Введение в API платформы Sailfish

Андрей Васильев

Ярославская лаборатория FRUCT, ЯрГУ
andrey.vasilyev@fruct.org

Летняя школа Sailfish OS
28 июня 2016

План

Обзор возможностей
Работа с данными о географическом положении
Отображение карты
Взаимодействие с датчиками
Мультимедиа возможности платформы
Особенности проектов

Возможности платформы Qt

Платформа Qt обладает большим набором модулей, позволяющих быстро разрабатывать приложения

- Qt Positioning позволяет определить текущее местоположение
- Qt Sensors предоставляет доступ к датчикам (акселерометр, гироскоп)
- Qt Location позволяет отобразить карты и построить маршруты
- Qt Multimedia обеспечивает доступ к музыке, фото и видео

Вне данной презентации

- Qt Bluetooth позволяет использовать протокол Bluetooth
- Qt NFC предоставляет доступ по работе с очень близкими устройствами
- Local Storage синглтон для взаимодействия с базой данных

План

Обзор возможностей
Работа с данными о географическом положении
Отображение карты
Взаимодействие с датчиками
Мультимедиа возможности платформы
Особенности проектов

Обзор возможностей Qt Positioning API

Positioning API предоставляет средства для определения местоположения устройства с помощью спутниковых систем или беспроводных сетей

Помимо собственно возможностей по определению долготы и ширины может предоставлять ещё и следующие данные:

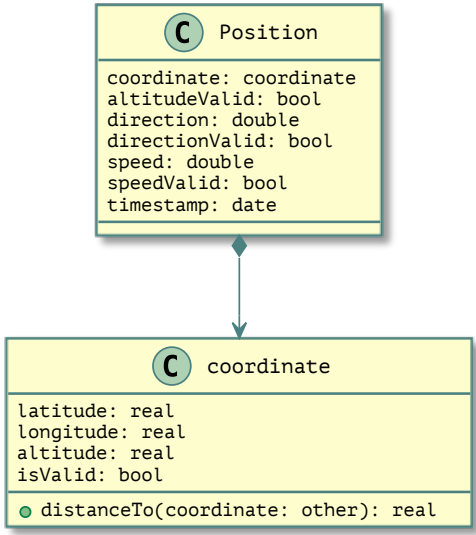
- Дата и время, в которые были получены данные координаты
- Скорость устройства, которая была зафиксирована
- Высота над уровнем моря
- Азимут направления устройства (угол отклонения от северного полюса)

Классы, описывающие данные Positioning API

Базовые данные описываются `coordinate`: ширина, долгота, высота и корректность

Дополнительные данные описываются с помощью объектов `Position`

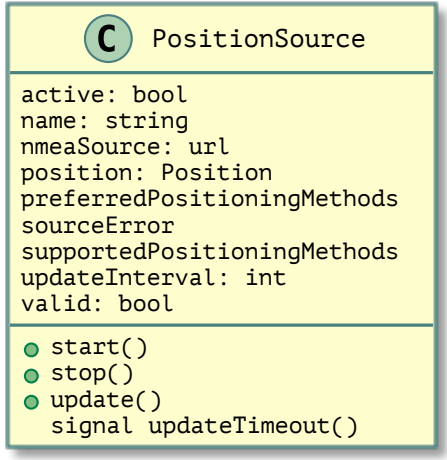
- `coordinate` — ссылка на соответствующий объект с данными
- `direction` — направление
- `speed` — скорость передвижения
- `timestamp` — время получения измерения



Источник данных о географических положениях

Класс `PositionSource` является источником данных о местоположении

- `name` — имя плагина-источника данных
- `nmeaSource` — путь к файлу из которого будут браться географические данные
- `preferredPositioningMethods` - предпочитаемые источники данных
- `sourceError` — источник ошибки
- `updateInterval` — частота обновления



Сложные значения свойств PositionSource

Возможные значения *PositioningMethods

- NoPositioningMethods — нет предпочитаемых методов
- SatellitePositioningMethods — спутниковое позиционирования
- NonSatellitePositioningMethods — следует использовать системы беспроводного доступа в сеть интернет
- AllPositioningMethods — можно использовать любые методы

Возможные значения sourceError

- AccessError — у приложения недостаточно привилегий
- ClosedError — пользователь отключил геопозиционирование
- SocketError — ошибка сетевого взаимодействия

Подключение Positioning API

Перед началом использования Posititioning API в приложении его поддержку необходимо включить, сформировав следующим образом зависимости:

```
# Runtime dependencies which are not automatically detected
Requires:
  - sailfishsilica-qt5 >= 0.10.9
  - qt5-qtdeclarative-import-positioning
```

После этого можно добавить элемент PositionSource на страницу:

```
import QtPositioning 5.2
Page {
    PositionSource {
        id: positionSource
        active: true
    }
}
```

Использование PositionSource

```
Column {
    Label {
        text: "Работает ли источник данных: " +
              positionSource.active
    }
    Label {
        text: "Ширина: " + positionSource.position.
                           coordinate.latitude
    }
    Label {
        text: "Долгота: " + positionSource.position.
                           coordinate.longitude
    }
}
```

Использование nmeaSource

Иногда у вас нет под рукой устройства, а хочется написать приложение, использующее информацию о глобальном позиционировании. Для этих целей можно использовать журнал в формате NMEA

Дистрибуция дополнительных файлов

Для того, чтобы разместить файл журнала на устройство, добавьте следующие строки в `project.pro` файл:

```
nmeafiles.files = nmeafiles/*
nmeafiles.path = /usr/share/$${TARGET}/nmeafiles
INSTALLS += nmeafiles

OTHER_FILES += nmeafiles/*.log
```

Указание файла как источника данных

Всё, что осталось - указать данный файл в качестве источника данных

```
PositionSource {
    id: positionSource
    active: true
    nmeaSource: "/usr/share/project/nmeafiles/gps.log"
}
```

Важные аспекты, которые стоит помнить:

- `project` надо заменить на название вашего проекта
- `nmeaSource` необходимо указывать только лишь в случае, если другие источники данных недоступны

План

Обзор возможностей
Работа с данными о географическом положении
Отображение карты
Взаимодействие с датчиками
Мультимедиа возможности платформы
Особенности проектов

Краткий обзор Location API

Location API предоставляет возможности по созданию приложений, ориентированных на показ пользователю карты

- Показ карт с пользовательскими слоями и подложками
- Отображение на картах точек интереса
- Построение маршрутов между точками
- Поиск точек интересов, рекомендации, категории

Официальное руководство покрывает все необходимые вопросы по работе с картами. Мы остановимся на этапе показа карты и текущего местоположения

Зависимости

Зависимости приложения

Для создания приложения, использующего карты необходимо подключить `Posititioning API` и источник картографических данных

```
Requires:
  - sailfishsilica-qt5 >= 0.10.9
  - qt5-qtdeclarative-import-positioning
  - qt5-qtdeclarative-import-location
  - qt5-plugin-geoservices-osm
```

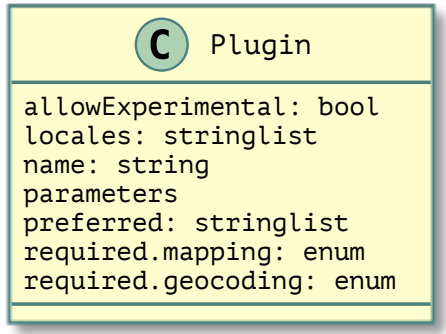
Зависимости в QML-файле

```
import QtPositioning 5.0
import QtLocation 5.0
```

Всё дело в плагинах (`Plugin`)

Location Plugin представляет из себя мост между Qt API и конкретной службой, предоставляющей услуги по работе с картами (OpenStretmaps, HereMaps)

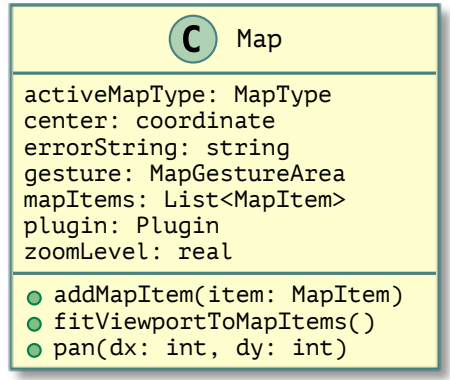
- `allowExperimental` — допускаются ли нестабильные плагины или нет
- `name`, `preferred` — указание имени плагина, которые можно использовать
- `required.mapping` — какая функциональность по отображению карты требуется в приложении
- `locales` — список предпочитаемых локалей для предоставления информации



Краткий взгляд на Map API

QML Map используется для отображения карт и изображений земли

- `activeMapType` — активный способ отображения карты (день, ночь и т.д.)
- `center` — указание места, в котором необходимо центрировать карту
- `mapItems` — набор элементов для отображения на карте
- `plugin` — источник данных для карты
- `zoomLevel` — уровень приближения
- `pan()` — передвинуть текущее отображение карты



Простейшее картографическое приложение

Создадим простое приложение, которое будет отображать карту и центрировать своё положение относительно пользователя. Нам потребуются:

- Элемент `PosititonSource` из предыдущей части
- Настроенный плагин для получения данных из OSM
- Карта, способная показывать данные

Настройка Plugin

```
Plugin {
    id: osmPlugin
    allowExperimental: true
    preferred: ["osm"]
    required.mapping: Plugin.AnyMappingFeatures
    required.geocoding: Plugin.AnyGeocodingFeatures
}
```

Добавление карты

```
Rectangle {
    anchors.fill: parent
    Map {
        id: map
        anchors.fill: parent
        plugin: osmPlugin
        gesture.enabled: true
        center: positionSource.position.coordinate
    }
}
```

- Обязательно поместите карту внутрь прямоугольника
- Не забудьте связать карту с соответствующим плагином

План

Обзор возможностей
Работа с данными о географическом положении
Отображение карты
Взаимодействие с датчиками
Мультимедиа возможности платформы
Особенности проектов

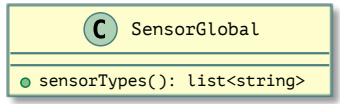
Типы датчиков на устройстве

Qt Sensors позволяет получить доступ к любому датчику, который может быть установлен на устройстве. Например, на Jolla Tablet присутствуют

- Orientation Sensor — ориентация устройства в пространстве
- Accelerometer — показания акселерометра
- Ambient Light Sensor — показания датчика окружающего света
- Light Sensor — показания датчика света
- Magnetometer — показания мощности электронного поля по трём осям
- Gyroscope — показания гироскопа
- Rotation Sensor — показания датчика вращения по трём осям
- Compass — данные компаса

Получение списка доступных датчиков

Не все устройства оборудованы данными датчиками. `SensorsGlobal` позволяет получить список названий доступных на данном устройстве датчиков.

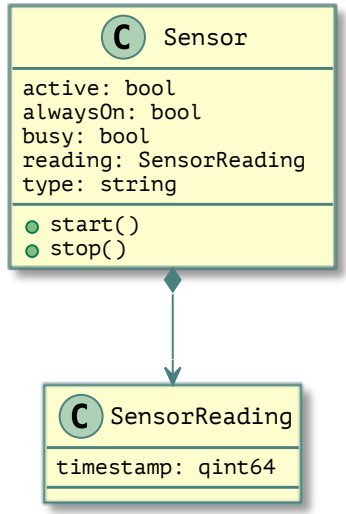



```
import QtSensors 5.0 as Sensors
ListModel {
    id: sensorsList
    Component.onCompleted: {
        var types = Sensors.QmlSensors.sensorTypes();
        for(var i = 0; i < types.length; i++) {
            append({ name: types[i] });
        }
    }
}
```

Архитектура датчика

Все объекты, предоставляющие доступ к датчикам, унаследованы от класса `Sensor`

- `active` — предоставляет ли датчик показания приложению или нет
- `alwaysOn` — должен ли работать датчик при закрытии экрана
- `busy` — занят ли (недоступен) датчик другими приложениями
- `reading` — текущие показания датчика
- `type` — строковое название датчика
- `start()`, `stop()` — запуск и остановка получения данных



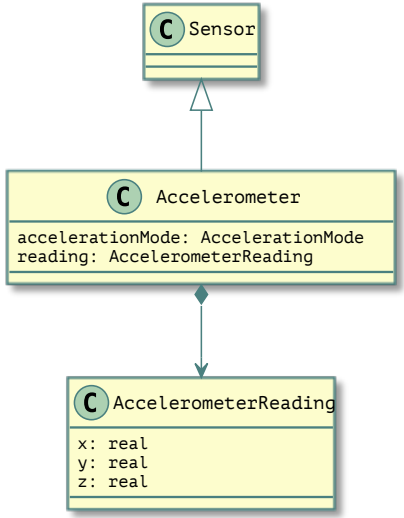
Accelerometer - типичный датчик

Акселерометр предоставляет данные об ускорении по всем трём осям. В зависимости от AccelerationMode предоставляет данные

- `Gravity` — только о гравитации
- `User` — только о действиях пользователя
- `Combined` — общие показания

`AccelerometerReading`

Содержит в себе 3 свойства, указывающие значения ускорения по трём осям `x`, `y`, `z`

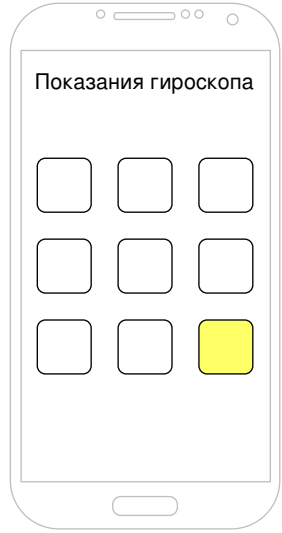


Пример использования акселерометра

```
import QtQuick 2.0
import Sailfish.Silica 1.0
import QtSensors 5.0 // Получаем доступ к сенсорам
Page {
    Accelerometer { // Создаём объект акселерометра
        id: accelerometer
        active: true // Обеспечиваем доступ к показаниям
    }
    Column { // Показываем данные акселерометра
        Label { text: "x: " + accelerometer.reading.x }
        Label { text: "y: " + accelerometer.reading.y }
        Label { text: "z: " + accelerometer.reading.z }
    }
}
```

Использование датчика RotationSensor

Создадим приложение, которое будет изображать значение гироскопа по осям X и Y. На экране отобразим 9 прямоугольников, которые будут загораться в зависимости от значений гироскопа.



Краткое руководство по реализации

1. Запустите Qt Creator и создайте приложение Sailfish
2. Разместите на странице 9 прямоугольников
3. Подключите поддержку датчиков в приложение
4. Создайте объект типа RotationSensor
5. Свяжите цвет прямоугольника со значением датчика

Зажигающийся квадратик

Создадим отдельный компонент, который будет зажигаться жёлтым цветом, если значение свойства `enabled` будет равным `true`.
Сохраним компонент в файл с названием `SensorVisualizer.qml`

```
Rectangle {
    property bool enabled: false
    width: parent.width / 4
    height: parent.width / 4
    radius: Theme.paddingMedium
    color: if(enabled) { "yellow" } else { "white" }
}
```

Структура страницы

```
import QtQuick.Layouts 1.0
import QtSensors 5.0
Page {
    allowedOrientations: Orientation.Portrait
    GridLayout {
        anchors.fill: parent
        columns: 3
        SensorVisualizer {
            enabled: rotationSensor.reading.x > 10
        }
    }
    RotationSensor { id: rotationSensor; active: true }
}
```

Добавим поддержку датчиков к приложению

Для корректной работы приложения на устройстве необходимо указать, что приложение зависит от подсистемы датчиков. Это делается путём указания зависимости от соответствующего пакета.

1. Найдите YAML-файл, в котором описан процесс сборки приложения. Он находится в каталоге `rpm` и совпадает с именем проекта.
2. Добавьте в список зависимостей (`Requires`) пакет `qt5-qtdeclarative-import-sensors`

```
# Runtime dependencies which are not automatically detected
Requires:
  - sailfishsilica-qt5 >= 0.10.9
  - qt5-qtdeclarative-import-sensors
```

План

Обзор возможностей
Работа с данными о географическом положении
Отображение карты
Взаимодействие с датчиками
Мультимедиа возможности платформы
Особенности проектов

Обзор Qt Multimedia

Qt Multimedia предоставляет базовые возможности для работой с мультимедиа содержимым. Qt Multimedia QML предоставляет компоненты для поддержки базовых сценариев работы с мультимедиа:

- воспроизведение аудио и видео
- получить доступ к камере и радио датчику
- запись видео файлов
- доступ к настройкам камеры

Рассмотрим базовые возможности данного модуля — воспроизведение звука и работу с изображениями

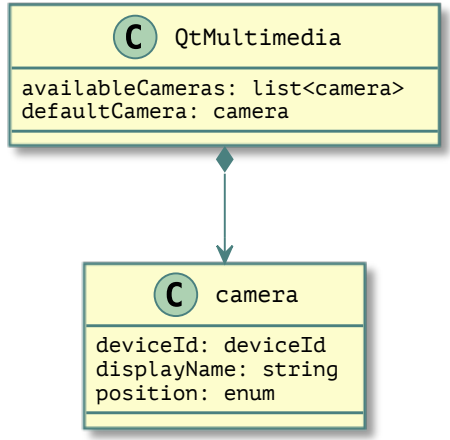
Глобальный объект Qt Multimedia

Глобальный объект предоставляет информацию о камерах, которые установлены в устройстве

- `availableCameras` — список доступных на устройстве камер
- `defaultCamera` — камера по умолчанию, свойство доступно только на чтение

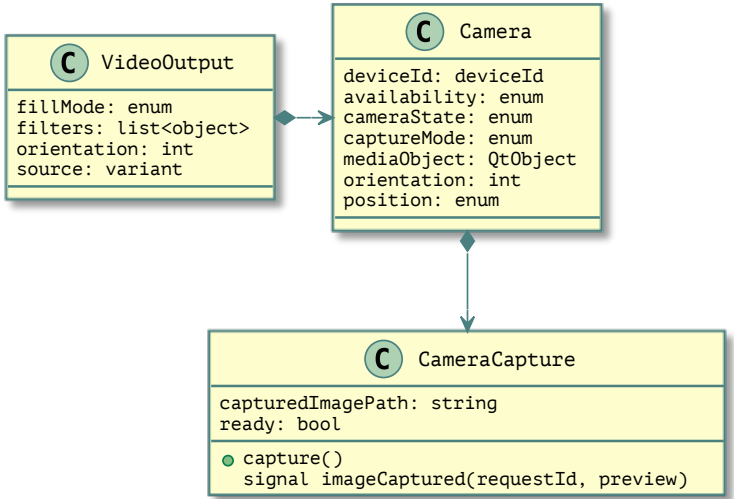
Каждая камера описывается следующими свойствами

- `deviceId` — уникальный идентификатор
- `displayName` — описание для человека
- `position` — положение камеры



Простейший фотоаппарат

CameraCapture API содержит в себе замечательный пример



Обзор элементов Qt Multimedia

Воспроизведение аудио

- SoundEffect позволяет быстро реагировать на действия пользователя
- `Audio` позволяет воспроизводить долгие аудио файлы и управлять воспроизведением

Воспроизведение видео

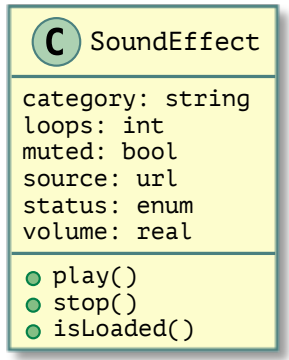
- Video — элемент для воспроизведения видео файлов
- MediaPlayer — универсальный проигрыватель любых файлов
- VideoOutput — элемент для вывода потока изображений

Дополнительно: Radio — доступ к радио; Torch — фонарик

Быстрые звуки с помощью SoundEffect

SoundEffect предоставляет возможности по проигрыванию звуков в ответ на действия пользователей с минимально возможной задержкой. Ввиду ускорения работы необходимо использовать несжатый формат

- `source` — путь к файлу, который необходимо проиграть
- `loops` — количество повторений для звука
- `play()` — начать воспроизведение звука
- `status` описывает текущий процесс загрузки файла



Использование SoundEffect

```
import QtMultimedia 5.0
Page {
    SoundEffect {
        id: shortSound
        source: "/usr/share/project/samples/martian-code-ding.
    }
    Button {
        anchors.centerIn: parent
        text: "Нажми для воспроизведения"
        onClicked: shortSound.play()
    }
}
```

Проигрывание звуковых файлов

Основой для воспроизведения мультимедиа файлов является QMediaService, который предоставляет доступ к объектам QMediaControl, способных воспроизвести файлы
На Sailfish OS используется фреймворк GStreamer

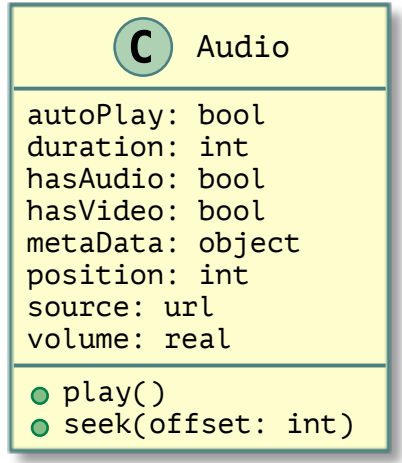
Настройка зависимостей приложения

```
Requires:
  - sailfishsilica-qt5 >= 0.10.9
  - qt5-qtmultimedia-plugin-mediaservice-gstmediaplayer
  - gst-plugins-bad
  - gst-plugins-bad-free
```

Обзор API Audio

Элемент Audio позволяет воспроизводить аудио файлы

- `source` — путь к файлу, который необходимо воспроизвести
- `duration` — продолжительность аудио файла в миллисекундах
- `hasAudio`, `hasVideo`, `metaData` — набор мета информации о данной записи
- `play()`, `pause()`, `stop()`, `seek(offset)` позволяют управлять потоком воспроизведения



Воспроизведение с помощью MediaPlayer

Элемент MediaPlayer позволяет воспроизводить не только аудио файлы, но и видео. Его API полностью совпадает с API Audio, но он может выступать в роли источника данных для VideoOutput.

Пример простого проигрывателя

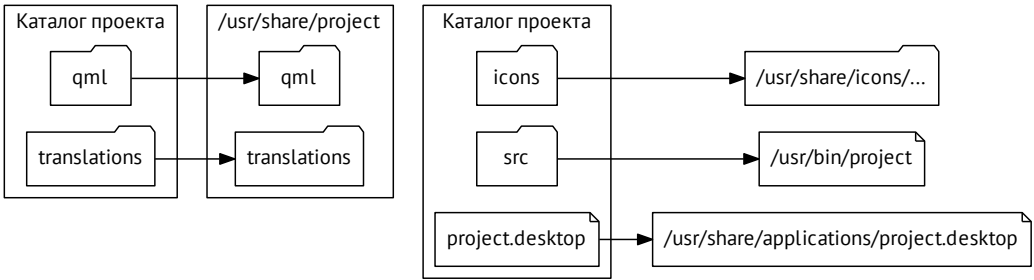
```
import QtMultimedia 5.0
Page {
    id: page
    Audio {
        id: audioPlayer
        source: "/usr/share/application/samples/audio.mp3"
    }
    Button {
        text: "Нажми для воспроизведения"
        enabled: audioPlayer.playbackState !=
                 Audio.PlayingState
        onClicked: audioPlayer.play()
    }
}
```

План

Обзор возможностей
Работа с данными о географическом положении
Отображение карты
Взаимодействие с датчиками
Мультимедиа возможности платформы
Особенности проектов

Структура проекта

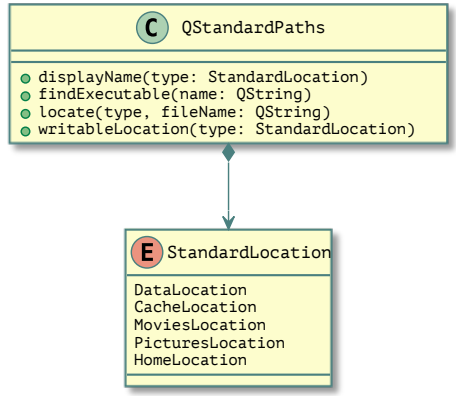
Всю информацию о структуре проекта можно найти в репозитории SDK Sailfish



Доступ к каталогам приложения

Элемент QStandardPaths позволяет получить местоположение стандартных путей
Все методы данного класса статические,
возвращают строки, а также пустую строку

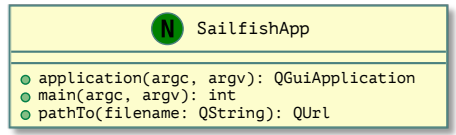
- `displayName` — показать локализованное имя данной локации
- `findExecutable` — найти исполняемый файл в PATH или указанных каталогах
- `locate` — найти обычный файл по указанному местоположению
- `writableLocation` — путь к каталогу, в который можно записать данные



libsailfishapp - библиотека, структурирующая приложения

libsailfishapp облегчает процесс создания приложения для платформы Sailfish, а также методы для доступа к файлам приложения. libsailfishapp.h содержит самую актуальную документацию

- `application` — создать объект QGuiApplication
- `main` — запустить приложение со стандартной конфигурацией
- `pathTo` — получить полный путь к файлу в каталоге с данными приложения



Создание собственного компонента

Официальная документация по интеграции C++ и QML

```
class PathProvider : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString dataDir READ dataDir NOTIFY dataDirChar
    Q_PROPERTY(QString privateDir READ privateDir NOTIFY priva
public:
    explicit PathProvider(QObject *parent = 0) {};
    QString dataDir() {
        return SailfishApp::pathTo("./").toString();
    }
    QString privateDir() {
        return QStandardPaths::writableLocation(
                QStandardPaths::DataLocation);
```

Использование компонента

Регистрация компонента

```
#include <QtQuick>
#include "pathprovider.h"
qmlRegisterType<PathProvider>("org.fruct.yar",
        1, 0, "PathProvider");
```

Использование компонента

```
import org.fruct.yar 1.0
Page {
    PathProvider {
        id: pathProvider
    }
    Label { text: pathProvider.dataDir }
```

Доступ к датчикам из C++

Если вам необходимо производить серьёзные расчёты на основании данных датчиков, тогда их следует реализовать на системном языке

```
#include <QAccelerometer>
// .pro: QT += sensors
// .yaml: Requires: - qt5-qtsensors
class Accelerometer : public QAccelerometer {
    Q_OBJECT
    Q_PROPERTY(qreal x READ x NOTIFY valuesChanged)
public:
    explicit Accelerometer(QObject *parent = 0);
    qreal x() const;
signals:
    void valuesChanged();
};
```