIP push-уведомлений:

```
registry = [[PKPushRegistry alloc] initWithQueue: dispatch_get_main_queue()];
registry.delegate = self;
registry.desiredPushTypes = [NSSet setWithObject: PKPushTypeVoIP];
```

4. Создать запрос Клиенту на разрешение отправки push-уведомлений:

```
UIUserNotificationSettings *settings = [UIUserNotificationSettings
settingsForTypes: (UIUserNotificationTypeBadge | UIUserNotificationTypeSound |
UIUserNotificationTypeAlert) categories:nil];
[[UIApplication sharedApplication] registerUserNotificationSettings: settings];
```

5. Выполнить вызов setEnablePushNotifications(true).

Доступна процедура отправки push-сообщений по deviceUid.

3.9. Привязка clientId к зарегистрированному приложению

Для привязки clientId к зарегистрированному приложению необходимо:

1. Выполнить вызов метода setEnablePushNotifications(true) (подробное описание методов представлено в таблице
2. Таблица 1).
3. Выполнить вызов метода setClientId.

   Доступна процедура отправки push-сообщений по clientId.

3.10. Привязка phoneNumber к зарегистрированному приложению

Для привязки phoneNumber к зарегистрированному приложению необходимо :

1. Выполнить вызов метода setEnablePushNotifications(true) (подробное описание методов представлено в таблице
2. Таблица 1).

3. Выполнить вызов метода setPhoneNumber (подробное описание методов представлено в таблице
Таблица 1). Доступна процедура отправки push-сообщений по phoneNumber.

4. СТРУКТУРА СООБЩЕНИЯ PUSHNOTIFICATIONMESSAGE

Структура сообщения PushNotificationMessage представлена в таблице Таблица 2.

Таблица 2. Структура сообщения PushNotificationMessage.

Имя	Тип	Значение
messageId	String	Идентификатор сообщения, readonly
sentAt	Int64	Время отправки (поступления на платформу), readonly
read	Bool	Флаг, означающий, что сообщение было прочитано Клиентом, read and write
secured	Bool	Сообщение было отправлено в безопасном режиме (с sessionKey), readonly
shortMessage	String	Заголовок сообщения, отправляемый PNS, readonly
fullMessage	String	Полный текст сообщения, readonly
markMessageAsRead	Boolean	Сообщить на push платформу что сообщение было прочитанно, значение read выставляется в true и отправляется на push платформу
hashValue	Int	Hash-значение объекта
description	String	Стандартное описание объекта, наследника от NSObject
debugDescription	String	Стандартное описание для вывода информации с помощью LLDV-дебагера

Пример сообщения:

```
{messageId: "10",
sentAt: "24.10.2012 12:41:57",
shortMessage: "Уважаемый, ФИО! Изменились …",
fullMessage: "0KPQstCw0LbQsNC10LzRi9C5INC60LvQuNC10L3RgiEg0JjQt",
secured: true
read: true
sessionKey: sessionKey
}
```

5. КОДЫ ОШИБОК

В метод, обрабатывающий ошибки взаимодействия с push-сервером, передаются коды ошибок. Описание кодов ошибок представлено в таблице 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
5.	IO_ERROR	Сетевая ошибка
6.	CLIENT_ID_INVALID	Указано некорректное значение идентификатора clientId
7.	PHONE_NUMBER_INVALID	Указано некорректное значение идентификатора phoneNumber
8.	PHONE_NUMBER_TEMPORARY_ BLOCKED	Превышено количество попыток ввода значения confirmationCode
9.	DEVICE_ADDRESS_TEMPORARY_ BLOCKED	Превышено количество попыток регистрации приложения
10.	CONFIRMATION_CODE_INVALID	Указано неверное значение confirmationCode
11.	CONFIRMATION_CODE_EXPIRED	Истек срок действия confirmationCode
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
16.	PHONE_NUMBER_IS_EMPTY	Не указано значение номера телефона
17.	CONFIRMATION_CODE_IS_EMPTY	Не указано значение confirmation code

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

Описание ошибки: errorDescription.