# 9. Лабораторная работа № 3 «Использование датчика ориентации для управления пользовательским интерфейсом»


## 9.1. Цель лабораторной работы

Демонстрация процесса разработки практического приложения с использованием датчика ориентации.

## 9.2. Инструкция по выполнению лабораторной работы

Для надежной и быстрой работы компьютера в его архитектуру входят различные датчиков, по которым определяется техническое состояние элементов компьютера (температура процессора, скорость вентилятора и т. п.)

Отличительная особенность современного смартфона или планшетного компьютера – наличие большого количества разнородных датчиков, предназначенных для определения положения и скорости устройства, освещенности и других параметров. В отличие от традиционных датчиков, например, температуры процессора или скорости вентилятора, они предназначены в первую очередь для уровня приложений, а не операционной системы. В отличие от внешних датчиков, например, GPS, они являются неотъемлемой частью устройства и не могут использоваться самостоятельно.

Наша задача – научиться работать с одним самых распространенных датчиков, датчиком ориентации, который позволяет определить, «на каком боку» в данный момент «лежит» устройство. Для этого необходимы базовые знания языка C++, знакомство с основами фреймворка *Qt* и двумя библиотеками *Qt*

Mobility и MeeGoTouch. Самое важное требование – необходимо устройство (смартфон или планшет) с датчиком ориентации, работающее под управлением Linux. Мы использовали MeeGo 1.1.99 Tablet на планшете 3Q TU1102T. Поскольку на эмуляторе или виртуальной машине информация от датчиков будет недоступна, разработку придется вести на реальном оборудовании.

Сначала рассмотрим интерфейс для работы с датчиками, который реализован как компонента sensors библиотеки *Qt* Mobility. Как обычно, для использования этой компоненты в файл проекта необходимо добавить строки

```
CONFIG += mobility
MOBILITY = sensors
```

В исходных текстах также надо употреблять макрос QTM_USE_NAMESPACE.

Основным классом для работы с датчиками является `QSensor`. От него унаследована масса классов для работы с различными типами датчиков, например, `QOrientationSensor` или `QAccelerometer`. Чтобы получить все доступные системе типы датчиков, используется статический метод `QSensor::sensorTypes()`, который возвращает список строк с названиями типов, строки упакованы в классы `QByteArray`. В свою очередь, для каждого типа с помощью статического метода `QSensor::sensorsForType()` можно получить список идентификаторов подходящих датчиков, также в виде списка строк в формате `QByteArray`.

Зная тип и идентификатор датчика, можно создать экземпляр соответствующего класса:

```
QSensor sensor(type);
sensor.setIdentifier(identifier);
```

Или явно указав тип датчика, если он известен заранее:

```
QOrientationSensor sensor;
sensor.setIdentifier(identifier);
```

Заметим, что созданный объект не получает никакого эксклюзивного контроля над датчиком, он просто дает возможность приложению получать от датчика данные и, возможно, настраивать некоторые его параметры. Драйвер датчика действует в ядре ОС, специальная библиотека координирует доступ к датчикам со

стороны программ. Можно создать несколько `QSensor` для получения данных от одного датчика.

Метод `QSensor::connectToBackend()` подключает объект к датчику и позволяет проверить, что датчик с таким идентификатором действительно существует и работоспособен.

Для доступа к данным датчика предназначен класс `QSensorReading` и его производные классы для разных типов датчиков. Они предоставляют удобный интерфейс для специфических данных.

Мы будем работать с датчиком ориентации, которому соответствует класс `QOrientationSensor` и класс данных `QOrientationReading`. Возможных показателей этого датчика всего 7, вот они:

```
QOrientationReading::Undefined
QOrientationReading::TopUp
QOrientationReading::TopDown
QOrientationReading::LeftUp
QOrientationReading::RightUp
QOrientationReading::FaceUp
QorientationReading::FaceDown
```

Последние два значения служат для определения того, что устройство лежит на какой-то поверхности вверх или вниз экраном. Наш планшет данные значения не предоставляет и таким образом мы оперировали четырьмя содержательными значениями – `TopUp, TopDown, LeftUp, RightUp`.

Важным является способ получения данных от датчика. Наиболее простым и самоочевидным способом является поллинг – периодический опрос датчика. Новые данные при этом обрабатываются в соответствии с логикой программы, старые данные или их отсутствие игнорируется. Главный недостаток поллинга – его вычислительная неэффективность, постоянное расходование ресурсов на бесполезные операции. Кроме того, современные приложения обычно строятся как событийно-управляемые (нет непрерывного потока управления, есть множество обработчиков различных событий) , и для организации классического поллинга нужно идти на технические ухищрения.

Более эффективен опрос датчика по прерываниям или иным способам нотификации. В этом случае датчик при готовности

новых данных генерирует прерывание, и его обработчик забирает данные. Механизм слотов и сигналов *Qt* идеально подходит для этой ситуации.

Для того, чтобы получить оповещение о готовности данных датчика, достаточно связать сигнал датчика `readingChanged()` с каким-либо обработчиком и активировать получение данных методом `start()`. Важно понимать, что и создание объекта `QSensor`, и начало приема данных совсем необязательно как-то повлияют на физический датчик или его ядерный драйвер. Например, датчик ориентации в MeeGo постоянно включен и поставляет данные приложениям на основе MeeGoTouch. Когда мы запускаем наше тестовое приложение и выполняем вышеописанные действия, мы просто добавляем еще одного слушателя к уже работающему датчику.

Есть датчики, которые работают с высокой частотой, и даже механизм слотов и сигналов оказывается для них слишком тяжелым. Для решения этой проблемы применяются фильтры, создаваемые программистом, которые решают, достаточно ли полученные данные важны, чтобы оповещать о них слушателей. В нашем примере фильтры не используются.

В слоте `readSensor()`, который был связан с сигналом датчика `readingChanged()`, мы получаем свежее значение датчика ориентации. Для наглядности оно будет использоваться для вращения одного из двух больших смайлов на переднем плане приложения – красный смайл будет ориентироваться с помощью стандартных средств, синий смайл будет ориентироваться вручную по данным датчика.

Поскольку приложение написано с использованием библиотеки MeeGoTouch, его GUI имеет встроенную поддержку датчика ориентации, т. е. поворачивается вслед за поворотами планшета. Пока не нажата кнопка «Find sensors», оба смайла поворачиваются одинаково. После нажатия кнопки и активации датчика ориентации красный смайл продолжает вращаться, а синий становится неподвижным относительно корпуса планшета. Для вращения изображения применяются стандартный способ *Qt* – класс `QPainter`.

Для дальнейшей работы с датчиками целесообразно ознакомится с примером *Qt* Mobility «Sensor Explorer», входящим в

пакет `qt-mobility-examples`. В MeeGo его расположение `/usr/lib/qtmobility/examples/sensor_explorer`. `Sensor Explorer` позволяет удобно просмотреть имеющиеся в системе датчики, их параметры и данные.

## 9.3. Задания для самостоятельной работы

1. Создать приложение для работы с двумя датчиками одновременно.
2. Создать *Qt*-приложение без использования MeeGoTouch, так чтобы его пользовательский интерфейс реагировал на датчик ориентации.

## 9.4. Выводы

В этой лабораторной работе рассмотрен пример приложения, которое использует информацию с датчика ориентации в пространстве (G-сенсор). Мы определили, что для мобильных устройств эффективней использовать метод прерываний для работы с датчиками. Также увидели, что можно использовать библиотеку MeeGoTouch, при этом в графическом интерфейсе мы автоматически получили поддержку датчика ориетации.

## Список литературы


1. MeeGo Tablet Developer Preview. https://meego.com/downloads/releases/1.2/meego-tablet-developer-preview
2. MeeGo Touch. http://apidocs.meego.com/git-tip/mtf/
3. Qt Mobility 1.2 Sensors API. http://doc.qt.nokia.com/qt-mobility-snapshot/sensors-api.html
4. Sensor Explorer. http://doc.qt.nokia.com/qt-mobility-snapshot/sensors-sensor-explorer.html