ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ АВРОРА SDK

Листов 86

2024

СОДЕРЖАНИЕ

1. Запуск проекта....4
1.1. Создание или открытие проекта....4
1.1.1. Создание нового проекта из шаблона....4
1.1.2. Создание нового проекта из примера....11
1.1.3. Открытие существующего проекта....16
1.2. Сборка проекта....18
1.2.1. Корректное подключение пространств имен Aurora и PushNotifications....20
1.3. Запуск приложения....21
1.4. Тихая установка приложений....25
2. Отладка проекта....26
2.1. Отладка приложений с изоляцией....26
2.2. Пересборка приложения с измененными зависимостями....29
3. Подпись пакета....30
4. Процесс валидации пакета....34
5. Примеры приложений в Аврора SDK....36
5.1. Открытие примера....36
5.2. Обновление примеров....39
6. Инструкция по подключению Мобильного устройства для отладки....45
7. Управление эмуляцией ОС Аврора....52
7.1. Эмуляция камер....52
7.1.1. Управление эмуляцией камер в Аврора IDE....53
7.1.2. Работа с камерой в приложении....54
7.1.3. Команды для проверки работы GStreamer....54
7.2. Эмуляция местоположения....57
7.2.1. Управление эмуляцией местоположения в Аврора IDE....57

7.2.2. Работа с местоположением в приложении 61
7.2.3. D-Bus-сервис для управления эмуляцией местоположения 61
7.3. Эмуляция встроенных датчиков (гироскоп, компас, акселерометр и т.д.) 66
7.3.1. Управление эмуляцией в Аврора IDE 66
7.3.2. Ориентация 72
7.3.3. Работа с датчиками 73
7.3.4. D-Bus-сервис для управления эмуляцией датчиков 73
8. Описание среды сборки и процесс сборки установочных пакетов 81
8.1. Среда сборки 81
8.1.1. Общая информация и структура 81
8.1.2. Авторизация в среде сборки 81
8.2. Сборка установочных пакетов 81
8.2.1. Локальная сборка пакетов 82
8.2.2. Установка недостающих зависимостей 82
8.2.3. Форматы пакетной сборки 83
8.2.4. Создание журнала изменений 84
Перечень терминов и сокращений 85

1. ЗАПУСК ПРОЕКТА

Приложения для операционной системы (ОС) Аврора пишутся на C++/Qt с использованием QML для описания интерфейса пользователя. Создание приложения осуществляется в IDE, основанной на Qt Creator (https://www.qt.io/product/development-tools), и практически совпадает с процессами создания приложений для множества настольных и мобильных платформ. Отличия связаны с тем, что сборка происходит в среде сборки (подраздел 8.1), а запуск — в эмуляторе или на внешнем устройстве с ОС Аврора.

Для того чтобы получить приложение, работающее в эмуляторе или на устройстве, необходимо выполнить три последовательных шага:

1) Создать или открыть проект (подраздел 1.1);

2) Собрать проект (подраздел 1.2);

3) Запустить приложение (подраздел 1.3).

Для сертифицированной версии ОС сборка и запуск требует дополнительных настроек.

1.1. Создание или открытие проекта

Приступить к работе над проектом можно одним из следующих способов:

- создать новый проект из шаблона (п. 1.1.1);
- создать новый проект из примера (п. 1.1.2);
- открыть существующий проект (п. 1.1.3).

1.1.1. Создание нового проекта из шаблона

Данный способ позволяет получить простое приложение с графическим интерфейсом с помощью мастера.

Для создания нового проекта из шаблона необходимо выполнить следующие действия:

1) Запустить Аврора IDE;

2) В основном окне Аврора IDE выбрать пункт меню «Файл» → «Создать файл или проект...» (Рисунок 1);

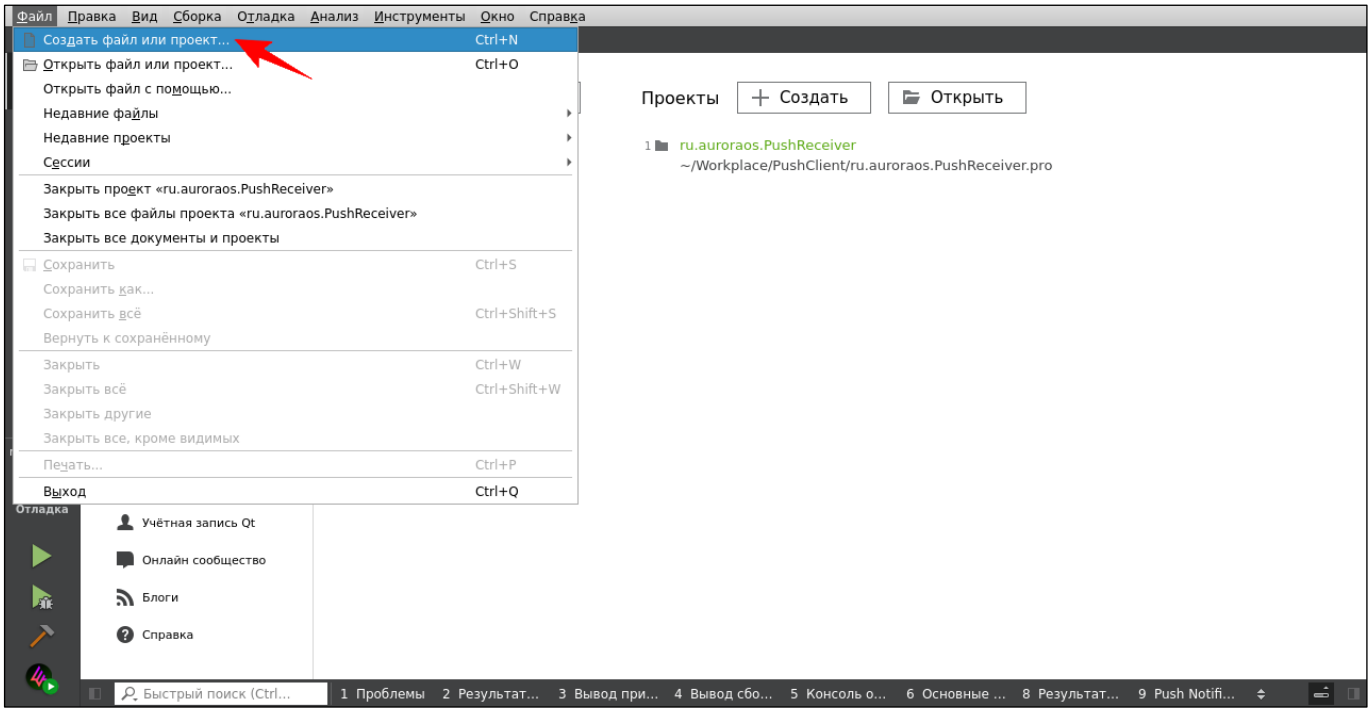


Рисунок 1

3) В открывшемся окне «Новый файл или проект» выбрать вкладку «Проекты» → «Приложение» и отметить «Приложение Qt Quick для ОС Аврора», после чего нажать кнопку «Выбрать...» (Рисунок 2);

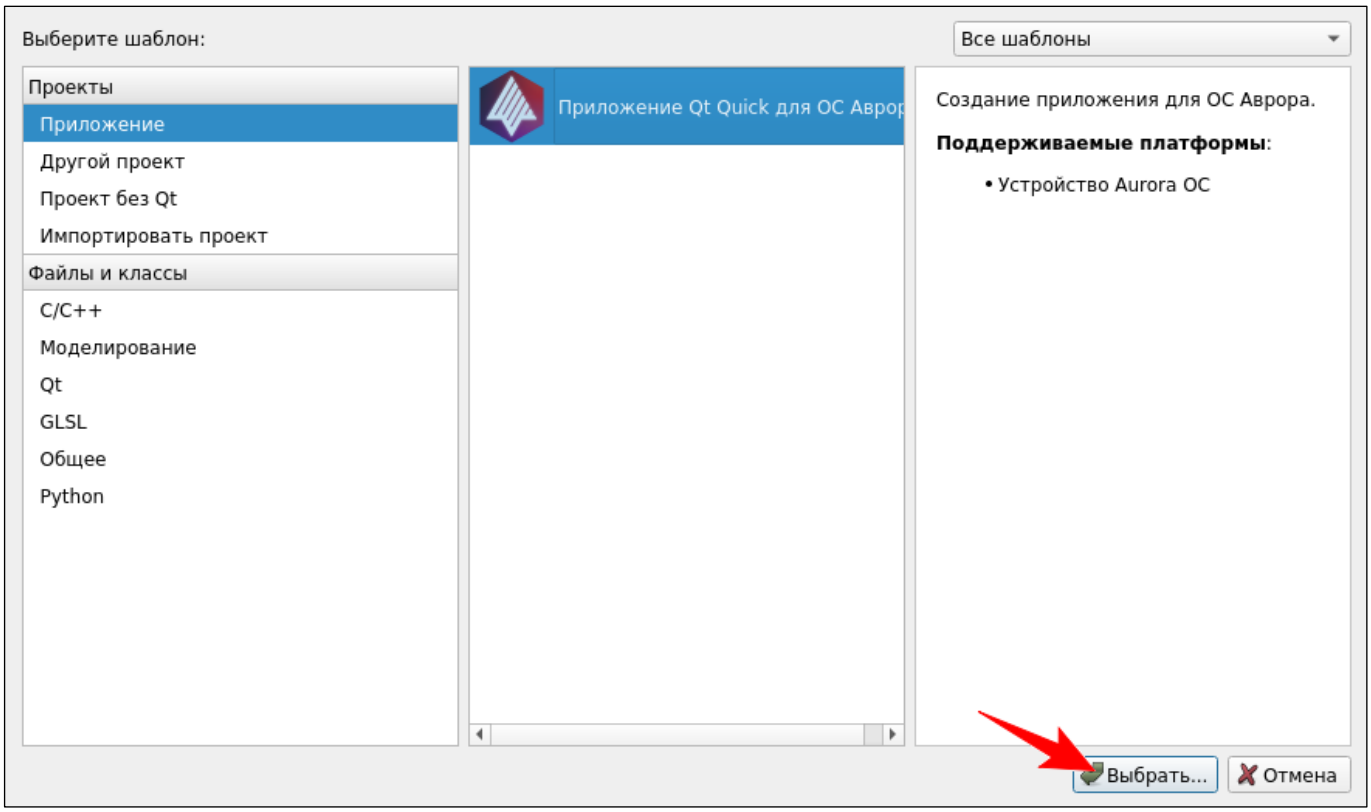


Рисунок 2

4) В появившемся окне «Размещение проекта» в левой части окна перечислены шаги, которые следует пройти для указания параметров проекта. В правой части окна следует указать название проекта, директорию и нажать кнопку «Далее» (Рисунок 3).

ВНИМАНИЕ! Проект должен находиться или в домашней директории пользователя, или в альтернативной директории, указанной при установке Аврора SDK.

Если отметить пункт «Размещение проекта по умолчанию», то указанная директория будет использоваться для следующих создаваемых проектов;

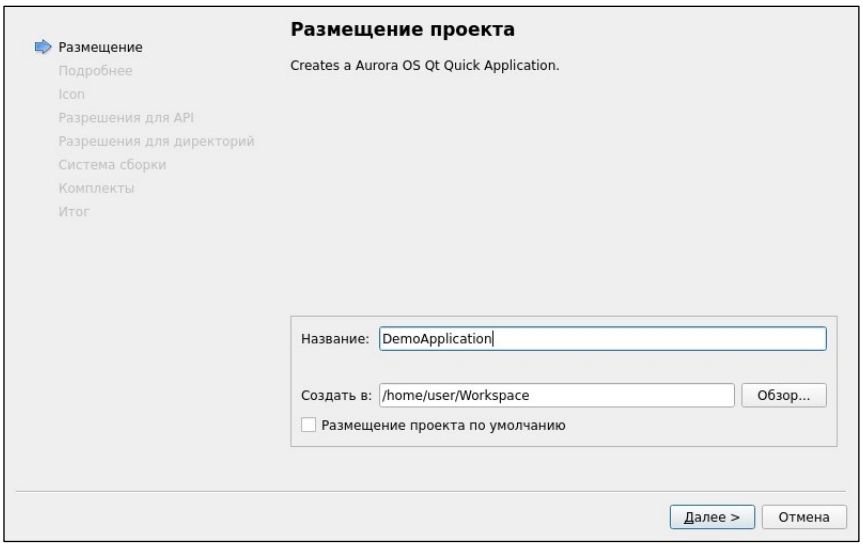


Рисунок 3

5) На следующем шаге «Описание приложения» указать название организации, название приложения латинскими буквами и на русском языке, краткое описание, версию, подробное описание и нажать кнопку «Далее» (Рисунок 4). Параметр «Название приложения» определяет название директории, в которой будет расположен проект;

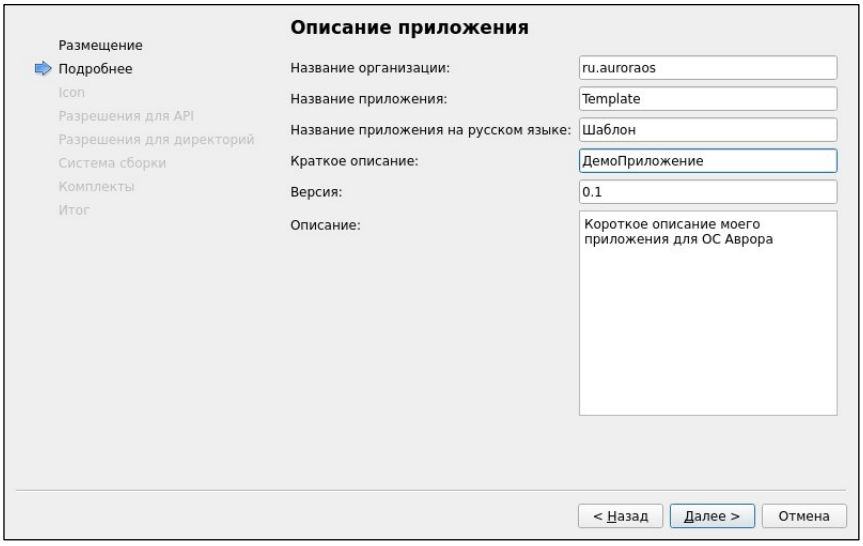


Рисунок 4

6) На следующем шаге «Иконка приложения» есть возможность выбрать основной и дополнительный цвет для иконки будущего приложения (Рисунок 5);

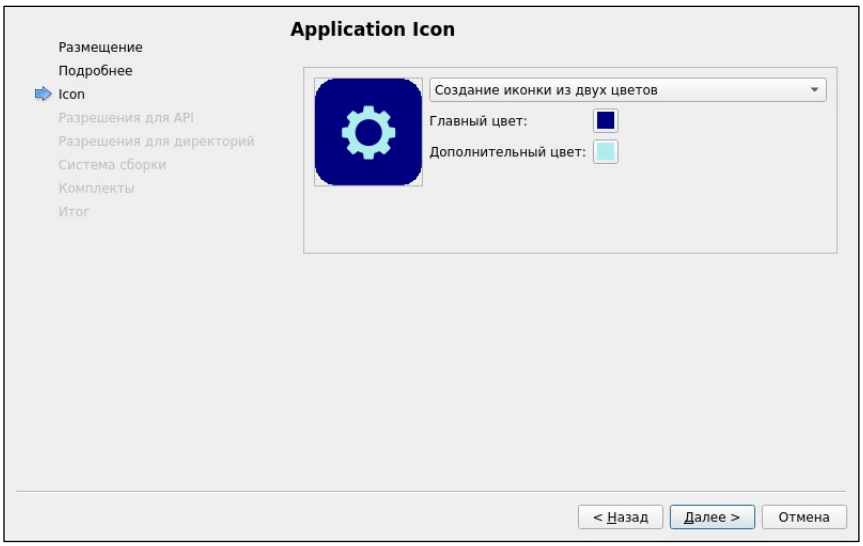


Рисунок 5

7) На следующем шаге «Разрешения для API» следует выбрать разрешения, которые потребуются приложению при работе, и нажать кнопку «Далее» (Рисунок 6). Позже набор требуемых разрешений можно изменить в настройках проекта;

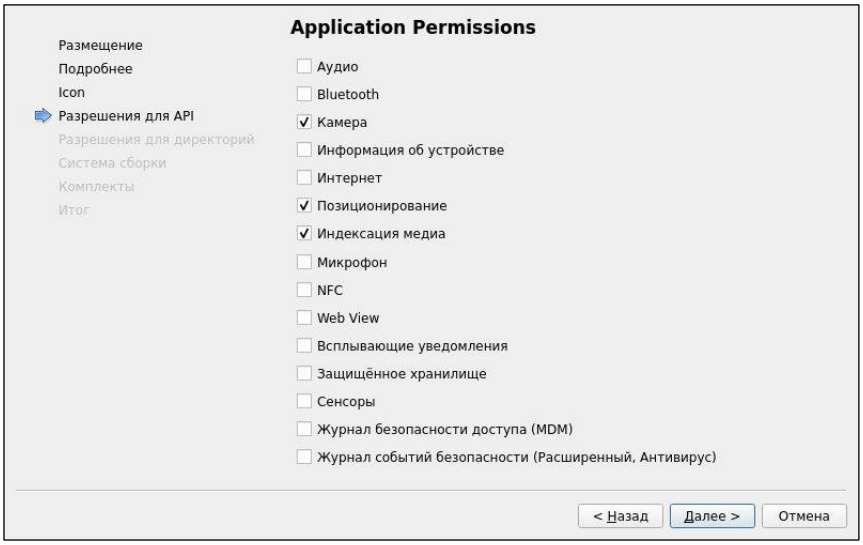


Рисунок 6

8) На следующем шаге «Разрешения для директорий» следует выбрать директории, к которым необходим доступ, и нажать кнопку «Далее» (Рисунок 7). Позже набор требуемых разрешений можно изменить в настройках проекта;

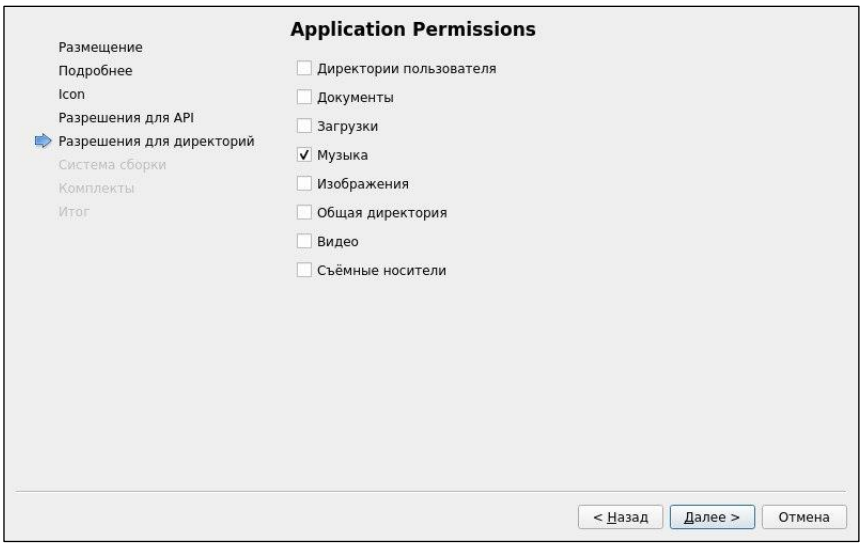


Рисунок 7

9) На следующем шаге «Система сборки» нужно указать систему сборки проекта: qmake или Cmake, и нажать кнопку «Далее» (Рисунок 8). Можно оставить вариант по умолчанию — qmake. Сборка проекта с Cmake аналогична системе сборки qmake;

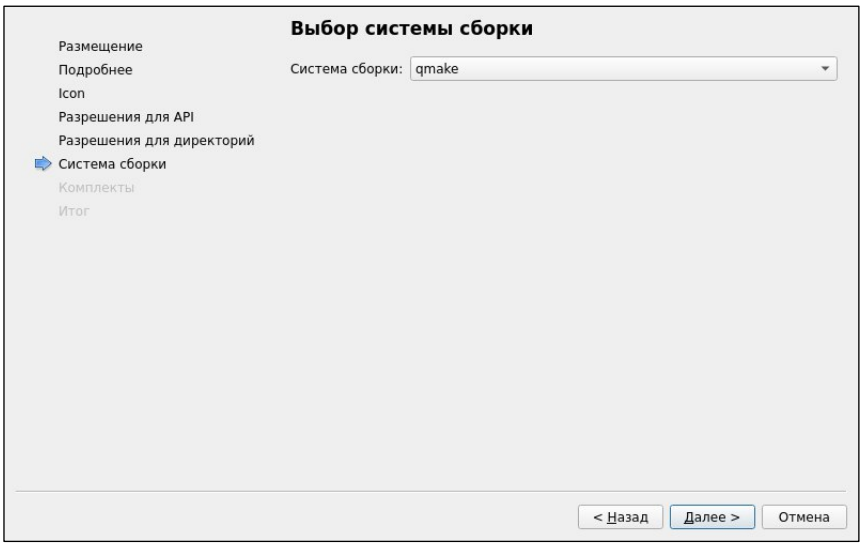


Рисунок 8

10) На следующем шаге «Выбор комплекта» выбрать необходимые комплекты для сборки и нажать кнопку «Далее» (Рисунок 9). Комплект arm7hl используется для мобильных устройств (МУ), i486 — для эмулятора. Позже набор комплектов можно изменить в настройках проекта;

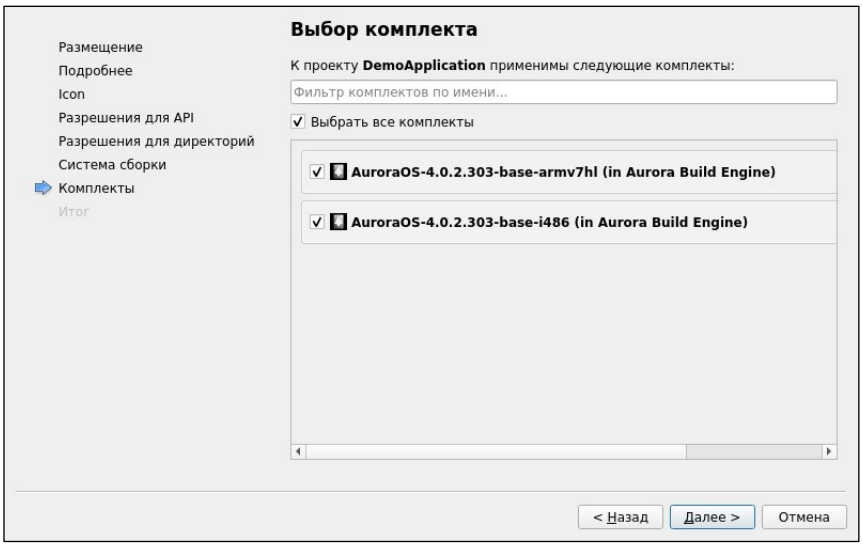


Рисунок 9

11) На последнем шаге «Итог» можно настроить взаимное положение проекта относительно других и подключить одну из систем контроля версий, доступных в ОС. После всех действий по начальной настройке проекта следует нажать кнопку «Завершить» (Рисунок 10);

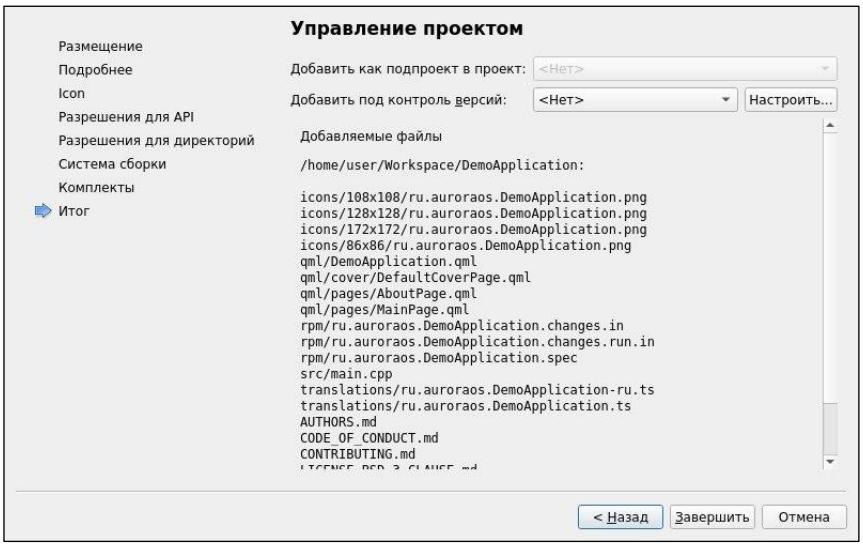


Рисунок 10

12) В открывшемся редакторе исходного кода можно приступить к работе над проектом (Рисунок 11).

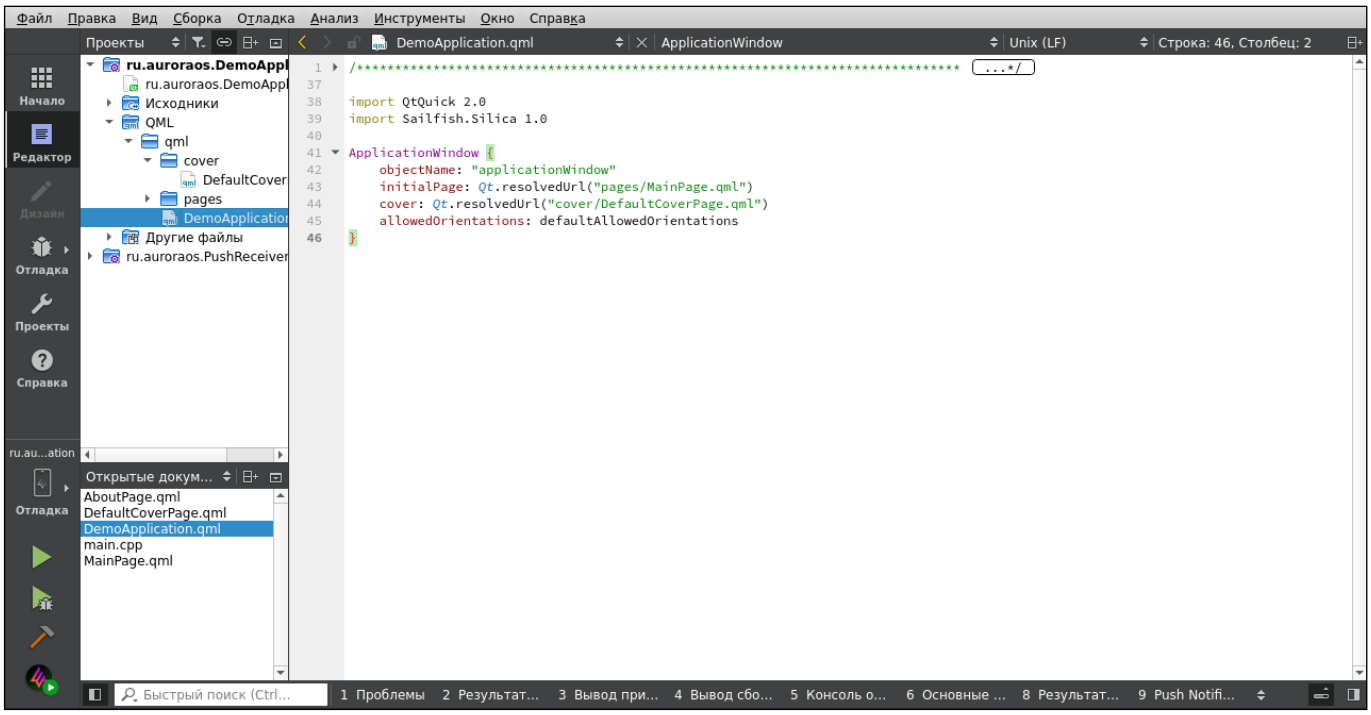


Рисунок 11

1.1.2. Создание нового проекта из примера

Данный способ позволяет создать приложение на базе существующего в Аврора SDK примера. Такой подход удобен для изучения на примерах способов реализации функций, связанных с особенностями разработки под ОС Аврора.

Для создания нового проекта из примера необходимо выполнить следующие действия:

1) Запустить Аврора IDE;

2) В основном окне Аврора IDE выбрать пункт «Начало» и нажать кнопку «Примеры»;

3) В открывшемся окне выбрать из выпадающего списка «Aurora ОС». Далее в галерее доступных примеров приложений выбрать интересующий пример двойным кликом мыши. При наведении курсором мыши на миниатюру примера будет показано его описание (Рисунок 12);

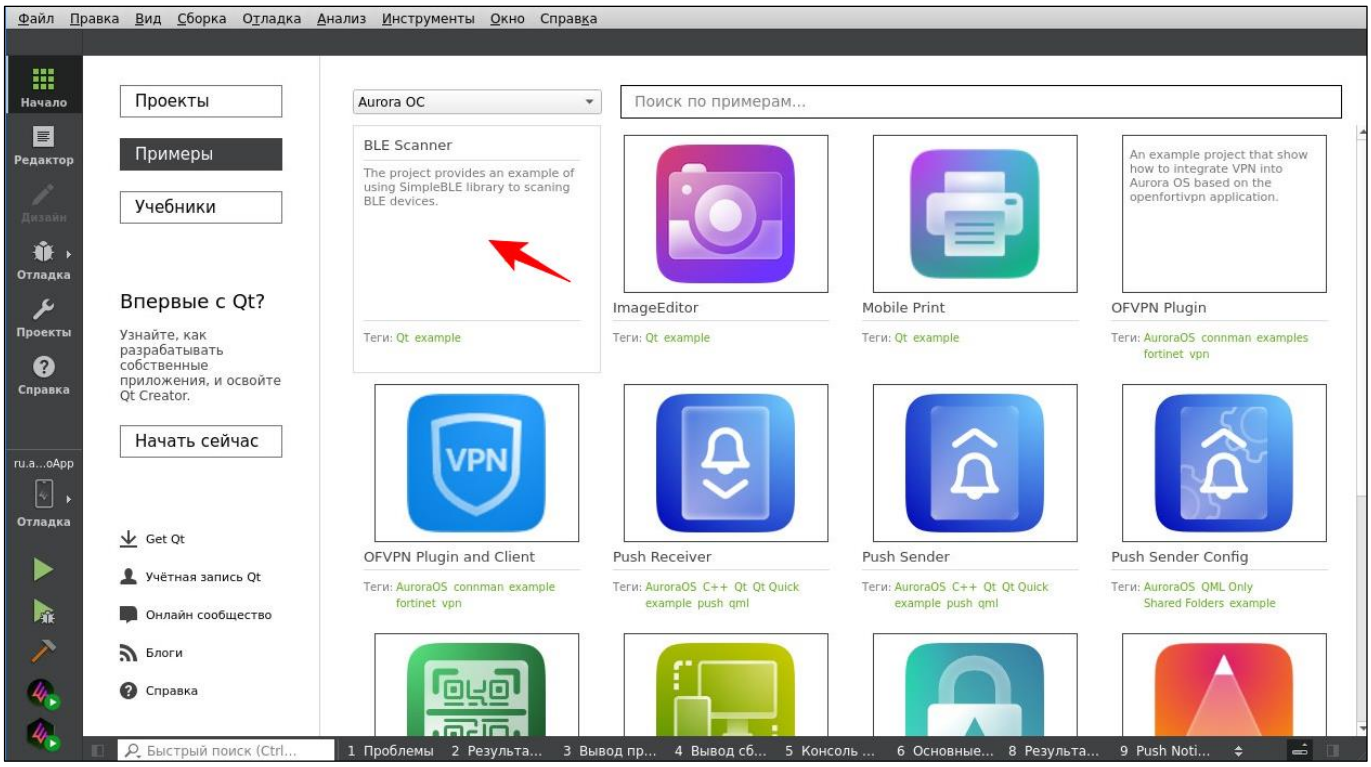


Рисунок 12

4) Во всплывающем окне «Copy Project to writable Location» указать директорию для проекта и нажать кнопку «ОК» (Рисунок 13).

ВНИМАНИЕ! Проект должен находиться или в домашней директории пользователя, или в альтернативной директории, указанной при установке Аврора SDK.

В данную директорию будет скопирована папка с файлами примера, которую можно будет модифицировать;

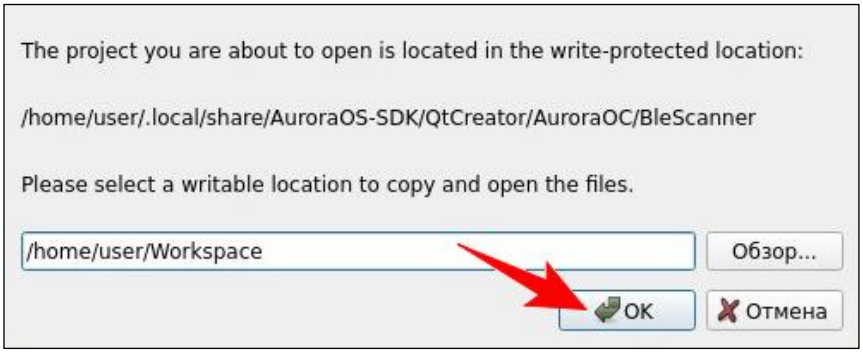


Рисунок 13

5) В следующем окне «Настройки сборки» (Рисунок 14) можно выбрать необходимые параметры для сборки. Комплект arm7hl используется для МУ, i486 — для эмулятора. Позже набор комплектов можно изменить в настройках проекта;

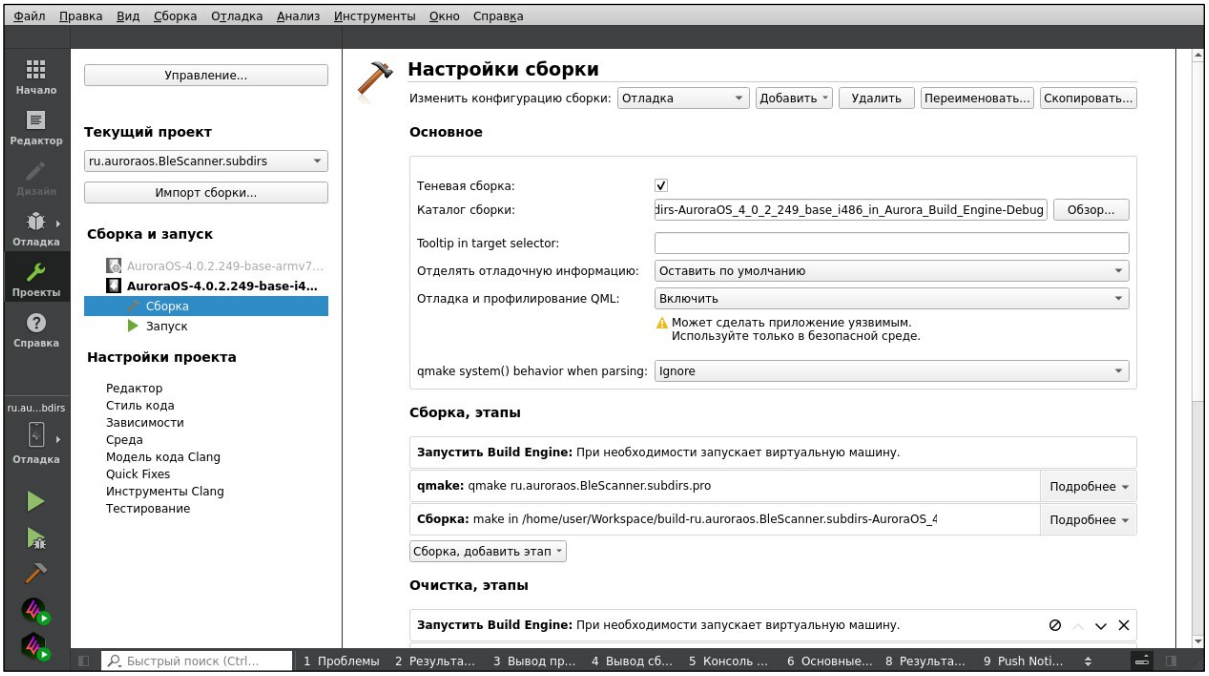


Рисунок 14

6) Далее, используя меню слева, переключиться в «Редактор» (Рисунок 15) и приступить к работе над проектом.

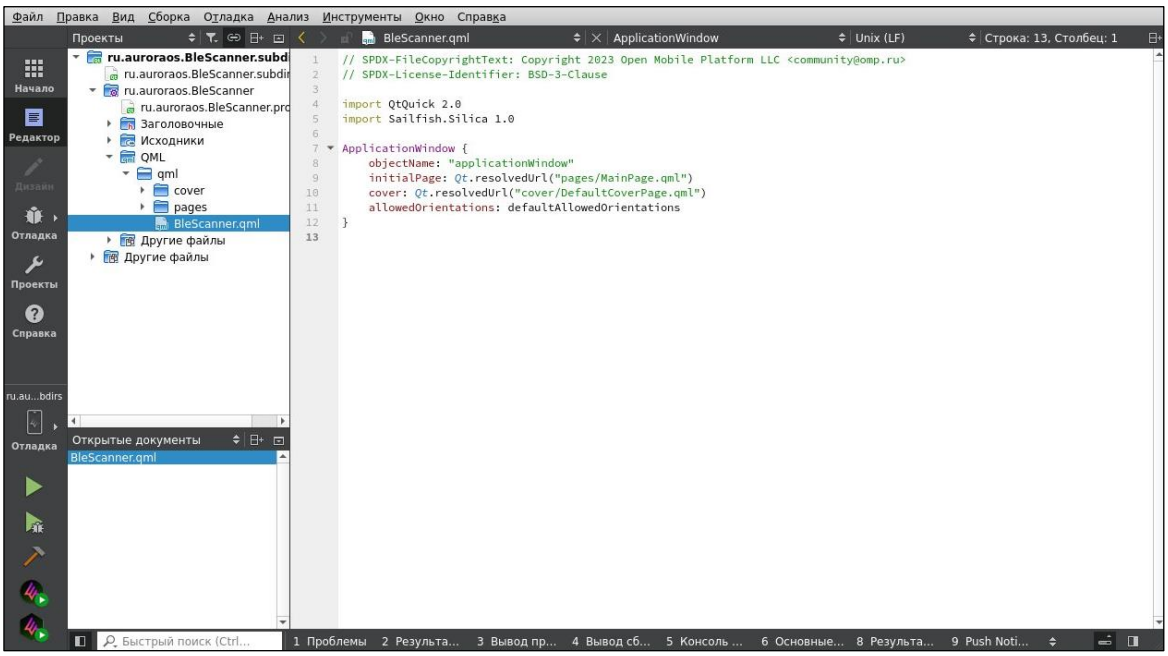


Рисунок 15

Для загрузки дополнительных примеров или обновления текущего списка нужно выполнить следующие шаги:

1) В меню «Инструменты» перейти в раздел «Параметры…» (Рисунок 16);

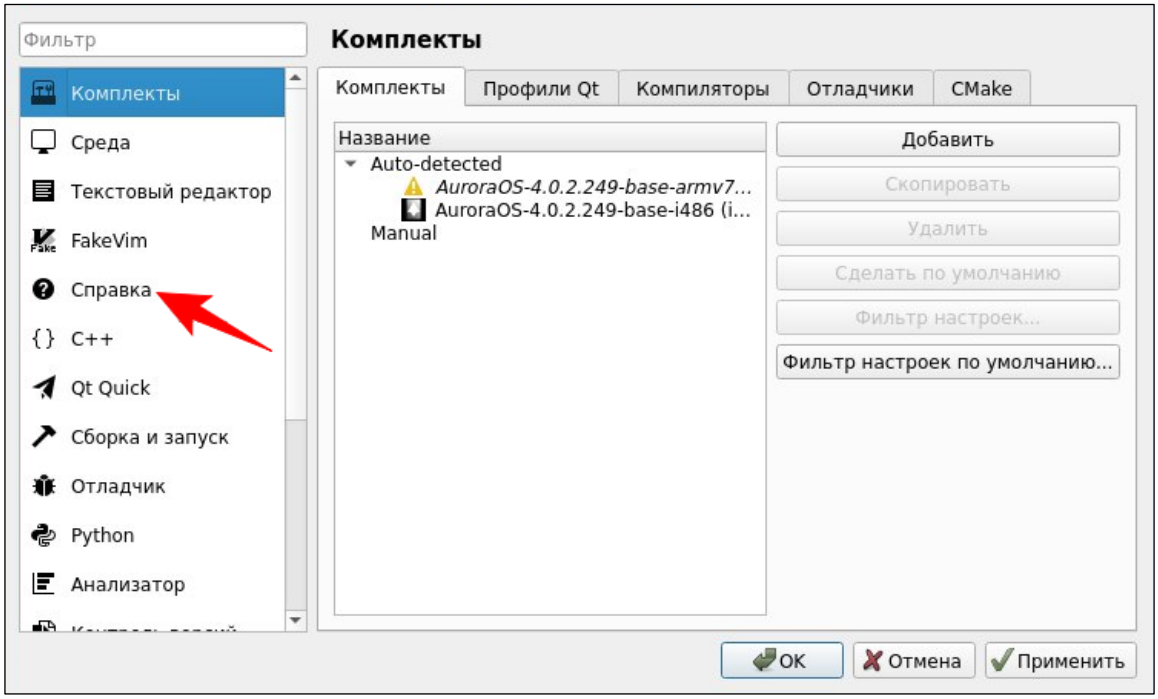


Рисунок 16

2) В окне «Параметры» перейти в раздел «Справка» (см. Рисунок 16) закладка «Aurora ОС примеры» (Рисунок 17);

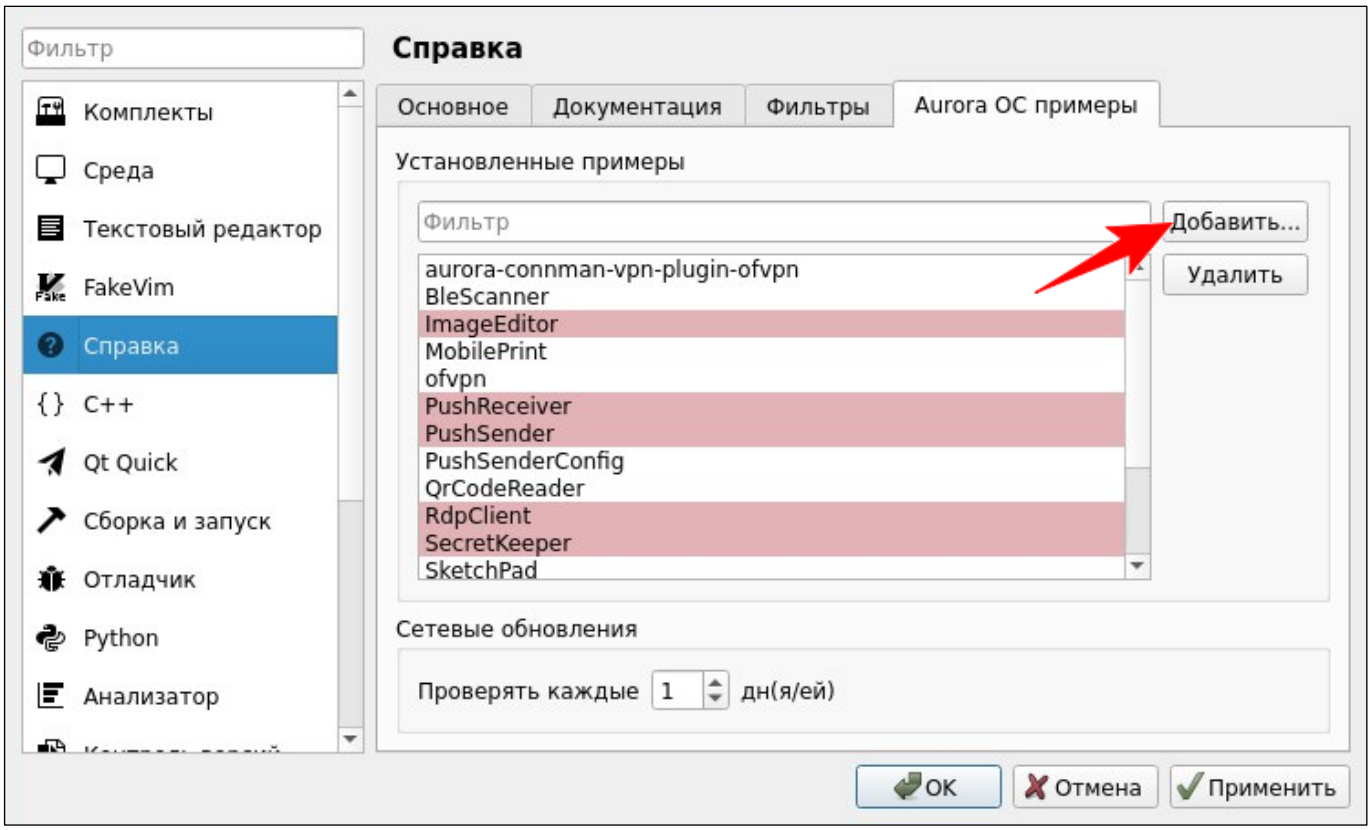


Рисунок 17

3) Нажать кнопку «Добавить» (см. Рисунок 17), чтобы загрузить в систему новые примеры проектов (Рисунок 18).

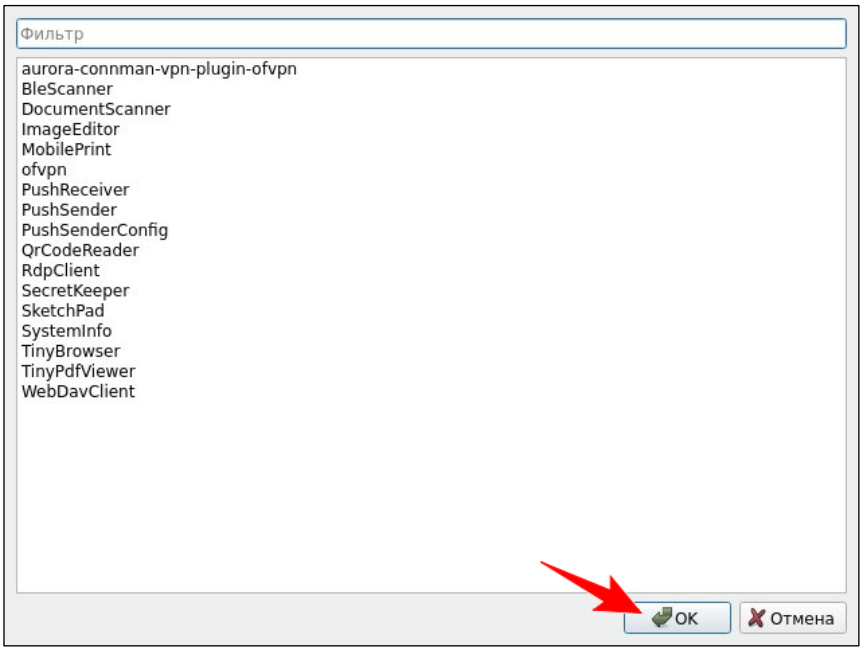


Рисунок 18

1.1.3. Открытие существующего проекта

Для проектов приложений, использующих систему сборки qmake, структура проектов для ОС Аврора определяется файлом `*.pro`. Для открытия существующего проекта необходимо указывать файл с данным расширением.

ВНИМАНИЕ! Проект должен находиться или в домашней директории пользователя, или в альтернативной директории, указанной при установке Аврора SDK.

Если проект используется не впервые, то в той же директории, в которой находится файл с расширением `*.pro`, будет располагаться файл с расширением `*.user` с настройками Аврора SDK для проекта.

Для открытия существующего проекта необходимо выполнить следующие действия:

1) Запустить Аврора IDE;

2) В основном окне Аврора IDE выбрать пункт меню «Файл» → «Открыть файл или проект...»;

3) В появившемся окне выбрать файл с расширением `.pro` и нажать кнопку «Открыть» (Рисунок 19);

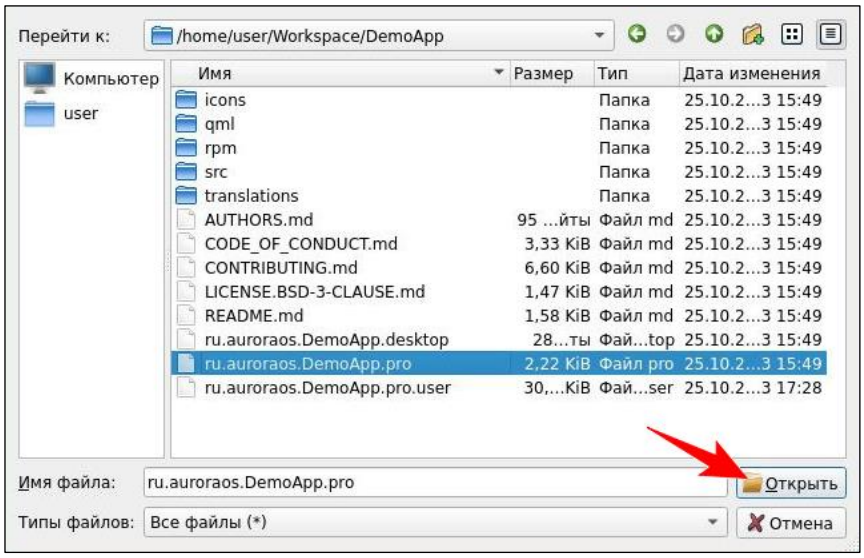


Рисунок 19

4) Если файл с расширением `*.user` отсутствует, или он некорректен, в окне Аврора IDE перейти в режим «Проекты» и выбрать необходимые комплекты для сборки (Рисунок 20). Комплект arm7hl используется для МУ, i486 — для эмулятора. Позже набор комплектов можно изменить в настройках проекта;

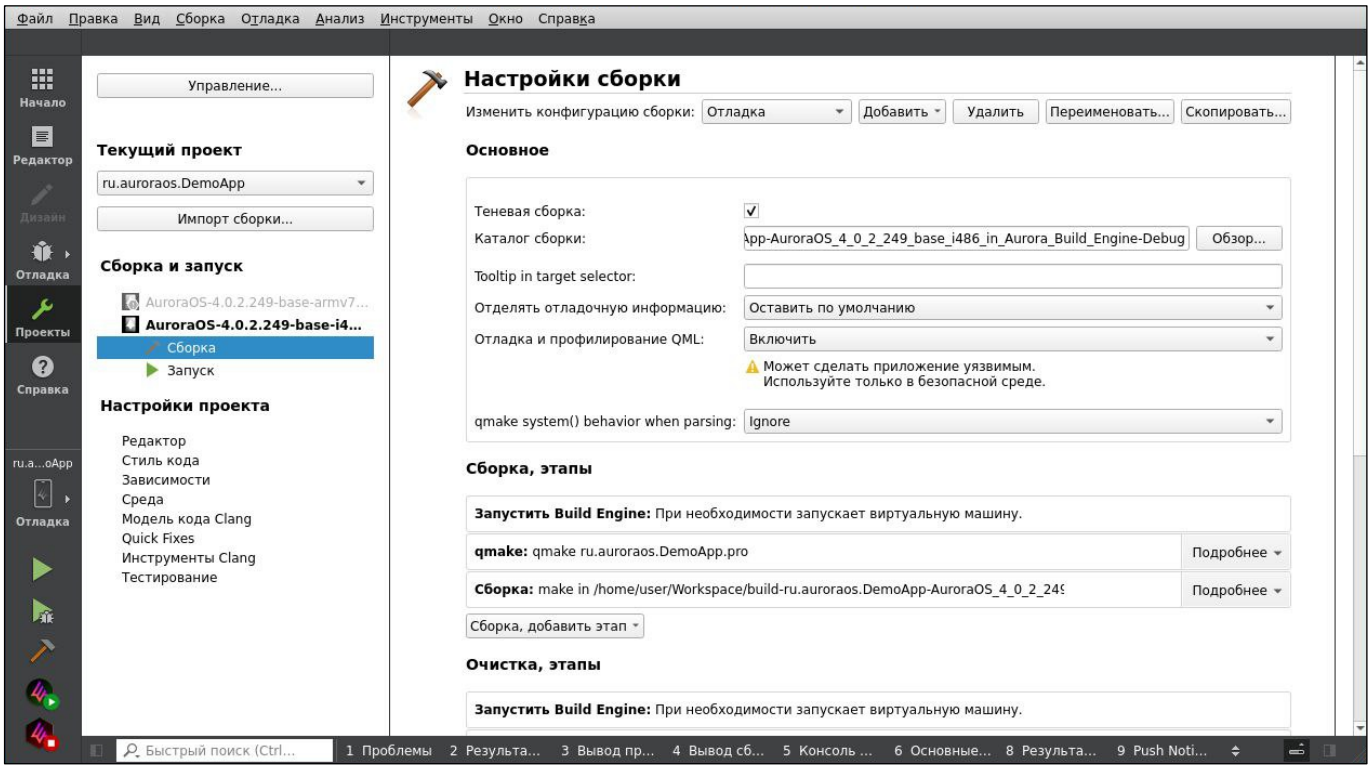


Рисунок 20

5) В окне Аврора IDE в режиме «Редактор» (Рисунок 21) можно приступить к работе над проектом.

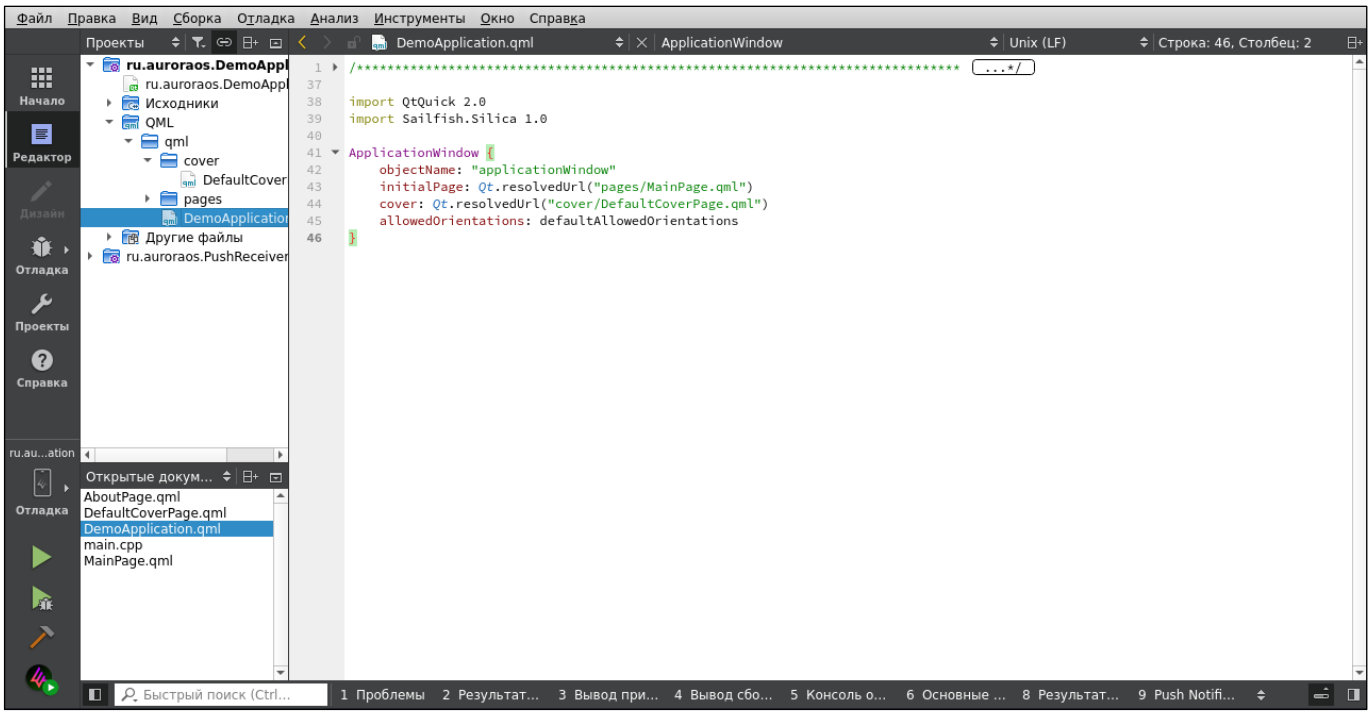


Рисунок 21

1.2. Сборка проекта

На этапе сборки предполагается, что в Аврора IDE существует открытый проект. Для сборки проекта используется среда сборки (подраздел 8.1), поэтому независимо от ОС процесс сборки приложения происходит одинаковым образом. В среде сборки настроено несколько общих папок для обмена файлами с домашней ОС. Поэтому важно расположить проект по одному из соответствующих им путей, чтобы он был доступен для сборки. По умолчанию для размещения проектов допустимы домашняя директория пользователя и альтернативная директория, указанная при установке Аврора SDK, а также все вложенные в них директории.

Для сборки проекта необходимо выполнить следующие действия:

1) Запустить среду сборки. Запуск, если требуется, происходит автоматически при начале сборки. Для управления виртуальной машиной в ручном режиме необходимо выполнить следующие действия:

– для запуска на панели слева нажать кнопку  и дождаться, пока она не примет вид  ;

– для остановки нажать кнопку  ;

2) На панели слева нажать кнопку  и выбрать комплекты и способы сборки. Для эмулятора необходимо выбрать AuroraOS-i486, для МУ — AuroraOS-armv7hl.

Здесь же можно выбрать способ сборки:

– * «Выпуск» — сборка пакетов для выпуска;

– * «Отладка» — в сборку пакетов будет добавлена информация для отладки приложения (пошаговое исполнение, наблюдение значений переменных и т. п.);

– * «Профилирование» — в сборку пакетов будет добавлена информация для профилирования и оптимизации быстродействия работы приложения (вычисление временных затрат на работу отдельных подпрограмм);

3) После завершения настроек нажать кнопку  для запуска сборки проекта (Рисунок 22).

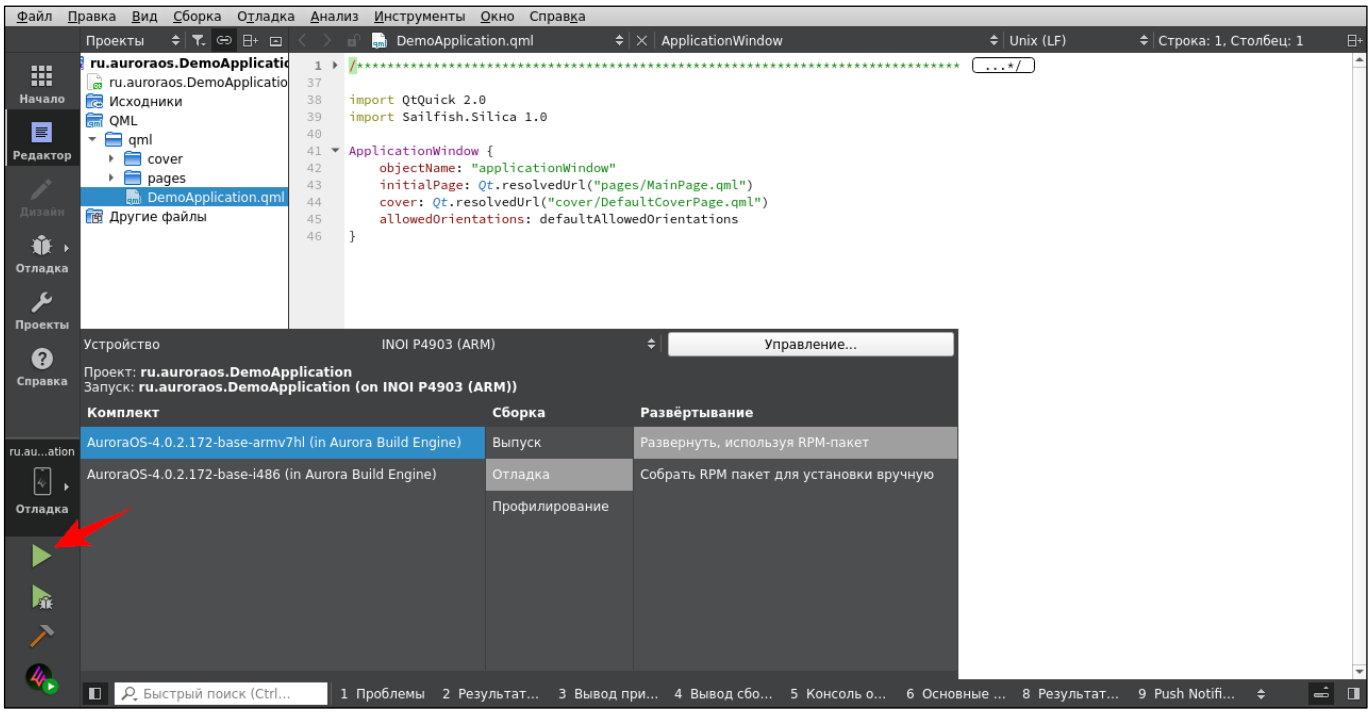


Рисунок 22

Для того, чтобы отладка и профилирование проекта были доступны, необходимо у настройки «Проекты» → «Сборка» → «Отладка и профилирование QML» установить значение «Включить» (Рисунок 23).

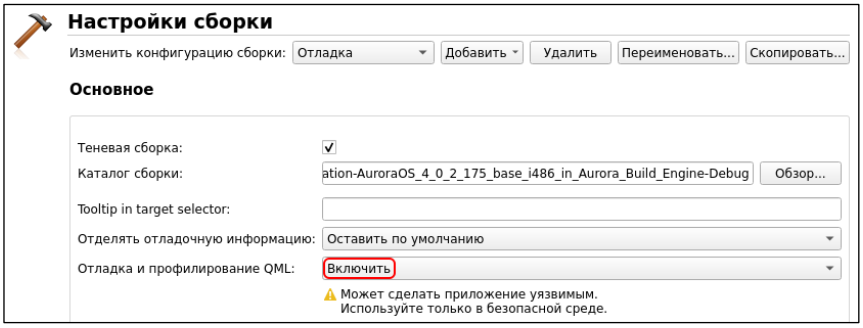


Рисунок 23

1.2.1. Корректное подключение пространств имен Aurora и PushNotifications

Для того, чтобы в проект в Аврора IDE подключить пространства имен Aurora и PushNotifications, следует корректно указать заголовочные файлы библиотеки, и настроить параметры сборки.

Подключение заголовочных файлов `pushclient` происходит стандартным образом, например:

```
#include <push_client.h>
```

Для настройки параметров сборки следует выполнить действия:

1) В pro-файле проекта добавить `pushclient` в переменную `PKGCONFIG`:

```
PKGCONFIG += pushclient
```

2) В разделе «Проекты» → «Сборка и запуск» открыть вкладку «Настройки» сборки для таргета, с которым в данный момент идет работа (Рисунок 24);

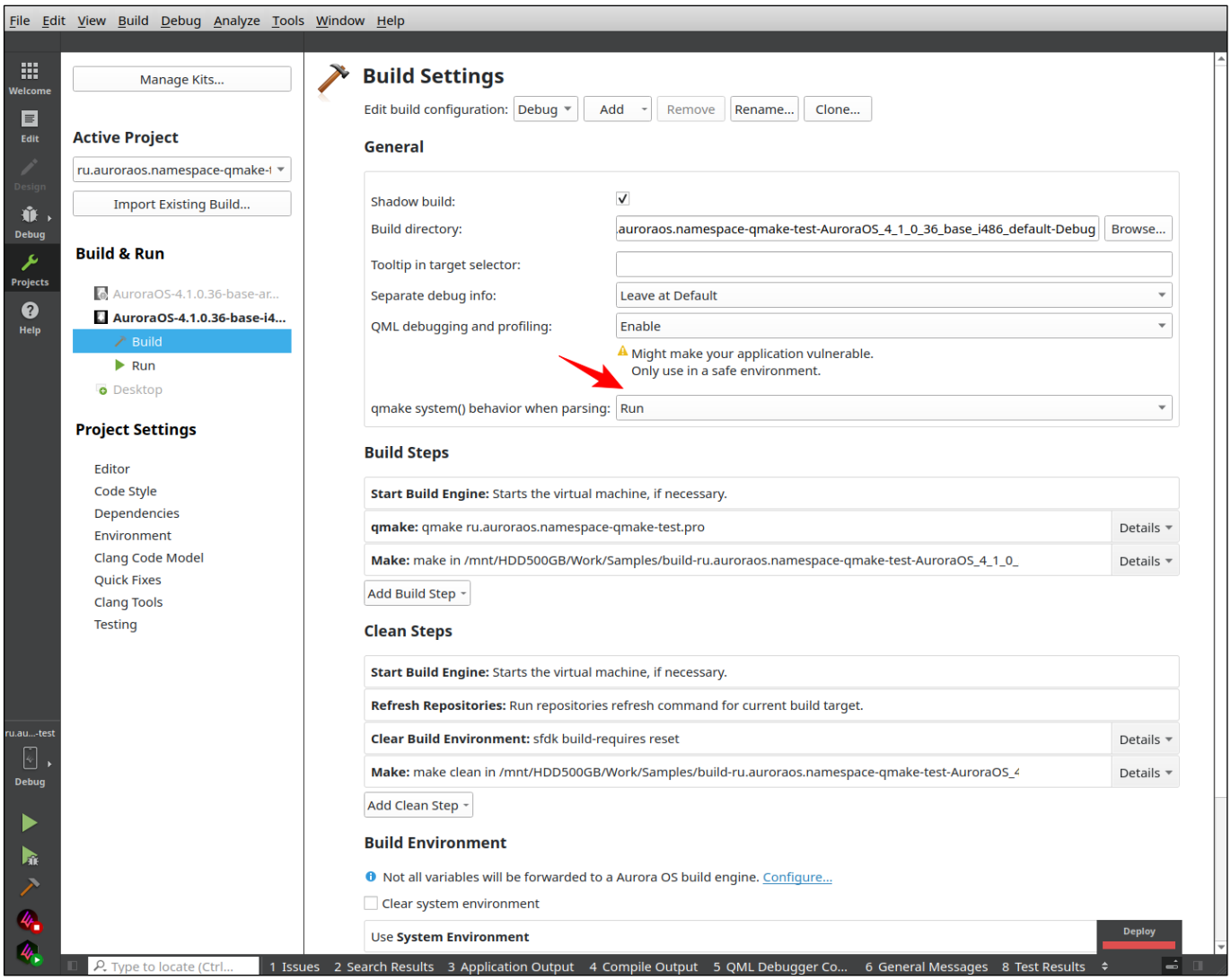


Рисунок 24

3) Изменить значение настройки «qmake system() behavior when parsing» с «Ignore» на «Run». После этого и перезапуска IDE, синтаксис пространств имен Aurora и PushNotifications должен подсвечиваться корректно.

Действия 2-3 следует повторить для всех таргетов, с которыми идет работа.

1.3. Запуск приложения

Приложение может быть запущено как на внешнем устройстве, работающем под управлением ОС Аврора, так и в эмуляторе, который устанавливается при инсталляции Аврора SDK.

Для запуска приложения на эмуляторе необходимо выполнить следующие действия:

1) На панели слева нажать кнопку  и выбрать комплект AuroraOS-i486;

2) Запустить эмулятор нажатием кнопки  и дождаться, пока она не примет вид  . Откроется новое окно VirtualBox, и загрузится эмулятор. Если нажать кнопку  , эмулятор остановится, и окно VirtualBox закроется;

3) Для запуска приложения нажать кнопку  , для отладки — кнопку  .

ПРИМЕЧАНИЕ. Чтобы кнопки стали активны, необходимо выбрать «Deploy As RPM Package» или «Deploy by Copying Binaries» в панели выбора комплектов и способов сборки.

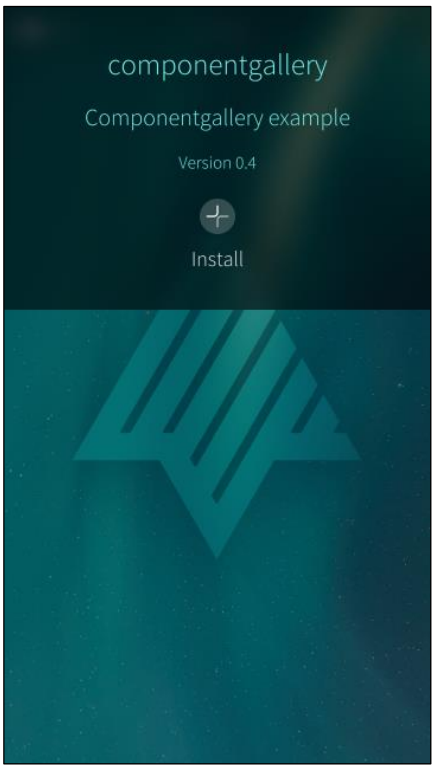


Рисунок 25

На экране эмулятора появится диалог подтверждения установки приложения (Рисунок 25);

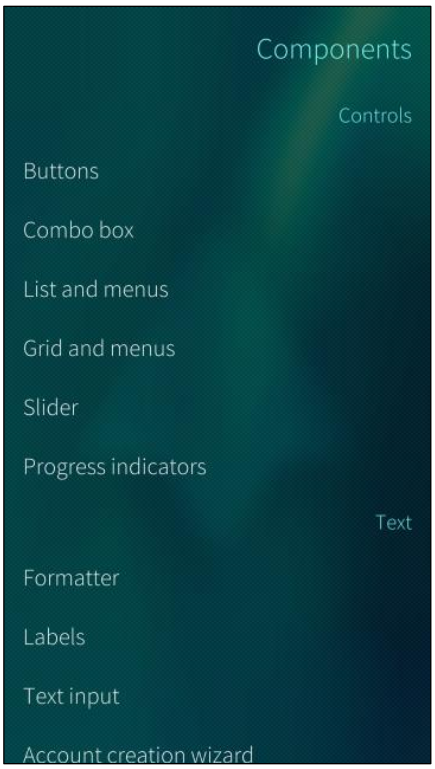


Рисунок 26

4) Для установки приложения коснуться значка «Install» на экране эмулятора. Приложение установится, и откроется его стартовая страница (Рисунок 26);

Запуск приложения на устройстве происходит аналогичным образом, но дополнительно требуется:

1) В основном окне Аврора IDE выбрать пункт меню «Инструменты» → «Параметры»;

2) Перейти на вкладку «Устройства» и подключить устройство;

3) Настроить подписание установочного пакета (https://developer.auroraos.ru/doc/software_development/guides/package_signing) (Рисунок 27). По умолчанию этап «RPM Sign» может быть отключен, для его включения следует убрать пиктограмму «Отключить».

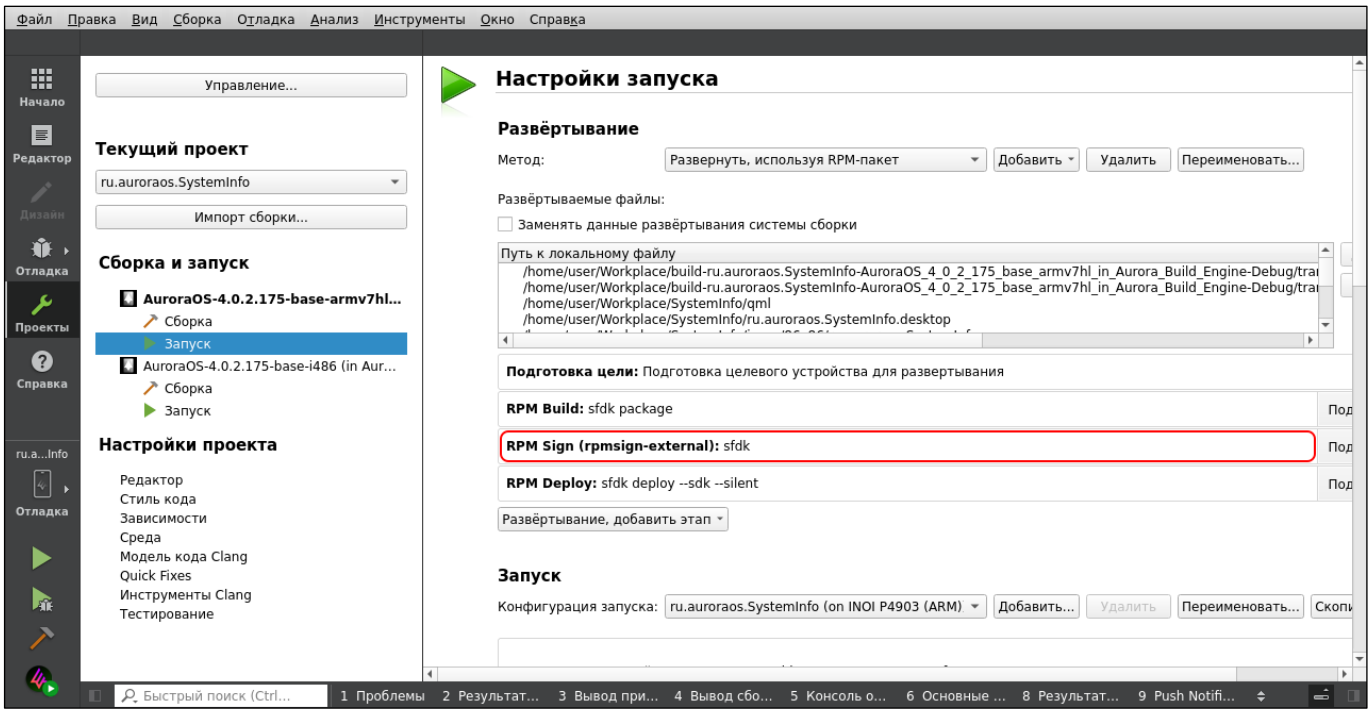


Рисунок 27

По умолчанию для МУ в поле «RPM Deploy» добавлен параметр `--silent` (тихая установка) (Рисунок 28). При необходимости данный параметр можно убрать вручную.

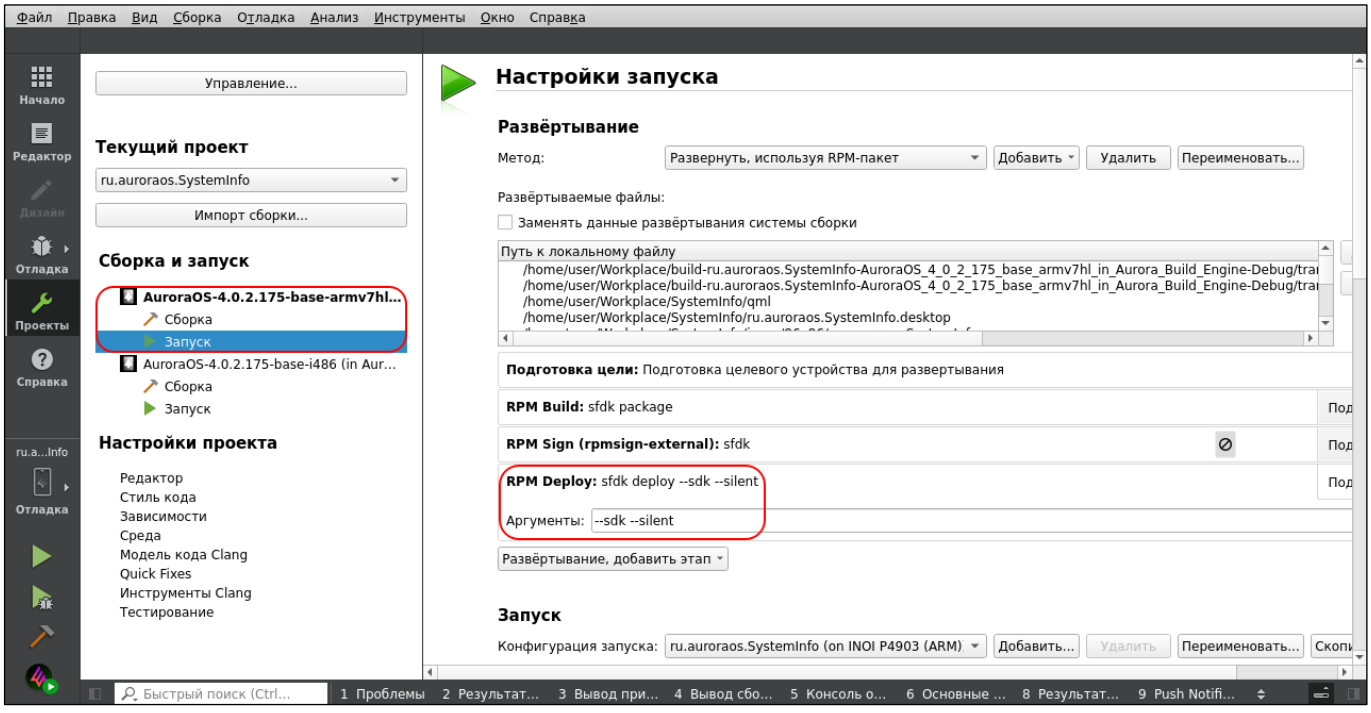


Рисунок 28

1.4. Тихая установка приложений

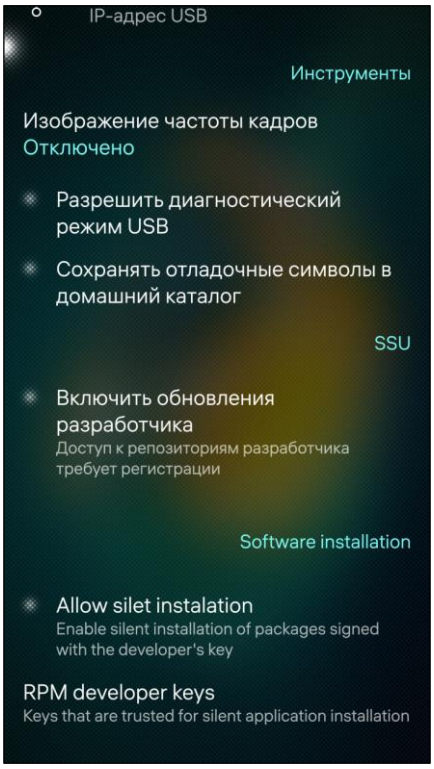


Рисунок 29

По умолчанию каждый раз при установке новой версии приложения требуется подтверждение. Режим тихой установки предоставляет пользователю Аврора SDK возможность устанавливать приложение на устройство или эмулятор во время разработки без подтверждения.

На устройстве в настройках разработчика необходимо разрешить тихую установку приложений. В приложении «Настройки» нужно включить режим разработчика. В разделе, посвященном этому режиму, имеется переключатель «Allow silent installation» для включения/выключения режима тихой установки (см. Рисунок 29).

2. ОТЛАДКА ПРОЕКТА

2.1. Отладка приложений с изоляцией

Отладчик gdb (https://sourceware.org/gdb/current/onlinedocs/gdb.html/) можно подключить к приложению по номеру процесса, например, через консоль или переконфигурацию gdb из Аврора IDE и двустороннюю передачу данных между gdb и Аврора IDE.

При запуске приложений из Аврора IDE с конфигурацией по умолчанию отладчик gdb не будет работать, так как не будет запущена утилита `sailjail`, необходимая для отладки.

Для старта `sailjail` нужно запустить уже установленное приложение на устройстве или на эмуляторе через значок.

Отладку можно выполнить следующим образом:

1) Запустить приложение в песочнице, используя `sailjail` или `firejail`:

– Вариант 1: `sailjail --dry-run - PATH`;

– Вариант 2 (для отладки c++-кода): `sailjail - PATH`;

– Вариант 3 (для отладки qml): `firejail` с параметрами.

Изменить у приложения конфигурацию по умолчанию в разделе «Проекты» → «Запуск»:

– полю «Сменить программу на устройстве» задать значение `/usr/bin/sailjail`, предварительно поставив галочку в чекбоксек «Использовать эту команду» (Рисунок 30);

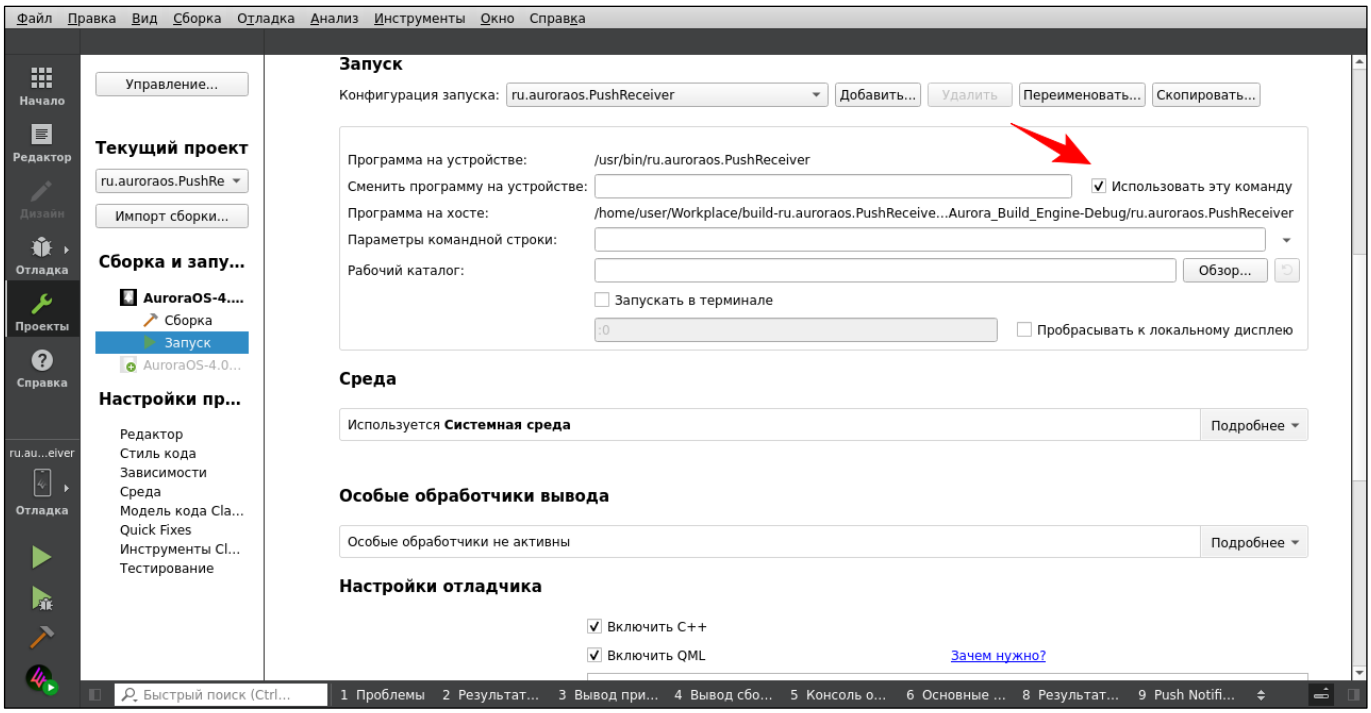


Рисунок 30

– полю «Параметры командной строки» (Рисунок 31) задать значение `-p <имя приложения>.desktop /usr/bin/<имя приложения>`.

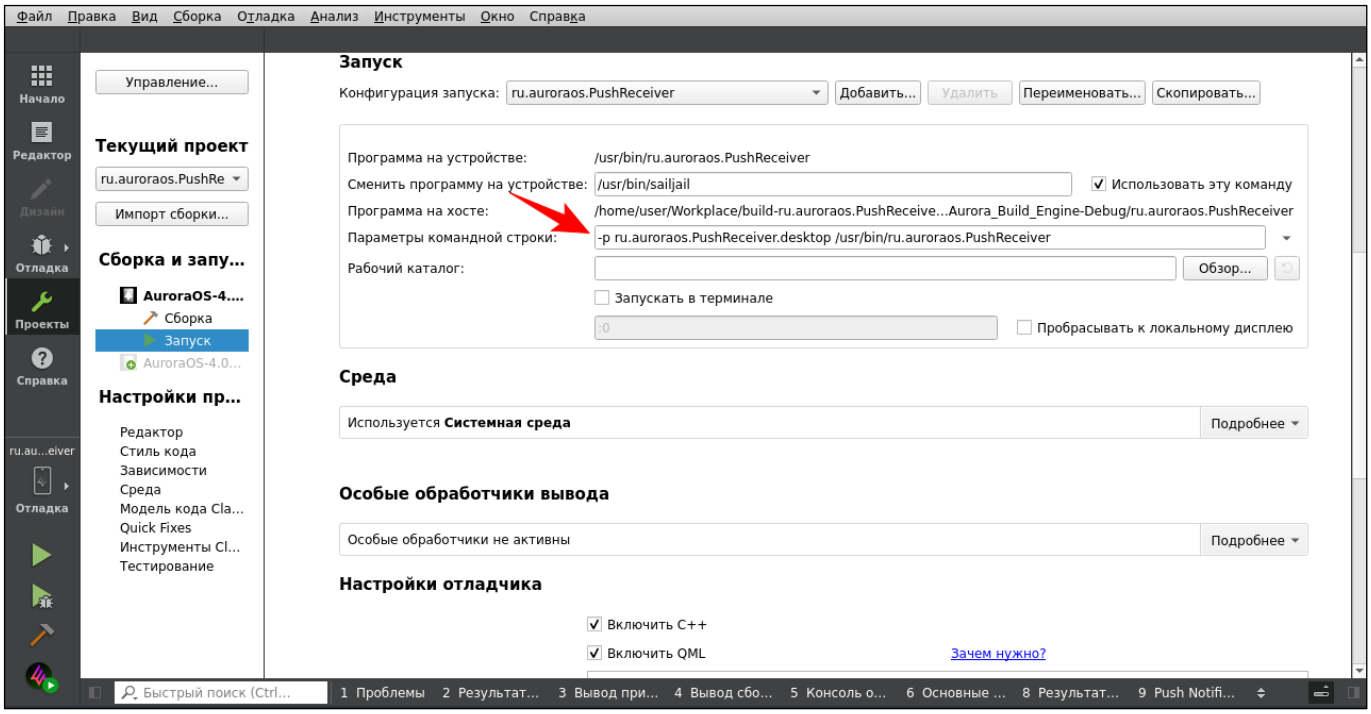


Рисунок 31

Данный способ ориентирован на физическое устройство, `sailjail` не применяется на эмуляторе.

Для отладки qml-кода нужно указать `firejail` вместо `sailjail` (Рисунок 32);

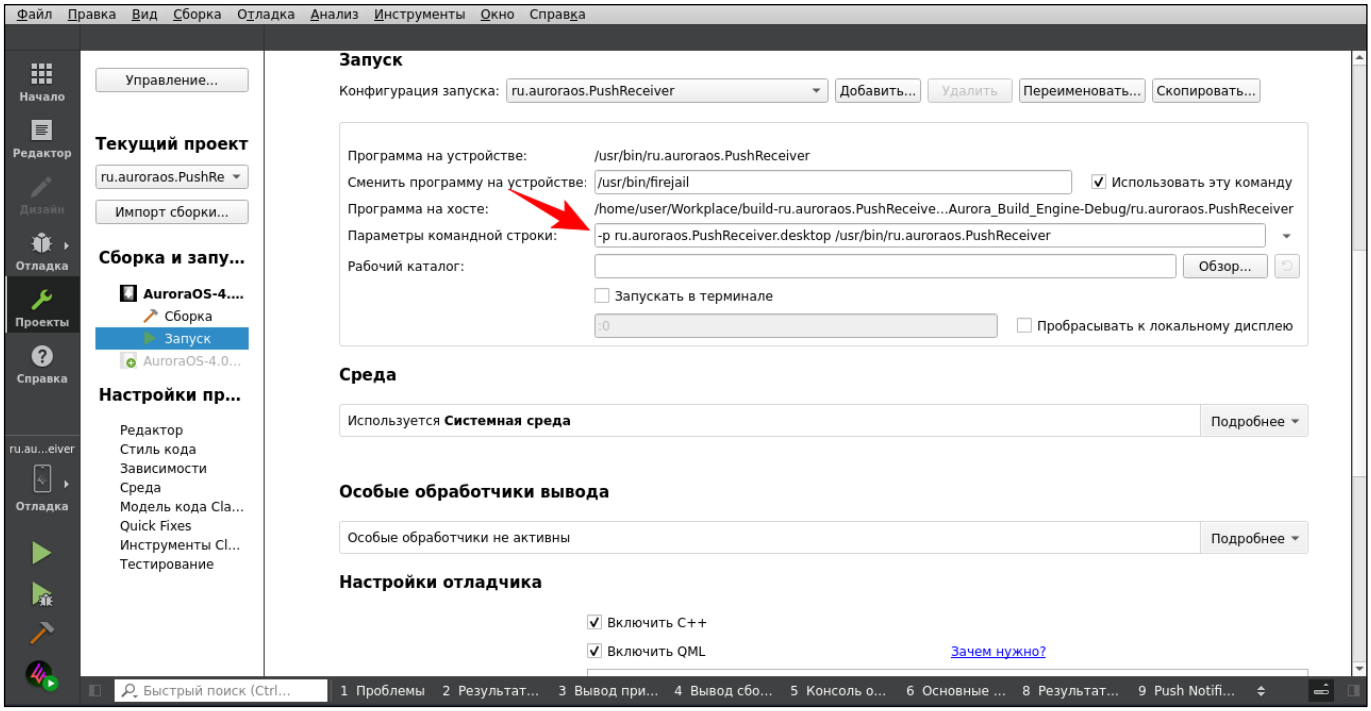


Рисунок 32

2) Получить pid приложения:

```
pidof <имя приложения>
```

И подключить gdb к процессу, используя pid:

```
gdb <имя приложения> -p <pid приложения>
```

Ограничения данного способа отладки:

– поскольку gdb не запускает процесс, а подключается к нему, то нет возможности отлаживать ошибки запуска программы;

– при подключении gdb к процессу, сообщения, которые выводит процесс в консоль методами qDebug и аналогичными, не передаются в окно Аврора IDE;

– при запуске приложения в песочнице не работает горячая перезагрузка QML - страниц через QmlLiveBench (https://developer.auroraos.ru/doc/software_development/sdk/qmllive);

– невозможно отладить инициализацию свободных статических переменных.

ВНИМАНИЕ! Крайне не рекомендуется использовать в программе большое количество свободных статических переменных.

Для отладки процесса запуска программы можно добавить в код вызов метода `QThread::sleep(ЧИСЛО_СЕКУНД)`. Параметр `ЧИСЛО_СЕКУНД` можно подобрать в зависимости от быстродействия компьютера разработчика: чем медленнее компьютер — тем больше стоит выбрать число.

Пример:

```
int main(int argc, char *argv[])
{
    QThread::sleep(2);
    QScopedPointer<QGuiApplication> app(SailfishApp::application(argc,
argv));
    ...
}
```

В этом случае gdb сможет гарантированно остановиться на строке, следующей после `QThread::sleep(2)`, что позволит отлаживать процесс запуска программы.

2.2. Пересборка приложения с измененными зависимостями

Если у приложения, которое уже собиралось ранее, изменился набор зависимостей, то при установке новой версии необходимо обновить снимок файловой системы. Для этого нужно выполнить пересборку проекта: «Сборка» → «Пересобрать проект…». В процессе пересборки сначала будут обновлены репозитории цели, и далее снимок файловой системы синхронизируется с таргетом.

Репозитории таргета можно обновить через Аврора IDE: «Инструменты» → «Параметры» → «Аврора ОС» → «Build Engine» → «Manage Build Targets» → «Refresh». При этом снимок файловой системы синхронизирован не будет.

3. ПОДПИСЬ ПАКЕТА

Установочные пакеты приложений для ОС Аврора должны быть подписаны (https://developer.auroraos.ru/doc/software_development/guides/package_signing) сертификатами, позволяющими идентифицировать поставщиков ПО и избежать попадания на МУ нежелательных приложений.

Один из способов подписи пакета через Аврора IDE — работа с соответствующим диалогом.

Все действия диалога валидации происходят в среде сборки. Для передачи параметров используется каталог, разделяемый с компьютером разработчика. Работа утилит валидации может происходить с задержкой.

Для того, чтобы подписать пакет с помощью диалога, нужно выполнить следующие действия:

1) В Аврора IDE выбрать «Инструменты» → «Аврора ОС» → «Подпись пакетов» (Рисунок 33);

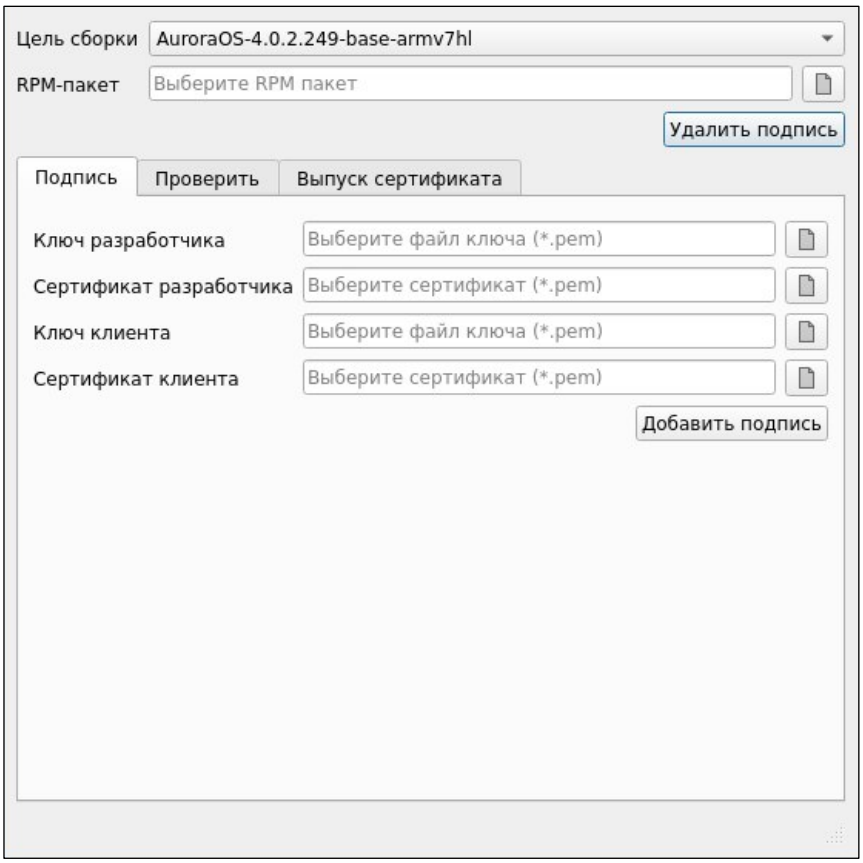


Рисунок 33

2) Убедиться, что в поле «Таргет» указана необходимая версия ОС Аврора. Если поле пустое и в выпадающем списке отсутствуют элементы, то необходимо запустить или перезапустить среду сборки и подождать загрузки виртуальной машины;

3) Выбрать необходимый RPM-пакет, нажав на кнопку «Обзор». RPM-пакет должен находиться в домашнем каталоге пользователя, так как среда сборки разделяет каталог с хостовой системой именно в нем. После выбора RPM-пакета автоматически запустится команда по дампу подписи (сбору информации о подписи пакета). Информация о подписи будет выведена в панели с основными сообщениями и на вкладке «Проверить»;

4) Кнопка «Удалить» подпись позволяет удалить всю подписанную часть RPM-пакета. Это действие не требует ключей или сертификатов;

5) На вкладке «Подпись» можно указать ключ и/или сертификат и нажать кнопку «Добавить подпись», чтобы подписать выбранный RPM-пакет. Подробнее о ключах и сертификатах описано в основной статье о подписи пакетов на Портале разработчиков (https://developer.auroraos.ru/doc/software_development/guides/package_signing);

6) На вкладке «Проверить подпись» (Рисунок 34) можно верифицировать подпись. Для этого нужно передать в поле «Сертификат УЦ» необходимый корневой сертификат, в противном случае будет произведена верификация корневым сертификатом на устройстве. Обычно эти действия не требуются, достаточно только выбрать RPM-пакет, и проверка произойдет автоматически;

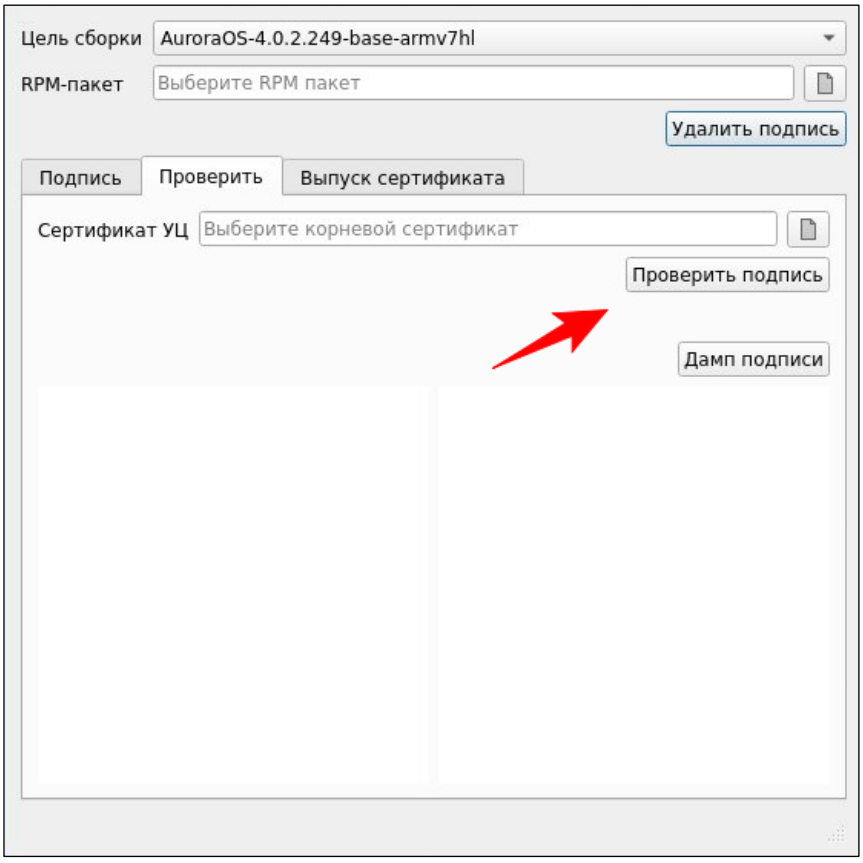


Рисунок 34

7) Кнопка «Дамп подписи» позволяет собрать информацию о подписи и сертификатах, которыми подписан пакет;

8) На вкладке «Выпуск сертификата» можно сгенерировать сертификат, указав каталоги и файлы для сохранения ключа и запроса, алгоритм шифрования RSA или ГОСТ, имя компании и поставив при необходимости галочку в чекбоксе «Зашифровать» (Рисунок 35).

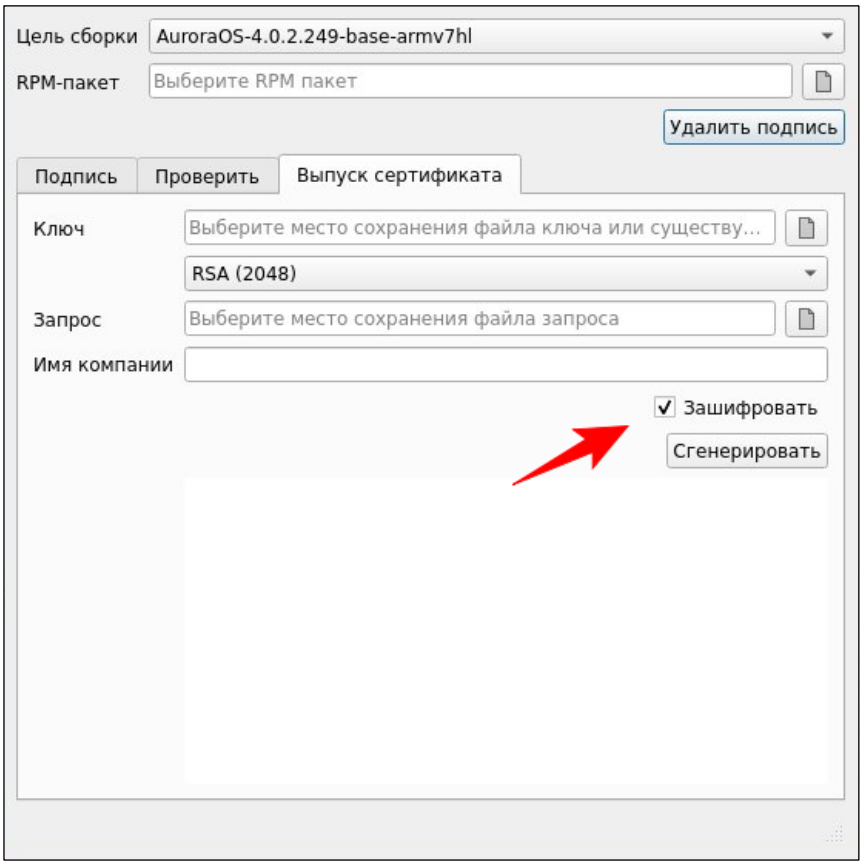


Рисунок 35

Файл сертификата (но не ключа) необходимо отправить письмом на dev-support@omp.ru. В ответ будет прислан сгенерированный файл `{cert}.pem`.

4. ПРОЦЕСС ВАЛИДАЦИИ ПАКЕТА

Для корректной установки и работы приложения для ОС Аврора, необходимо, чтобы RPM - пакет соответствовал требованиям (https://developer.auroraos.ru/doc/software_development/guidelines/rpm_requirements). Пакеты, не прошедшие валидацию, не могут быть установлены.

Самостоятельно проверить установочный пакет можно с помощью диалога в Аврора IDE, описанного ниже, или валидатора RPM-пакетов (https://developer.auroraos.ru/doc/tools/rpm-validator).

Все действия с диалогом валидации происходят в среде сборки. Для передачи параметров используется каталог, разделяемый с компьютером разработчика. Работа утилит валидации может происходить с задержкой.

Для того, чтобы валидировать пакет с помощью диалога, нужно выполнить следующие действия:

1) В Аврора IDE выбрать «Инструменты» → «Аврора ОС» → «Валидация RPM» (Рисунок 36);

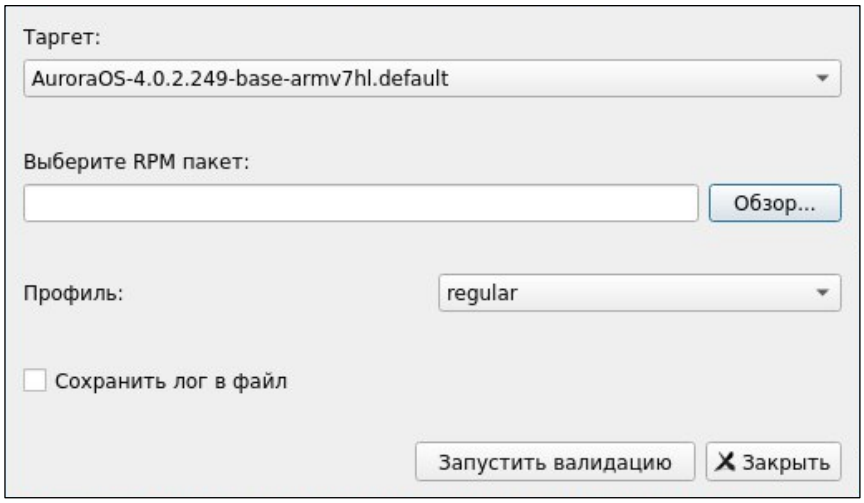


Рисунок 36

2) Убедиться, что в поле «Таргет» указана необходимая версия ОС Аврора. Если поле пустое и в выпадающем списке отсутствуют элементы, то необходимо запустить Build Engine (или перезапустить) и подождать загрузки виртуальной машины;

3) Выбрать необходимый RPM-пакет, нажав на кнопку «Обзор». RPM-пакет должен находиться в домашнем каталоге пользователя, так как среда сборки разделяет каталог с хостовой системой именно в нем;

4) Выбрать профиль валидации из выпадающего списка. По умолчанию установлен regular;

5) Опционально можно выбрать пункт «Сохранить лог в файл» и в открывшемся поле указать путь к текстовому файлу, в который нужно записать лог;

6) Нажать кнопку «Запустить валидацию» (Рисунок 37). Результат будет выведен как сообщение в теле диалога. Более подробная информация появится в панели с основными сообщениями.

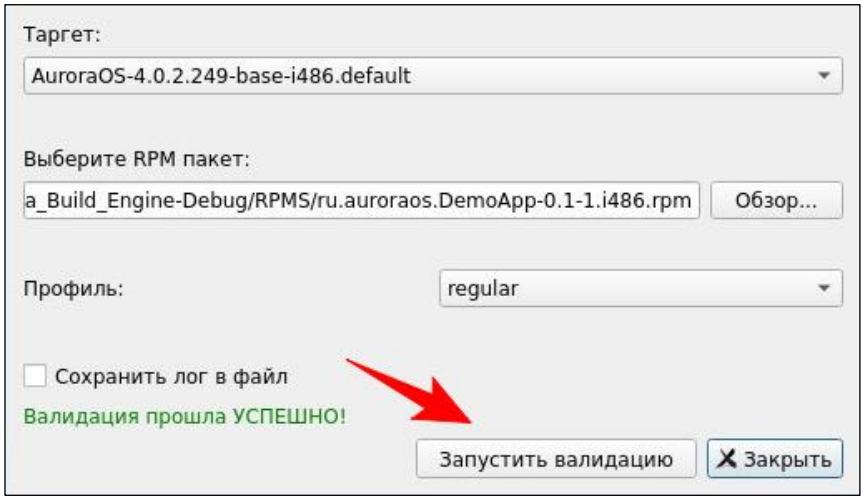


Рисунок 37

5. ПРИМЕРЫ ПРИЛОЖЕНИЙ В АВРОРА SDK

На главном экране Aurora IDE на вкладке «Примеры» доступны проекты приложений, которые демонстрируют подходы к разработке и работу с различными API в ОС Аврора.

Подробнее с ними можно ознакомиться на Портале разработчиков в разделе проектов с открытым исходным кодом (https://developer.auroraos.ru/doc/software_development/guides/examples_open_source).

5.1. Открытие примера

Для того, чтобы открыть пример в IDE, нужно выполнить следующие действия:

1) Открыть вкладку «Примеры» на главном экране и выбрать пункт «Aurora ОС», если он не указан по умолчанию (Рисунок 38);

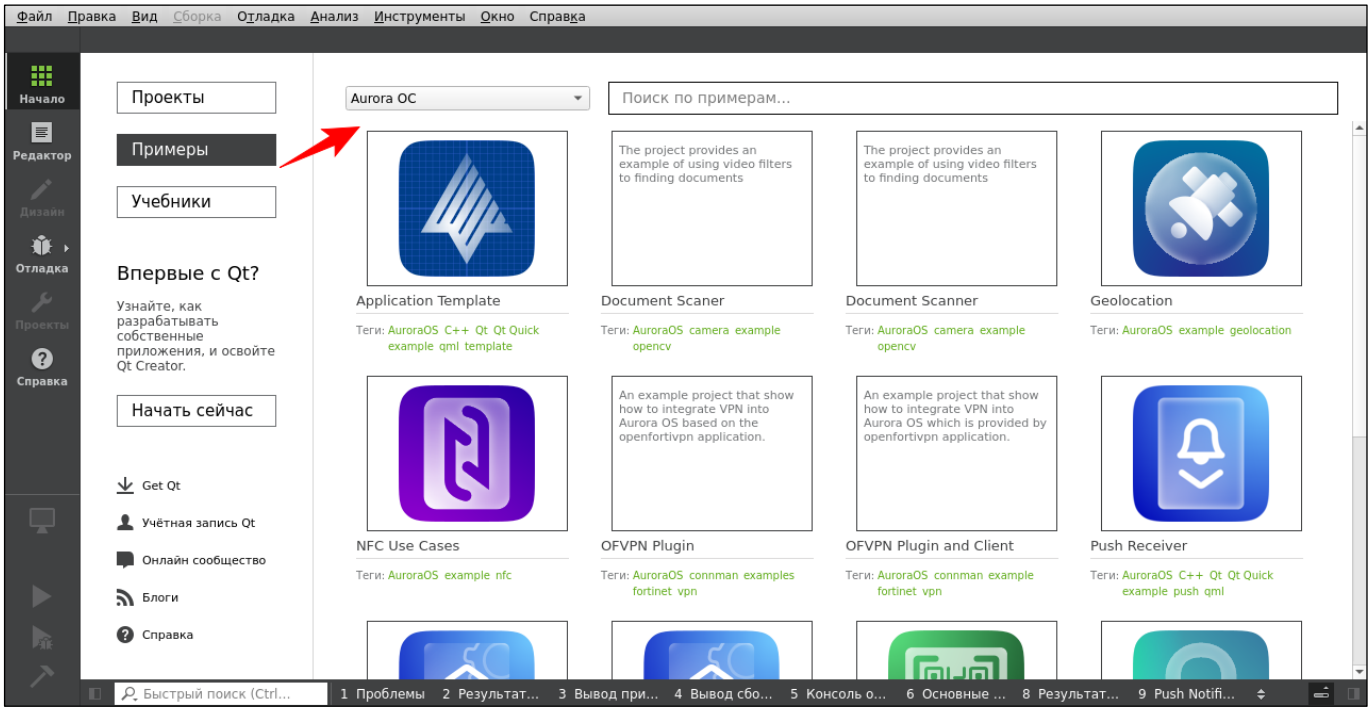


Рисунок 38

У каждого примера имеется описание, которое показывается вместо картинки, если навести на нее курсор (Рисунок 39).

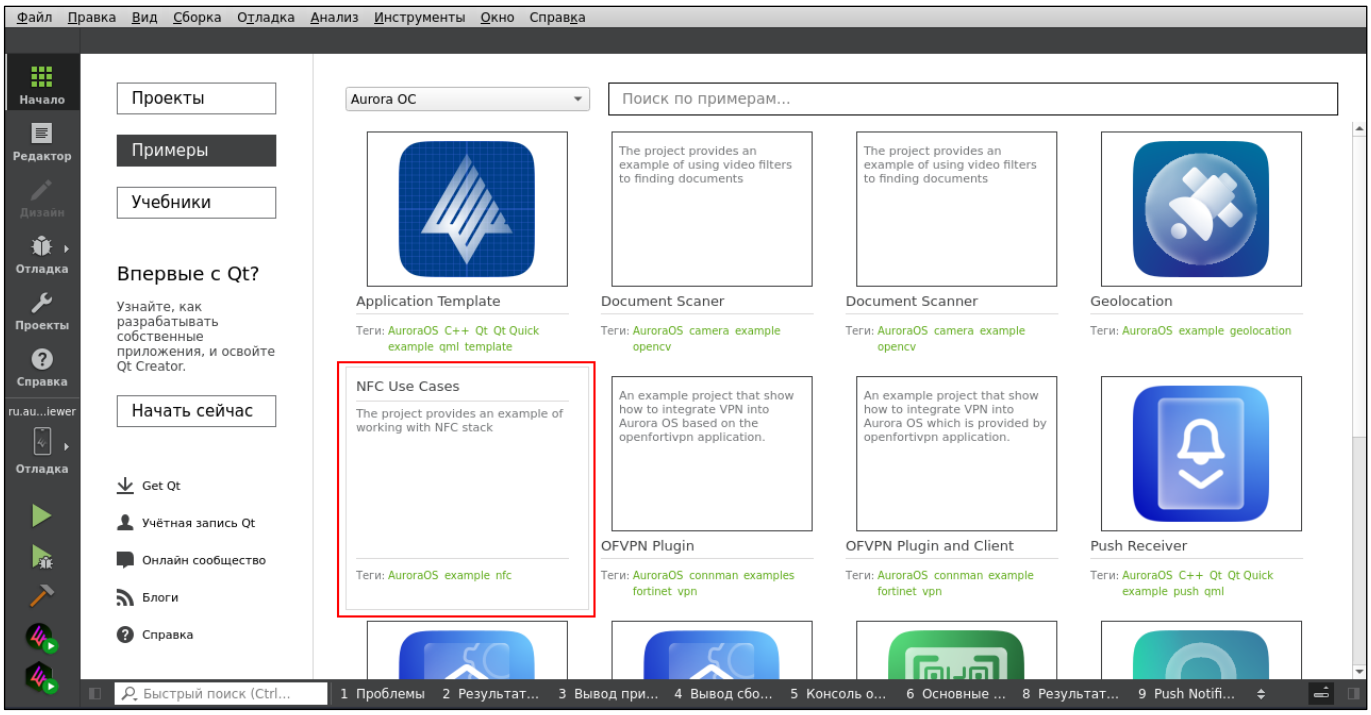


Рисунок 39

В строке поиска можно искать примеры по названию или тегам (Рисунок 40);

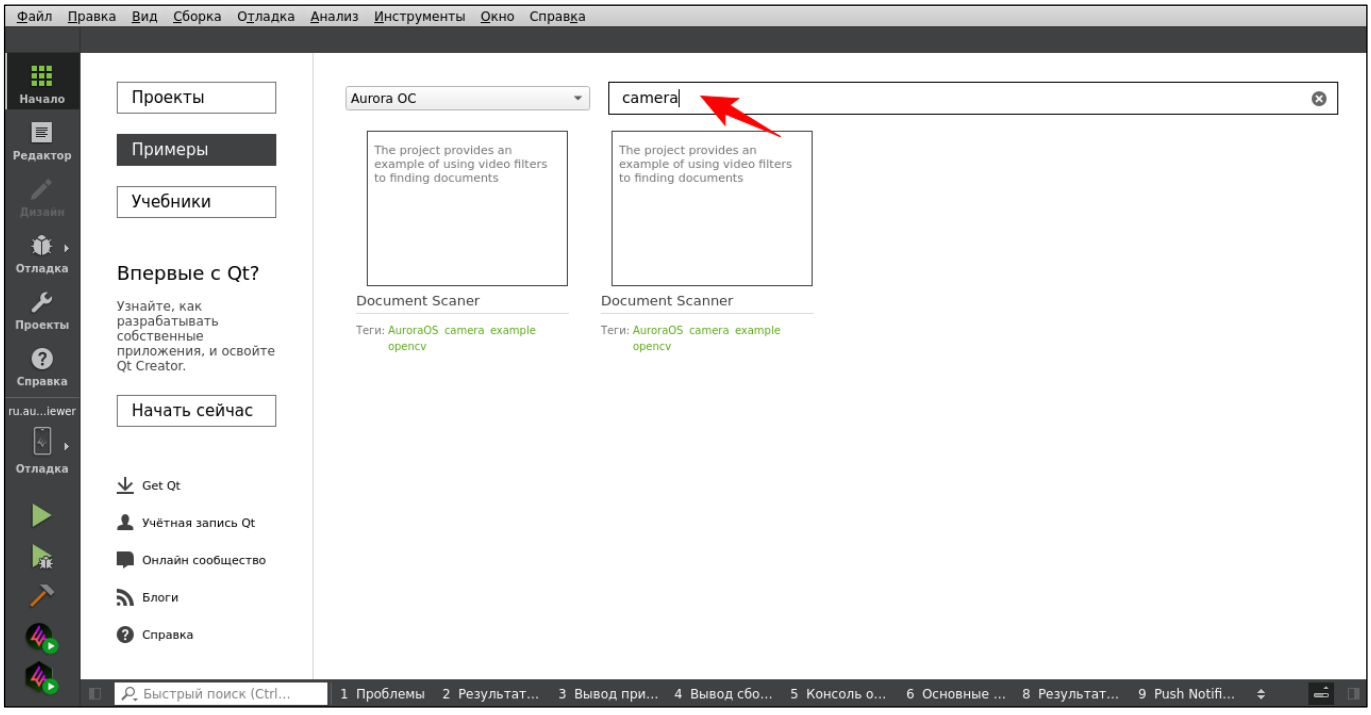


Рисунок 40

2) Выбрать пример и нажать на него. IDE предложит скопировать проект в каталог, из которого приложение можно будет запускать. По умолчанию это каталог проектов, указанный при установке IDE, но его можно изменить в диалоговом окне (Рисунок 41).

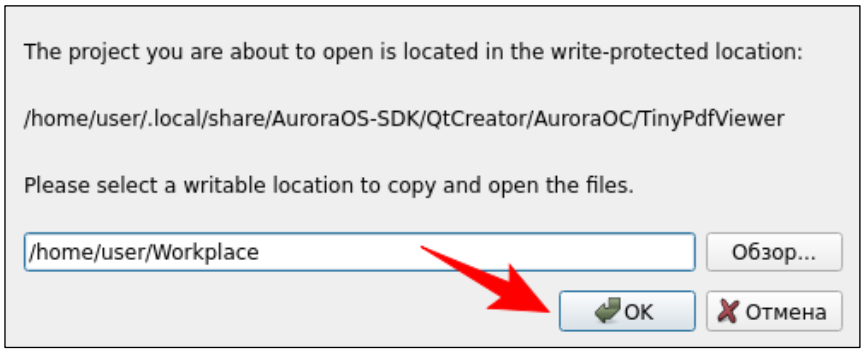


Рисунок 41

После выбора каталога следует нажать кнопку «ОК»;

3) Пример будет скопирован в выбранный каталог и открыт в IDE (Рисунок 42). С примером можно работать так же, как и с обычным проектом: редактировать код, настраивать параметры сборки и запускать (Рисунок 43).

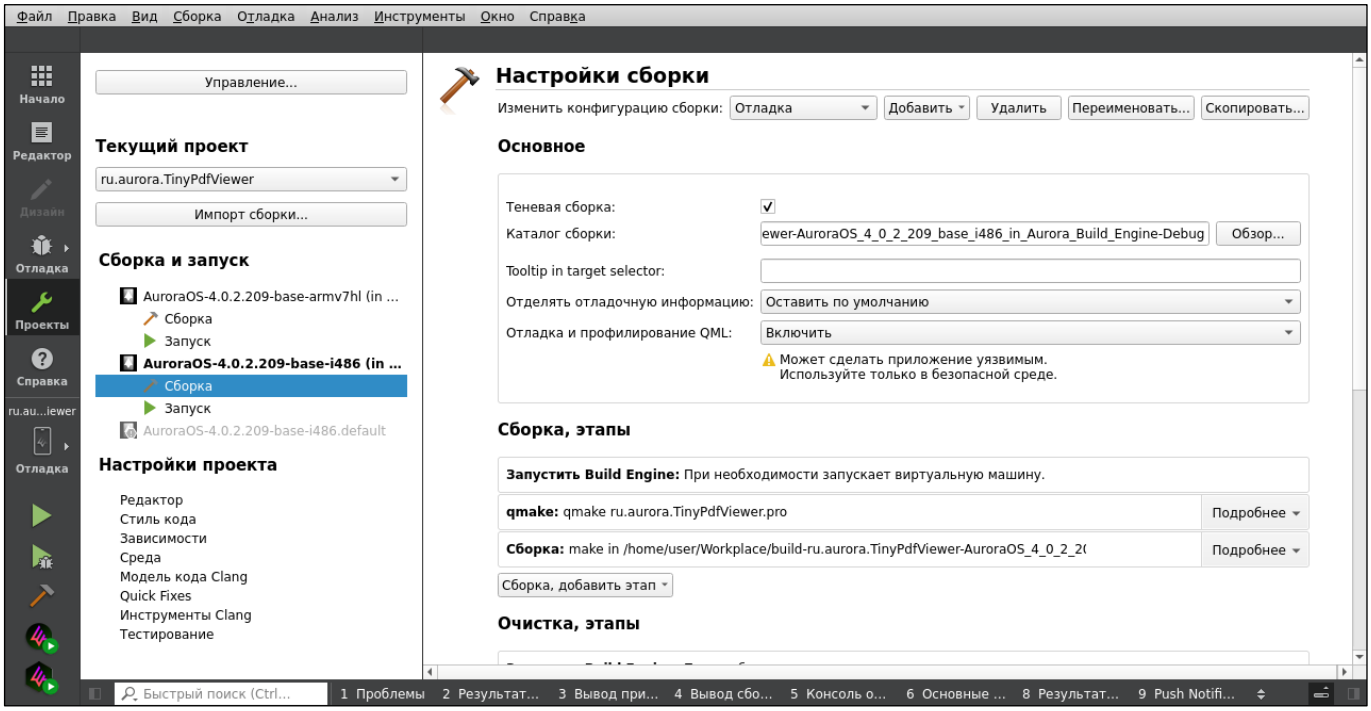


Рисунок 42

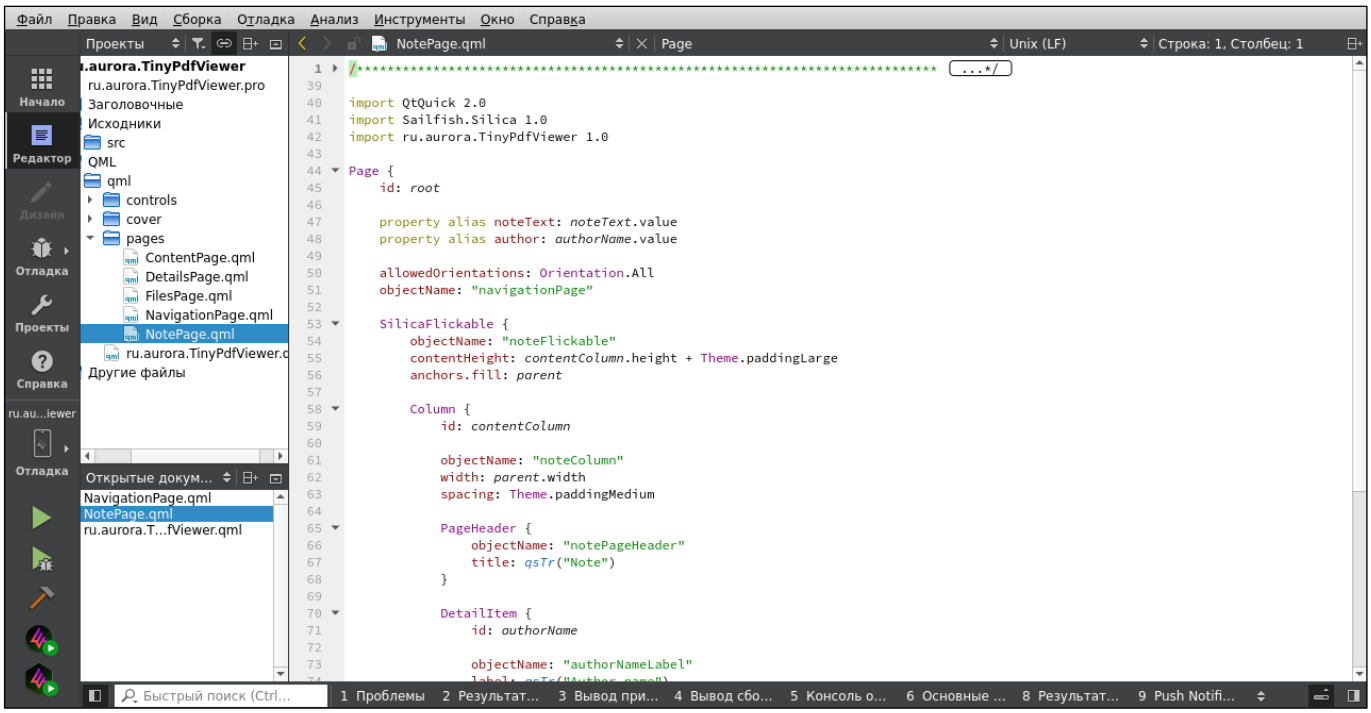


Рисунок 43

5.2. Обновление примеров

Добавить новые примеры, обновить или удалить пример можно с помощью плагина в IDE.

Чтобы открыть настройки плагина, следует выбрать: «Инструменты» → «Параметры» → «Справка» → «Aurora ОС примеры» (Рисунок 44).

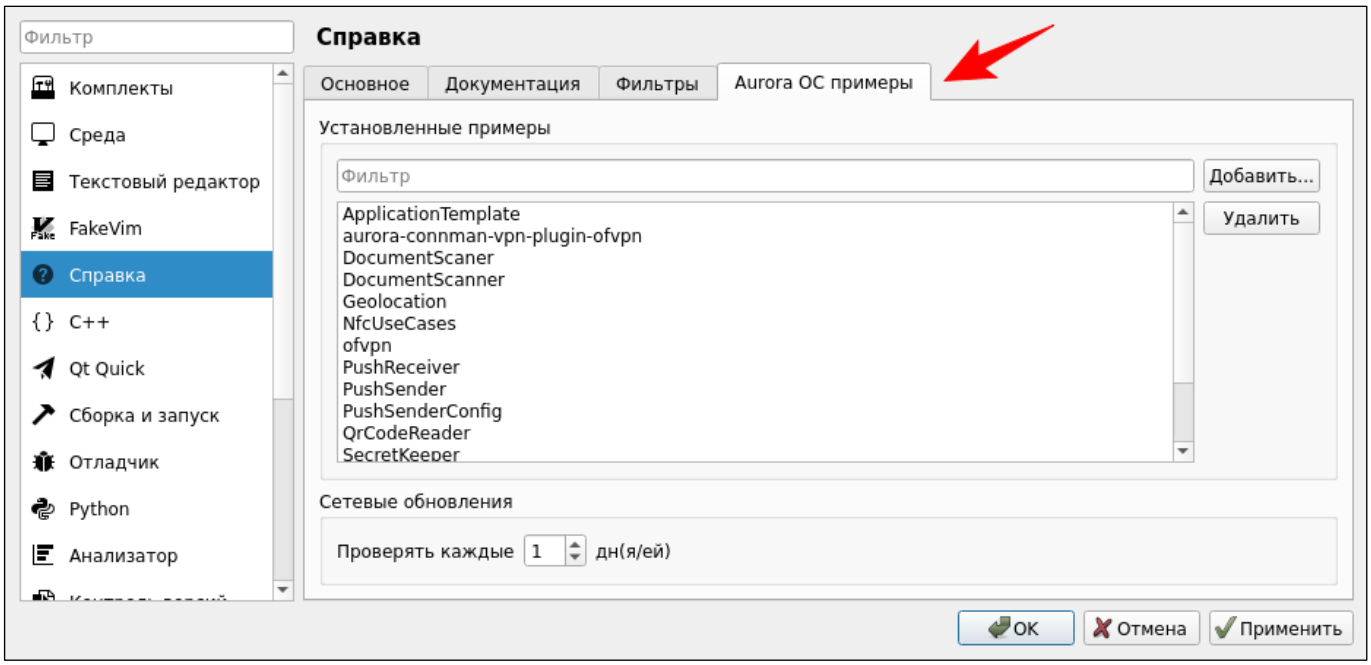


Рисунок 44

Дополнительную информацию о примере можно получить во всплывающей подсказке, если навести на его название курсор (Рисунок 45). Можно узнать статус его обновления и расположение на компьютере разработчика.

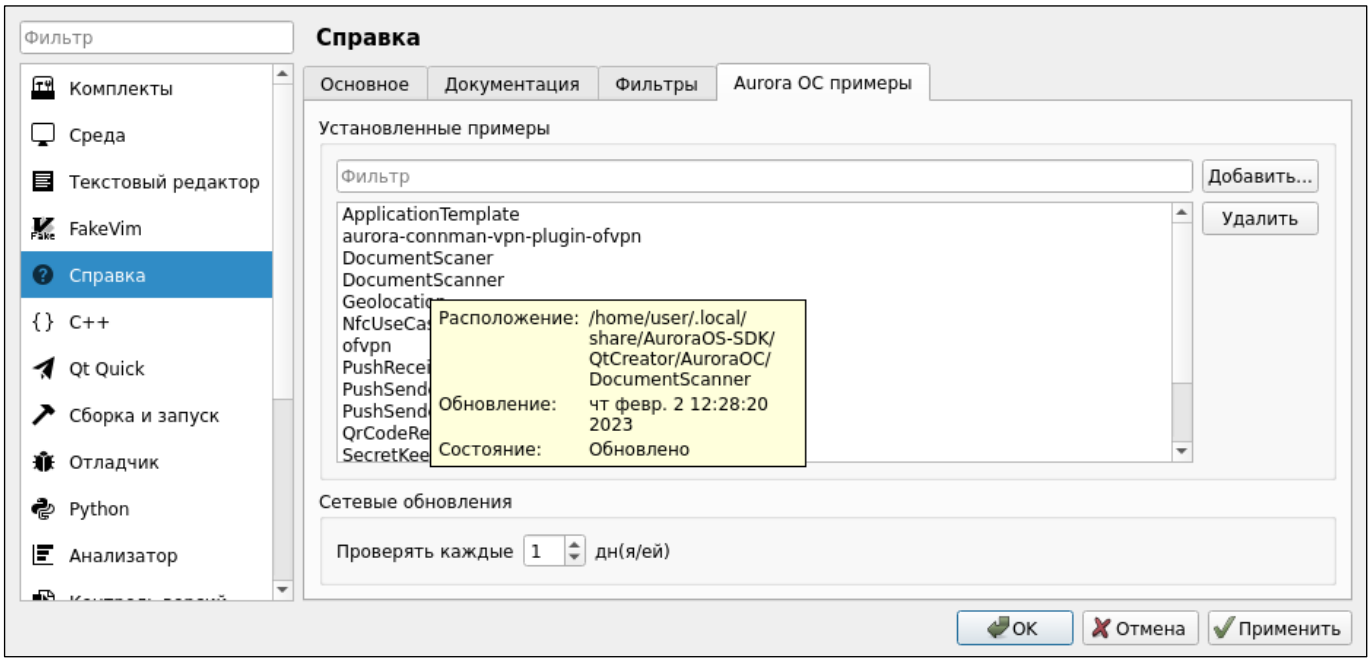


Рисунок 45

Пример, для которого доступно обновление, подсвечивается красным.

Каталог с примером на диске можно открыть, нажав правой кнопкой мыши по примеру и выбрав пункт контекстного меню «Открыть расположение» (Рисунок 46).

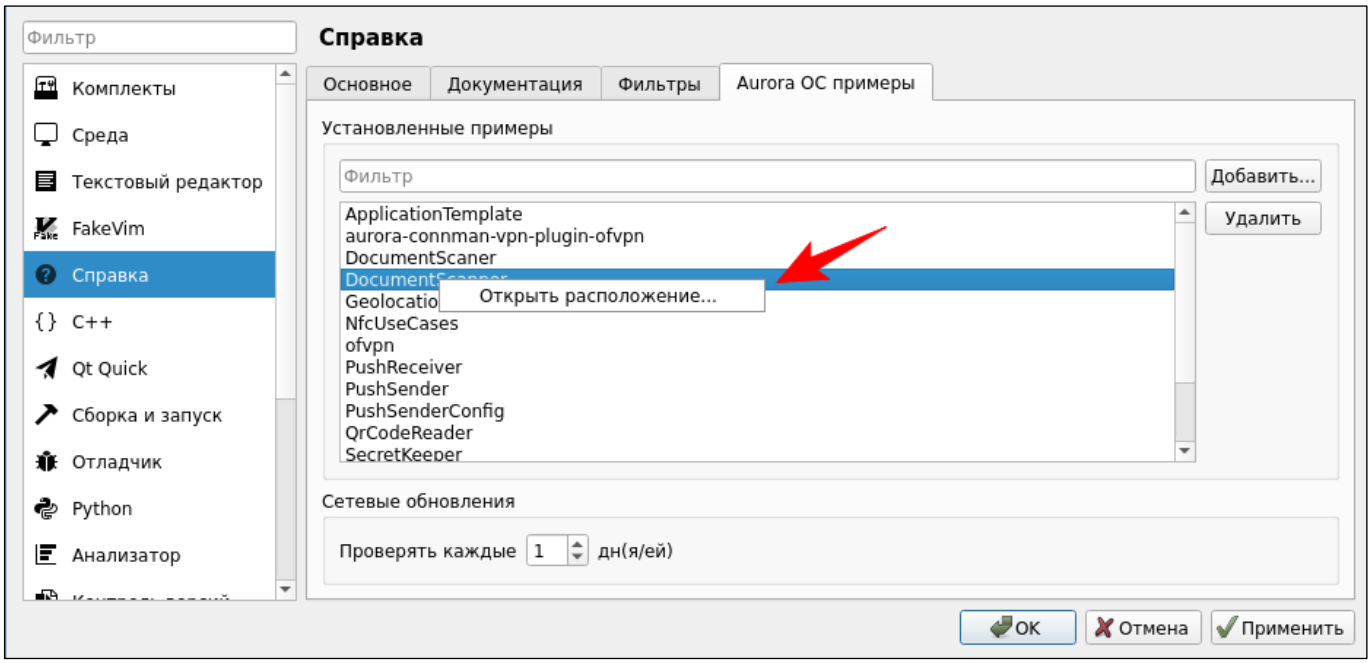


Рисунок 46

Примеры обновляются автоматически из публичных репозиториев при наличии связи с ними. Репозитории примеров с подмодулями не поддерживаются. Задать частоту проверки обновлений примеров можно в настройке «Проверять каждые N дней», где N находится в диапазоне от 0 до 99, при этом 0 означает остановку автоматического обновления (Рисунок 47).

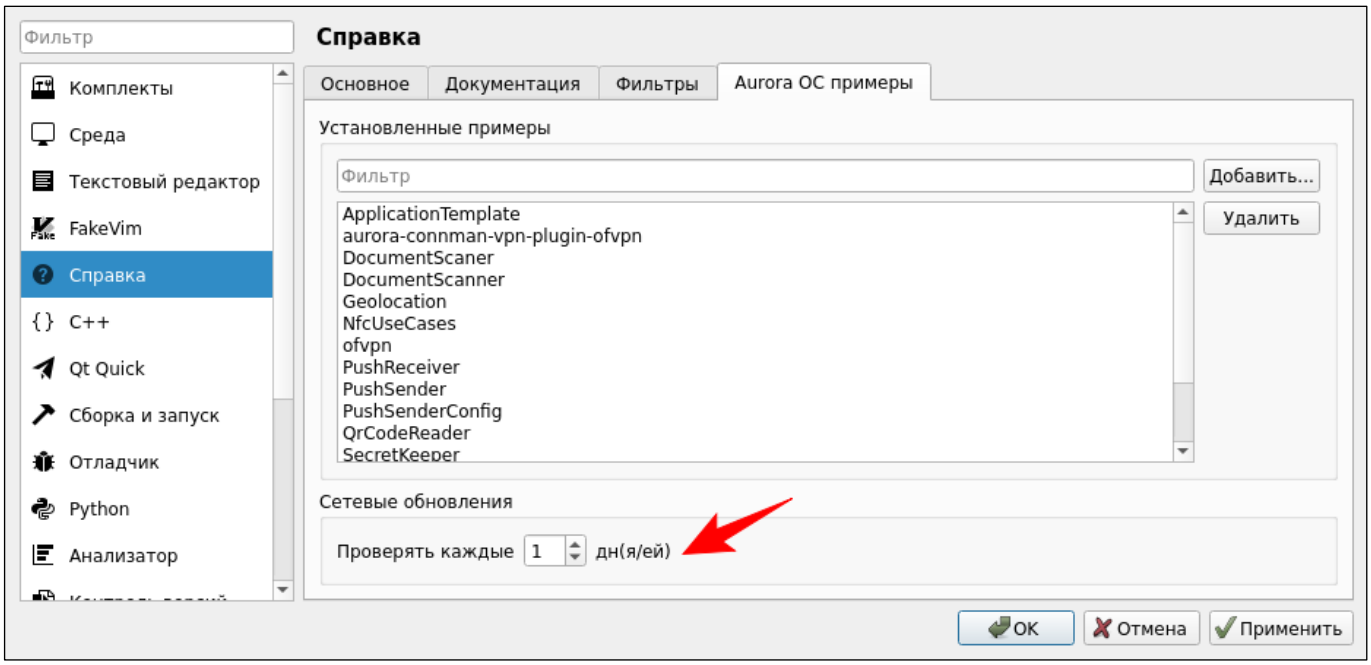


Рисунок 47

Добавить новые примеры (если они появились) можно, нажав кнопку «Добавить», выбрать нужные примеры в появившемся окне и нажать кнопку «Ок» (Рисунок 48).

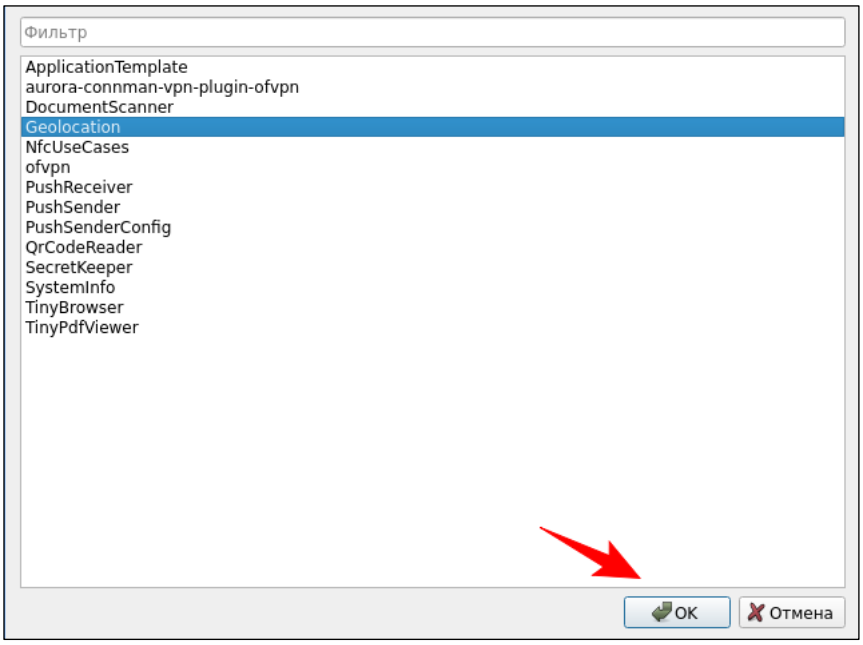


Рисунок 48

Также в окне добавления примеров можно посмотреть дополнительную информацию о примере во всплывающей подсказке (Рисунок 49) и открыть в браузере его репозиторий, нажав правой кнопкой мыши по примеру и выбрав пункт контекстного меню «Открыть расположение» (Рисунок 50).

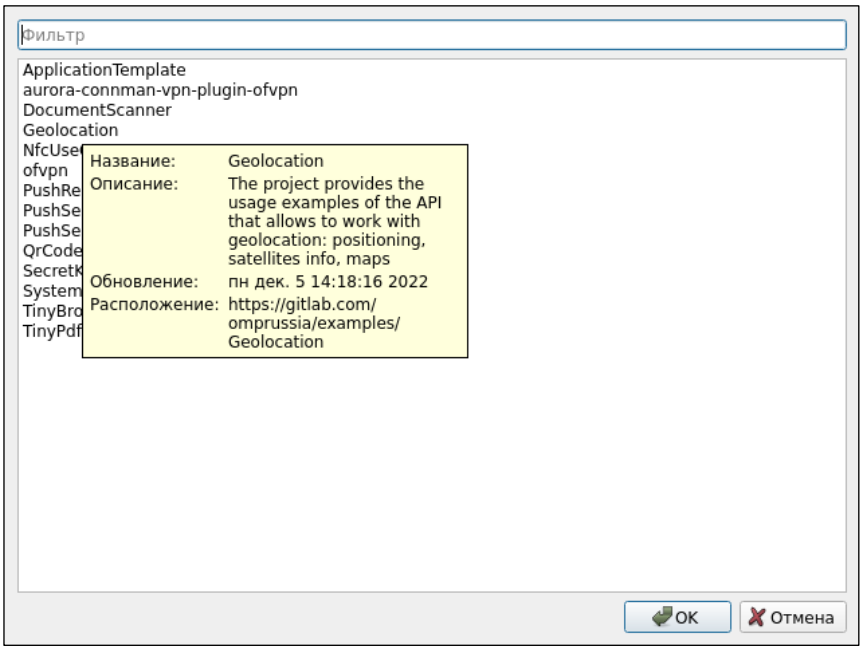


Рисунок 49

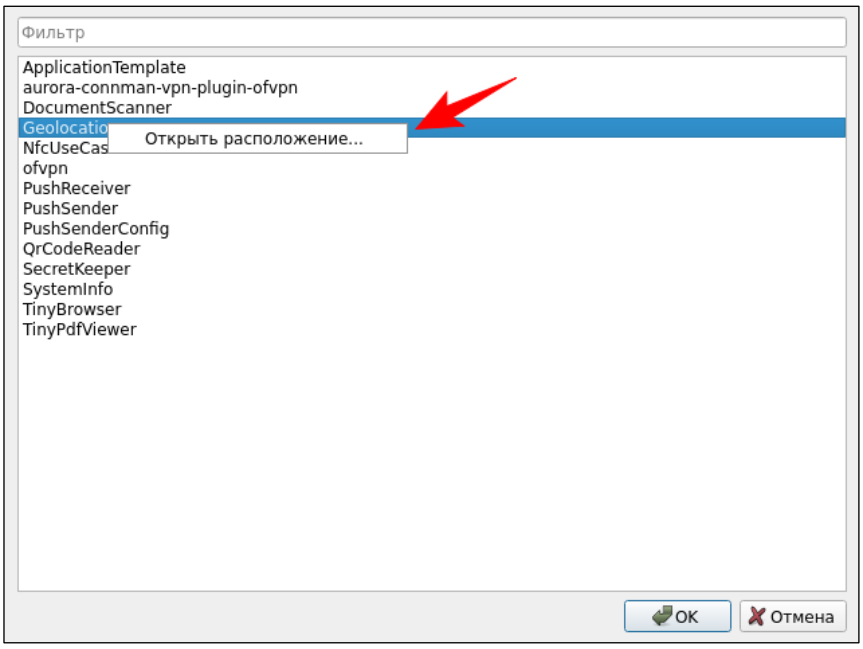


Рисунок 50

После добавления примеров следует нажать кнопку «Применить», чтобы обновить список на главном экране (Рисунок 51).

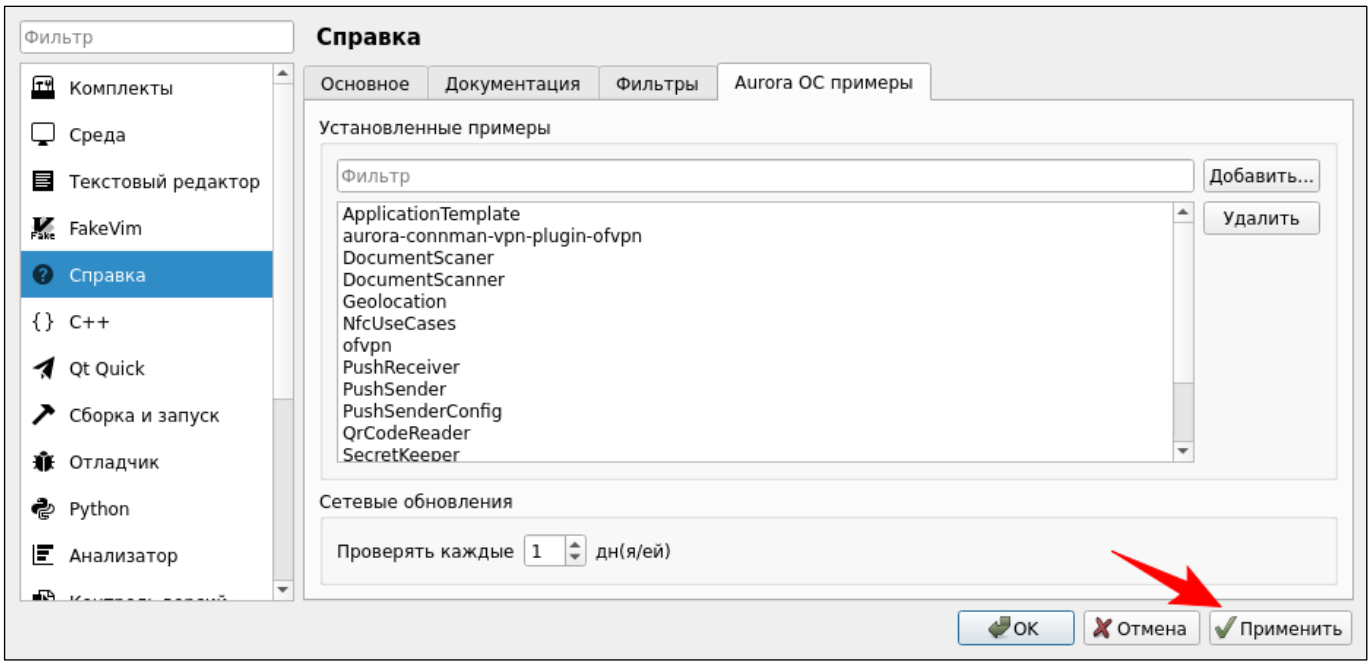


Рисунок 51

Чтобы обновить устаревшие примеры, нужно сначала их выбрать и удалить, нажав кнопку «Удалить» (Рисунок 52).

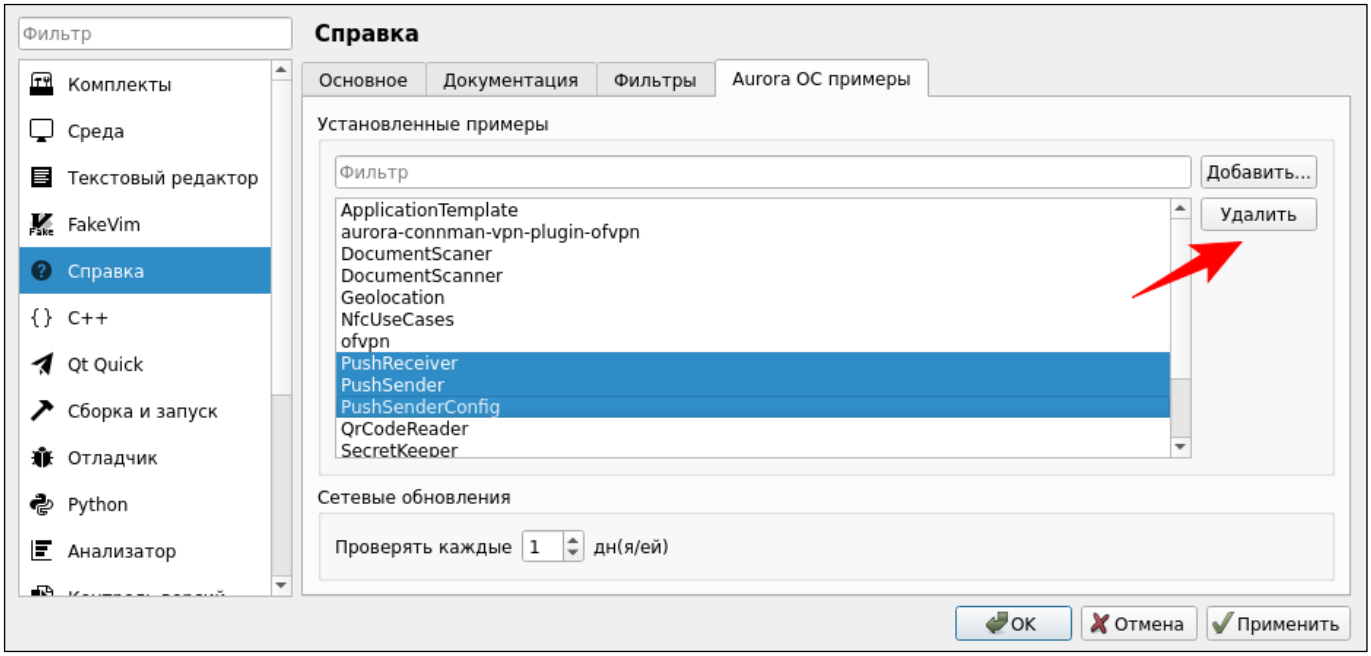


Рисунок 52

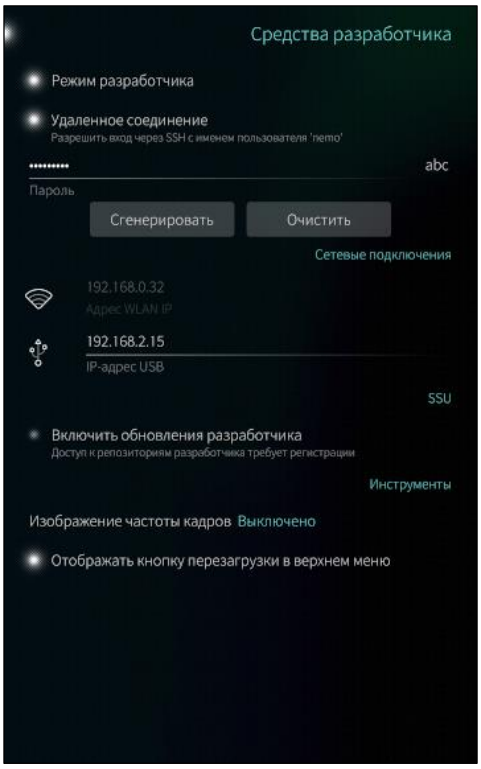
После удаления примеров следует нажать кнопку «Применить». Затем примеры нужно добавить заново, как описано выше.

6. ИНСТРУКЦИЯ ПО ПОДКЛЮЧЕНИЮ МОБИЛЬНОГО УСТРОЙСТВА ДЛЯ ОТЛАДКИ

Для подключения МУ к среде разработки необходимо подключить его к персональному компьютеру разработчика по USB-проводу. Альтернативно можно подключить МУ к той же сети WLAN, что и компьютер.

Перед добавлением МУ необходимо настроить подключение по SSH. Для этого на МУ необходимо выполнить следующие действия:

1) Перейти в «Настройки» → «Средства разработчика» → пункт «Удаленное соединение»;



2) Задать SSH-пароль и нажать кнопку «Сохранить» (Рисунок 53).

Рисунок 53

Чтобы подключить МУ к Аврора IDE, необходимо выполнить следующие действия:

1) В IDE открыть вкладку «Инструменты» → «Параметры» → «Устройства» (Рисунок 54, Рисунок 55);

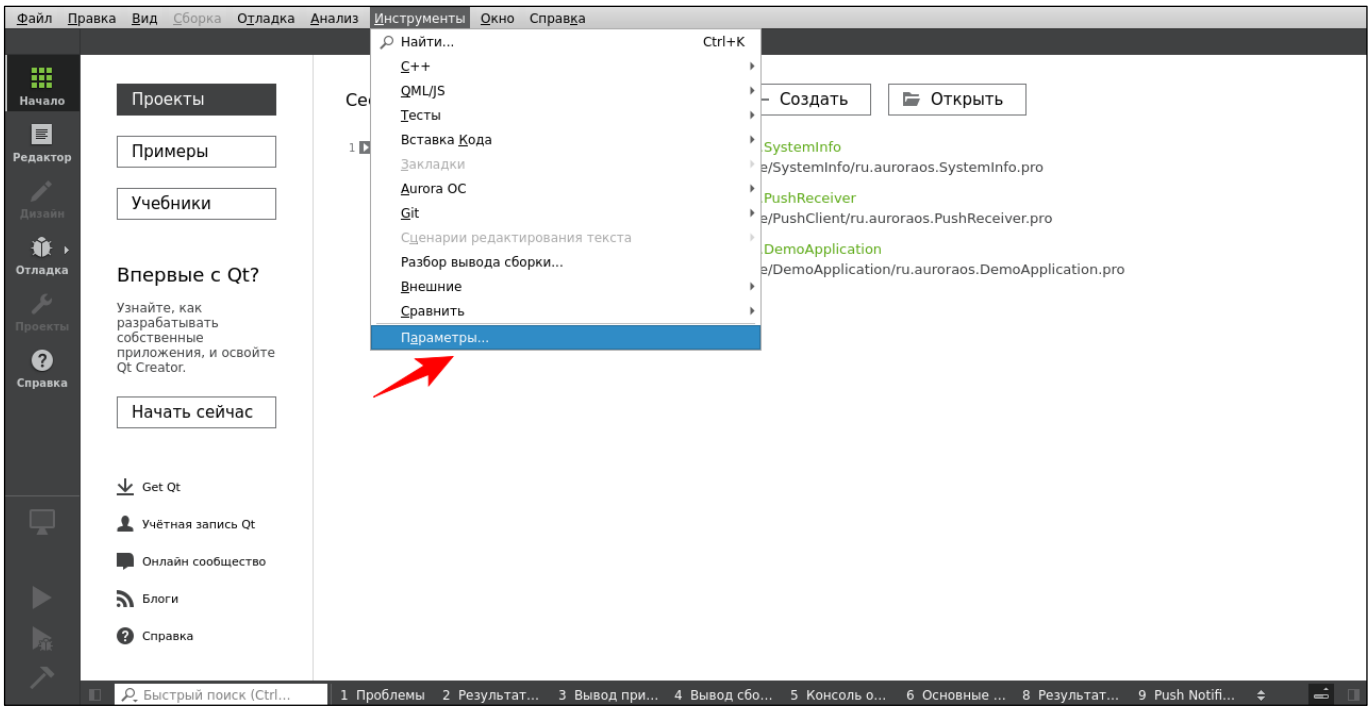


Рисунок 54

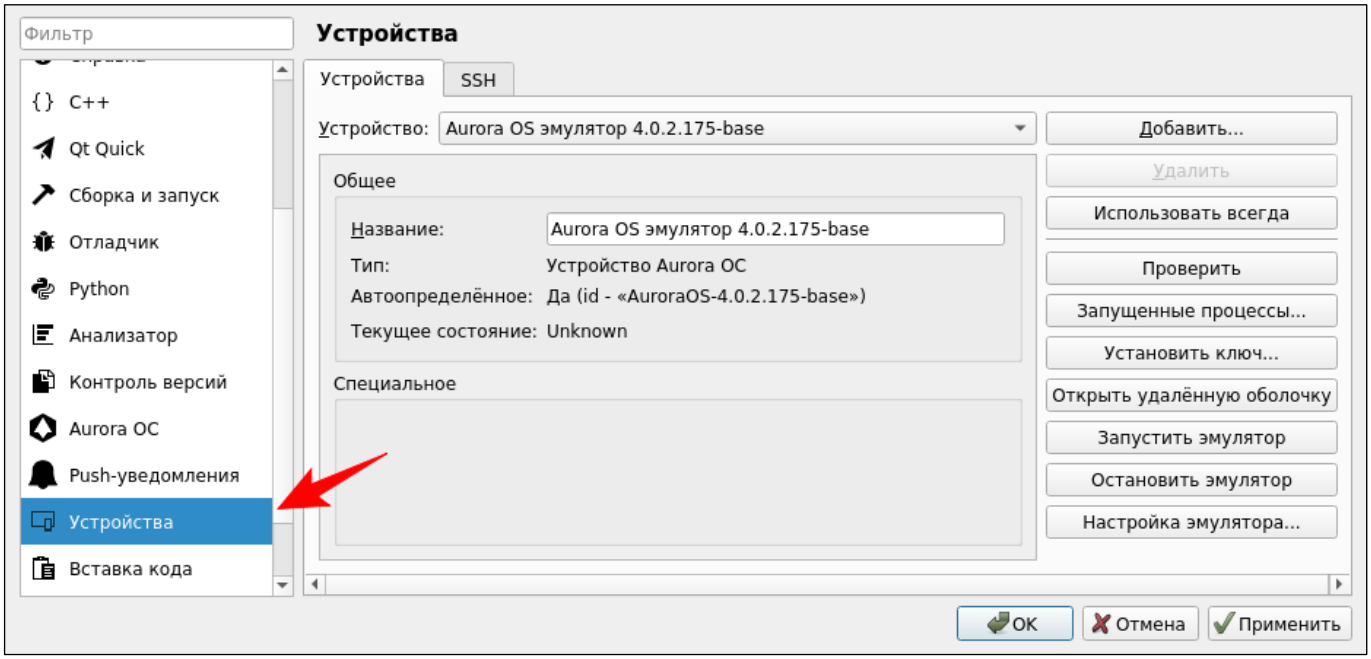


Рисунок 55

2) Нажать кнопку «Добавить устройство»;

3) Выбрать пункт «Устройство Aurora OC» и нажать кнопку «Запустить мастера» (Рисунок 56);

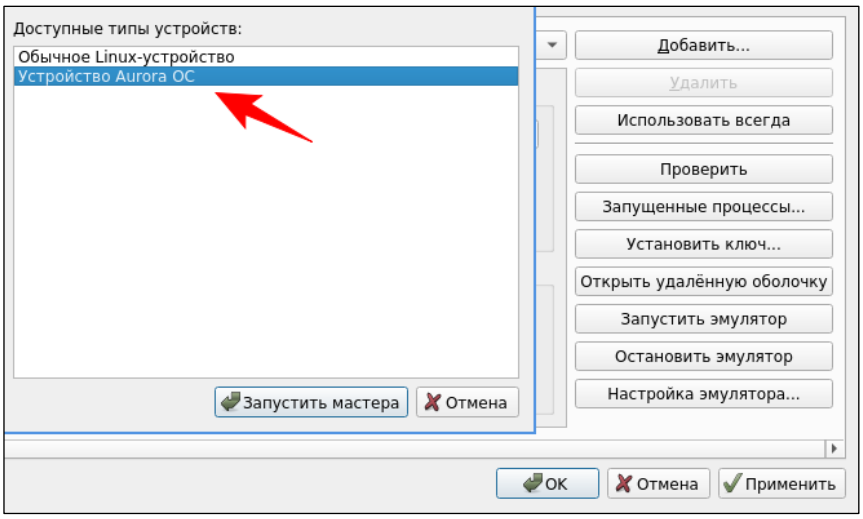


Рисунок 56

4) Ввести имя пользователя и IP-адрес МУ (по умолчанию IP-адрес SSH-соединения и пользователь `defaultuser`) (Рисунок 57);

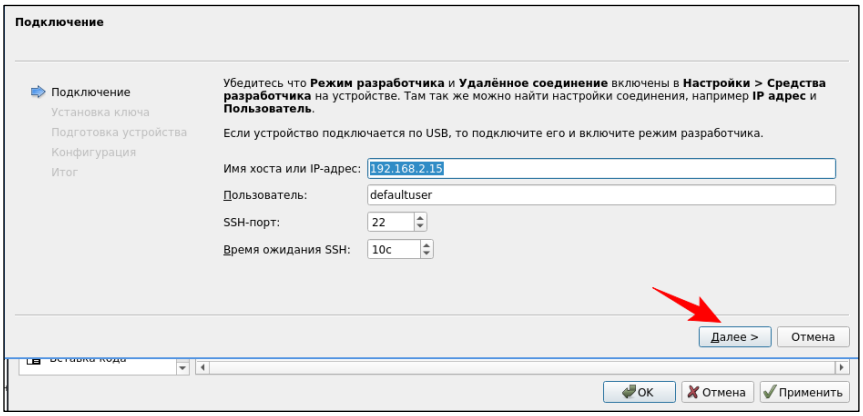


Рисунок 57

5) Создать пару ключей или выбрать существующие (Рисунок 58, Рисунок 59);

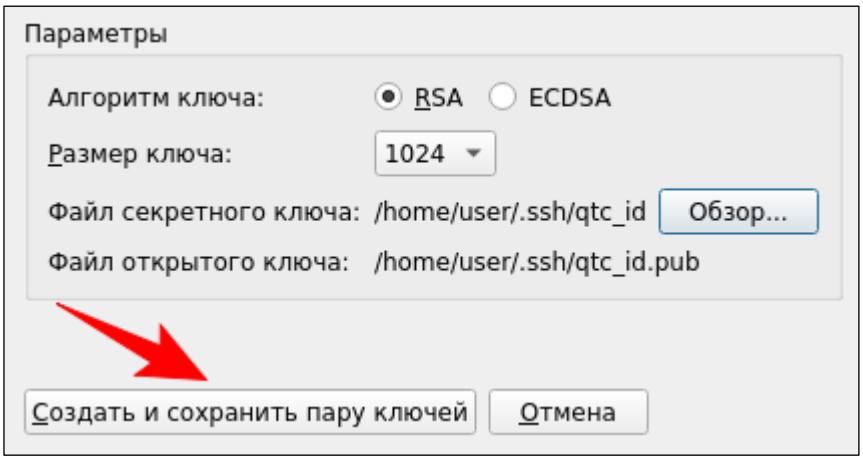


Рисунок 58

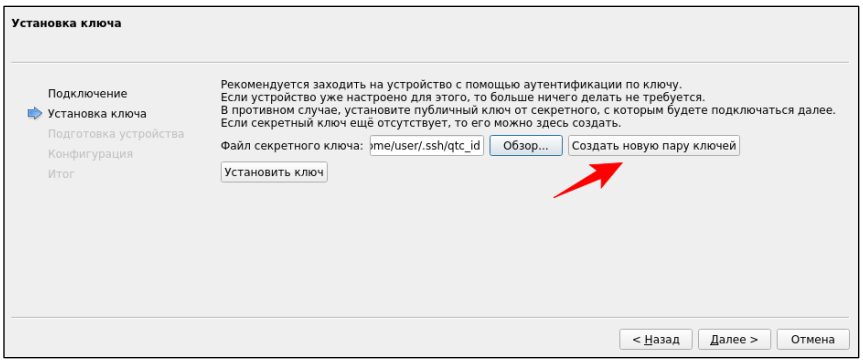


Рисунок 59

6) Нажать кнопку «Установить ключ» (см. Рисунок 59), после в диалоговом окне (Рисунок 60) ввести имя пользователя и пароль (по умолчанию это пользователь defaultuser).

Рисунок 60

Если пароль введен верно, ключ будет успешно установлен (Рисунок 61).

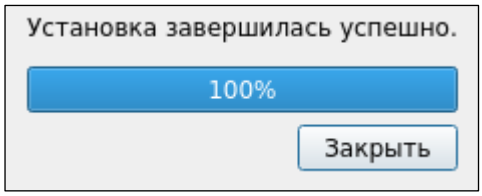


Рисунок 61

После успешной установки ключа рядом с кнопкой должна появиться зеленая галочка (Рисунок 62);

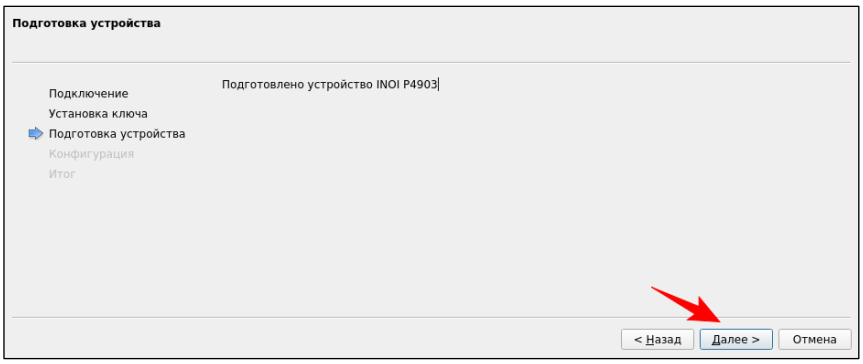


Рисунок 62

7) Нажать кнопку «Далее», после чего МУ будет подготовлено к соединению (Рисунок 63).

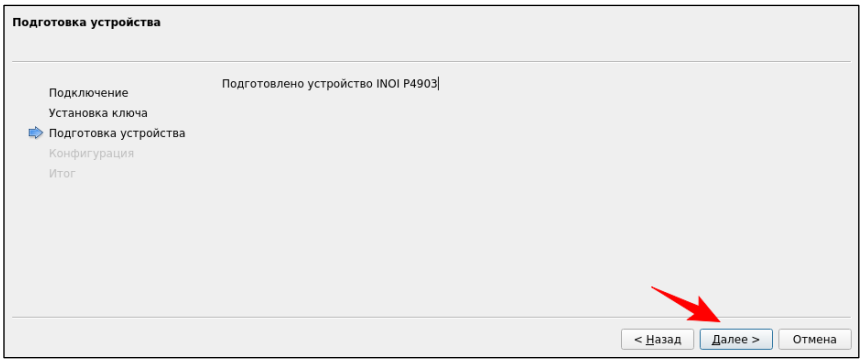


Рисунок 63

Когда подготовка завершится, снова нужно нажать кнопку «Далее»;

8) При необходимости на вкладке «Конфигурация» ввести название новой конфигурации и диапазон свободных портов, а затем нажать кнопку «Далее» (Рисунок 64). По умолчанию в названии конфигурации будет записана модель МУ и тип архитектуры. Количество свободных портов должно быть не менее двух, это необходимо для отладки;

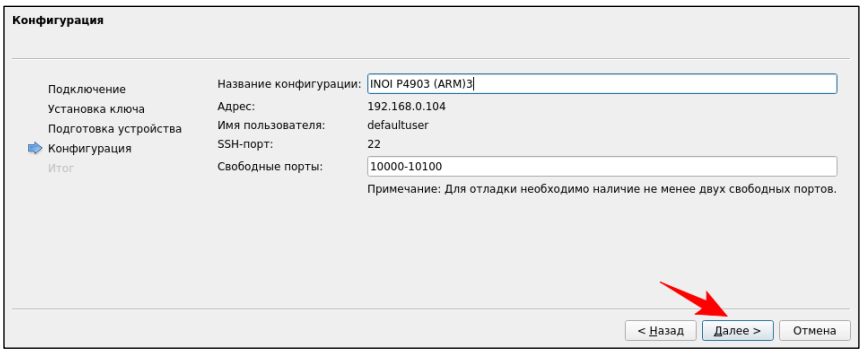


Рисунок 64

9) На вкладке «Итог» нажать кнопку «Завершить» (Рисунок 65). Подключение МУ будет завершено (Рисунок 66);

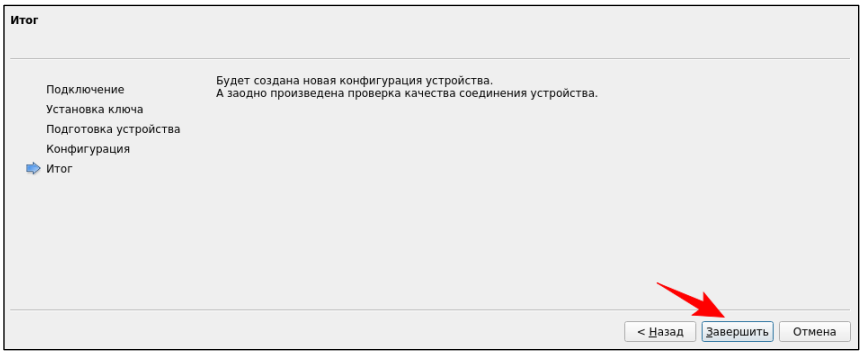


Рисунок 65

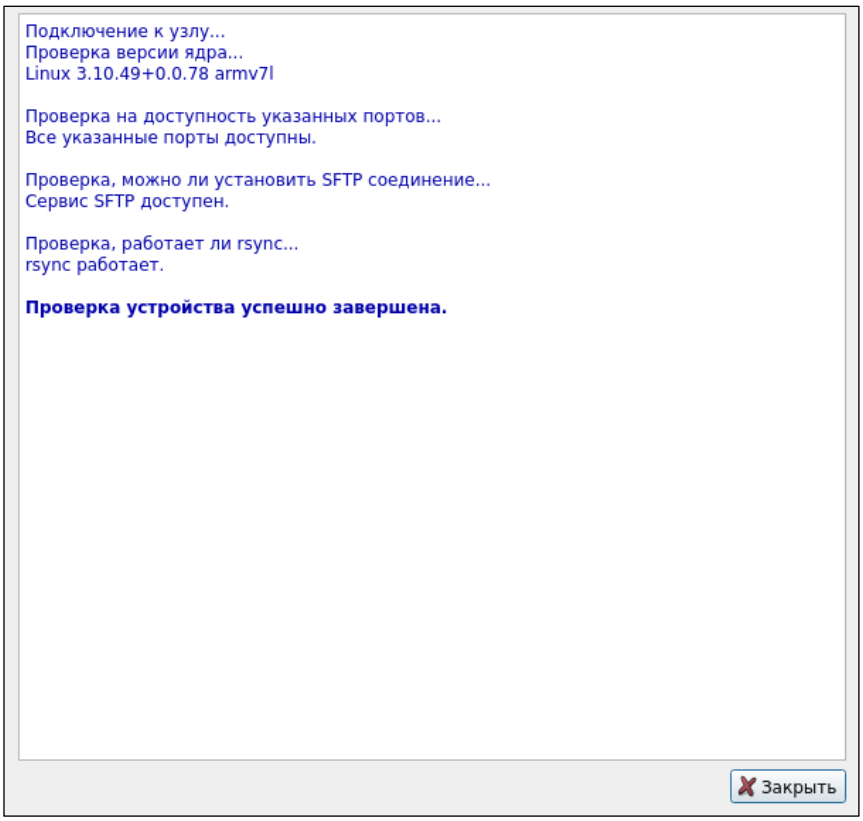


Рисунок 66

10) На вкладке «Устройства» появится новое МУ (Рисунок 67).

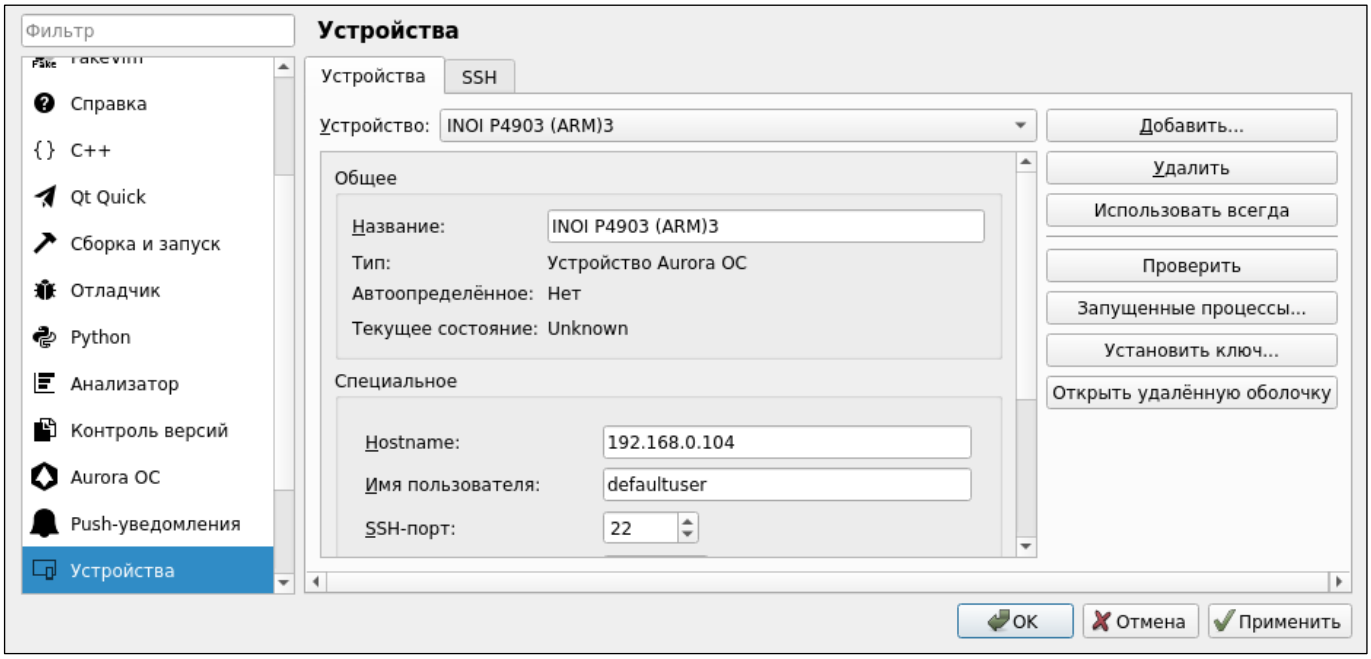


Рисунок 67

7. УПРАВЛЕНИЕ ЭМУЛЯЦИЕЙ ОС АВРОРА

Эмулятор в Аврора IDE может имитировать работу с камерой (подраздел 7.1), геопозицией (подраздел 7.2) и датчиками (подраздел 7.3).

Для запуска интерфейса пользователя для управления эмуляцией необходимо выбрать пункт меню «Инструменты» → «Аврора ОС» → «Управление эмуляцией».

При первом запуске будут установлены пакеты для эмуляции. В результате будет открыто окно для управления эмуляцией, которое разделено на две части: слева — список доступных разделов, справа — рабочая область с виджетами для управления эмуляцией в рамках выбранного раздела.

Дополнительно, эмулятор предоставляет интерфейсы D-Bus:

– GeoclueEmulationManagement (п. 7.2.3) для управления геопозицией;

– SensorfwEmulationManagement (п. 7.3.4) для управления датчиками.

Информация об управлении эмуляторами доступна в соответствующей статье на Портале разработчиков (https://developer.auroraos.ru/doc/software_development/sdk/setup/manage_emulators).

7.1. Эмуляция камер

На устройствах ОС Аврора имеется одна или несколько камер. Эмулятор ОС Аврора может имитировать работу с камерой. Средствами Аврора IDE можно запустить мастер управления эмуляцией, задать в качестве данных камеры видеофайл в формате `.webm` и проверить работу программы с камерой.

7.1.1. Управление эмуляцией камер в Аврора IDE

Для работы с эмулятором камер следует использовать раздел «Камера» (Рисунок 68).

Рисунок 68

Данный раздел содержит:

– переключатели для выбора источника данных (камеры хост-машины или видеофайла);

– текущие настройки камеры и элементы для выбора камеры, разрешения и частоты кадров из списков;

– кнопки «Установить настройки», «Обновить список камер» и «Запустить» для управления камерой»;

– поле для выбора и загрузки видеофайла;

– кнопки «Запустить» и «Остановить» для запуска и остановки передачи видеопотока.

7.1.1.1. Передача видеопотока из выбранного видеофайла

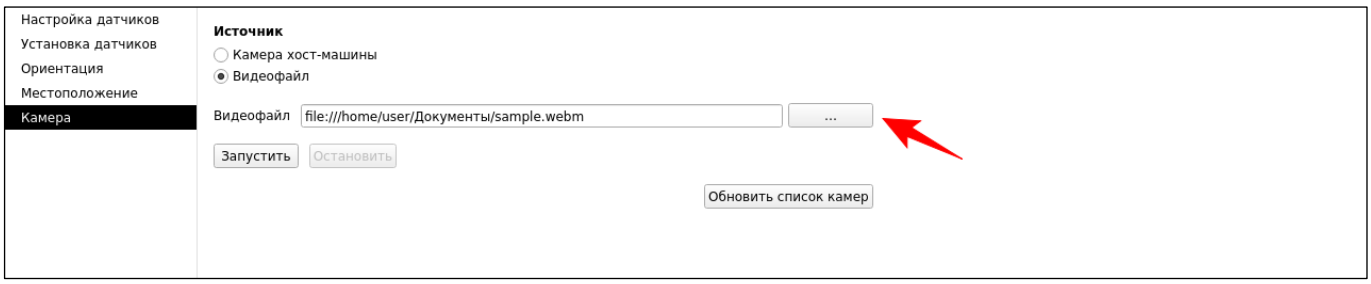


Рисунок 69

Для запуска передачи потока данных из видеофайла, необходимо выбрать пункт «Видеофайл» в разделе «Источник», нажать на кнопку «...» рядом с текстовым полем «Видеофайл» (см. Рисунок 69). Откроется диалоговое окно для выбора файлов. Для передачи данных поддерживаются файлы, закодированные в VP9, с расширением `.webm`.

После выбора конкретного файла будет запущен процесс передачи потока данных на эмулятор. Передача видеопотока из файла будет выполняться циклически, т. е. по окончанию передачи данных из файла процесс будет запущен снова.

Для остановки процесса передачи видеоданных необходимо нажать кнопку «Остановить». Для возобновления процесса передачи необходимо нажать кнопку «Запустить».

7.1.2. Работа с камерой в приложении

В проекте для работы с камерой можно использовать компонент QML Camera (https://doc.qt.io/archives/qt-5.6/qml-qtmultimedia-camera.html), класс C++ QCamera (https://doc.qt.io/archives/qt-5.6/qcamera.html) и другие классы и компоненты QtMultimedia (https://doc.qt.io/archives/qt-5.6/qtmultimedia-index.html).

Подробнее о работе с камерой можно прочитать в документации Qt.

7.1.3. Команды для проверки работы GStreamer

Если для эмуляции требуется использовать камеру у компьютера разработчика, но при старте плагина эмуляции камера не найдена автоматически, или в процессе запуска возникает ошибка, можно использовать команды из следующих разделов для диагностики работы камеры.

Следует отметить, что для эмуляции поддерживаются все камеры, с которыми работает фреймворк GStreamer. Диагностика производится с помощью команды `gst-launch-1.0`.

7.1.3.1. Команды для Windows

Для выполнения проверки работы камеры под Windows следует выполнить следующие действия:

– в консоли PowerShell вывести отфильтрованный список устройств:

```
gst-device-monitor-1.0.exe Video/Source | Select-String -Pattern
"ksvideosrc device.path"
```

Например, ответ может содержать следующую информацию о пути к файлам устройств:

```
gst-launch-1.0 ksvideosrc device-
path="\\\\\?\\usb\#vid_046d&pid_0825&mi_00\#7&1e2afdec&0&0000\#\{6994a
d05-93ef-11d0-a3cc-00a0c9223196\}\\global" ! ...
```

```
gst-launch-1.0 ksvideosrc device-
path="\\\\\?\\usb\#vid_0bda&pid_565c&mi_00\#6&a611cca&0&0000\#\{6994ad
05-93ef-11d0-a3cc-00a0c9223196\}\\global" ! ...
```

– далее выполнить команду `gst-launch-1.0` для проверки запуска веб-камеры, указав путь к файлу устройства:

- встроенная камера:

```
gst-launch-1.0 ksvideosrc device-
path="\\\\\?\\usb\#vid_0bda&pid_565c&mi_00\#6&a611cca&0&0000\#\{6994ad
05-93ef-11d0-a3cc-00a0c9223196\}\\global" ! videoconvert !
autovideosink
```

- внешняя USB-камера:

```
gst-launch-1.0 ksvideosrc device-
path="\\\\\?\\usb\#vid_046d&pid_0825&mi_00\#7&1e2afdec&0&0000\#\{6994a
d05-93ef-11d0-a3cc-00a0c9223196\}\\global" ! videoconvert !
autovideosink
```

7.1.3.2. Команды для Linux

Для выполнения проверки работы камеры на Linux следует выполнить следующие действия:

– установить пакет для работы с утилитой `v4l2-ctl`, если он не был установлен ранее:

```
sudo apt install v4l-utils
```

– вывести список доступных камер и ссылок на файлы их устройств. Например:

```
v4l2-ctl --list-devices
```

Например, ответ может содержать следующую информацию о пути к файлам устройств:

```
C270 HD WEBCAM (usb-0000:00:14.0-2.4):
        /dev/video2
        /dev/video3
        /dev/media1

Integrated_Webcam_HD: Integrate (usb-0000:00:14.0-6):
        /dev/video0
        /dev/video1
        /dev/media0
```

В данном случае, встроенная камера — это устройство /dev/video0, внешняя USB-камера — устройство /dev/video2;

– и протестировать камеру, указав путь к файлу устройства:

- встроенная камера:

```
gst-launch-1.0 v4l2src device=/dev/video0 ! videoconvert !
autovideosink
```

- внешняя USB-камера:

```
gst-launch-1.0 v4l2src device=/dev/video2 ! videoconvert !
autovideosink
```

Команда в случае успеха должна запустить камеру в отдельном окне на компьютере разработчика и вывести сообщение, например:

```
Установка конвейера в состояние PAUSED…
Конвейер работает и не требует состояния PREROLL…
Конвейер подготовлен (PREROLLED)…
Установка конвейера в состояние PLAYING…
New clock: GstSystemClock
```

7.1.3.3. Команды для MacOS

Проверить работу камеры на MacOS можно с помощью команды:

```
gst-launch-1.0 avfvideosrc ! autovideosink
```

Пример вывода:

```
Установка конвейера в состояние PAUSED…
Конвейер работает и не требует состояния PREROLL…
Установка конвейера в состояние PLAYING…
New clock: GstSystemClock
^Chandling interrupt.
Прерывание: Остановка конвейера…
Execution ended after 0:00:11.129066355
Установка конвейера в состояние NULL…
Освобождение конвейера…
```

7.2. Эмуляция местоположения

На устройствах ОС Аврора можно получить данные о геолокации устройства с помощью классов и компонентов из модулей Qt Positioning (https://doc.qt.io/archives/qt-5.6/qtpositioning-index.html) и Qt Location (https://doc.qt.io/archives/qt-5.6/qtlocation-index.html).

Эмулятор ОС Аврора может также имитировать работу с местоположением. Средствами Аврора IDE можно запустить мастер управления эмуляцией, задать местоположение или GPS-трек для эмуляции перемещения и проверить работу программы с геопозицией.

7.2.1. Управление эмуляцией местоположения в Аврора IDE

Для эмуляции местоположения можно использовать раздел «Местоположение» (Рисунок 70).

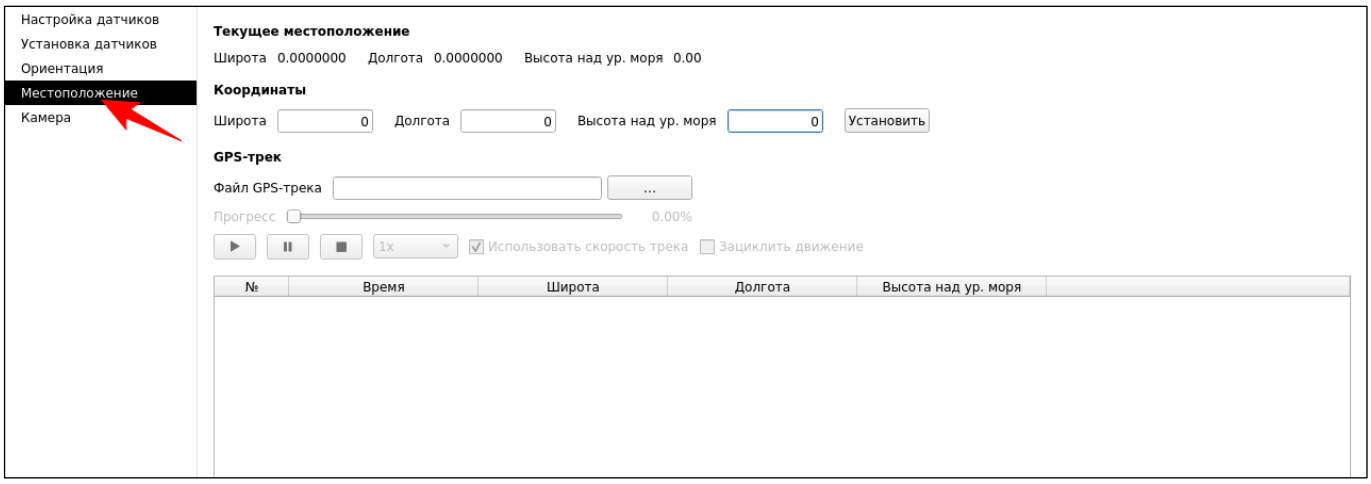


Рисунок 70

Раздел содержит:

– текстовые поля, отображающие координаты текущей позиции (широту, долготу и высоту над уровнем моря);

– поля для ввода координат новой текущей позиции (координаты вводятся в формате «ГГ.ммсс», где ГГ - радусы, мм - минуты, сс - секунды);

– поле для загрузки файла GPS-трека в формате NMEA;

– таблицу, отображающую точки загруженного GPS-трека (номер, время, широту, долготу и высоту над уровнем моря);

– слайдер для отображения (в процентах) и установки текущего положения на треке;

– кнопки «Воспроизведение», «Пауза» и «Стоп» для управления движением по загруженному GPS-треку;

– компоненты для настройки скорости передвижения по загруженному GPS-треку и зацикливания движения.

7.2.1.1. Установка новой текущей позиции

Для установки новой текущей позиции необходимо ввести новые координаты в поля «Широта», «Долгота» и «Высота над уровнем моря» и нажать на кнопку «Установить» (Рисунок 71).

Диапазоны допустимых значений приведены в таблице (Таблица 1).

Таблица 1

Параметр	Мин	Макс
Широта	-90°	90°
Долгота	-180°	180°
Высота	-1000м	10000м

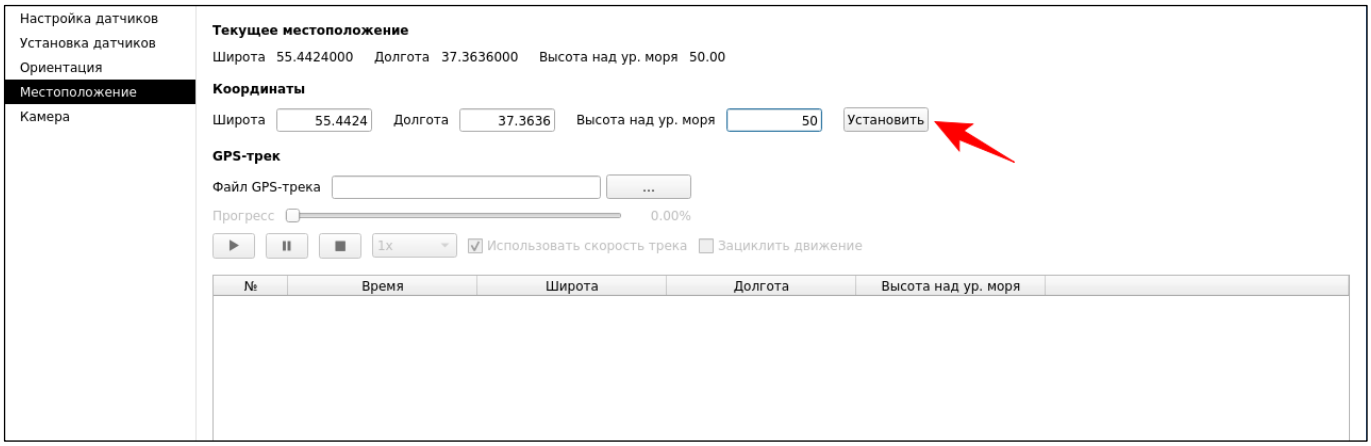


Рисунок 71

Если установка новой текущей позиции была выполнена во время движения по GPS-треку, то движение по треку будет приостановлено (поставлено на паузу).

7.2.1.2. Эмуляция движения по GPS-треку

Для загрузки GPS – трека в формате NMEA (https://web.archive.org/web/20070322070358/http://www.tronico.fi/OH6NT/docs/NMEA0183.pdf) необходимо воспользоваться кнопкой «...» рядом с полем «Файл GPS-трека» (Рисунок 72). Откроется диалоговое окно для выбора файла с расширением `.nmea`.

После выбора файла будет выполнена его загрузка, таблица в окне плагина Аврора IDE будет заполнена информацией о точках GPS-трека.

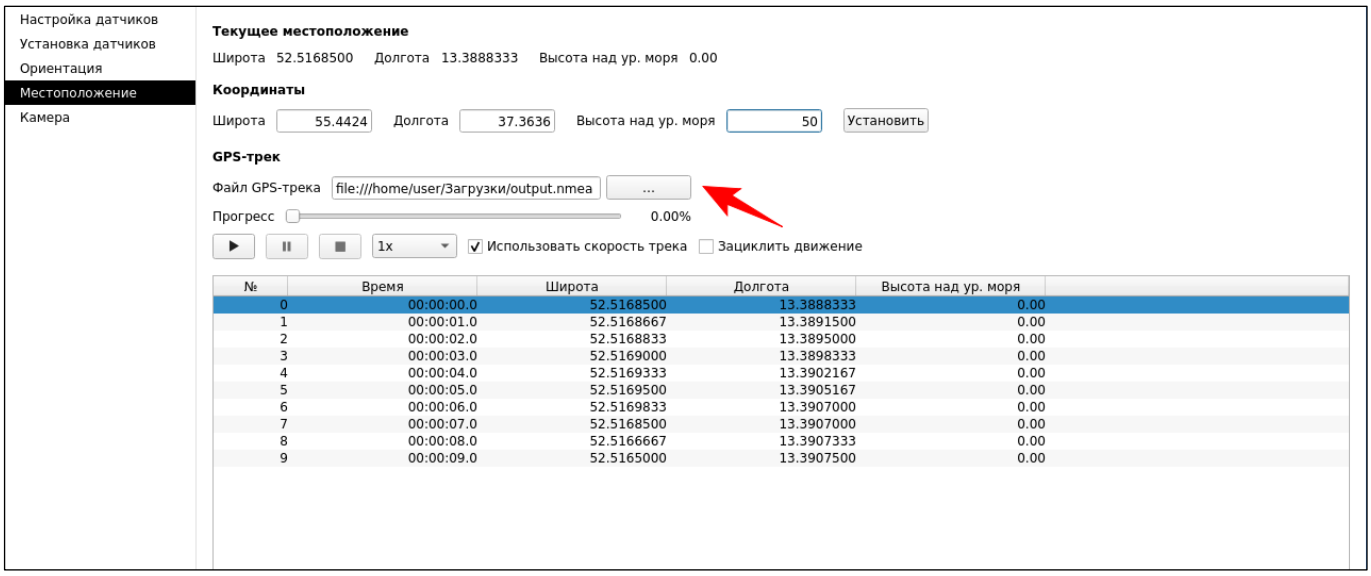


Рисунок 72

Нажатие на кнопку «Воспроизведение» запускает движение по загруженному GPS-треку. Нажатие на кнопку «Пауза» приостанавливает движение по треку. Нажатие на кнопку «Стоп» полностью останавливает движение по треку. Если движение по треку приостановлено (поставлено на паузу), то при нажатии на кнопку «Воспроизведение» движение будет возобновлено с той точки, на которой оно было приостановлено.

Слайдер «Прогресс» отображает текущий прогресс движения по треку в процентах, а также позволяет выполнять перемещение по треку. Также выполнять перемещение к конкретной позиции на треке можно с помощью двойного нажатия на строку в таблице с точками GPS-трека.

Выпадающий список с элементами 1x, 2x, 3x, 4x и 5x позволяет настраивать текущую скорость движения по GPS-треку.

Переключатель «Использовать скорость трека» позволяет менять режим скорости движения по треку. Если переключатель включен, то будет использоваться скорость самого трека, которая определяется временными интервалами между точками. Если же переключатель выключен, то будет использована стандартная скорость движения по треку 1 точка в 2 секунды.

Переключатель «Зациклить движение» позволяет сделать движение по GPS-треку зацикленным.

7.2.2. Работа с местоположением в приложении

В проекте для работы с местоположением можно использовать:

– QML-компоненты:

- Position (https://doc.qt.io/archives/qt-5.6/qml-qtpositioning-position.html),
- PositionSource (https://doc.qt.io/archives/qt-5.6/qml-qtpositioning-positionsource.html),
- coordinate (https://doc.qt.io/archives/qt-5.6/qml-coordinate.html),

– классы C++:

- QGeoCoordinate (https://doc.qt.io/archives/qt-5.6/qgeocoordinate.html),
- QNmeaPositionInfoSource (https://doc.qt.io/archives/qt-5.6/qnmeapositioninfosource.html);

– и другие классы и компоненты модулей Qt Positioning (https://doc.qt.io/archives/qt-5.6/qtpositioning-index.html) и Qt Location (https://doc.qt.io/archives/qt-5.6/qtlocation-index.html).

7.2.3. D-Bus-сервис для управления эмуляцией местоположения

Сервис (Таблица 2) позволяет управлять эмуляцией местоположения. Подробнее в пп. 7.2.3.2.

Таблица 2

Шина:	сессионная
Служба:	ru.omp.GeoclueEmulationManagement
Объект:	/ru/omp/GeoclueEmulationManagement
Интерфейс:	ru.omp.GeoclueEmulationManagement

7.2.3.1. Методы

Методы эмуляции местоположения:

- goToPositionOnTrack(int index);
- loadTrack(variant[] positions);
- pauseTrack();
- resumeTrack();
- setAltitude(double altitude);
- setLatitude(double latitude);
- setLongitude(double longitude);
- setPosition(double latitude, double longitude, double altitude);
- setTrackIntervalMode(bool useDefaultInterval);
- setTrackLooped(bool looped);
- setTrackSpeed(int speed);
- startTrack();
- stopTrack().

7.2.3.2. Подробное описание

Сервис позволяет управлять эмуляцией местоположения: устанавливать координаты и настраивать GPS-трек. Эта функциональность эмулятора предназначена для экспертов.

Примеры использования DBus-сервиса с помощью вызова утилиты `dbus-send`:

- установка новой текущей позиции (широты, долготы, высоты):

```
dbus-send --session --type=method_call --print-reply --
dest=ru.omp.GeoclueEmulationManagement
/ru/omp/GeoclueEmulationManagement
ru.omp.GeoclueEmulationManagement.setPosition double:-1.1
double:54.2345 double:100
```

– запуск движения по загруженному GPS-треку:

```
dbus-send --session --type=method_call --print-reply --
dest=ru.omp.GeoclueEmulationManagement
/ru/omp/GeoclueEmulationManagement
ru.omp.GeoclueEmulationManagement.startTrack
```

– переход к позиции на загруженном GPS-треке:

```
dbus-send --session --type=method_call --print-reply --
dest=ru.omp.GeoclueEmulationManagement
/ru/omp/GeoclueEmulationManagement
ru.omp.GeoclueEmulationManagement.goToPositionOnTrack int32:12
```

Взаимодействие Аврора IDE и эмулятора для эмуляции местоположения и датчиков устроено не только через службы DBus, но и с помощью передачи сообщений по WebSockets.

Утилита `dbus-send` не поддерживает параметры методов в виде массивов и объектов `variant`. Поэтому для загрузки треков необходимо использовать другой способ, например, отправить сообщение по WebSockets на порт 1234 следующего формата:

```
session serviceName objectPath interfaceName methodName
methodArguments
```

Например:

```
session ru.omp.GeoclueEmulationManagement
/ru/omp/GeoclueEmulationManagement ru.omp.GeoclueEmulationManagement
loadTrack
array:array:double:1.1,2.2,30,1000;2,3,4,2000;10,11,12,1000;0,1,3,1000
;22,33,44,2000;70,111,121,1000
```

Здесь `array:array:double` означает, что метод принимает массив массивов действительных чисел. Далее перечислены загружаемые точки GPS-трека. Точкой с запятой отделяются точки на треке, а только запятыми отделяются широта, долгота, высота и интервал конкретной точки.

7.2.3.3. Описание методов

```
void goToPositionOnTrack(int index)
```

Меняет текущую позицию на GPS-треке на позицию по переданному индексу в списке предзагруженных позиций. `index` — индекс позиции в списке позиций, загруженных методом `loadTrack()`. После смены текущей позиции на треке движение будет приостановлено (будет выставлена пауза). Если переданный индекс имеет значение меньшее, чем 0, то DBus-метод вернет ошибку.

```
void loadTrack(variant[] positions)
```

Загружает GPS-трек для дальнейшей эмуляции движения по нему. `positions` — список из элементов-точек GPS-трека, где каждая точка (`variant`) — это массив, содержащий 4 вещественных числа:

- широту (`latitude`);
- долготу (`longitude`);
- высоту (`altitude`);
- интервал от предыдущей до данной позиции в миллисекундах (`interval`).

Таким образом, `positions` — это массив из массивов чисел, где каждый элемент содержит 4 числа.

```
void pauseTrack()
```

Ставит движение по GPS-треку на паузу. Если движение по GPS-треку не производится, то метод ничего не делает.

```
void resumeTrack()
```

Возобновляет движение по GPS-треку, если оно приостановлено. Если движение по GPS-треку выполняется в данный момент или движение остановлено полностью, то метод ничего не делает.

```
void setAltitude(double altitude)
```

Устанавливает новое значение высоты над уровнем моря в метрах. Диапазон допустимых значений: `[-1000:10000]`.

```
void setLatitude(double latitude)
```

Устанавливает новое значение широты в градусах. Диапазон допустимых значений: `[-90:90]`.

```
void setLongitude(double longitude)
```

Устанавливает новое значение долготы в градусах. Диапазон допустимых значений: `[-180:180]`.

```
void setPosition(double latitude, double longitude, double altitude)
```

Устанавливает новые значения широты и долготы в градусах, а также высоты над уровнем моря в метрах. Диапазон допустимых значений широты: `[-90:90]`. Диапазон допустимых значений долготы: `[-180:180]`.

```
void setTrackIntervalMode(bool useDefaultInterval)
```

Устанавливает режим использования временных интервалов между точками GPS-трека. `useDefaultInterval` — `true`, если необходимо использовать интервал между точками по умолчанию (2 секунды), `false` — если нужно использовать интервалы, указанные в списке загруженных позиций трека.

```
void setTrackLooped(bool looped)
```

Делает движение по GPS-треку зацикленным. `looped` — `true`, если трек необходимо зациклить, в противном случае — `false`.

```
void setTrackSpeed(int speed)
```

Устанавливает скорость движения по GPS-треку. `speed` — множитель скорости движения. Диапазон допустимых значений: `[1:5]`.

```
void startTrack()
```

Запускает эмуляцию движения по загруженному GPS-треку. Если GPS-трек не был загружен или движение по GPS-треку уже выполняется, то метод ничего не делает. Если движение по GPS-треку приостановлено (стоит на паузе), то метод продолжает движение по треку. Если движение по GPS-треку остановлено полностью (вызван `stopTrack()`), то метод начинает движение по треку сначала.

```
void stopTrack()
```

Полностью останавливает движение по GPS-треку. Если движение по GPS-треку не производится, то метод ничего не делает.

7.3. Эмуляция встроенных датчиков (гироскоп, компас, акселерометр и т.д.)

На устройствах ОС Аврора имеются встроенные датчики, например, гироскоп, компас, акселерометр. Эмулятор ОС Аврора не имеет реальных датчиков, но может имитировать работу с ними. Средствами Аврора IDE можно запустить плагин управления эмуляцией, задать различные значения датчиков и проверить работу программы.

7.3.1. Управление эмуляцией в Аврора IDE

Для взаимодействия с датчиками можно использовать три раздела (Рисунок 73):

– «Настройка датчиков», содержит список переключателей для включения/выключения каждого из датчиков;

– «Установка датчиков», содержит виджеты (ползунки, текстовые поля, выпадающие списки) для изменения значений конкретных датчиков;

– «Ориентация», содержит виджеты для управления вращением и движением устройства.

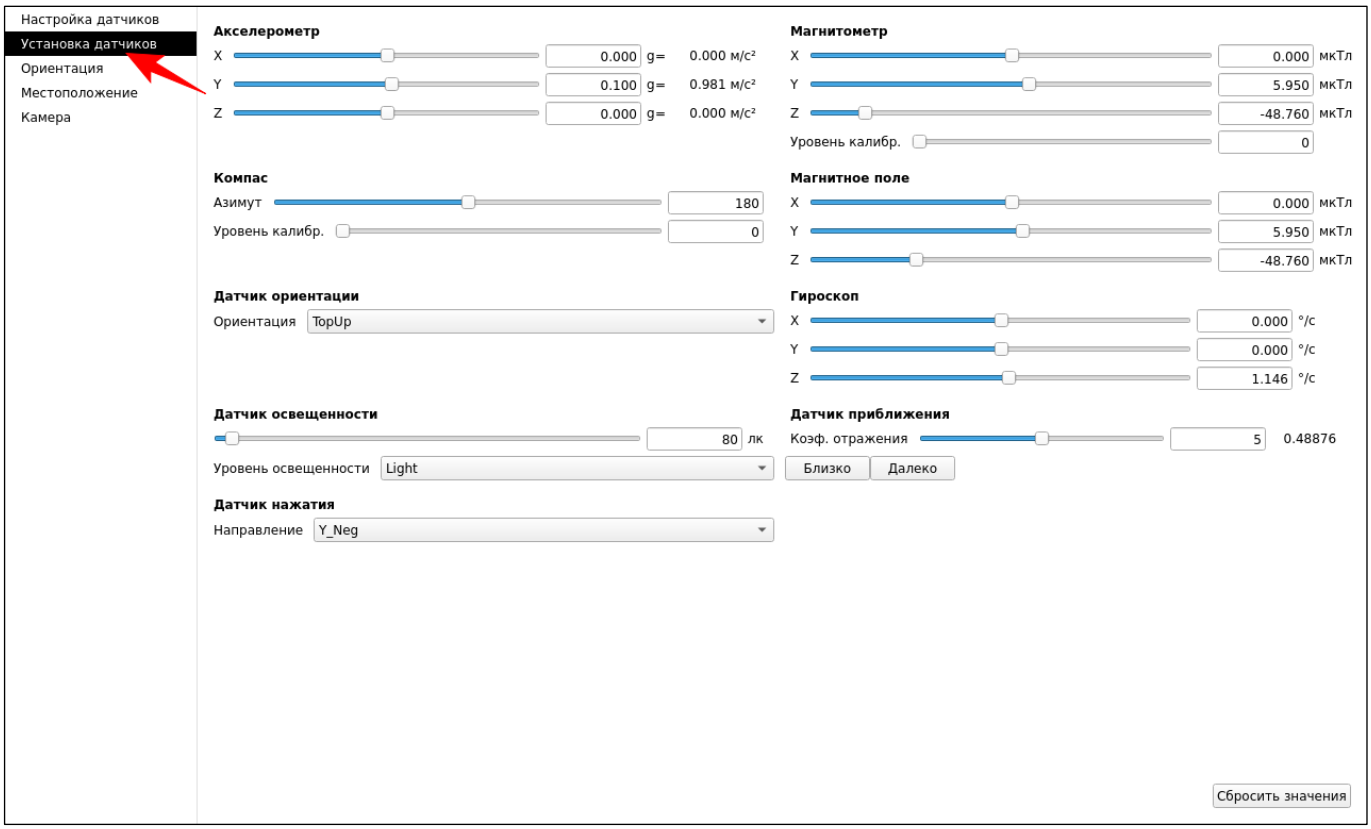


Рисунок 73

7.3.1.1. Включение/выключение датчиков

Для управления состояниями датчиков (включения/выключения) необходимо перейти в раздел «Настройка датчиков» (Рисунок 74).

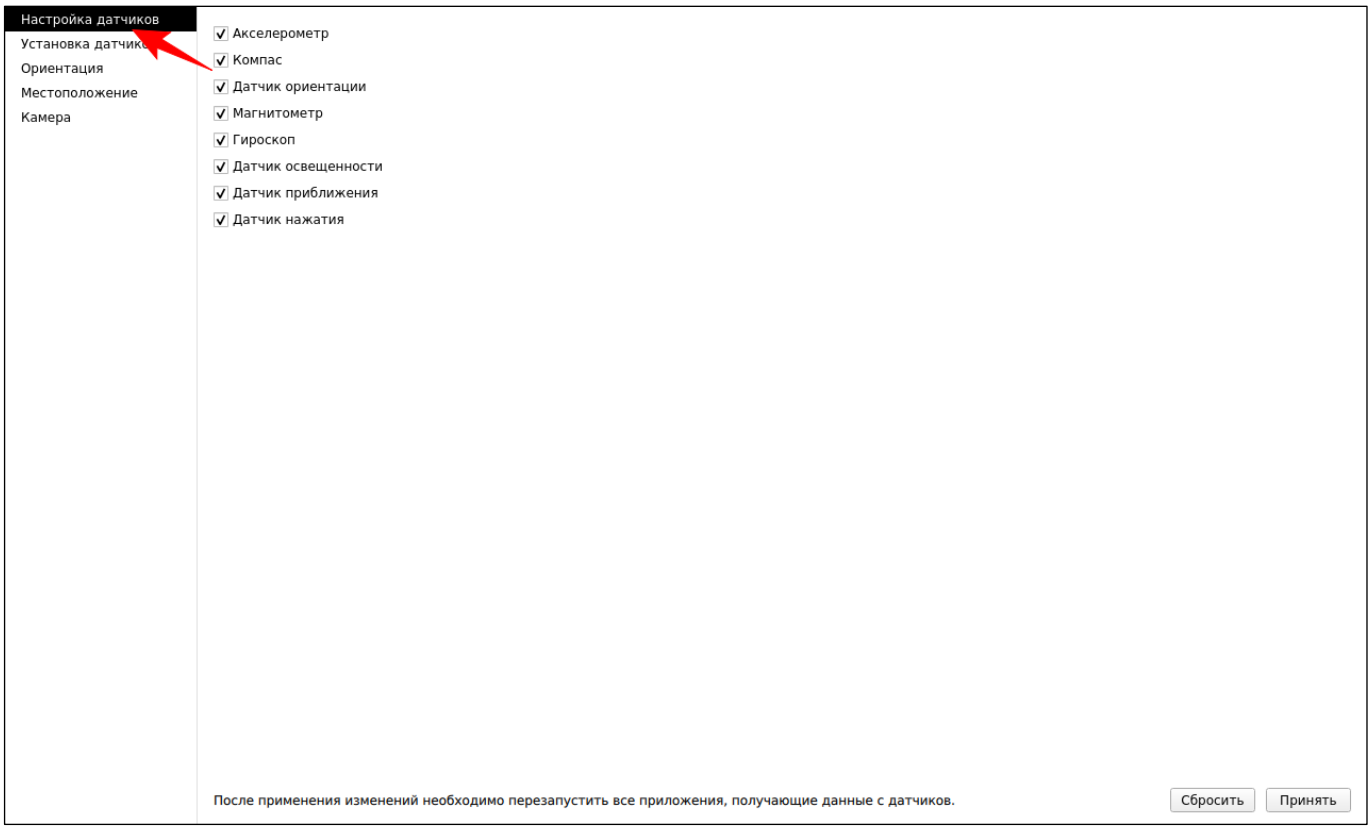


Рисунок 74

Рабочая область данного раздела содержит список из переключателей, по одному для каждого датчика.

Для изменения состояния датчиков, необходимо последовательно выполнить следующие шаги:

1) Поставить или убрать галочки на переключателях тех датчиков, которые необходимо включить или выключить;

2) Нажать кнопку «Принять» для применения изменений.

Для отмены изменений и возврата переключателей к предыдущему принятому состоянию следует нажать кнопку «Сбросить».

После применения изменений виджеты датчиков, помеченных как выключенные, станут недоступными в разделах «Установка датчиков» и «Ориентация».

После применения изменений необходимо перезапустить все приложения на эмуляторе, работающие с датчиками.

7.3.1.2. Зависимость датчиков друг от друга

Значения некоторых датчиков зависят от других. Компас зависит от акселерометра и магнитометра. Датчик ориентации зависит от акселерометра, Датчик поворота зависит от акселерометра и компаса, и от магнитометра.

Остальные датчики не являются ни зависимыми, ни теми, от которых зависят.

Изменение состояния датчиков будет влиять на состояние зависимых датчиков и наоборот:

– при выключении акселерометра будут выключены компас, датчики ориентации и поворота, так как они зависят от него и не могут без него работать.

– при выключении магнитометра будут выключены компас и датчик поворота, так как они зависят от него.

– при включении компаса будут включены акселерометр и магнитометр, от которых он зависит.

– при включении датчика ориентации будет включен акселерометр, от которого он зависит.

– при включении датчика поворота будут включены акселерометр, компас и магнитометр, от которых он зависит.

7.3.1.3. Изменение значений датчиков

Для изменения значений датчиков необходимо перейти в раздел «Установка датчиков» (Рисунок 75). Рабочая область данного раздела содержит наборы виджетов для каждого из датчиков.

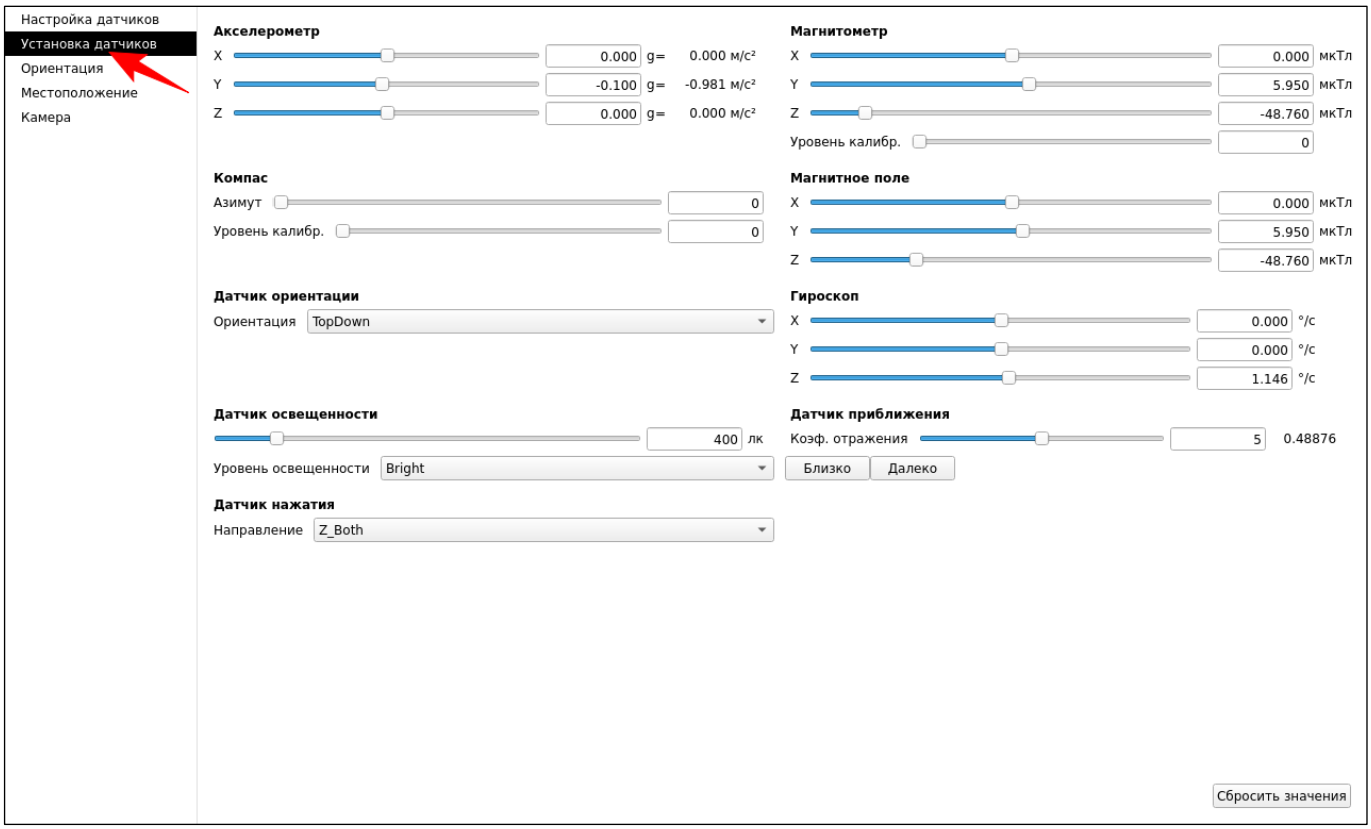


Рисунок 75

Ползунки позволяют менять значения в рамках определенного диапазона значений. Они связаны с текстовыми полями, где указаны текущие установленные значения. При редактировании значений в текстовом поле изменения принимаются по нажатию на кнопку «Enter», либо при потере фокуса данным текстовым полем. Выпадающие списки позволяют выбрать одно из заранее определенных значений.

При изменении какого-либо значения какого-либо датчика будет выполнен запрос к эмулятору на установку данного значения.

7.3.1.4. Особенности значений датчиков

Значения датчиков имеют следующие особенности:

1) Значения акселерометра.

Значения акселерометра задаются в долях g (ускорения свободного падения, взятого равным 9.80665).

Ползунки и текстовые поля для акселерометра позволяют вводить количество долей *g*, которые пользователь хочет задать. Справа от текстового поля указываются значения акселерометра в м/с², которое будет в результате установлено на эмуляторе;

2) Значения уровня калибровки магнитометра и компаса.

Значения уровня калибровки магнитометра и компаса имеют диапазон `[0:3]`, однако результирующие значения на эмуляторе будут иметь диапазон `[0:1]`, так как они будут равны установленному значению, деленному на 3 (а именно, `[0, 0.(3), 0.(6), 1]`). Это связано с особенностями плагина Sensorfw для магнитометра;

3) Значения датчика освещенности.

Датчик освещенности имеет шесть заранее определенных значений (в люксах), приведенных в таблице (Таблица 3);

Таблица 3

Параметр	Мин, лк	Макс, лк
Undefined	-1	-1
Dark	0	9
Twilight	10	79
Light	80	399
Bright	400	2499
Sunny	2500	3000

4) Значения датчика приближенности.

Значение датчика приближенности задается как степень приближения и имеет диапазон `[0:10]`, однако результирующее значение на эмуляторе будет иметь диапазон `[0:1]`. Это также связано с особенностями плагина Sensorfw, как и в случае с установкой уровня калибровки магнитометра.

7.3.2. Ориентация

На вкладке «Ориентация» можно управлять вращением и движением модели устройства (Рисунок 76). Внизу расположены элементы для настройки:

– радиокнопки для переключения между ползунками вращения или движения;

– ползунки для трех осей вращения или движения, значения которых влияют на значения акселерометра (справа), гироскопа и магнитометра;

– кнопки для поворота устройства в одну из шести позиций, которые устанавливают значения датчика ориентации, а также значения ползунков и акселерометра.

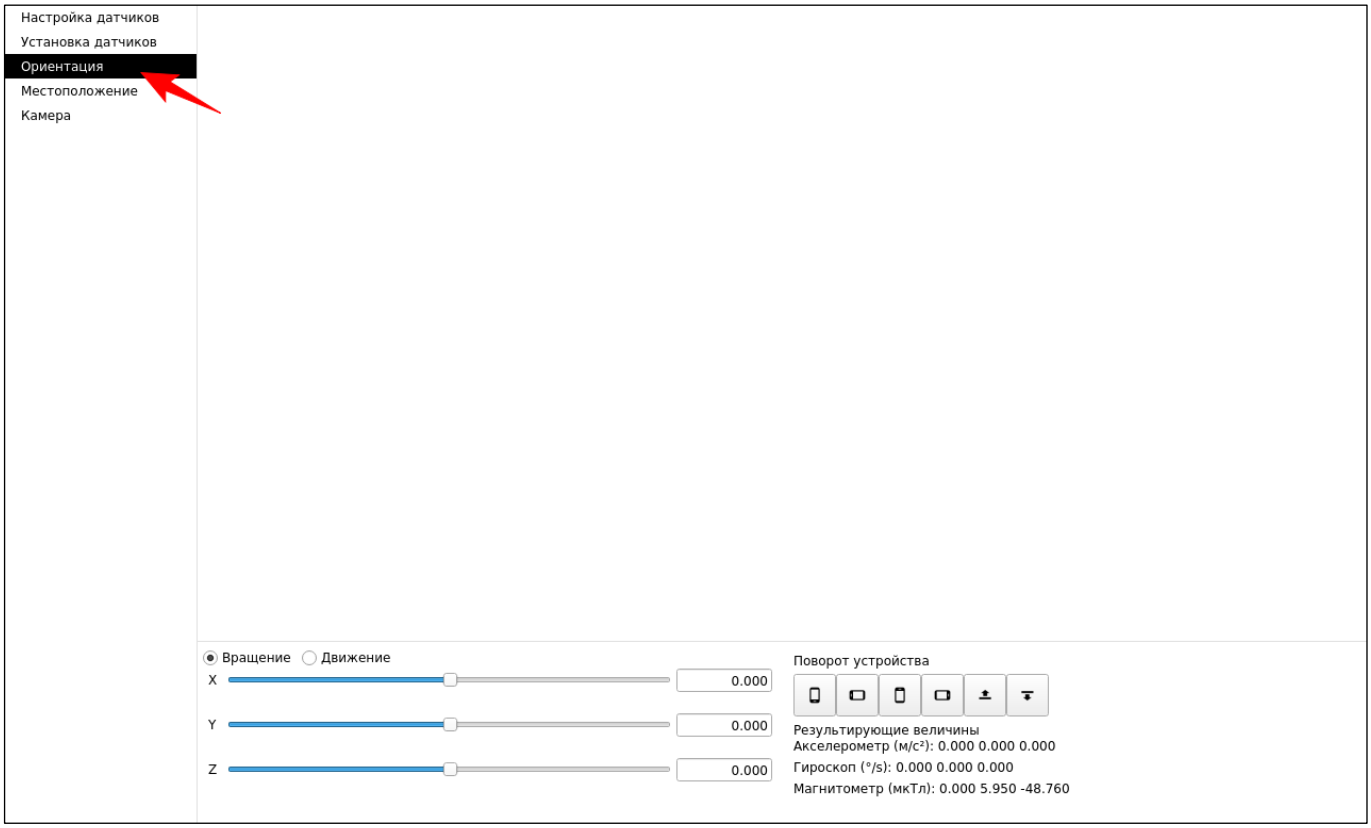


Рисунок 76

7.3.3. Работа с датчиками

В проекте для работы с местоположением можно использовать компоненты QML:

– Accelerometer (https://doc.qt.io/archives/qt-5.6/qml-qtsensors-accelerometer.html);

– AmbientLightSensor (https://doc.qt.io/archives/qt-5.6/qml-qtsensors-ambientlightsensor.html);

– Compass (https://doc.qt.io/archives/qt-5.6/qml-qtsensors-compass.html);

– Gyroscope (https://doc.qt.io/archives/qt-5.6/qml-qtsensors-gyroscope.html);

– LightSensor (https://doc.qt.io/archives/qt-5.6/qml-qtsensors-lightsensor.html);

– Magnetometer (https://doc.qt.io/archives/qt-5.6/qml-qtsensors-magnetometer.html);

– OrientationSensor (https://doc.qt.io/archives/qt-5.6/qml-qtsensors-orientationsensor.html);

– ProximitySensor (https://doc.qt.io/archives/qt-5.6/qml-qtsensors-proximitysensor.html);

– QTapSensor (https://doc.qt.io/archives/qt-5.6/qtapsensor.html);

– RotationSensor (https://doc.qt.io/archives/qt-5.6/qml-qtsensors-rotationsensor.html);

– и соответствующие им C++-классы (https://doc.qt.io/archives/qt-5.6/qtsensors-module.html), а также другие классы и компоненты Qt Sensors (https://doc.qt.io/archives/qt-5.6/qtsensors-index.html).

7.3.4. D-Bus-сервис для управления эмуляцией датчиков

Сервис (Таблица 4) позволяет управлять эмуляцией датчиков (Таблица 5). Подробнее в пп. 7.3.4.2.

Таблица 4

Шина:	системная
Служба:	ru.omp.SensorfwEmulationManagement
Объект:	/ru/omp/SensorfwEmulationManagement
Интерфейс:	ru.omp.SensorfwEmulationManagement

Таблица 5

Датчик	sensorName	Компонент QML
Акселерометр	accelerometer	Accelerometer, RotationSensor
Компас	compass	Compass
Датчик ориентации	orientation	OrientationSensor
Магнитометр	magnetometer	Magnetometer, RotationSensor
Гироскоп	gyroscope	Gyroscope
Датчик освещенности	als	AmbientLightSensor, LightSensor
Датчик приближения	proximity	ProximitySensor
Датчик нажатия	tap	QTapSensor

7.3.4.1. Методы

Методы эмуляцией датчиков:

- disableSensor(string sensorName);
- enableSensor(string sensorName);
- setAccelerometerValues(int32 x, int32 y, int32 z);
- setAccelerometerX(int32 x);
- setAccelerometerY(int32 y);
- setAccelerometerZ(int32 z);
- setAlsValue(int32 lux);
- setGyroscopeValues(double x, double y, double z);
- setGyroscopeX(double x);
- setGyroscopeY(double y);

- setGyroscopeZ(double z);
- setMagnetometerCalibrationLevel(int32 calibrationLevel);
- setMagnetometerValues(double x, double y, double z);
- setMagnetometerX(double x);
- setMagnetometerY(double y);
- setMagnetometerZ(double z);
- setProximitySensorValue(double reflectance);
- setTapSensorValue(int32 tapDirection).

7.3.4.2. Подробное описание

Сервис позволяет управлять эмуляцией датчиков: включать/выключать датчики и устанавливать значения. Эта функциональность эмулятора предназначена для экспертов.

Примеры использования DBus-сервиса с помощью вызова утилиты `dbus-send`:

- включение датчика освещенности:

```
dbus-send --system --type=method_call --print-reply --
dest=ru.omp.SensorfwEmulationManagement
/ru/omp/SensorfwEmulationManagement
ru.omp.SensorfwEmulationManagement.enableSensor string:als
```

- установка значений акселерометра:

```
dbus-send --system --type=method_call --print-reply --
dest=ru.omp.SensorfwEmulationManagement
/ru/omp/SensorfwEmulationManagement
ru.omp.SensorfwEmulationManagement.setAccelerometerValues int32:1100
int32:900 int32:2000
```

- установка значения магнитометра по оси Y:

```
dbus-send --system --type=method_call --print-reply --
dest=ru.omp.SensorfwEmulationManagement
/ru/omp/SensorfwEmulationManagement
ru.omp.SensorfwEmulationManagement.setMagnetometerY double:0.0000012
```

7.3.4.3. Описание методов

```
void disableSensor(string sensorName)
```

Выключает датчик с именем `sensorName`.

После вызова метода необходимо перезапустить все приложения на эмуляторе, работающие с датчиками.

```
void enableSensor(string sensorName)
```

Включает датчик с именем `sensorName`.

После вызова метода необходимо перезапустить все приложения на эмуляторе, работающие с датчиками.

```
void setAccelerometerValues(int32 x, int32 y, int32 z)
```

Устанавливает значения ускорения по осям X, Y и Z (Рисунок 77) для акселерометра. Значения акселерометра задаются в долях *g* (ускорения свободного падения, взятого равным 9.80665), умноженных на 1000. Диапазон допустимых значений: `[-3000:3000]`.

```
void setAccelerometerX(int32 x)
```

Аналог метода setAccelerometerValues() для установки одного значения ускорения по оси X.

```
void setAccelerometerY(int32 y)
```

Аналог метода setAccelerometerValues() для установки одного значения ускорения по оси Y.

```
void setAccelerometerZ(int32 z)
```

Аналог метода setAccelerometerValues() для установки одного значения ускорения по оси Z.

```
void setAlsValue(int32 lux)
```

Устанавливает значение уровня освещенности в люксах.

```
void setGyroscopeValues(double x, double y, double z)
```

Устанавливает значения угловой скорости вокруг осей X, Y и Z для гироскопа в/с.

```
void setGyroscopeX(double x)
```

Аналог метода setGyroscopeValues() для установки угловой скорости вокруг оси X.

```
void setGyroscopeY(double y)
```

Аналог метода setGyroscopeValues() для установки угловой скорости вокруг оси Y.

```
void setGyroscopeZ(double z)
```

Аналог метода setGyroscopeValues() для установки угловой скорости вокруг оси Z.

```
void setMagnetometerCalibrationLevel(int32 calibrationLevel)
```

Устанавливает значение точности калибровки магнитометра `calibrationLevel`.

Значение уровня калибровки магнитометра имеет диапазон `[0:3]`, однако результирующие значения на эмуляторе будут иметь диапазон `[0:1]`, так как они будут равны установленному значению, деленному на 3 (а именно `[0, 0.(3), 0.(6), 1]`). Это связано с особенностями плагина Sensorfw для магнитометра.

```
void setMagnetometerValues(double x, double y, double z)
```

Устанавливает значения плотности магнитного потока по осям X, Y, Z для магнитометра в теслах. Диапазон допустимых значений: `[-2.0:2.0]`.

```
void setMagnetometerX(double x)
```

Аналог метода setMagnetometerValues() для установки значения плотности магнитного потока по оси X.

```
void setMagnetometerY(double y)
```

Аналог метода setMagnetometerValues() для установки значения плотности магнитного потока по оси Y.

```
void setMagnetometerZ(double z)
```

Аналог метода setMagnetometerValues() для установки значения плотности магнитного потока по оси Z.

```
void setProximitySensorValue(double reflectance)
```

Устанавливает значение коэффициента отражения `reflectance` для датчика приближенности. Диапазон допустимых значений: `[0:10]`.

Значений `reflectance` определяет близость некоторого объекта к датчику приближенности. Если `reflectance` равно нулю, то объект находится в непосредственной близости.

Значение датчика приближенности задается как степень приближения и имеет диапазон `[0:10]`, однако результирующее значение на эмуляторе будет иметь диапазон `[0:1]`. Это связано с особенностями плагина Sensorfw для датчика приближенности.

```
void setTapSensorValue(int32 tapDirection)
```

Устанавливает значение направления нажатия на устройство `tapDirection` для датчика нажатий.

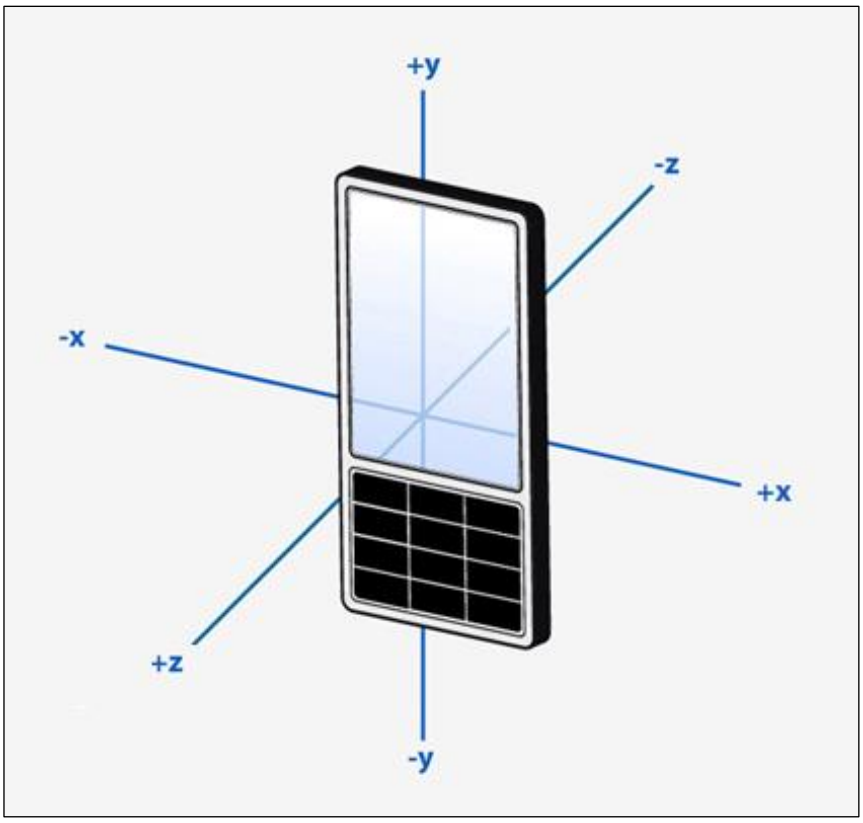


Рисунок 77

Параметр `tapDirection` может принимать одно из значений, приведенных в таблице (Таблица 6).

Таблица 6

Значение	Направление	Описание
-1	Undefined	Направление нажатия не определено
0	X_Both	Нажатие по оси X, но не определено, в каком направлении
1	Y_Both	Нажатие по оси Y, но не определено, в каком направлении
2	`Z_Both	Нажатие по оси Z, но не определено, в каком направлении
3	X_Pos	Нажатие по оси X в положительном направлении
4	X_Neg	Нажатие по оси X в отрицательном направлении
5	Z_Neg	Нажатие по оси Z в отрицательном направлении

Значение	Направление	Описание
6	Z_Pos	Нажатие по оси Z в положительном направлении
7	Y_Pos	Нажатие по оси Y в положительном направлении
8	Y_Neg	Нажатие по оси Y в отрицательном направлении

8. ОПИСАНИЕ СРЕДЫ СБОРКИ И ПРОЦЕСС СБОРКИ УСТАНОВОЧНЫХ ПАКЕТОВ

8.1. Среда сборки

8.1.1. Общая информация и структура

Build Engine — это среда сборки, которая предоставляет доступ к ряду инструментов, например, к Scratchbox2 (https://developer.auroraos.ru/doc/tools/scratchbox2), в котором можно собрать специфичные для Linux-архитектуры исполняемые файлы с использованием эмуляции на QEMU.

В среде сборки настроены несколько общих каталогов для обмена файлами с домашней ОС. Для размещения проектов допустимы:

- домашний каталог пользователя;
- альтернативный каталог, указанный при установке Аврора SDK;
- все вложенные в них каталоги.

8.1.2. Авторизация в среде сборки

Авторизоваться с правами пользователя в среде сборки можно с помощью команды:

```
$ ssh -p 2222 -i {путь к SDK}/vmshare/ssh/private_keys/sdk
mersdk@localhost
```

Авторизоваться с правами суперпользователя в среде сборки можно с помощью команды:

```
$ ssh -p 2222 -i {путь к SDK}/vmshare/ssh/private_keys/sdk
root@localhost
```

8.2. Сборка установочных пакетов

Пакеты могут быть собраны локально (с помощью команд `mb2` или `sb2` для Build Engine).

8.2.1. Локальная сборка пакетов

После установки среды сборки, настройки целей, возможна локальная сборка пакетов для целевой архитектуры с использованием скрипта `mb2` (вызывается из Build Engine). Данный скрипт является оберткой Scratchbox2 (`sb2`) и `rpmbuild`, которая упрощает создание пакетов из исходных репозиториев, предоставляемых в файле `*.spec`.

Скрипт `mb2` для сборки пакета вызывается следующим образом:

```
mb2 -s [путь к *.spec] -t [цель] build
```

При наличии единственного файла формата `*.spec`, содержащегося в подкаталоге `/rpm`, первые два параметра являются необязательными.

Команда ниже собирает большинство проектов и упаковывает результаты в RPM-файлы, готовые для установки (при условии, что цель AuroraOS-armv7hl была установлена):

```
mb2 -t AuroraOS-armv7hl build
```

Выходные RPM-файлы будут находиться в подкаталоге `/RPMS`. Подробная информация об установке на устройство представлена в статье Интерфейс командной строки.

8.2.2. Установка недостающих зависимостей

В некоторых случаях разработчик должен добавить или подключить репозитории для цели, если выводится сообщение о том, что зависимость не может быть установлена. Для этого пользователь должен войти в chroot ScratchBox2.

Для добавления репозитория следует использовать команду:

```
ssu ar [наименование_репозитория] [URL_репозитория]
```

Для подключения репозитория следует использовать команду:

```
ssu er [наименование_репозитория]
```

Для обновления информации об устанавливаемых пакетах следует использовать команду:

```
zypper ref -f
```

Для вывода списка известных репозиториев следует использовать команду:

```
ssu lr
```

Для удаления репозитория следует использовать команду:

```
ssu rr [наименование_репозитория]
```

8.2.3. Форматы пакетной сборки

8.2.3.1. Структура пакетной сборки tar_git

Пакеты в ОС Аврора в основном упакованы в формат `tar_git`. Данный формат относительно прост и, при соблюдении базовых правил, такие инструменты как `mb2`, функционируют корректно, а значит с большой вероятностью, пакет соберется корректно.

RPM-файлы `<имя файла>.spec` расположены в `rpm/<имя файла>.spec`. Наличие файла `<имя файла>.changes` необязательно, но необходимо, если файлы размещены в `rpm/<имя файла>.changes`. Кроме того, все остальные файлы в каталоге `rpm/` должны быть отмечены как SourceX: или PatchX:.

8.2.3.2. Расположение исходного кода пакета tar_git в Git

Исходный код, как правило, размещен внутри одного дерева директорий Git, если первоисточником кода является ОС Аврора. Код может размещаться в корне дерева директорий Git, в каталоге `Src` или в аналогичном месте в зависимости от пакета. Примером такого пакета является `mce`.

Если первоисточник пакета существует извне и не содержит значительных изменений для исходного кода, то источники обычно располагаются в каталоге с таким же наименованием, что и у пакета в `*.spec`-файле.

В случае если существует множество модификаций, обращающихся к первоисточнику, но не являющихся его частью, то подмодуль обычно называют первоисточником (upstream), как и соответствующий каталог в `*.spec`-файле, который содержит копию первоисточника с модификациями. Примером такого пакета является `connman`.

В некоторых случаях в дереве директорий Git подмодули отсутствуют, если первоисточник — архив `.tar_`, `_.cvs` или `*.svn`, из-за чего подмодуль Git не может быть создан.

8.2.4. Создание журнала изменений

Журнал изменений генерируется из коммитов Git в пакетах `tar_git`. Каждая строка добавляется в журнал изменений в следующем формате:

```
[key] Summary. Contributes to xyz#123
[packaging] Updated Y to version X. Fixes xyz#124
```

Для вызова справки о журнале изменений, выполните команду внутри SDK:

```
mb2 --help
```

ПЕРЕЧЕНЬ ТЕРМИНОВ И СОКРАЩЕНИЙ

Используемые в настоящем документе термины и сокращения приведены в таблице (Таблица 7).

Таблица 7

Термин/ Сокращение	Расшифровка
МУ	Мобильное устройство
ОС	Операционная система
Поток	Основная единица, которой операционная система выделяет время процессора; последовательность инструкций в программе, которая может выполняться параллельно с другими
Таргеты	Набор бинарных файлов ОС под разные архитектуры (i486/ARM), в том числе инструменты для компиляции, линковки и пакетирования
Хост-машина	Основной компьютер, предоставляющий информационные или вычислительные ресурсы. При использовании эмулятора хост-машиной называется компьютер, на котором эмулятор запускается
Эмулятор	Программное обеспечение, с помощью которого можно запускать приложения, предназначенные для других ОС

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

№ версии документа	Содержание изменения	ФИО	Дата