РУКОВОДСТВО РАЗРАБОТЧИКА

Часть 1

Подсистема Сервис уведомлений

для операционной системы Аврора

Версия 1.1

Листов 52

2026

АННОТАЦИЯ

Настоящий документ является первой частью руководства разработчика подсистемы Сервис уведомлений (ПСУ), входящей в состав прикладного программного обеспечения «Аврора Центр» (далее – ППО) релиз 5.7.0.

ПРИМЕЧАНИЯ:

✓ Подробная информация о составе и назначении ППО, а также требования к условиям выполнения приведены в документе «Руководство администратора»;

✓ Подробная информация об особенностях резервного копирования приведена в документе «Рекомендации по резервному копированию».

Настоящий документ содержит инструкции для разработчика приложений, выполняющих отправку push-уведомлений на устройства, функционирующие под управлением операционной системы (ОС) Аврора.

СОДЕРЖАНИЕ

1. Общая информация 5
1.1. Работа с тестовым Сервером приложений ПСУ 5
1.2. Получение доступа к тестовому Серверу приложений ПСУ 7
1.3. Плагин для эмуляции push-уведомлений 8
2. Описание работы 11
2.1. Подготовка к работе 11
2.2. Разработка приложения 12
2.2.1. Общее описание 12
2.2.2. Работа с push-уведомлениями от версии ОС 4.0 и выше 16
2.2.3. UI уведомления (для ОС от версии 5.0 и выше) 26
2.3. Отправка push-уведомлений 27
2.3.1. Запрос токена авторизации 27
2.3.2. Подпись токена авторизации 29
2.3.3. Оправка push-уведомлений 31
2.3.4. Получение информации о проекте push-уведомлений 35
2.3.5. Рекомендации использования запроса 39
2.3.6. Автоматическая смена ключа безопасности проекта 39
2.4. Пример кода сервера внешнего приложения 44
2.4.1. Код на языке Python 44
2.4.2. Код на языке Java 44
3. Ограничения 46
4. Диагностика и устранение неполадок 47
4.1. Проверка известных проблем 47
4.2. Проверка счетчика принятых push-уведомлений 48
4.3. Диагностика при отсутствии известных проблем 48

4.4. Локализация проблемы, если используемый push-сервер развернут в составе Аврора Центр 49

Перечень терминов и сокращений 50

1. ОБЩАЯ ИНФОРМАЦИЯ

ПРИМЕЧАНИЕ. Функциональность получения push-уведомлений доступна в ОС Аврора начиная с версии 3.2.2.

Если на момент отправки push-уведомления устройство недоступно, приложение получит push-уведомление после подключения к сети, обеспечивающей доступ к Серверу приложений ПСУ.

Для использования push-уведомлений в своих системах разработчикам необходимо расширить функциональность приложений и сервера внешнего приложения, отправляющего push-уведомления.

ПРИМЕЧАНИЕ. Не рекомендуется с помощью push-уведомлений осуществлять передачу конфиденциальной и чувствительной информации в незащищенных сетях, т.к. протокол взаимодействия устройства с Сервером приложений ПСУ не подразумевает передачу конфиденциальной информации.

При разработке приложений доступны следующие способы тестирования функциональности push-уведомлений:

– подключение к тестовому серверу;

– использование плагина для эмуляции Сервера приложений ПСУ.

1.1. Работа с тестовым Сервером приложений ПСУ

Для того, чтобы использовать в разработке тестовый Сервер приложений ПСУ для отправки push-уведомлений, необходимо выполнить следующие шаги:

1) Получить доступ к тестовому Серверу приложений ПСУ:

– направить запрос в техническую поддержку предприятия-разработчика dev-support@omp.ru;

– получить ключи для доступа к тестовому Серверу приложений ПСУ;

– использовать ключи при взаимодействии сервера внешнего приложения с тестовым Сервером приложений ПСУ;

2) Создать и установить приложение на устройстве:

– установить зависимости;

– передать в приложение `applicationId`, полученный от поддержки разработчиков;

– зарегистрироваться в push-демоне с помощью `applicationId`, получив в ответ на запрос `registrationId` для тестового Сервера приложений ПСУ;

3) Организовать передачу push-уведомлений сервера внешнего приложения через тестовый Сервер приложений ПСУ:

– передать `registrationId` серверу внешнего приложения доверенным способом от приложения. Это позволит серверу внешнего приложения отправлять таргетированные push-уведомления к своим приложениям на конкретном устройстве;

– получить токен доступа на тестовом Сервере приложений ПСУ через сервер внешнего приложения;

– передать через сервер внешнего приложения push-уведомления, используя токен доступа и идентификатор связки приложение – устройство `registrationId`.

Процесс регистрации приложений приведен на рисунке (Рисунок 1).

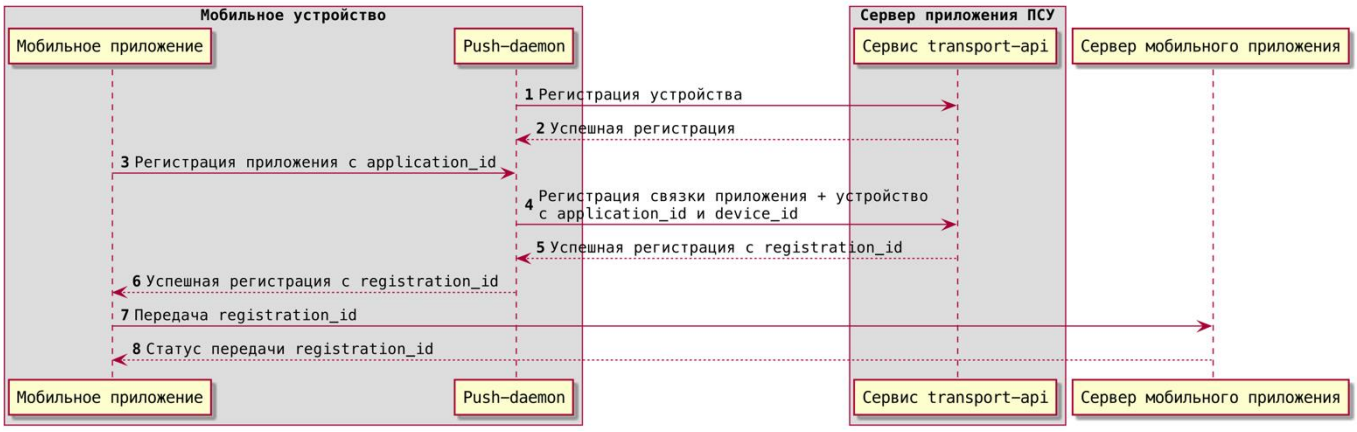


Рисунок 1

Запрос на регистрацию приложений должен происходить при каждом его запуске, т.к `registrationId` имеет срок жизни, не зависящий от приложений.

1.2. Получение доступа к тестовому Серверу приложений ПСУ

ПРИМЕЧАНИЕ. Для тестирования сервера внешнего приложения, который отправляет push-уведомления, и приложения, которое получает push-уведомления, необходимо использовать тестовый Сервер приложений ПСУ.

Для получения настроек подключения к тестовому Серверу приложений ПСУ необходимо направить письмо в техническую поддержку предприятия-разработчика: dev-support@omp.ru, указав название организации, которой требуется предоставить доступ. В ответном письме будут направлены адрес и порт, а также файл с настройками Сервера приложений ПСУ и настройками приложений. Настройки Сервера приложений ПСУ выдаются в единственном экземпляре на организацию, при этом приложений (каждое из которых имеет настройки) может быть несколько.

Настройки необходимо указать в конфигурации сервера внешнего приложения, который после прохождения аутентификации получает токен доступа и отправляет push-уведомления через Сервер приложений ПСУ.

Список настроек для подключения сервера внешнего приложения к тестовому Серверу приложений ПСУ:

– `project_id` – уникальный идентификатор (ID) проекта;

– `push_public_address` – адрес тестового Сервера приложений ПСУ, на который будут отправляться запросы;

– `api_url` – адрес API тестового Сервера приложений ПСУ;

– `client_id` – аккаунт учетной записи разработчика в подсистеме безопасности тестового Сервера приложений ПСУ, обычно совпадает с ID проекта;

– `scopes` – список действий, которые могут быть разрешены приложению;

– `audience` – предполагаемый потребитель токена, как правило, это сервер ресурсов (API), к приложению которого необходимо получить доступ;

– `token_url` – адрес получения токена авторизации для запросов к тестовому Серверу приложений ПСУ;

– `key_id` – идентификатор приватного ключа RSA для передачи сообщений между сервером внешнего приложения и Сервером приложений ПСУ;

– `private_key` – приватный ключ RSA для передачи сообщений между сервером внешнего приложения и тестовым Сервером приложений ПСУ. Рекомендации по ротации приватного ключа проекта приведены в п. 2.3.6.

Для приложений доступна настройка следующих параметров:

– `application_id` – уникальный идентификатор приложений, который необходимо добавить в приложение;

– `project_id` – уникальный идентификатор проекта.

ПРИМЕЧАНИЕ. `Application_id` необходимо использовать для регистрации в push-демоне на устройстве. Каждому `application_id` может соответствовать только 1 приложение.

1.3. Плагин для эмуляции push-уведомлений

В Аврора SDK предусмотрена возможность протестировать работу стороннего приложения, отправляющего push-уведомления, на эмуляторе.

Для настройки плагина push-уведомлений необходимо выполнить следующие действия:

1) Сгенерировать сертификат `cert.pem` и ключ `key.pem`, выполнив команду `openssl` в терминале в системе разработчика. Вместо многоточий допускается заполнить поля в опции `-subj` собственными значениями или опустить. Ограничения на данные значения отсутствуют.

Пример:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days
365 -nodes -subj
"/C=RU/ST=.../L=.../O=.../OU=.../CN=.../emailAddress=..."
```

2) В Аврора IDE запустить машину сборки Аврора Build Engine, нажав кнопку запуска, расположенную в нижнем левом углу окна Аврора IDE;

3) Создать файл настроек для службы push-уведомлений: «Параметры» -> «Push-уведомления» -> вкладка «Служба» -> «Файл настроек» -> «Создать»;

4) На той же вкладке «Служба» в разделе «Конфигурация» указать сертификат и ключ из шага 1 в полях «Файл сертификата» и «Файл ключа»;

5) При необходимости добавить значение по умолчанию для полей «Данные» и «Заголовок» во вкладке «Значение по умолчанию для уведомлений»;

6) Нажать кнопку «Применить»;

7) Установить на эмуляторе пакет `push-example` из системных репозиториев. Для этого необходимо подключиться к эмулятору и выполнить команду `pkcon`:

```
ssh -p 2223 -i ~/AuroraOS/vmshare/ssh/private_keys/Aurora_OS-Emulator-
latest/nemo nemo@localhost
pkcon install push-example
```

8) Открыть приложение `push-example` на эмуляторе;

9) В Аврора IDE открыть нижнюю вкладку «Push-уведомления» или нажать сочетание клавиш Alt+9;

10) Выбрать идентификатор устройства;

11) Ввести идентификатор приложения в поле справа от идентификатора устройства. Для push-example это `testApplication`;

12) Ввести текст уведомления и нажать кнопку «Отправить push-уведомление».

Плагин для эмуляции push-уведомлений позволяет отправлять уведомления не только на тестовое, но и на собственные приложения. Для этого необходимо выполнить следующие действия:

– запустить приложение;

– в Аврора IDE во вкладке «Push-уведомления» указать в качестве идентификатора `applicationId`, с которым приложение регистрируется в push-демоне. Идентификатор для эмулятора может быть любым.

ПРИМЕЧАНИЕ. Подробное описание процесса разработки собственного приложения для push-уведомлений приведено в настоящем документе.

2. ОПИСАНИЕ РАБОТЫ

2.1. Подготовка к работе

Для получения push-уведомлений на устройстве используется push-демон, являющийся системным компонентом ОС Аврора.

ПРИМЕЧАНИЕ. При разработке приложений следует убедиться, что в Аврора SDK и на устройствах имеются необходимые пакеты.

Для входа в Аврора Build Engine необходимо выполнить команду:

```
ssh -p 2222 -i ~/AuroraOS/vmshare/ssh/private_keys/engine/mersdk
mersdk@localhost
```

Для установки пакетов (если они отсутствуют) в Аврора Build Engine необходимо выполнить команду:

```
zypper in push-daemon-libs push-daemon-devel
```

Для входа на устройство необходимо выполнить команду (`device ip` следует уточнить в настройках устройства; `username` для версий ОС Аврора начиная с 4.0 – `defaultuser`, для версий ниже – `nemo`):

```
ssh <username>@<device ip>
```

Для устройства необходимо выполнить команду установки библиотек:

```
pkcon install push-daemon push-daemon-libs
```

Для проверки наличия пакета в Аврора Build Engine необходимо выполнить команду:

```
zypper search <имя пакета>
```

Пример команды для устройства:

```
pkcon search <имя пакета>
```

2.2. Разработка приложения

2.2.1. Общее описание

ПРИМЕЧАНИЕ. Функционал получения push-уведомлений доступен в ОС Аврора начиная с версии 3.2.2.

ВНИМАНИЕ! Приведенное в настоящем подразделе описание классов и функций применимо для ОС Аврора начиная с версии 4.0.

Для полноценной работы push-демона приложение должно поддерживать работу в фоновом режиме, т.е. процесс должен выполняться и при закрытии графического интерфейса.

Основной класс, используемый для работы с push-уведомлениями, – `Aurora::PushNotifications::Client`.

В начале работы приложения необходимо установить `applicationId` с помощью метода `Aurora::PushNotifications::Client::setApplicationId()`.

Значение `applicationId` необходимо получить в технической поддержке предприятия-разработчика. Данный идентификатор можно вынести в конфигурацию приложений или программно запрашивать у Сервера приложений ПСУ до начала процесса регистрации.

ПРИМЕЧАНИЕ. Каждому `applicationId` может соответствовать только одно приложение.

На основе `applicationId` push-демон выполняет поиск соответствующего ему `registrationId`, после чего при вызове метода `registrate()` эта пара отправляется на сервер, где производится проверка ее актуальности. Если пара не актуальна (или если `registrationId` пустой в случае первой регистрации), то генерируется новый `registrationId`, который возвращается приложением через push-демон. Если же пара уже была актуальна, то возвращается существующий `registrationId`, поэтому `registrationId` является постоянным для связки приложения и устройства.

В случае неактивности пользователя в течение длительного времени будет отправлен сигнал `clientInactive`, при этом если приложение работает в фоновом режиме и не выполняется никаких активных процессов, то оно должно завершить работу в целях экономии заряда аккумулятора. В дальнейшем для возобновления работы приложения его потребуется запустить заново.

Каждое push-уведомление содержит поля для заголовка, тело сообщения и объект с данными, который представляет собой набор пар «ключ-значение». Структура с описанием push-уведомления:

```
struct Push
{
    QString data; // строка, содержащая сериализованный JSON-объект
    QString title; // заголовок сообщения
    QString message; // тело сообщения
    QString action; // зарезервировано для будущего использования
    QString image; // зарезервировано для будущего использования
    QString sound; // зарезервировано для будущего использования
    quint32 categoryId = 0; // зарезервировано для будущего использо-
вания
};
```

ПРИМЕЧАНИЕ. Все поля, кроме `data`, предназначены для системы и нет гарантии, что они будут передаваться в приложение всегда.

`Data` является сериализованным в строку JSON-объектом. Формат JSON-объекта в `data` может быть произвольным. Атрибут `data`, в частности, можно использовать для указания типа push-уведомления — `text`, `command` и т.д., например, «`{«type»: «command», «text»: «value»}`».

Список этих структур `Aurora::PushNotifications::PushList` возвращается с сигналом `Aurora::PushNotifications::Client::notifications` (см. API `Aurora::PushNotifications::Client`). Каждое push-уведомление, полученное таким образом, в дальнейшем может быть выведено как уведомление (п. 2.2.2).

API клиента `Aurora::PushNotifications::Client` для работы с push-демоном состоит из методов и сигналов.

Основные методы:

– `void setApplicationId(const QString &applicationId)` — устанавливает объекту класса `Client applicationId` приложения, должен быть вызван непосредственно перед вызовом `startHandler`.

Пример:

```
 bool Application::start(const QStringList &arguments)
{
    auto applicationId = getApplicationId();
    m_client->setApplicationId(applicationId); // m_client - это
объект класса `Aurora::PushNotifications::Client`
    if (!m_client->startHandler()) {
        qWarning() << "Failed to start push client, can not start han-
dler";
        return false;
    }
    if (arguments.indexOf(QStringLiteral("--gui")) != -1) {
        startGui();
    }
    return true;
}
```

– `QString applicationId()` — возвращает установленный в объекте класса `Client applicationId`.

Пример:

```
qDebug() << "Current client applicationId:" << m_client-
>applicationId(); // m_client - это объект класса
Aurora::PushNotifications::Client
```

– `void registrate()` – отправляет запрос на Сервер приложений ПСУ на регистрацию приложения (следует вызывать при каждом запуске приложения);

– `bool isPushSystemReady()` – возвращает статус готовности push-демона:

- `true`, если push-демон может обрабатывать запросы приложения;
- `false` — в противоположном случае;

– `int error()` – позволяет получить код последней ошибки регистрации;

– `QString errorMessage()` – позволяет получить текстовое описание последней ошибки регистрации.

Основные сигналы:

– `void pushSystemReadinessChanged(bool status)` – передается при изменении состояния push-демона. `Status` имеет значение `true`, если устройство способно получать push-уведомления, `false` – в противоположном случае.

Пример:

```
connect(m_client, &Aurora::PushNotifications::Client::pushSystemReadi-
nessChanged, this, [](bool status){
    qDebug() << "Daemon can receive push notifications: " << status;
});
```

– `void registrationId(const QString &registrationId)` – передается при получении объектом класса `Client` `registrationId`, содержащего идентификатор приложения. Данный идентификатор должен быть отправлен на Сервер приложений ПСУ для реализации возможности отправки push-уведомлений.

Пример:

```
connect(m_client, &Aurora::PushNotifications::Client::registrationId,
this, [this](const QString &registrationId){
    qDebug() << QString("Registration is successful for %1. Registra-
tionId: %2").arg(applicationId()).arg(registrationId);
    setRegistrationId(registrationId);
});
```

– `void registrationError()` – передается в случае неудачной регистрации.

Пример:

```
connect(m_client, &Aurora::PushNotifications::Client::registrationEr-
ror, this, [this]() {
    qWarning() << "Registration error for " << applicationId();
});
```

– `void notifications(const Aurora::PushNotifications::PushList &pushList)` – передается при получении объектом класса `Client` push-уведомлений. `PushList` содержит список сообщений.

Пример:

```
connect(m_client, &Aurora::PushNotifications::Client::notifications,
this, [this](const Aurora::PushNotifications::PushList &pushList){
    for (const Aurora::PushNotifications::Push &push : pushList) {
        qInfo() << "Push title" << push.title;
        qInfo() << "Push message" << push.message;
    }
});
```

– `void clientInactive()` – передается в случае, если приложение не взаимодействовало с push-уведомлениями длительное время. Если приложение запущено в фоновом режиме, его рекомендуется выключить.

Пример:

```
connect(m_client, &Aurora::PushNotifications::Client::clientInactive,
this, [this](){
    if (!m_guiStarted) {
        qInfo() << "Handling inactivity in background mode";
        QGuiApplication::instance()->quit();
        return;
    }
    qInfo() << "Ignoring inactivity signal";
});
```

2.2.2. Работа с push-уведомлениями от версии ОС 4.0 и выше

Пример стороннего приложения Push Receiver демонстрирует использование API Push daemon, получение и обработку push-уведомлений, а также их отображение. Пример полностью опубликован на ресурсе https://hub.mos.ru/auroraos/examples.

Архитектура приложения включает следующие элементы:

– `main.cpp` – стартовая точка приложения;

– `ApplicationController` – класс, реализующий клиентское API для работы с push-уведомлениями. Унаследован от QObject;

– `ApplicationService` – класс, реализующий сервис для контроля работы приложения в фоновом режиме/на переднем плане с помощью D-Bus-сервиса и аргументов командной строки. Унаследован от QDBusAbstractAdaptor;

– вспомогательные классы и qml-код для графического интерфейса (в настоящем документе не приводятся, могут быть определены произвольно).

2.2.2.1. Объявление приложения

Фрагмент кода класса, отвечающего за взаимодействие API push-демона и интерфейса приложения:

```
// SPDX-FileCopyrightText: 2023 Open Mobile Platform LLC <commu-
nity@omp.ru>
// SPDX-License-Identifier: BSD-3-Clause

#ifndef APPLICATIONCONTROLLER_H
#define APPLICATIONCONTROLLER_H

#include <QtCore/QObject>

#include <push_client.h>
#include <auroraapp.h>

class QQuickView;
class NotificationsModel;
class ApplicationController : public QObject
{
    Q_OBJECT

    Q_PROPERTY(NotificationsModel *notificationsModel READ notifica-
tionsModel CONSTANT)
    Q_PROPERTY(QString applicationId READ applicationId NOTIFY appli-
cationIdChanged)
    Q_PROPERTY(QString registrationId READ registrationId NOTIFY reg-
istrationIdChanged)

public:
    explicit ApplicationController(QObject *parent = nullptr);
    ApplicationController(const QStringList &arguments, QObject *par-
ent = nullptr);

    NotificationsModel *notificationsModel() const;
    QString applicationId() const;
    QString registrationId() const;

signals:
    void applicationIdChanged(const QString &applicationId);
    void registrationIdChanged(const QString &registrationId);

public slots:
    void showGui();
```

```
private slots:
    void _setApplicationId(const QString &applicationId);
    void _setRegistrationId(const QString &registrationId);
    void pushListToNotificationList(const Aurora::PushNotifica-
tions::PushList &pushList);
    void quitOnClientInactive();

private:
    QString _readApplicationId() const;

private:
    Aurora::PushNotifications::Client *m_notificationsClient{ nullptr
};
    NotificationsModel *m_notificationsModel{ nullptr };
    QQuickView *m_view{ nullptr };

    QString m_registrationId{};
};

#endif // APPLICATIONCONTROLLER_H
```

API `Aurora::PushNotifications` доступно при подключении библиотеки `libpushclient`, основные классы объявлены в заголовочном файле `push_client.h`.

В конструкторе происходит подключение сигналов `Aurora::PushNotifications::Client` к слотам `Application`:

```
    connect(m_notificationsClient, &Client::clientInactive, this,
            &ApplicationController::quitOnClientInactive);
    connect(m_notificationsClient, &Client::pushSystemReadiness-
Changed, [](bool status) {
        qWarning() << "Push system is" << (status ? "available" : "not
available");
    });
    connect(m_notificationsClient, &Client::registrationId, this,
            &ApplicationController::_setRegistrationId);
    connect(m_notificationsClient, &Client::registrationError,
            [] () { qWarning() << "Push system have problems with reg-
istrationId"; });
    connect(m_notificationsClient, &Client::notifications, this,
            &ApplicationController::pushListToNotificationList);
```

2.2.2.2. Идентификатор приложений

Для успешной работы приложения необходимо создать файл `/usr/share/ru.auroraos.PushReceiver/applicationid` и добавить в него идентификатор приложения, который был получен вместе с другими настройками подключения к Серверу приложений ПСУ (список настроек приведен в подразделе 1.2). Для этого требуется выполнить на устройстве команду:

```
echo "идентификатор" >
/usr/share/ru.auroraos.PushReceiver/applicationid
```

Следует убедиться, что у файла достаточно прав, чтобы пользователь мог читать этот файл. Для выдачи прав файлу можно выполнить на устройстве команду:

```
chmod 644 /usr/share/ru.auroraos.PushReceiver/applicationid
```

В классе `ApplicationController` идентификатор из `/usr/share/ru.auroraos.PushReceiver/applicationid` считывается в методе `_readApplicationId()` следующим образом:

```
QString ApplicationController::_readApplicationId() const
{
    QFile applicationIdFile(QStringLiteral(APP_ID_FILE_PATH));
    return applicationIdFile.exists() &&
applicationIdFile.open(QIODevice::ReadOnly)
            ? applicationIdFile.readAll().trimmed()
            : QString();
}
```

где `APP_ID_FILE_PATH = /usr/share/ru.auroraos.PushReceiver/applicationid`.

Альтернативным вариантом указания `applicationId` является задать его непосредственно в коде приложения.

В конструкторе класса `ApplicationController` считывается идентификатор приложения, который затем передается push-демону:

```
ApplicationController::ApplicationController(const QStringList
&arguments, QObject *parent)
    : QObject(parent),
      m_notificationsClient(new Client(this)),
      m_notificationsModel(new NotificationsModel(this))
{
    qRegisterMetaType<NotificationsModel *>("NotificationsModel *");

    _setApplicationId(_readApplicationId());

    if (arguments.indexOf(QStringLiteral("/no-gui")) == -1)
        showGui();
```

Также при старте обрабатываются аргументы командной строки `arguments`. Отсутствие аргумента `/no-gui` может служить сигналом запуска графического интерфейса в методе `showGui()`:

```
void ApplicationController::showGui()
{
    if (m_view) {
        m_view->raise();
        m_view->showFullScreen();
    } else {
        m_view = Aurora::Application::createView();
        m_view->rootContext()->setContextProperty(QStringLiteral("Ap-
plicationController"), this);
        m_view->setSource(Aurora::Application::pathTo(QStringLit-
eral("qml/PushReceiver.qml")));
        m_view->show();
    }
}
```

2.2.2.3. Регистрационный идентификатор

Результат регистрации в push-демоне будет обработан в слоте `_setRegistrationId()` `ApplicationController`:

```
void ApplicationController::_setRegistrationId(const QString &regis-
trationId)
{
    if (registrationId == m_registrationId)
        return;

    m_registrationId = registrationId;

    emit registrationIdChanged(registrationId);
}
```

После запуска приложение пытается зарегистрироваться на Сервере приложений ПСУ и в случае успеха получает идентификатор регистрации `registrationId`. В консоли `/journalctl` отобразится сообщение следующего вида:

```
„62bba07e-1000-4d6e-ad8b-47935e1b7d64"
```

Приложение должно передать `registrationId` в свой сервер внешнего приложения, чтобы сервер внешнего приложения мог отправлять push-уведомления для конкретного `registrationId`.

2.2.2.4. Push-уведомления

Получив push-уведомление, приложение сначала проверяет, не находится ли приложение на переднем плане, а затем выполняет анализ поля данных push-уведомления и поиск ключа `mtype` в полученном JSON.

Если `mtype` равно:

– `action` — уведомление не будет отображаться;

– `notify` — будет отображен только верхний баннер;

– `system_notify` — уведомление будет отображаться только в центре уведомлений.

При любом другом значении, а также при пустом значении или отсутствии `mtype` будут отображаться 2 push-уведомления (баннер и центр уведомлений).

Выбор способа отображения push-уведомления реализован в методе `pushListToNotificationList` класса `ApplicationController`:

```
void ApplicationController::pushListToNotificationList(const PushList
&pushList)
{
    for (const auto &push : pushList) {
        m_notificationsModel->insertPush(push);

        QJsonDocument jsonDcoument = QJsonDocu-
ment::fromJson(push.data.toUtf8());
        QString notifyType = jsonDcoument.ob-
ject().value("mtype").toString();

        if (notifyType == QStringLiteral("action"))
```

```
            continue;

        static QVariant defaultAction = Notification::remoteAction(
                QStringLiteral("default"), tr("Open app"), Applica-
tionService::notifyDBusService(),
                ApplicationService::notifyDBusPath(), Application-
Service::notifyDBusIface(),
                ApplicationService::notifyDBusMethod());

        Notification notification;
        notification.setAppName(tr("Push Receiver"));
        notification.setSummary(push.title);
        notification.setBody(push.message);
        notification.setIsTransient(false);
        notification.setItemCount(1);
        notification.setHintValue("x-nemo-feedback", "sms_exists");
        notification.setRemoteAction(defaultAction);
        notification.publish();
    }
}
```

2.2.2.5. Сервис для управления приложениями

Для управления приложением, способным как работать в фоновом режиме, так и запускать графический интерфейс, создается класс `ApplicationService`, который наследуется от `QDBusAbstractAdaptor`:

```
// SPDX-FileCopyrightText: 2023 Open Mobile Platform LLC <commu-
nity@omp.ru>
// SPDX-License-Identifier: BSD-3-Clause

#ifndef APPLICATIONSERVICE_H
#define APPLICATIONSERVICE_H

#include <QtDBus/QDBusAbstractAdaptor>

class ApplicationService : public QDBusAbstractAdaptor
{
    Q_OBJECT

    Q_CLASSINFO("D-Bus Interface", DBUS_INTERFACE)

public:
    explicit ApplicationService(QObject *parent = nullptr);
    ~ApplicationService() override;

    static bool isRegistered();
```

```
    static QString notifyDBusService();
    static QString notifyDBusPath();
    static QString notifyDBusIface();
    static QString notifyDBusMethod();

    static int updateApplicationArgs(const QStringList &arguments);

signals:
    void guiRequested();

public slots:
    void handleApplicationArgs(const QStringList &arguments);
    void handleApplicationWakeUp();
};

#endif // APPLICATIONSERVICE_H
```

Название сервиса, путь и интерфейс, которые может использовать приложение, ограничены форматом orgName.appName (подробнее в пп. 2.2.2.6).

В конструкторе регистрируется D-Bus-сервис:

```
ApplicationService::ApplicationService(QObject *parent) : QDBusAb-
stractAdaptor(parent)
{
    setAutoRelaySignals(true);

    QDBusConnection dbus = QDBusConnection::sessionBus();
    dbus.registerObject(dbusPathStr, this, QDBusConnection::Expor-
tAllSlots);
    if (!isRegistered()) {
        bool success = dbus.registerService(dbusServiceStr);
        if (!success)
            qApp->quit();
    }
}
```

Метод `updateApplicationArgs(const QStringList &arguments)` использует этот сервис, чтобы применить для приложения аргументы командной строки, вызвав `handleApplicationArgs(const QStringList &arguments)`:

```
int ApplicationService::updateApplicationArgs(const QStringList
&arguments)
{
    QDBusMessage message =
QDBusMessage::createMethodCall(dbusServiceStr, dbusPathStr,
dbusIfaceStr,
                                                        dbusMeth-
odStr);
    message.setArguments(QList<QVariant>() << arguments);
    QDBusMessage reply = QDBusConnection::sessionBus().call(message);

    return 0;
}
```

Метод `handleApplicationArgs(const QStringList &arguments)` отправляет сигнал о необходимости открыть графический интерфейс, если в аргументах командной строки имеется `-gui`:

```
void ApplicationService::handleApplicationArgs(const QStringList &ar-
guments)
{
    if (arguments.indexOf(QStringLiteral("/no-gui")) != -1)
        return;

    emit guiRequested();
}
```

Сервис создается и запускается в `main.cpp`:

```
int main(int argc, char *argv[])
{
    QScopedPointer<QGuiApplication> application(Aurora::Applica-
tion::application(argc, argv));
    application->setOrganizationName(QStringLiteral("ru.auroraos"));
    application->setApplicationName(QStringLiteral("PushReceiver"));

    QStringList applicationArgs = application->arguments();
    applicationArgs.removeFirst();

    if (ApplicationService::isRegistered()) {
        return ApplicationService::updateApplicationArgs(application-
Args);
    } else {
        QScopedPointer<ApplicationService> applicationService(
                new ApplicationService(application.data()));
        QScopedPointer<ApplicationController> applicationController(
                new ApplicationController(applicationArgs, applica-
tion.data()));
```

```
        QObject::connect(applicationService.data(), &Application-
Service::guiRequested,
                         applicationController.data(), &Application-
Controller::showGui);

        return application->exec();
    }
}
```

Сначала проверяется, не был ли сервис уже запущен. В этом случае у него только обновляются аргументы командной строки. Иначе создается и запускается новый сервис, а также приложение, которое получает аргументы командной строки.

2.2.2.6. Дополнительные настройки конфигурации

Для приложения необходимо в файле `.desktop` указать возможность принимать `uri` в качестве аргумента:

```
Exec=/usr/bin/ru.auroraos.PushReceiver  %u
```

Для доступа к push-демону приложению необходимо разрешение `PushNotifications`, которое можно задать в поле `Permissions` секции `X-Application` файла `.desktop`:

```
[X-Application]
Permissions=PushNotifications
```

Ограничение на имена D-Bus шин, которые может поднимать приложение. Доступны только имена вида `orgname.appname`, где `orgname` и `appname` задаются в секции `X-Application` файла `.desktop`:

```
[X-Application]
OrganizationName=ru.auroraos
ApplicationName=PushReceiver
```

В pro-файле необходимо указать, что приложение использует Qt D-Bus и дополнительные библиотеки `pushclient` и `nemonotifications-qt5`:

```
QT += dbus

PKGCONFIG += \
    pushclient \
    nemonotifications-qt5
```

ПРИМЕЧАНИЕ. После регистрации приложения в push-демоне оно отправляет на свой сервер внешнего приложения уникальный `registrationId`, при этом на сервере внешнего приложения необходим процесс, способный получать и сохранять `registrationId` для дальнейшей таргетированной отправки push-уведомлений.

2.2.3. UI уведомления (для ОС от версии 5.0 и выше)

В ОС версии 5.0 реализован новый механизм доставки push-уведомлений напрямую в UI (подраздел 2.3). Данный механизм убирает необходимость создавать уведомления непосредственно из приложения. Переход на новый механизм требует только подключения новой библиотеки `libpushapi`:

```
PKGCONFIG += pushapi
```

ПРИМЕЧАНИЕ. Если ранее приложение создавало уведомления с данными, пришедшими в push-уведомления, то при переходе на новый механизм, для избежания дублирования уведомлений, стоит избавиться от самостоятельного создания.

Приложение может реализовать в своем D-Bus сервисе метод, который будет вызван при клике на push-уведомление, если его имя было передано в поле `action` push-уведомления. На текущий момент поддерживается только 1 метод на push-уведомление и не поддерживаются методы с аргументами. Для того чтобы метод выполнялся (в том числе при выключенном приложении), необходимо сделать приложение DBus activatable путем добавления поля `ExecDBus` в секцию `X-Application` файла `.desktop`:

```
[X-Application]
ExecDBus=/usr/bin/ru.auroraos.PushReceiver %u
```

ВНИМАНИЕ! Поддержка механизма, описанного в пп. 2.3.3.2, сохраняется в данный момент, для продолжения работы не требуется дополнительных изменений в приложении или пересборки.

ПРИМЕЧАНИЕ. Библиотеки `libpushclient` и `libpushapi` конфликтуют друг с другом, одновременно можно использовать только одну из них.

2.3. Отправка push-уведомлений

Push-уведомления отправляются с сервера внешнего приложения через API Сервера приложений ПСУ, который защищен протоколом OAUTH2 OpenID Connect согласно JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication. При этом используется метод аутентификации `private_key_jwt`.

При регистрации проекта на Сервере приложений ПСУ разработчику будут направлены конфигурационные настройки приложения и сервера внешнего приложения. В конфигурации представлены параметры, необходимые для настройки клиента протокола авторизации OpenID Connect и формирования URL для отправки push-уведомлений, приведенные в подразделе 1.2.

Этапы отправки push-уведомлений:

1) Получение токена, позволяющего в течение времени его жизни отправлять push-уведомления (п. 2.3.1);

2) Непосредственно отправка push-уведомления (п. 2.3.3).

2.3.1. Запрос токена авторизации

Для получения токена необходимо выполнить неавторизованный запрос по адресу из параметра `token_url` (подраздел 1.2).

Запрос:

```
POST {token_url}
```

Пример:

```
POST http://example.ru/auth/public/oauth2/token
```

Заголовок `Content-Type` должен иметь значение `application/json`.

Обязательные параметры тела запроса приведены в таблице (Таблица 1).

Таблица 1

Обязательность	Параметр	Тип	Описание
Да	`scope`	`String`	Список действий, которые могут быть разрешены приложению. Значения из файла с настройками проекта (подраздел 1.2). Пример: `openid offline message:update project:read`
Да	`audience`	`String`	Предполагаемый потребитель токена, как правило, это сервер ресурсов (API), к которому приложению необходимо получить доступ. Значения из файла с настройками проекта (подраздел 1.2). Пример: `http://example.ru/auth/public http:// example.ru /push/public`
Да	`grant_type`	`String`	Тип авторизации сервера внешнего приложения в Подсистеме безопасности Push-сервера. Передать значение: «`client_credentials`»
Да	`client_assertion_type`	`String`	Передать значение, как строку: «`urn:ietf:params:oauth:client-assertion-type:jwt-bearer`»
Да	`client_assertion`	`String`	Сформированный самостоятельно JWT, содержащий информацию для аутентификации клиента. Он должен быть подписан цифровой подписью с использованием закрытого ключа, значение которого находится в параметре `private_key` из файла с настройками проекта (подраздел 1.2). Требования к формированию токена JWT можно посмотреть в таблице (Таблица 2)

JWT должен содержать обязательные параметры, приведенные в таблице (Таблица 2).

Таблица 2

Обязательность	Параметр	Тип	Описание
Да	`iss`	`String`	Issuer. Должен содержать `client_id` клиента OAuth
Да	`sub`	`String`	Subject. Должен содержать `client_id` клиента OAuth
Да	`aud`	`String`	Audience. Предполагаемый потребитель токена, как правило, это сервер ресурсов (API), к которому приложению необходимо получить доступ. Сервер авторизации должен подтвердить, что он является целевой аудиторией для токена. Значение должно представлять собой URL конечной точки сервера авторизации
Да	`jti`	`String`	JWT ID. Уникальный идентификатор токена, который может быть использован для предотвращения повторного использования токена. Такие маркеры должны использоваться только один раз, если только условия повторного использования не были оговорены между сторонами; любые такие переговоры выходят за рамки данной спецификации
Да	`exp`	`String`	Время истечения срока действия токена, по истечении которого JWT не должен приниматься к обработке

2.3.2. Подпись токена авторизации

JWT должен быть подписан цифровой подписью с использованием закрытого ключа в асимметричной криптографии RS256.

Клиент, использующий метод аутентификации, должен заранее зарегистрировать свой открытый ключ (параметр `key_id`) на сервере авторизации, чтобы сервер мог проверить `assertion`.

Проверить корректность выпущенного токена можно, к примеру, на сайте https://jwt.io/.

Ответ на запрос получения токена приведен в таблице (Таблица 3).

Таблица 3

Обязательность	Параметр	Тип	Описание
Да	`access_token`	`String`	Токен авторизации, с которым необходимо выполнять запросы к серверу
Да	`expires_in`	`String`	Время действия токена авторизации
Да	`scope`	`String`	Список действий, которые могут быть разрешены приложениям
Да	`token_type`	`String`	Тип токена
Да	`expires_at`	`String`	Время окончания действия токена авторизации

Из ответа следует считать атрибуты, которые возвращают токен, и время его действия.

Примеры кода на Python и Java для получения токена доступа приведены в подразделе 2.4.

Примеры кода приведены на портале разработчиков ОС Аврора: https://community.omprussia.ru/documentation/software_development/guides/push/server.html#example.

В случае, если вышел срок действия `access_token`, необходимо инициировать процедуру получения токена (п. 2.3.1).

2.3.3. Оправка push-уведомлений

2.3.3.1. Общая информация

Для отправки push-уведомления необходимо сформировать запрос на Сервер приложений ПСУ, указанный в `api_url` (подраздел 1.2).

```
POST  {api_url}/projects/{project_id}/messages
```

Заголовок `Content-Type` должен иметь значение: `application/json`.

Заголовок `Authorization` должен иметь значение: `Bearer access_token`, полученный в запросе авторизации (п. 2.3.1).

К адресу Сервера приложений ПСУ необходимо добавить фрагмент с указанием `project_id`.

Параметры тела запроса приведены в таблице (Таблица 4).

Таблица 4

Обязательность	Параметр	Тип	Описание
Да	`target`	`String`	Указывается `registrationId`, полученный из приложения после его регистрации на Сервере приложений ПСУ
Да	`type`	`Enum (String)`	Поддерживается значение `device`
Да	`ttl`	`String`	Время жизни push-уведомления, например, 5h30m. Может быть задано только в следующих единицах: `s` — секунды, `h` — часы, `m` — минуты
Да	`notification`	`OBJ Notification`	Содержание push-уведомления

Параметры объекта `Notification` приведены в таблице (Таблица 5).

Таблица 5

Обязательность	Параметр	Тип	Описание
Нет	`title`	`string`	Заголовок (до 512 символов)
Нет	`message`	`string`	Тело сообщения (до 2048 символов)

Обязательность	Параметр	Тип	Описание
Нет	`data`	`Object`	Объект для параметров ключ-значение, можно передавать любые необходимые приложению поля (до 1024 байт)
Нет	`action`	`string`	Действие, которое будет выполнено в приложении, после нажатия пользователем на уведомление. Действия задаются разработчиком приложений. Например, если у приложения есть D-Bus сервис, в котором определен метод «Open», тогда «Open» можно передать в поле и метод будет вызван системой при клике на уведомление. Поле должно содержать не более 255 символов. Метод должен быть 1 на поле и метод не должен содержать параметры. Для того, чтобы была возможность вызвать `action`, push-уведомление должно отобразиться в системном трее

Возможные коды ошибок приведены в таблице (Таблица 6).

Таблица 6

Код и статус ошибки	Описание	Действия для устранения ошибки
`400 Bad Request`	`ttl limit is exceeded`	Необходимо проверить допустимое время жизни push-уведомления в соответствии с настоящим документом
	`unsupported message type`	Необходимо проверить доступные типы доставки уведомлений в соответствии с настоящим документом
	`invalid notification title length`	Необходимо проверить доступные ограничения на создание уведомлений в соответствии с настоящим документом

Код и статус ошибки	Описание	Действия для устранения ошибки
	`invalid notification message length`	Необходимо проверить доступные ограничения на создание уведомлений в соответствии с настоящим документом
	`invalid target`	Необходимо проверить доступные значения в соответствии с настоящим документом
	`target not found` (указанный `registration_id` не найден)	`registration_id` необходимо получить после регистрации устройства и приложения на Сервере приложений ПСУ
`401 Unauthorized`	Истек срок действия токена авторизации	Необходимо получить новый токен авторизации (п. 2.3.1). Дополнительно см. раздел 4
	`Client authentication failed, the provided client JSON Web key is expired` (не удалось выполнить аутентификацию. Срок действия предоставленного ключа истек)	Необходимо обратиться к Администратору Сервиса уведомлений с просьбой выпустить новый ключ безопасности проекта и прислать его, после чего следует установить новый ключ и повторить запрос
`403 Forbidden`	Отсутствуют необходимые права доступа для выполнения запроса	Необходимо обратиться к системному администратору для назначения прав
`413 Request Entity Too Large`	Общий размер JSON-объекта в теле сообщения превышает 4 кб	Необходимо уменьшить размер JSON-объекта в теле сообщения
`429 Too Many Requests`	Достигнут лимит количества запросов в секунду	Необходимо сократить количество запросов в секунду для проекта
`500 Internal Server Error`	Внутренняя системная ошибка Сервера приложений ПСУ	Необходимо обратиться к системному администратору. Дополнительно см. раздел 4
`504 Gateway Timeout`	Инфраструктура Сервиса уведомлений недоступна	Необходимо обратиться к системному администратору

Пример запроса:

```
POST https://global-push.domain/api/projects/project-from-ui-
bthsc2iq44tso869omt0/messages
Content-Type: application/json
{
    "target": "442125d4-6041-4f72-ab4b-1e5fd3ad389a",
    "ttl": "2h",
    "type": "device",
    "notification": {
        "title": "some title",
        "message": "some message",
        "data": {
            "another_key": "value"
        },

"action": "command"
    }
}
```

Пример ответа в случае успеха имеет следующий вид:

```
{
    "expiredAt": "2020-10-06T07:44:43.967576Z",
    "id": "1526c09c-1ca4-48ea-8357-e1b8aa2f3fc3",
    "notification": {
        "data": {
            "another_key": "value"
        },
        "message": "some message",
        "title": "some title",

"action": "command"
    },
    "status": "",
    "string": "", "target": "442125d4-6041-4f72-ab4b-1e5fd3ad389a",
    "type": "device"
}
```

2.3.3.2. Создание push-уведомления для ОС версии 3.2.2 и выше

Push-уведомление с данными в полях `data`, `title` и/или `message` будет доставляться напрямую в приложение, минуя отображение в системном трее устройства. Если приложение не было запущено, оно будет запущено в фоне.

2.3.3.3. Создание push-уведомления для ОС от версии 5.0 и выше

ПРИМЕЧАНИЕ. В случае, если приложение не было адаптировано для применения в ОС версии 5.0, то необходимо обращаться к следующей информации:

– логика создания push-уведомлений (пп. 2.3.3.2);

– описание адаптации приложения (п. 2.2.3).

Push-уведомление с данными в полях `title` и/или `message` будет отображаться в системном трее устройства. Исключением является случай, когда приложение запущено и развернуто, тогда все данные попадут в приложение и push-уведомление не будет отображаться в системном трее. Если приложение не было запущено в момент получения устройством push-уведомления, тогда приложение не будет запускаться, а push-уведомление будет отображаться в системном трее.

Push-уведомление с данными в поле `data` будет доставляться напрямую в приложение, минуя отображение в системном трее устройства. Если приложение не было запущено, оно будет запущено в фоне.

Push-уведомление с данными в полях `data`, `title` и/или `message` будет отображено в системном трее и доставлено в приложение одновременно. Исключением будет случай, когда приложение запущено и развернуто, тогда все данные попадут в приложение и push-уведомление не будет отображаться в системном трее.

2.3.4. Получение информации о проекте push-уведомлений

Сервер приложений ПСУ предоставляет возможность получить следующую информацию:

– о доступности Сервера приложений ПСУ;

– об актуальности ключей безопасности проекта push-уведомлений;

– об активности проекта push-уведомлений.

Необходимо сформировать запрос на Сервер приложений ПСУ, указанный в `api_url` (подраздел 1.2):

```
GET  {api_url}/projects/{project_id}
```

Заголовок `Content-Type` должен иметь значение: `application/json`.

Заголовок `Authorization` должен иметь значение: `Bearer access_token`, полученный в запросе авторизации (п. 2.3.1).

При формировании URL для запроса вместо `{project_id}` необходимо указать идентификатор проекта из конфигурации.

Параметры ответа приведены в таблице (Таблица 7).

Таблица 7

Обязательность	Параметр	Тип	Описание
Да	`id`	`String`	Идентификатор проекта
Да	`name`	`String`	Название проекта
Да	`isActive`	`Boolean`	Активность проекта
Да	`serviceAccount`	`OBJ serviceAccount`	Объект с информацией о сервисном аккаунте проекта
Да	`createdAt`	`String`	Дата создания проекта
Да	`updatedAt`	`String`	Дата обновления проекта

Параметры объекта `serviceAccount` приведены в таблице (Таблица 8).

Таблица 8

Обязательность	Параметр	Тип	Описание
Да	`clientId`	`string`	Идентификатор сервисного аккаунта, совпадает с идентификатором проекта
Да	`clientName`	`string`	Название сервисного аккаунта, совпадает с названием проекта
Да	`publicKeys`	`Object`	Объект с информацией о сроке действия ключей безопасности проекта
Да	`scope`	`string`	Список действий, которые могут быть разрешены приложению
Да	`audience`	`Array[string]`	Предполагаемый потребитель токена, как правило, это сервер ресурсов (API), к которому приложению необходимо получить доступ

Параметры объекта `publicKeys` приведены в таблице (Таблица 9).

Таблица 9

Обязательность	Параметр	Тип	Описание
Да	`meta`	`Object`	Объект с информацией о ключе безопасности проекта. Содержит значение параметров: – `public:{значение}` – идентификатор публичного ключа; – `assigned_at` – дата назначения ключей безопасности проекта; – `expired_at` – дата окончания действия ключей безопасности проекта

Ответ будет представлен в следующем формате:


```
{
   "createdAt":"2023-09-13T21:43:29.712093+03:00",
   "id":"acs_1384_test_ck105k9pfb781l4a8n8g",
   "isActive":true,
   "name":"acs-1384-test",
   "serviceAccount":{
      "audience":[
         "http://example.ru/auth/public",
         "http://example.ru/push/public"
      ],
      "clientId":"acs_1384_test_ck105k9pfb781l4a8n8g",
      "clientName":"acs-1384-test",
      "publicKeys":{
         "meta":{
            "public:WcKEcj7wQQ":{
               "assigned_at":"2023-09-13T18:43:58Z",
               "expired_at":"2024-12-13T00:43:58Z"
            }
         }
      },
      "scope":"openid offline message:update project:read"
   },
   "updatedAt":"2023-09-13T21:44:43.73058+03:00"
}
```

Возможные коды ошибок приведены в таблице (Таблица 10).

Таблица 10

Код и статус ошибки	Описание	Действия для устранения ошибки
`401 Unauthorized`	Истек срок действия токена авторизации	Необходимо получить новый токен авторизации (п. 2.3.1). Дополнительно см. раздел 4
	`Client authentication failed, the provided client JSON Web key is expired` (не удалось выполнить аутентификацию. Срок действия предоставленного ключа истек)	Необходимо обратиться к Администратору Сервиса уведомлений с просьбой выпустить новый ключ безопасности проекта и прислать его, после чего следует установить новый ключ и повторить запрос
`403 Forbidden`	Доступ к ресурсу запрещен	Проверить корректность настроек переданного проекта. В случае повторения ошибки обратиться к Администратору Сервиса уведомлений
`404 Not Found`	Ресурс не найден	Проверить корректность настроек переданного проекта. В случае повторения ошибки обратиться к Администратору Сервиса уведомлений
`429 Too Many Requests`	Достигнут лимит количества запросов в секунду	Необходимо сократить количество запросов в секунду для проекта
`500 Internal Server Error`	Внутренняя системная ошибка Сервера приложений ПСУ	Необходимо обратиться к системному администратору. Дополнительно см. раздел 4
`504 Gateway Timeout`	Инфраструктура Сервиса уведомлений недоступна	Необходимо обратиться к системному администратору

2.3.5. Рекомендации использования запроса

Периодически, с частотой не менее одного раза в сутки, с сервера внешнего приложения необходимо выполнять запрос на Сервер приложений ПСУ, чтобы получать актуальные данные по проекту:

– атрибут `isActive` должен быть равен `true`. В случае нарушений связаться с Администратором Сервиса уведомлений;

– значение `expired_at` для используемого ключа должно быть с действующей датой. Рекомендуется получить новый ключ безопасности проекта, не дожидаясь даты окончания действия текущего. Процесс автоматической смены ключа безопасности проекта приведен в п. 2.3.6;

– эндпоинт должен возвращать статус `200 OK` – это значит, что Сервер приложений ПСУ физически доступен для отправки сообщений из сервера внешнего приложения. В случае нарушений связаться с Администратором Сервиса уведомлений.

2.3.6. Автоматическая смена ключа безопасности проекта

В случае, если при выполнении запроса на получение информации о проекте (п. 2.3.4), будет замечено, что срок действия ключа истекает, необходимо заблаговременно выполнить следующие действия:

– сгенерировать новую пару, состоящую из открытого и закрытого ключа;

– установить публичный ключ для аккаунта своего проекта;

– обновить конфигурацию своего сервера внешнего приложения для использования нового ключа при получении авторизационного токена.

Для генерации пары ключей необходимо выполнить запрос ниже:

```
POST  {api_url}/keyPairs
```

Заголовок `Content-Type` должен иметь значение: `application/json`.

Параметры тела запроса приведены в таблице (Таблица 11).

Таблица 11

Обязательность	Параметр	Тип	Описание
Да	`alg`	`String`	Алгоритм для подписи токена. Должно быть указано значение `RS256`.
Да	`kid`	`String`	Идентификатор ключа. Строка длиной 10 символов, содержащая цифры и буквы латинского алфавита в нижнем и верхнем регистрах
Да	`use`	`String`	Целевое назначение алгоритма шифрования. Должно быть указано значение `sig`

Заголовок `Authorization` должен иметь значение: `Bearer access_token`, полученный в запросе авторизации (п. 2.3.1).

Коды ошибок аналогичны тем, что приведены в таблице (см. Таблица 6).

Для случая успеха параметры ответа приведены в таблице (Таблица 12).

Таблица 12

Обязательность	Параметр	Тип	Описание
Да	`private`	`OBJ private`	Объект с информацией о закрытом ключе
Да	`public`	`OBJ public`	Объект с информацией об открытом ключе

Параметры объекта `private` приведены в таблице (Таблица 13).

Таблица 13

Обязательность	Параметр	Тип	Описание
Да	`jwk`	`OBJ privateJWK`	Объект с описанием закрытого ключа
Да	`pem`	`String`	Строка с закрытым ключом в формате `.pem`

Параметры объекта `privateJWK` приведены в таблице (Таблица 14). Детальная информация доступна в RFC 7518.

Таблица 14

Обязательность	Параметр	Тип	Описание
Да	`alg`	`String`	Криптографический алгоритм, в котором предполагается использование ключа
Да	`d`	`String`	Экспонента RSA, участвующая в формировании закрытого ключа. Для подключения к ПСУ не используется
Да	`e`	`String`	Публичная экспонента RSA. Для подключения к ПСУ не используется
Да	`kid`	`String`	Идентификатор ключа. Строка длиной 10 символов, содержащая цифры и буквы латинского алфавита в нижнем и верхнем регистрах
Да	`kty`	`String`	Семейство алгоритмов шифрования, используемое при генерации ключа. Значение будет равно RSA
Да	`n`	`String`	Модуль RSA. Для подключения к ПСУ не используется
Да	`p`	`String`	Первый простой множитель. Для подключения к ПСУ не используется
Да	`q`	`String`	Второй простой множитель. Для подключения к ПСУ не используется
Да	`use`	`String`	Целевое назначение алгоритма шифрования. Должно быть указано значение `sig`

Параметры объекта `public` приведены в таблице (Таблица 15).

Таблица 15

Обязательность	Параметр	Тип	Описание
Да	`jwk`	`OBJ publicJWK`	Объект с описанием открытого ключа

Параметры объекта `publicJWK` приведены в таблице (Таблица 16). Детали доступны в RFC 7518.

Таблица 16

Обязательность	Параметр	Тип	Описание
Да	`alg`	`String`	Криптографический алгоритм, в котором предполагается использование ключа
Да	`e`	`String`	Публичная экспонента RSA. Для подключения к ПСУ не используется
Да	`kid`	`String`	Идентификатор ключа
Да	`kty`	`String`	Семейство алгоритмов шифрования, используемое при генерации ключа. Значение будет равно RSA
Да	`n`	`String`	Модуль RSA. Для подключения к ПСУ не используется
Да	`use`	`String`	Целевое назначение алгоритма шифрования. Должно быть указано значение `sig`

Пример ответа представлен ниже:

```
{
   "private":{
      "jwk":{
         "alg":"RS256",
         "d":"HD ..... PAHPkp_V6byeaf-G0", // сокращено для
отображения
         "e":"AQAB",
         "kid":"private:tt9jjwRJxx",
         "kty":"RSA",
         "n":"39a ....... LRs9J_ROgPUk", // сокращено для отображения
         "p":"8BU ....... fZW-RclDaPw", // сокращено для отображения
         "q":"7q3 ....... PdGvp62dw", // сокращено для отображения
         "use":"sig"
      },
    "pem":"-----BEGIN RSA PRIVATE KEY-----\nMIIJQ ...... jKqthG\n-----
END RSA PRIVATE KEY-----\n" // сокращено для отображения
   },
   "public":{
      "jwk":{
         "alg":"RS256",
         "e":"AQAB",
         "kid":"public:tt9jjwRJxx",
         "kty":"RSA",
         "n":"39a ...... BLRs9J_RogPUk", // сокращено для отображения
         "use":"sig"
      },
      "pem":""
   }
}
```

После генерации пары ключей необходимо применить публичный ключ для аккаунта своего проекта. Для этого необходимо сделать следующий запрос:

```
PUT  {api_url}/projects/{project_id}/serviceAccounts/{client_id}/pub-
licKeys
```

Параметры `{project_id}` и `{client_id}` подставляются из конфигурационного файла проекта, полученного от владельца ПСУ.

Заголовок `Content-Type` должен иметь значение: `application/json`.

Параметры тела запроса приведены в таблице (Таблица 17).

Таблица 17

Обязательность	Параметр	Тип	Описание
Да	keys	Array of OBJ publicJWK	Массив с данными об открытом ключе, описанный в таблице (см. Таблица 16)

В тело запроса в массив `keys` должен быть добавлен объект `public.jwk` из ответа на запрос по генерации ключей.

Пример тела запроса:

```
{
   "keys":[
      {
         "alg":"RS256",
         "e":"AQAB",
         "kid":"public:tt9jjwRJxx",
         "kty":"RSA",
         "n":"39a ......... Rs9J_RogPUk", // сокращено для отображения
         "use":"sig"
      }
   ]
}
```

Заголовок `Authorization` должен иметь значение: `Bearer access_token`, полученный в запросе авторизации (п. 2.3.1).

Коды ошибок в ответе аналогичны тем, что приведены в таблице (см. Таблица 6).

В случае успеха пример ответа может выглядеть так:

```
{
   "audience":[
      "https://ocs-int.ompcloud.ru/auth/public",
      "https://ocs-int.ompcloud.ru/push/public"
   ],
   "clientId":"kushnarev_test_cla7thirvkjfc4j9g1e0",
   "clientName":"kushnarev-test",
   "scope":"openid offline message:update project:read keyPairs:create
serviceAccount:update"
}
```

На заключительном этапе необходимо обновить конфигурацию сервера внешнего приложения (и всех узлов кластера при наличии) для использования нового ключа при получении авторизационного токена.

При этом полученные ранее на основании старого ключа токены могут быть использованы до истечения срока их действия.

2.4. Пример кода сервера внешнего приложения

2.4.1. Код на языке Python

Тестовый пример на языке программирования Python демонстрирует все описанные ранее шаги. Пример полностью доступен на ресурсе https://hub.mos.ru/auroraos/tools/PushSenderPython.

Программа считывает настройки проекта из YAML-файла, указанного в параметре `config`, делает запрос на получение токена доступа и в зависимости от параметра `action` выполняет ту или иную логику, доступную через API.

Детали использования утилиты описаны в сопровождающем код README.md файле.

2.4.2. Код на языке Java

Тестовый пример на Java демонстрирует ту же функциональность и в полном виде доступен на ресурсе https://hub.mos.ru/auroraos/tools/PushSenderJava.

Детали использования утилиты описаны в сопровождающем код README.md файле.

Назначение классов в примере сервера внешнего приложения:

– классы `PushNotificationSystem` и `Config` в пакете `ru.omp.push.example.config` отвечают за чтение и хранение настроек клиента приложения и Сервера приложений ПСУ;

– класс `ru.omp.push.example.auth.TokenFetcher` — запрос и получение токена от Сервера приложений ПСУ;

– класс `TestPushClient` — стартовая точка приложения, загрузка настроек из файла, получение токена с помощью `TokenFetcher`, проверка токена, отправка push-уведомления, получение информации о проекте и обновление ключей.

Необходимо обратить внимание на следующее:

– перед тем, как делать запрос на отправку push-уведомлений, требуется проверить токен на актуальность (см. метод `validateToken` класса `TokenFetcher`);

– если запрос на отправку push-уведомления вернул HTTP code 401, токен невалидный. Требуется обновить его через метод `authenticate` класса `TokenFetcher` и произвести попытку отправить push-уведомление (такое поведение может зависеть от стратегии сервера внешнего приложения).

3. ОГРАНИЧЕНИЯ

В ОС начиная с версии 4.0 был расширен протокол взаимодействия Сервера приложений ПСУ и push-демона.

Общая логика взаимодействия осталась прежней, однако изменение протокола произошло ввиду изменения взаимодействия push-демона с компонентами управления приложениями в ОС Аврора.

Для версии 1 push-демона было необходимо запускать GUI, а в версии 2 push-демона запуск GUI не требуется ввиду наличия используемого ключа `nogui`.

Соответственно, для обработки push-уведомлений без запуска GUI необходимо реализовать поддержку соответствующего режима в самом приложении.

ПРИМЕЧАНИЕ. Дополнительные ограничения приведены в документе «Руководство администратора».

4. ДИАГНОСТИКА И УСТРАНЕНИЕ НЕПОЛАДОК

В случае возникновения ошибок 401, 500, а также при отсутствии явных ошибок и одновременном отсутствии push-уведомлений рекомендуется выполнить подпункты, описанные ниже.

4.1. Проверка известных проблем

Для проверки известных проблем:

– используется корректный проект и приложение, а также их настройки (см.: 1.1, 1.2, 2.2.1, а также документ «Руководство пользователя. Часть 5. Подсистема Сервис уведомлений» подраздел «Регистрация проекта»;

– push-уведомление отправляется на актуальный `registration ID` (см.: 1.1, 2.2.1, 2.2.2.3, 2.3.3, Таблица 6 – код и статус ошибки «target not found»);

– push-уведомления в проекте включены (см.: 2.3.4, 2.3.5, атрибут `isActive`, а также документ «Руководство пользователя. Часть 5. Подсистема Сервис уведомлений» подраздел «Карточка проекта»;

– время серверов, на которых развернут push-сервер, синхронизировано со временем серверов, формирующих push-уведомления;

– используется действующий токен доступа (см.: 2.3.1, 2.3.3, Таблица 6 – код и статус ошибки «401 Unauthorized»);

– сервер, формирующий push-уведомления, использует актуальные ключи безопасности своего проекта (см.: 2.3.4 – 2.3.6, а также документ «Руководство пользователя. Часть 5. Подсистема Сервис уведомлений» подраздел «Настройки проекта»);

– доступны push-порт для устройства и порт 80/443 для сервера, формирующего push-уведомления (проверить, например, утилитой `telnet`);

– версия ОС Аврора на устройстве соответствует той, под которую адаптировано приложение. Логика доставки push-уведомлений различается для ОС до 5.0.0 и ОС 5.0.0 и выше (см.:2.3.3.2, 2.3.3.3; раздел 3).

4.2. Проверка счетчика принятых push-уведомлений

Если доступна карточка проекта, в блоке «Статистика» отображается поле «Общее количество push-уведомлений» за выбранный период (по умолчанию — текущий месяц). Если созданное push-уведомление было принято сервером, после обновления страницы значение счетчика увеличивается на единицу. Счетчик отражает только факт создания и приема сообщений сервером и не связан с доставкой push-уведомлений на устройство (см. документ «Руководство пользователя. Часть 5. Подсистема Сервис уведомлений» подраздел «Раздел «Статистика»).

4.3. Диагностика при отсутствии известных проблем

Если перечисленные выше известные проблемы отсутствуют, требуется дальнейший анализ и поиск проблем на стороне push-сервера и устройства, на котором возникают проблемы с получением push-уведомлений.

Для диагностики на стороне push-сервера требуется доступ к серверам и логам сервисов, поэтому необходимо обратиться к системному администратору (см.: Таблица 6 – код и статус ошибки «500 Internal Server Error»; Таблица 10, а также документ «Руководство пользователя. Часть 5. Подсистема Сервис уведомлений» раздел «Сообщения об ошибках»).

На первом этапе рекомендуется выполнить следующие проверки:

1) Проверить статус сервисов push-сервера. Для этого требуется на сервере выполнить скрипт `consul_service_health_check.sh`.

2) Проверить отсутствие проблем с созданием партиций в БД. При наличии проблем в логах присутствуют сообщения вида:

```
ERROR: no partition of relation "notifications" found for row
DETAIL: Partition key of the failing row contains (created_at) =
(2025-05-13 05:26:44.94745+00)
```

Если указанные проверки не выявили причину, необходимо собрать логи с push-сервера и с проблемного устройства за период возникновения проблемы, чтобы передать в службу технической поддержки для дальнейшего анализа.

4.4. Локализация проблемы, если используемый push-сервер развернут в составе Аврора Центр

В случае, когда используемый push-сервер развернут в составе ППО и его интеграция с ППО работает корректно, но при этом проблемы с доставкой push-уведомлений возникают для сторонних приложений, тогда требуется обратиться к разработчику стороннего приложения. Дальнейшую диагностику имеет смысл сосредоточить на приложении, формирующем push-уведомления, и на устройстве-получателе.

ПЕРЕЧЕНЬ ТЕРМИНОВ И СОКРАЩЕНИЙ

Используемые в настоящем документе термины и сокращения приведены в таблице (Таблица 18).

Таблица 18

Термин/ Сокращение	Расшифровка
Аврора SDK	Среда разработки Аврора, включающая в себя следующие компоненты: – Аврора IDE; – Аврора Emulator; – Аврора Build Engine
ОС	Операционная система
ППО	Прикладное программное обеспечение «Аврора Центр»
Приложение	Приложением является мобильное приложение, функционирующее под управлением ОС Аврора
Предприятие-разработчик	Общество с ограниченной ответственностью «Открытая мобильная платформа» (ООО «Открытая мобильная платформа»)
ПСУ	Подсистема Сервис уведомлений, реализующая логику управления жизненным циклом проекта push-уведомлений
Сервер внешнего приложения	Внешняя система, по отношению к ПСУ, реализующая бизнес-логику. Использует функционал Сервера приложения Сервиса уведомлений для доставки push-уведомлений на устройства
Сервер приложений ПСУ	Backend подсистемы Сервиса уведомлений предоставляет API для сервера приложений по созданию push-уведомления
Устройство	Под устройством подразумевается мобильное устройство, на котором функционируют соответствующие компоненты ППО
API	Application Programming Interface — программный интерфейс приложения, описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой
D-Bus	Desktop Bus — система межпроцессного взаимодействия, которая позволяет приложениям в операционной системе сообщаться друг с другом

Термин/ Сокращение	Расшифровка
GUI	Graphical User Interface — разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки), представленные пользователю на дисплее, исполнены в виде графических изображений
HTTP	HyperText Transfer Protocol — протокол прикладного уровня передачи данных (изначально в виде гипертекстовых документов). Основой HTTP является технология «клиент-сервер», т.е. предполагается существование потребителей (клиентов), которые инициируют соединение и посылают запрос, и поставщиков (серверов), ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом
IP	Internet Protocol — основной протокол сетевого уровня, использующийся в сети Интернет и обеспечивающий единую схему логической адресации устройств в сети и маршрутизацию данных
JSON	Текстовый формат обмена данными, основанный на JavaScript
JWT	JSON Web Token
Push-уведомления	Текстовые сообщения, предназначенные для оперативной (мгновенной) доставки на устройства пользователей
RSA	Криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел
UI	User Interface – пользовательский интерфейс
URL	Uniform Resource Locator — единообразный локатор (определитель местонахождения) ресурса

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

№ версии документа	Содержание изменения	ФИО	Дата
1.0	Начальная версия	Абрамов А.	07.09.2026 г.
1.1	Внесены изменения	Абрамов А. Пенджиев М.	28.09.2026 г.