СЛАЙД 1

РАЗРАБОТКА ПРИЛОЖЕНИЙ НА C++

Раздел 1. **Основы Qt Quick**

Тема № 2. **Фреймворк Qt и технология Qt Quick.**

СЛАЙД 2

Занятие № 2/1

ЛЕКЦИЯ № 2

Технология Qt Quick и инструменты разработки приложений.

1. Знакомство с фреймворком Qt и технологией Qt Quick.	
2. Обзор основных инструментов разработки приложений в ОС АВРОРА.	
3. Структура проекта: особенности и общие сведения.	

СЛАЙД 3

ПЕРВЫЙ УЧЕБНЫЙ ВОПРОС.
ЗНАКОМСТВО С ФРЕЙМВОРКОМ QT И ТЕХНОЛОГИЕЙ QT
1.1. Общая справка по технологиям кроссплатформенной и нативной разработки.

Нативная разработка – это создание продукта, который пишется на оригинальных языках программирования, созданных специально для выбранной платформы. Например, родными языками для **Android** являются **Java** и **Kotlin**, для **iOS** - **Swift** и **Objective-C**. Нативное приложение будет работать только на «своей» платформе.

Нативные мобильные приложения ориентированы на конкретную операционную систему, поэтому:

-более безопасны;

-интуитивно понятны;

-лучше работают;

-предоставляют разработчикам полный доступ к функциям целевого устройства (GPS, камера, телефон и т.д.).

При этом создание нативных приложений требует больше времени и финансовых затрат.

Кроссплатформенная разработка – это реализация приложения, которое работает на нескольких операционных системах. Это становится возможным с помощью универсального кода в кроссплатформенном фреймворке.

Для такой разработки используются универсальные фреймворки: **Flutter**, **React Native**, **Apache Cordova**, **Ionic** и другие.

Плюсы и минусы нативной разработки мобильных приложений

Разработка нативного приложения имеет свои преимущества и недостатки.

Итак, к ПЛЮСАМ нативной разработки относятся:

●БЕЗОПАСНОСТЬ: нативные приложения более защищены от вредоносных программ, включая вредоносное ПО и вирусы.

●БОЛЕЕ БЫСТРАЯ СКОРОСТЬ РАБОТЫ. При создании приложения используется понятный и привычный для платформы код, поэтому оно способно работать более быстро и качественно. При этом в кроссплатформенной разработке приложение может работать не так оперативно.

●БОЛЕЕ ШИРОКИЕ ВОЗМОЖНОСТИ: нативная разработка позволяет использовать все функции и возможности операционной системы и устройства, что в свою очередь расширяет возможности приложения.

К МИНУСАМ разработки нативных приложений можно отнести

●ВЫСОКАЯ ЦЕНА. Если вам необходимы две версии нативного приложения под разные операционные системы, платить нужно будет за два отдельных приложения.

СЛАЙД 4

Плюсы и минусы кроссплатформенной разработки

У кроссплатформенной разработки мобильных приложений также есть свои преимущества и недостатки. Расскажем о них подробнее.

К ПРЕИМУЩЕСТВАМ кроссплатформенной разработки можно отнести:

●ВОЗМОЖНОСТЬ ПОЛУЧИТЬ ПРИЛОЖЕНИЕ, которое работает сразу на нескольких ОС. Вам не нужно отдельно разрабатывать приложение под каждую систему. Приложение, разработанное по кроссплатформенной технологии будет доступно для установки как в App Store, так и в Google Play.

●ВЫГОДА. Кроссплатформенное приложение будет стоить дешевле двух отдельных версий нативного. Что касается разницы стоимости одного нативного и одного кроссплатформенного приложения, то сравнивать здесь будет не совсем корректно. Ведь стоимость разработки зависит еще и от требований к функционалу.

К НЕДОСТАТКАМ создания кроссплатформенных приложений относятся:

●МЕНЕЕ ГИБКАЯ ИНТЕГРАЦИЯ. Из-за разницы в операционных системах реализация некоторых функций будет затруднительной, а приложение может работать не так функционально. Кроме того, страдает оперативность.

●БОЛЕЕ ЖЕСТКИЕ ТРЕБОВАНИЯ. В магазинах приложений, в частности, в AppStore требования к кроссплатформенному приложению будут более высокими, а модерация – более долгой, и могут возникнуть проблемы.

●БОЛЕЕ МЕДЛЕННАЯ СКОРОСТЬ РАБОТЫ. Из-за разницы в интерфейсе операционных систем, кроссплатформенное приложение обычно работает более медленно, чем нативное.

СЛАЙД 5

1.2. Происхождение, текущее состояние и возможности фреймворка Qt

Фреймворк (англ. framework) - это набор библиотек, инструментов, принципов и подходов, которые предоставляют удобный функционал для разработки более сложного программного обеспечения. Он задает структуру проекта.

Qt – кроссплатформенный фреймворк, а для ОС Аврора и ряда десктопных дистрибутивов Linux на нем создаются нативные приложения.

Происхождение

Средства разработки Qt впервые стали известны общественности в мае 1995 года. Первоначально Qt разрабатывалось Хаарвардом Нордом (исполнительный директор) и Айриком Чеймб-Ингом (президент), которые познакомились в Норвежском институте технологии г. Тронхейм.

Хаавард начал интересоваться проблемами создания графического интерфейса на C++ с 1988 года. Тогда он получил от Шведской компании заказ на разработку библиотеки, средствами которой можно было бы реализовать графический интерфейс приложений. Спустя пару лет, летом 1990 года, Хаавард и Эрик начали совместную работу над приложением баз данных, которое обрабатывало снимки, получаемые с аппарата ультразвукового обследования. Система должна была иметь возможность

работы через графический интерфейс с пользователем, под управлением операционных систем Unix, Macintosh и Windows. Однажды, Хаавард и Эрик вышли на улицу, чтобы подышать свежим воздухом и насладиться летним солнцем. Они присели на скамейку в парке и Хаавард сказал: «НАМ НУЖНА ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ СИСТЕМА ОТОБРАЖЕНИЯ ИНФОРМАЦИИ». В результате обсуждения была заложена основа, для создания объектно-ориентированной, мультиплатформенной библиотеки, к разработке которой они должны были вскоре приступить

В 1991 году Хаарвард начал писать классы, которые фактически образовали Qt, причём проектные решения принимались совместно с Эриком.

В следующем году, то есть, в 1992 году, Эрику пришла идея реализации «СИГНАЛОВ и СЛОТОВ» - простой, но мощной парадигмы программирования GUI, которая в настоящее время заимствована некоторыми другими инструментами. Хаарвард воспринял эту идею с восторгом и реализовал её. К 1993 году Хаарвард и Эрик разработали первое графическое ядро Qt и могли создавать собственные виджеты. А в конце этого года Хаарвард предложил совместно заняться бизнесом и построить «самые лучшие в мире инструментальные средства разработки на C++ графического пользовательского интерфейса».

Но начало 1994 года не предвещало ничего хорошего, когда два молодых программиста собирались выйти на установившийся рынок, не имея ни заказчиков, ни законченного продукта, ни денег. К счастью, жены обоих имели работу и могли поддержать своих мужей в течение двух лет, которых, как считали Эрик и Хаарвард, будет достаточно для разработки программного продукта, позволяющего начать зарабатывать деньги.

Буква «**Q**» была выбрана в качестве префикса классов, поскольку эта буква имела красивое начертание в шрифте **Emacs**, которым пользовался Хаарвард. Была добавлена буква «**t**», означающая «**toolkit**» (инструментарий).

Компания зарегистрирована 4 марта 1994 года и по началу называлась «**Quasar Technologies**», затем «**Troll Tech**», затем «**Trolltech**», а потом просто «**Qt Software**».

В апреле 1995 года через посредничество одного университетского профессора, знакомого Хаарварда, норвежская компания «**Metis**» заключила с ними контракт на разработку программного обеспечения на основе Qt. Примерно в это же время «Trolltech» приняла на работу Арнта Гулдбрансена, которые в течение своих шести лет работы в этой компании продумал и реализовал оригинальную систему документирования, а также внёс определённый вклад в программный код Qt.

20 мая 1995 года Qt 0.90 был установлен на сайте **sunsite.unc.edu**[1]. Спустя

[1] **ibiblio** (ранее **SunSITE.unc.edu** и **MetaLab.unc.edu**) - «коллекция коллекций», сайт, на котором располагается разнообразная общедоступная информация и программное обеспечение с открытым кодом. ibiblio - один из старейших интернет-сайтов и заповедник самой разнообразной общедоступной информации, включающей в себя свободное программное обеспечение, музыку, литературу и др

шесть дней о выпуске этой версии было объявлено на comp.os.linux.announce. Это была первая публичная версия Qt. Qt можно было использовать в разработках как Windows, так и Unix, причём программный интерфейс был одинаковый на обеих платформах. С первого дня предусматривались две лицензии применения Qt: коммерческая лицензия предназначалась для коммерческих разработок, и свободно распространяемая версия предназначалась для разработок Open-source проектов. Контракт с «Metis» сохранил компанию на плаву, хотя в течение долгих 10-ти месяцев не было продано ни одной коммерческой лицензии Qt.

В марте 1996 года Европейское управление космических исследований стало вторым заказчиком Qt, которое приобрело десять коммерческих лицензий. Верящие в удачу Эрик и Хаарвард приняли на работу ещё одного разработчика.

Qt 0.97 был выпущен в конце мая, а 24 сентября 1996 года вышла версия Qt 1.0.

К концу 1996 года вышла версия Qt 1.1 и восемь заказчиков - все из разных стран - приобрели в общей сложности 18 лицензий.

СЛАЙД 6

В этом году был также основан Маттиасом Эттричем проект KDE[2].

Принятое Маттиасом решение по применению Qt для построения KDE помогло Qt стать фактическим стандартом по разработке на C++ графического пользовательского интерфейса в системе Linux.

Маттиас присоединился к «Trolltech» в 1998 году.

Сентябрь 1998 года – появление последней значимой версии Qt первого выпуска, Qt 1.40.

Qt 2 имела новую лицензию для открытого исходного кода - Q Public License (QPL), которая соответствовала ОПРЕДЕЛЕНИЮ ОТКРЫТОГО ИСХОДНОГО КОДА.

В августе 1999 года Qt выиграла премию журнала «Linux World» за лучшую библиотеку или инструментальное средство. Примерно в это же время была образована компания «Trolltech Pty Ltd» (Австралия).

Qt 3.0 была выпущена в 2001 году. Qt теперь работала в системах Windows, Mac Os X, Unix.

Qt 3.0 содержала 42 новых класса, и объем её программного кода превышал 500 000 строк.

Qt 3 представляла собой важный шаг вперёд по сравнению с Qt 2, которая, в частности, значительно улучшила поддержку локализации и кодировки Unicode[3], ввела совершенно новые виджеты по просмотру и редактированию текста и класс

[2] **KDE** - международное сообщество, разрабатывающее свободную среду рабочего стола KDE Plasma, набор связанных между собой программ, а также несколько веб-сервисов. Программное обеспечение KDE построено на основе кроссплатформенного инструментария разработки пользовательского интерфейса Qt.

[3] Юникод -стандарт кодирования - это **16-разрядная кодировка символов, обеспечивающая достаточно кодирования для всех языков**.

регулярных выражений[4], аналогичных применяемым языкам **Perl**[5].

Qt 3.0 была удостоена премии «Software Development Times» в категории «Высокая продуктивность» в 2002 году.

СЛАЙД 7

Летом 2005 года была выпущена Qt 4.0. Имея около 500 классов и более 9000 функций.

Qt 4 оказалась больше и богаче любой предыдущей версии. Qt 4 была разбита на несколько библиотек, чтобы разработчики могли использовать только нужные им части Qt.

Версия Qt 4 представляет собой большой шаг вперёд по сравнению с предыдущими версиями. Она содержит полностью новый набор эффективных и простых в применении классов-контейнеров, усовершенствованную функциональность архитектуры модель/представление, быстрый и гибкий фреймфорк графики 2D и мощные классы для просмотра и редактирования текста в кодировке Unicode, не говоря уже о тысячах небольших улучшений по всему спектру классов Qt.

Qt 4 является первой версией Qt, доступной на всех поддерживаемых платформах как для коммерческой разработки, так и для разработки с открытым исходным кодом.

17 июня 2008 года Nokia приобрела компанию Trolltech ASA и сменила название сначала на Qt Software, а затем на Qt Development Framework.

Nokia сосредоточилась на превращении Qt в основную платформу разработки для своих устройств, включая порт для платформы **Symbian S60**. Версия 1.0 Nokia Qt SDK была выпущена 23 июня 2010 года. Исходный код был доступен через Gitorious, репозиторий исходного кода git, ориентированный на сообщество, с целью создания более широкого сообщества, использующего и улучшающего Qt.

14 января 2009 года в Qt версии 4.5 была добавлена еще одна опция, LGPL[6], чтобы сделать Qt более привлекательным как для проектов с открытым исходным кодом, не связанных с GPL, так и для закрытых приложений.

В феврале 2011 года Nokia объявила о своем решении отказаться от технологий Symbian и вместо этого основывать свои будущие смартфоны на платформе Windows Phone (и с тех пор поддержка этой платформы также была прекращена).

Месяц спустя Nokia объявила о продаже коммерческого лицензирования и профессиональных услуг Qt компании Digia с непосредственной целью распространения поддержки Qt на платформы Android, iOS и Windows 8, а также продолжения сосредоточения внимания на разработке настольных и встраиваемых систем, хотя Nokia должна была оставаться основной силой разработки фреймворка на тот момент.

---

[4] Регулярные выражения (их еще называют regexp, или regex) - это **механизм для поиска и замены текста**. В строке, файле, нескольких файлах... Их используют разработчики в коде приложения, тестировщики в автотестах, да просто при работе в командной строке!

[5] Perl - это язык программирования, который создали специально для обработки текста. Высокоуровневый язык, по синтаксису похожий на C

[6] Лицензия на свободное программное обеспечение

В марте 2011 года Nokia продала часть коммерческого лицензирования Qt компании Digia, создав Qt Commercial.

В августе 2012 года Digia объявила о приобретении Qt у Nokia. Команда Qt в Digia начала свою работу в сентябре 2012 года. Они выпустили Qt 5.0 в течение месяца и более новые версии каждые шесть месяцев с новыми функциями и дополнительными поддерживаемыми платформами.

Qt 5 был официально выпущен 19 декабря 2012 года. Эта новая версия ознаменовала собой серьезные изменения в платформе, в которой важную роль играли графика с аппаратным ускорением, QML и JavaScript. Традиционные QWidgets, работающие только на C++, продолжали поддерживаться, но не выиграли от улучшений производительности, доступных благодаря новой архитектуре. Qt 5 вносит значительные улучшения в скорость и простоту разработки пользовательских интерфейсов.

Разработка фреймворка Qt 5 перешла на открытое управление в qt-project.org, что позволило разработчикам за пределами Digia отправлять патчи на проверку.

В сентябре 2014 года Digia передала бизнес Qt и авторские права своей дочерней компании Qt Company, которая владела 25 брендами, связанными с Qt.

В мае 2016 года Digia и Qt полностью разделились на две независимые компании.

СЛАЙД 8

Во вторник, 8 декабря 2020 года компания Qt Company обновила свой фреймворк для кроссплатформенной разработки. Если сравнить с пятой версией 2012 года, в Qt 6.0 появились новые API для 3D-графики, улучшенная поддержка C++17 и система сборки cmake.

За годы разработки между версиями пятого Qt стало значительно труднее сохранять совместимость на уровне исходных кодов и двоичных файлов.

Разработчики оправдывают историческую неизбежность Qt 6.0 необходимостью адаптировать его к новому миру и его изменчивым требованиям.

Подводя итоги, разработчики полностью переработали базовые классы, их используют для системы сигнала-слотов и для реализации нового типа контейнера, который объединяет в себе свойства массива и связанного списка.

Qt 6 работает в едином интерфейсе аппаратного рендеринга[7] с поддержкой:

-Direct 3D;

-Metal;

-Vulkan;

-OpenGL.

[7] Рендеринг относится к процессу создания автоматизированных изображений с помощью компьютерных программ. Это может быть выполнено с помощью аппаратного или программного рендеринга.
Аппаратный рендеринг выполняется с помощью компьютерного чипа, который возвращает изображения непосредственно на экран.
Программный рендеринг обрабатывается без помощи какого-либо оборудования и выполняется в процессоре, в то время как аппаратный рендеринг опирается на графический блок
Рендеринг программного обеспечения осуществляется исключительно с помощью компьютерного кода или приложений.

СЛАЙД 9

Текущее состояние и возможности фреймворка Qt

Qt предоставляет большой НАБОР ВОЗМОЖНОСТЕЙ для работы с GUI (графический интерфейс пользователя), включая графические элементы управления, взаимодействие с мышью и клавиатурой, работу с окнами и диалоговыми окнами, прорисовку графических элементов и многое другое.

Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML.

Является полностью объектно-ориентированным, расширяемым и поддерживающим технику КОМПОНЕНТНОГО ПРОГРАММИРОВАНИЯ[8].

КОМПИЛЯЦИЯ – это превращение программного кода в исполняемый код для процессора: на входе было то, что могли прочитать вы, а на выходе – то, что может прочитать и исполнить компьютер.

КОМПИЛЯТОР – это программа, которая переводит текст, написанный на языке программирования, в набор машинных кодов.

Отличительная особенность – использование МЕТАОБЪЕКТНОГО КОМПИЛЯТОРА – предварительной системы обработки исходного кода.

Расширение возможностей обеспечивается системой ПЛАГИНОВ, которые возможно размещать непосредственно в панели визуального редактора.

Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

ПЛАГИН – (от англ. plug in – подключать, plugin – подключаемый модуль). Это дополнение к базовой программе или приложению, которое расширяет её возможности и функции.

СЛАЙД 10

Фреймворк Qt (иногда, что некорректно, называют библиотекой) разделен на ряд модулей. Далее несколько подробнее рассмотрим наиболее важные и распространённые модули феймворка Qt.

●***QtCore*** – классы ядра библиотеки, используемые другими модулями;

Все остальные модули Qt полагаются на этот модуль. Чтобы включить определения классов модуля, используется следующая директива:

```
#include <QtCore>
```

Если для сборки своих проектов используется qmake, ***QtCore*** включен по умолчанию.

---

[8] **Компонентно-ориентированное программирование** - парадигма программирования, опирающаяся на понятие **компонента** - независимого модуля исходного кода программы, предназначенного для повторного использования и развёртывания и реализующегося в виде множества языковых конструкций, объединённых по общему признаку и организованных в соответствии с определёнными правилами и ограничениями.

QtCore добавляет в C++ следующие возможности:

-очень мощный механизм для бесшовной объектной связи, называемый сигналами и слотами;

-запрашиваемые и проектируемые свойства объекта;

-иерархические и запрашиваемые деревья объектов, которые организуют владение объектом естественным способом с помощью защищенных указателей (QPointer);

-динамическое приведение, которое работает за пределами библиотеки;

-Qt Core предоставляет независимый от платформы механизм хранения двоичных файлов в исполняемом приложении;

-Qt Core предоставляет некоторые ключевые платформы Qt.

●**Qt Quick** – модуль для поддержки QML;

Модуль Qt Quick - это стандартная библиотека для написания приложений QML. В то время как модуль Qt QML предоставляет механизм QML и языковую инфраструктуру, модуль Qt Quick предоставляет все основные типы, необходимые для создания пользовательских интерфейсов с помощью QML. Он предоставляет визуальный холст и включает типы для создания и анимации визуальных компонентов, получения пользовательского ввода, создания моделей и представлений данных, а также отложенного создания экземпляров объектов.

Модуль Qt Quick предоставляет как API QML, который предоставляет типы QML для создания пользовательских интерфейсов с языком QML, так и API C++ для расширения приложений QML с помощью кода C++

Для создания пользовательских интерфейсов также доступен набор элементов управления пользовательского интерфейса на основе Qt Quick.

Важные понятия в Qt Quick

Qt Quick предоставляет все необходимое для создания многофункционального приложения с гибким и динамичным пользовательским интерфейсом. Он позволяет строить пользовательские интерфейсы на основе поведения компонентов пользовательского интерфейса и того, как они соединяются друг с другом, а также предоставляет визуальный холст с собственной системой координат и механизмом рендеринга. Эффекты анимации и перехода являются первоклассной концепцией в Qt Quick, а визуальные эффекты могут быть дополнены специализированными компонентами для эффектов частиц и шейдеров.

СЛАЙД 11

●**QtScript** – классы для работы с Qt Scripts или классы внутреннего скриптового языка Qt Scripts;

QtScript - скриптовый язык, который, начиная с версии 4.3.0, является составной частью Qt.

Язык основан на стандарте ECMAScript с некоторыми расширениями, такими как возможность соединения с сигналами и слотами объектов QObject.

Использование QtScript (или QSA для более ранних версий Qt) позволяет легко превратить Qt-приложение в полностью переконфигурируемую программную платформ.

С выходом Qt 5.5 (выпущен 1 июля 2015 года), QtScript был объявлен устаревшим

●**QTGUI** – компонент для создания графического интерфейса;

Модуль Qt GUI предоставляет классы для интеграции оконной системы, обработки событий, интеграции OpenGL и OpenGL ES, 2D-графики, базовых изображений, шрифтов и текста. Эти классы используются внутри технологий пользовательского интерфейса Qt, а также могут использоваться напрямую, например, для написания приложений с использованием низкоуровневых графических API OpenGL ES.

Для разработчиков приложений, пишущих пользовательские интерфейсы, Qt предоставляет API более высокого уровня, такие как Qt Quick, которые гораздо более подходят, чем средства реализации, находящиеся в модуле Qt GUI.

Чтобы включить определения классов модуля, используется следующая директива:

```
#include <QtGui>
```

Если вы используете qmake для создания своих проектов, Qt GUI включен по умолчанию.

Чтобы отключить графический интерфейс Qt, добавьте следующую строку в свой файл .pro:

```
QT -= gui
```

СЛАЙД 12

●**QtNetwork** – набор классов для сетевого программирования.

Модуль Qt Network предоставляет набор API для программирования приложений, использующих TCP/IP. Такие операции, как запросы, файлы cookie и отправка данных по HTTP, обрабатываются различными классами C++.

Поддержка различных высокоуровневых протоколов может меняться от версии к версии. В версии 4.2.x присутствуют классы для работы с протоколами FTP и HTTP.

Для работы:

-с протоколами TCP/IP предназначены такие классы, как **QTcpServer**;

-с протоколами TCP предназначены такие классы, как **QTcpSocket**;

-с протоколами UDP предназначены такие классы, как **QUdpSocket**;

●**QtOpenGL** – набор классов для работы с OpenGL[9];

OpenGL - это стандартный API для рендеринга 3D-графики. OpenGL занимается только 3D-рендерингом и практически не обеспечивает поддержку проблем программирования графического пользовательского интерфейса. Пользовательский

[9] **OpenGL (Open Graphics Library)** - это стандарт, который используется для рендеринга (процесс получения изображения по модели с помощью компьютерной программы) 2D и 3D графики.
OpenGL (Open Graphics Library) - спецификация, определяющая платформонезависимый (независимый от ЯП) программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

интерфейс для приложения OpenGL должен быть создан с помощью другого набора инструментов, например Cocoa на платформе macOS, Microsoft Foundation Classes (MFC) под Windows или Qt на обеих платформах.

Модуль **Qt OpenGL** упрощает использование **OpenGL** в приложениях Qt. Он предоставляет класс виджета OpenGL, который можно использовать так же, как и любой другой виджет Qt, за исключением того, что он открывает буфер отображения OpenGL, где вы можете использовать API OpenGL для визуализации содержимого.

Чтобы включить определения классов модуля, необходимо использовать следующую директиву:

```
#include <QtOpenGL>
```

Чтобы связать модуль (линковка), необходимо добавить эту строку в файл qmake .pro:

```
QT += opengl
```

Модуль Qt OpenGL реализован как независимая от платформы оболочка Qt/C++ вокруг зависящих от платформы API-интерфейсов GLX (версия 1.3 или новее), WGL или AGL C. Хотя предоставляемая базовая функциональность очень похожа на библиотеку GLUT Марка Килгарда, приложения, использующие модуль Qt OpenGL, могут использовать весь Qt API для реализации функций графического интерфейса, не связанных с OpenGL.

СЛАЙД 13

●**QtSql** – набор классов для работы с базами данных с использованием SQL.

Основные классы данного модуля в версии 4.2.x:

-**QSqlDatabase** - класс для предоставления соединения с базой, для работы с какой-нибудь конкретной базой данных требует объект, унаследованный от класса QSqlDriver;

-**QSqlDriver** - абстрактный класс, который реализуется для конкретной базы данных и может требовать для компиляции SDK базы данных.

Например, для сборки драйвера под СУБД Firebird или InterBase требуются .h-файлы и библиотеки статической компоновки, входящие в комплект поставки данной СУБД;

●**QtSvg** – классы для отображения и работы с данными **Scalable Vector Graphics** (SVG)[10];

Масштабируемая векторная графика (SVG) - это язык на основе XML для описания двумерной векторной графики.

Qt предоставляет классы для рендеринга и отображения рисунков SVG в виджетах[11] и на других устройствах рисования.

---

[10] **SVG (Scalable Vector Graphics)** - это один из форматов векторной графики, которая, в отличие от растровой, имеет одно большое преимущество: она не меняет свое качество и четкость при изменении размера. Дословная расшифровка аббревиатуры - *«масштабируемая векторная графика»*

[11] Виджеты - это **«строительные блоки» для создания пользовательского интерфейса**.
Визуальные объекты на форме, такие как кнопки, метки, поля, меню, раскрывающиеся списки и т. д. - всё это относится к виджетам.

СЛАЙД 14

●**QtXml** – модуль для работы с XML[12], поддерживаются модели SAX[13] и DOM[14];

Модуль QtXml обеспечивает работу с потоками чтения и записи XML документов и реализацию их в форме SAX и DOM.

Для включения определений классов этого модуля используйте следующую директиву:

```
#include <QtXml>
```

Для линковки[15] приложения с этим модулем, добавьте в ваш qmake файл проекта .pro:

```
QT += xml
```

Стоит оратить внимание, что, начиная с QtXML 5.15.18, модуль больше не получает дополнительных возможностей. Для итеративного чтения или записи XML-документов (SAX) мы рекомендуем использовать классы QXmlStreamReader и QXmlStreamWriter Qt Core. Эти классы проще в использовании и более совместимы со стандартом XML.

●**QtAssistant** – справочная система;

Qt Assistant - это инструмент для просмотра онлайн-документации в формате файла справки Qt.

Иными словами, Qt Assistant - инструментом для представления интерактивной документации.

СЛАЙД 15

●**QtTest** – классы для поддержки модульного тестирования;

Модуль **QtTest** предоставляет классы для модульного тестирования (юнит-тестирование) приложений и библиотек Qt.

Все общедоступные методы находятся в пространстве имен QTest.

Приложения, которые используют классы юнит-тестирования Qt, должны быть настроены для сборки с модулем QtTest. Для включения определений классов этого модуля используется следующая директива:

```
#include <QtTest>
```

---

[12] **XML** (eXtensible Markup Language) - это расширяемый язык разметки, предназначенный для хранения и передачи данных в структурированном виде. Данные находятся внутри тегов, которые помогают организовать данные в логической иерархии, понятной человеку и компьютеру.
XML-популярный формат для обмена данными между различными системами и приложениями. Его универсальность делает XML важным инструментом в веб-разработке, настройке программного обеспечения и других областях.

[13] Существуют две стратегии обработки XML документов: DOM (Document Object Model) и SAX (Simple API for XML).
SAX - Способ последовательного чтения/записи XML-файлов.

[14] **DOM** - это объектная модель документа, которую браузер создает в памяти компьютера на основании HTML-кода, полученного им от сервера. Иными словами, это представление HTML-документа в виде дерева тегов.
Основное их отличие связано с тем, что использование DOM позволяет читать и вносить изменения в существующий XML-документ, а также создавать новый. Стратегия использования SAX основывается на том, что содержимое XML-документа только анализируется. XML-текст может быть больших размеров: DOM должен весь документ «заглотить» и проанализировать, а SAX-парсер обрабатывает XML-документ последовательно и не требует дополнительной памяти.

[15] **Линковка** это процесс компоновки различных кусков кода и данных вместе, в результате чего получается один исполняемый файл. Линковка может быть выполнена во время компиляции, во время загрузки (загрузчиком) и также во время исполнения (исполняемой программой)

Для линковки приложения с этим модулем, добавьте в ваш qmake файл проекта .pro:

```
CONFIG += qtestlib
```

Кроме того, класс **QSignalSpy** обеспечивает простой самоанализ сигналов и слотов Qt.

Header: `#include <QSignalSpy>`

CMake:
```
find_package(Qt6 REQUIRED COMPONENTS Test)
target_link_libraries(mytarget PRIVATE Qt6::Test)
```

qmake: `QT += testlib`

Класс **QAbstractItemModelTester** позволяет проводить неразрушающее тестирование моделей элементов.

Header: `#include <QAbstractItemModelTester>`

CMake:
```
find_package(Qt6 REQUIRED COMPONENTS Test)
target_link_libraries(mytarget PRIVATE Qt6::Test)
```

qmake: `QT += testlib`

Для модуля Qt Test не существует гарантии двоичной совместимости. Это означает, что приложение, использующее Qt Test, гарантированно будет работать только с той версией Qt, для которой оно было разработано. Однако совместимость источников гарантирована.

●**Qt3Support** – модуль с классами, необходимыми для совместимости с библиотекой Qt версии 3.x.x;

Модуль Qt3Support предоставляет классы, которые облегчают портирование[16] с Qt 3 на Qt 4.

Для включения определений классов этого модуля используйте следующую директиву:

```
#include <Qt3Support>
```

Для линковки приложения с этим модулем, добавьте в ваш qmake файл проекта .pro:

```
QT += qt3support
```

СЛАЙД 16

●**QtCLucene** – модуль для поддержки полнотекстового поиска. Применяется в новой версии Assistant в Qt 4.4;

●**QtXmlPatterns** – модуль обеспечивает поддержку проверки XPath[17], XQuery[18], XSLT[19] и XML Schema[20].

---

[16] Портируемость – компиляция кода (обычно в некоторый промежуточный код, который затем интерпретируется или компилируется во время исполнения, «на лету», англ. Just-In-Time), затем запускать его на множестве платформ без каких-либо изменений.

[17] XPath –*это* язык запросов для навигации по разметке XML.

[18] **XQuery** – язык запросов и функциональный язык программирования, разработанный для обработки данных в формате XML, простого текста, JSON или других предметно-специфичных форматах. XQuery использует XML как свою модель данных. Предназначен для запроса и преобразования коллекций структурированных и неструктурированных данных.

[19] *XSLT (eXtensible Stylesheet Language Transformations)* – язык преобразования XML-документов.

Для включения определения классов модуля, используется следующая директива:

```
#include <QtXmlPatterns>
```

Чтобы связать модуль, добавляется эта строку в файл qmake .pro:

```
QT += xmlpatterns
```

В Qt для ОС АВРОРА модуль Qt XML Patterns обеспечивает поддержку **XQuery 1.0** и **Xpath 2.0**;

СЛАЙД 17

●**Phonon** – модуль для поддержки воспроизведения и записи видео и аудио, как локально, так и с устройств, и по сети.

Phonon – кроссплатформенный мультимедийный каркас, который даёт возможность использовать аудио- и видеоконтент в приложениях Qt.

Пространство имен Phonon содержит перечень всех предоставляемых модулем классов, функций и пространств имен.

Приложения, использующие классы Phonon, нужно сконфигурировать для сборки вместе с модулем Phonon.

Следующая декларация в файле проекта qmake гарантирует, что приложение будет скомпилировано и связано с данным модулем:

```
QT += phonon
```

Для включения определений классов этого модуля используйте следующую директиву:

```
#include <phonon>
```

Начиная с Qt 5 заменён на QtMultimedia.

●**QtMultimedia** – модуль для поддержки воспроизведения и записи видео и аудио

Qt Multimedia – это важный модуль, предоставляющий богатый набор типов QML и классов C++ для обработки мультимедийного контента. Он также предоставляет необходимые API для доступа к функциям камеры и радио. Входящий в комплект Qt Audio Engine предоставляет типы для позиционного трехмерного воспроизведения звука и управления контентом.

Функционал данного модуля разделен на следующие подмодули:

-подмодуль **Qt Multimedia** предоставляет API для конкретных случаев использования мультимедиа и обеспечивает функции аудио, видео, радио и камеры.

Чтобы включить Qt Multimedia в проекте, необходимо добавить директиву в файлы C++:

```
#include <QtMultimedia>
```

Чтобы связать библиотеки C++, необходимо добавить в файл проекта qmake следующую строку:

```
QT += multimedia
```

[20] ML Schema – *язык описания структуры XML-документа*

-подмодуль **Qt Multimedia Widgets** – предоставляет мультимедийный API на основе виджетов.

Классы, предоставляемые подмодулем Qt Multimedia Widgets:

-*QCameraViewfinder* - предоставляет виджет видоискателя камеры;

-*QGraphicsVideoItem* - графический элемент, отображающий видео, созданное QMediaObject;

-*QVideoWidget* - виджет, отображающий видео, созданное медиа-объектом.

-*QVideoWidgetControl* - медиа-элемент управления, реализующий видео-виджет.

Эти классы являются частью модуля Qt Multimedia Widgets.

Чтобы включить виджеты Qt Multimedia в проекте, необходимо добавить следующую директиву в файлы C++:

```
#include <QtMultimediaWidgets>
```

Чтобы связать библиотеки C++, необходимо добавить в файл проекта qmake следующую строку:

```
QT += multimediawidgets
```

Типы QML можно импортировать в разрабатываемое приложение с помощью следующего оператора импорта в файле .qml.:

```
import QtMultimedia 5.15
```

Итак, мы отметили, что если вы собираетесь использовать классы C++ в своем приложении, включите определения C++, используя следующие директивы:

```
#include <QtMultimedia>
#include <QtMultimediaWidgets>
```

Если вы используете только некоторые классов из этих модулей, рекомендуется включать только эти конкретные классы вместо модуля (модулей) полностью.

Чтобы связать соответствующие библиотеки C++, добавьте в файл проекта qmake следующее:

```
QT += multimedia multimediawidgets
```

СЛАЙД 18

●**QtDeclarative** – модуль, предоставляющий декларативный фреймворк для создания динамичных, настраиваемых пользовательских интерфейсов.

Модуль Qt Declarative предоставляет декларативную структуру для создания высокодинамичных настраиваемых пользовательских интерфейсов.

Для включения определений классов этого модуля необходимо испольтзовать следующую директиву:

```
#include <QtDeclarative>
```

Для линковки приложения с этим модулем, необходимо добавить в qmake файл проекта .pro следующую строку:

```
QT += declarative
```

Одним из преимуществ фреймворка Qt - подробная документация, сопровождающаяся большим количеством примеров. Исходный код примеров содержит подробные комментарии и описание, что также упрощает изучение Qt.

Также Qt поддерживает множество платформ, включая все основные настольные и встраиваемые платформы.

Благодаря абстракции платформы Qt теперь проще, чем когда-либо, при необходимости перенести Qt на вашу собственную платформу.

СЛАЙД 19

1.3. Что такое декларативное описание UI, как устроены приложения с Qt Quick.

Что такое UI?

UI – это user interface, пользовательский интерфейс, проще говоря – оформление сайта:

-сочетания цветов;

-шрифты;

-иконки и кнопки.

Данный IT-термин означает визуальную часть интерфейса.

Пользовательский интерфейс - это способ взаимодействия пользователя и программы. Давайте разбираться дальше зачем он нужен

Пользовательский интерфейс – это все, что помогает людям управлять устройствами и программами с помощью голоса, нажатий, жестов, через командную строку и даже силой мысли (такое теперь тоже есть). Самый популярный вид интерфейсов сейчас - UI приложений.

Зачем нужен интерфейс в принципе?

Интерфейс помогает двум объектам понимать друг друга и обмениваться информацией.

Интерфейс - это «язык общения», который понимают оба объекта, которые взаимодействуют друг с другом с целью решить определенный вопрос.

Если каждое приложение или программа, установленная на компьютере, планшете или смартфоне, — это помощник, то интерфейс - это способ общаться (взаимодействовать) с ней, чтобы она помогала в вашем деле на работе и в жизни.

К примеру, у цифровых систем пользовательские интерфейсы бывают графические, голосовые, командной строки, жестовые - все это интерфейсы. Через пользовательский интерфейс мы получаем доступ к новым возможностям, которые дает приложение для обучения, работы, творчества, развлечений.

Также распространены программный, аппаратный, аппаратно-программный интерфейсы. Такие интерфейсы обеспечивают взаимодействие не только между человеком и машиной (устройством), но и между программами, оборудованием или компьютерами:

- **аппаратный:** соединяет друг с другом два объекта, например, помогает подключить смартфон к ноутбуку с помощью WiFi или кабеля;

- **программный (API):** создает связь между приложениями/программами, к примеру, подключение API одного приложения к другому. Самый популярный сценарий работы - авторизация через соцсети на сайтах;
- **аппаратно-программный:** комбинация технических элементов под управлением программного обеспечения.

Виды пользовательского интерфейса

Пользовательские интерфейсы бывают жестовые, тактильные, голосовые, графические, командной строки и даже нейронные.

Интерфейс командной строки и текстовый интерфейс (Command Line Interface или CLI)

Командная строка все еще очень популярна среди системных администраторов и программистов. Это один из первых методов взаимодействия с компьютером. Она обладает особым шармом — создает ощущение общения тет-а-тет с машиной без посредников. Командная строка — как бесконечный лист А4, на котором пользователь вводит текст команд и получает результаты работы в виде текста.

Графический пользовательский интерфейс (Graphical User Interface или GUI)

Самый популярный тип UI. Представляет собой окошко с различными элементами управления. Пользователи взаимодействуют с ними с помощью клавиатуры, мыши и голосовых команд: жмут на кнопки, тыкают мышкой, смахивают пальцем.

Жестовый, голосовой, тактильный, нейронный

«Любая достаточно развитая технология неотличима от магии», — как-то сказал английский писатель-фантаст и футуролог Артур Кларк.

Например, через Voice User Interface вы можете отдавать команды своему смартфону через голосовых помощников: Siri от Apple, Alexa от Amazon или Алиса от Яндекса.

NUI (жестовые, естественные) применяют в играх для приставок Xbox, Nintendo Wii или PlayStation. Эту же технологию вы найдете в оборудовании «умного дома», например, при включении света или регулировании громкости Яндекс.Станции с помощью изменения положения руки.

Производители качают технологии и расширяют возможности машин, и наслаждаться новыми фишками гаджетов можно, даже посылая мысли напрямую в компьютер.

Аббревиатура UX[21] расшифровывается как user experience – «пользовательский опыт».

UX - это совокупность ощущений от взаимодействия пользователя с интерфейсом системы, продукта или услуги.

UX - это что-то про юзабилити?

[21] Питер Морвиль - информационный архитектор, основатель компании Semantic Studios.
Илья Бирман -продуктовый и информационный дизайнер

Немного нет. Юзабилити - это степень удобства использования или оценка качества интерфейса, а UX - это результат взаимодействия с интерфейсом. Проще говоря, если юзабилити - это то, насколько хорошо грабли убирают листья, то UX - синяк на лбу от граблей. Но, в общем-то да, всё это про грабли.

Что влияет на UX?

- прозрачность логики работы интерфейса и её соответствие ментальной модели пользователей;
- очевидность каждого из действий;
- единообразие интерфейсных решений;
- качество контента и понятность языка.

И много чего ещё: каждый из идеологов находит факторы, влияющие на пользовательский опыт.

Как управлять пользовательским опытом?

- узнать потребности ваших клиентов, например, во время сессий глубинных интервью, анализа запросов в службу поддержки или чтения отзывов на другом сайте;
- создавать и совершенствовать существующие интерфейсы для достижения потребностей ваших клиентов.

Как оценить пользовательский опыт?

Провести юзабилити-тестирование, опрос удовлетворённости или просто увидеть большое количество отказов среди клиентов.

Простыми словами, UX - это то, каким образом пользователь взаимодействует с интерфейсом и насколько приложение для него понятны и удобны. В UX входит навигация по приложению, функционал меню и результат взаимодействия со страницами.

СЛАЙД 20

Знакомы ли вам эти понятия: императивный и декларативный стиль программирования?

Для того, чтобы описать отличия, нужно сначала понять, что это в принципе такое.

В нативных средствах IOS и Android используется **Императивный** стиль создания **UI**.

То есть мы вручную создаем UI сущность, которую впоследствии мутируем различными способами. Получается мы создаём общий объект (Application), а потом поочередно меняем его поля так, чтобы он стал таким, какой нам нужен.

Ещё часто говорят, что императивный способ – это о том **«как».**

СЛАЙД 21

В **Декларативном UI** всё иначе. Этот подход описывает **«что»** надо сделать? Без ненужных деталей реализации. Все они все вынесены в какие-то отдельные чистые функции. Тут для того, чтобы изменить UI нужно создать новый объект описания текущего состояния, просто поменять пару полей не выйдет.

То есть, в декларативном стиле UI – всегда результат от функции состояния, и никак иначе.

СЛАЙД 22

Таким образом, **ИМПЕРАТИВНЫЙ** подход использует последовательность утверждений, меняющих состояние программы для достижения результата.

При **ДЕКЛАРАТИВНОМ** же подходе описывается результат, которого необходимо достичь. При этом без указания шагов, необходимых для его получения

Преимущество декларативного подхода перед императивным:

●Декларативный подход зачастую легче понять и использовать, благодаря тому, что описывает то, что нужно получить, а не как это нужно сделать;

●Декларативный код часто проще читать и отлаживать, потому что он чаще всего читается как естественный язык;

●Декларативное программирование позволяет вам писать более общие функции, которые потенциально могут использоваться для нескольких целей.

СЛАЙД 23

QtQuick

QtQuick – представляет собой набор технологий для создания динамических пользовательских интерфейсов при помощи языка QML.

QML – язык программирования и одновременно его интерпретатор, который предоставляет API для совместного использования QML-кода, JavaScript и C++.

QtQuick разбивает пользовательский интерфейс на более мелкие элементы, которые можно объединить в компоненты.

QtQuick описывает внешний вид и поведение этих элементов пользовательского интерфейса.

Это описание пользовательского интерфейса можно дополнить кодом JavaScript, чтобы обеспечить не только простую, но и более сложную логику.

СЛАЙД 24

Таким образом, Qt Quick является коллекцией следующих технологий:

●QML - язык разметки для пользовательских интерфейсов

●JavaScript - динамический скриптовый язык

●Qt C++ - используется для внутренней реализации (back-end)

Подобно HTML, QML - это язык разметки. Он состоит из тегов, называемых типами в Qt Quick, которые заключены в фигурные скобки: *Item {}*.

Он был разработан с нуля для создания пользовательских интерфейсов, ускорения и облегчения чтения разработчиками. Пользовательский интерфейс может быть дополнительно улучшен с помощью кода JavaScript.

Qt Quick легко расширяется с помощью вашей собственной встроенной функциональности с использованием Qt C++.

Проще говоря, ДЕКЛАРАТИВНЫЙ ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС называется front-end, а собственные части называются back-end. Это позволяет

разработчику отделить трудоемкую и встроенную работу вашего приложения от части пользовательского интерфейса.

В типичном проекте интерфейс разрабатывается на QML/JavaScript. Внутренний код, который взаимодействует с системой и выполняет тяжелую работу, разработан с использованием Qt C++. Это обеспечивает естественное разделение между разработчиками, более ориентированными на дизайн, и функциональными разработчиками. Как правило, серверная часть тестируется с помощью Qt Test, платформы модульного тестирования Qt, и экспортируется для использования разработчиками внешнего интерфейса.

СЛАЙД 25

1.4. Описание Silica

Основы использования Silica

Корни модуля Silica связаны с операционной системой *Sailfish OS*, а именно с разработкой под мобильную платформу *Sailfish OS*.

В состав *SailfishOS SDK* входит *Sailfish Silica* – QML модуль, использующийся для создания *Sailfish* приложений. Данный модуль содержит QML компоненты, которые выглядят и управляются в соответствии со стандартами приложений для *Sailfish*. Помимо прочего, *Sailfish Silica* так же содержит инструменты для создания специфических элементов *Sailfish* приложений. Для того, чтобы воспользоваться модулем *Sailfish Silica* и его инструментами и компонентами, необходимо импортировать данный модуль в QML файлы кода приложения.

Но вернёмся к ОС АВРОРА. В состав Аврора SDK входит **Silica** – QML-модуль, используемый для создания приложений для ОС Аврора.

Для написания приложений для платформы ОС АВРОРА используется комбинация из **QML** и **C++**.

QML - декларативный язык программирования на основе **Qt**, использование которого заметно упрощает создание пользовательских интерфейсов с плавными переходами и анимациями. Пользовательские интерфейсы на основе **QML** могут быть связаны с более сложными функциональными возможностями приложения, реализованными на языке **C++** и обращающимися к сторонним библиотекам **C++.**

Несмотря на то, что фреймворк **Qt** включает в себя модуль **QtQuick**, содержащий основные типы для создания пользовательских интерфейсов на основе **QML**, модуль **Silica** предоставляет дополнительные типы, предназначенные для создания приложений с внешним видом, поведением и уникальными возможностями, соответствующими стилю стандартных приложений ОС Аврора. При создании приложений для ОС Аврора в QML-файлы необходимо импортировать оба модуля **Silica** и **QtQuick**.

Модуль **Silica** предоставляет основные компоненты для построения пользовательского интерфейса приложений ОС Аврора.

Сюда входят:

-компоненты пользовательского интерфейса, основанные на **QtQuick**;

-средства для стилизации приложений ОС Аврора и управления их поведением.

Модуль **Silica** не только облегчает построение интерфейсов приложений, но и обеспечивает единство стиля оформления с другими приложениями ОС Аврора.

В QML-файле описывается компоновка пользовательского интерфейса приложения.

Рассмотрим возможности модуля **Silica**.

Модуль **Silica** позволяет создавать пользовательские интерфейсы, которые:

●имеют оформление в стиле стандартных приложений ОС Аврора;

●ведут себя аналогично стандартным приложениям ОС Аврора (например, списки должны плавно затухать при прокрутке за пределы их границ);

●используют уникальные для ОС Аврора возможности приложений, такие как раскрывающиеся меню и обложки приложений.

СЛАЙД 26

Сформулируем некоторые теоретические основы, связанные с модулем Silica, которые станут полезными при разработке мобильных приложений.

QML-типы Silica

Основные типы

Основные QML-типы, относящиеся к ОСНОВНЫЕ ТИПЫ, представлены в табл. 1.1.

Каждое приложение ОС Аврора начинается с **ApplicationWindow** и имеет один объект **PageStack (**обеспечивает стековую модель навигации), в котором содержатся все его страницы.

Специальный объект **Them** позволяет оформить пользовательский интерфейс приложения в соответствии со стилем ОС Аврора.

Тип ApplicationWindow – компонент верхнего уровня приложения Ос АВРОРА – используется для создания элемента верхнего уровня в приложении ОС Аврора.

Каждое приложение ОС Аврора должно иметь один компонент ApplicationWindow определённый в корне его иерархии. Окно приложения - это точка входа для загрузки приложения.

Тип ApplicationWindow позволяет приложению:

- управлять стеком страниц;
- устанавливать активную обложку;
- устанавливать фоновый рисунок;
- обрабатывать изменения ориентации для обновления пользовательского интерфейса.

Самое простое окно приложения состоит из одной initialPage - страницы, которая отображается при открытии приложения (рис. 1.1)

```
import Sailfish.Silica 1.0

ApplicationWindow {
    initialPage: Component {
        Page {
            Text { text: "Привет!" }
        }
    }
}
```

Рисунок 1.1 – Окно приложения, состоящего из одной страницы

Таблица 1.1 - Основные QML-типы Silica

Clipboard	Обеспечивает базовые функции буфера обмена
StandardPaths	Предоставляет расположение стандартных системных каталогов для хранения пользовательской информации
ApplicationWindow	Компонент верхнего уровня приложения ОС Аврора
PageStack	Хранит и управляет страницами приложения
SafeZoneRect	Объект содержит информацию об отступах
SafeZoneRectInsets	Объект содержит внешние поля определённого отступа
Screen	Предоставляет параметры экрана устройства
Theme	Определяет свойства, позволяющие оформить пользовательский интерфейс приложения в стиле ОС Аврора

Страницы и диалоговые окна

Каждая страница или диалоговое окно содержит один экран с содержимым приложения. Диалоговое окно - это тип страницы, которая представляет содержимое при запросе подтверждения или отмены действия от пользователя.

Основные QML-типы, относящиеся к СТРАНИЦЫ И ДИАЛОГОВЫЕ ОКНА, представлены в табл. 1.2.

Таблица 1.2 – Основные QML-типы, относящиеся к СТРАНИЦЫ И ДИАЛОГОВЫЕ ОКНА

ColorPickerDialog	Диалог для выбора цвета из ColorPicker
ColorPickerPage	Страница для выбора цвета из ColorPicker
DatePickerDialog	Диалог для выбора даты из DatePicker
Dialog	Страница, закрываемая действием подтверждения или отмены
DialogHeader	Заголовок для использования в типе Dialog
Page	Предоставляет контейнер для содержимого одной страницы внутри приложения
PageHeader	Обеспечивает оформленный в стиле ОС Аврора заголовок страницы
TimePickerDialog	Предоставляет диалог для выбора времени с помощью TimePicker

СЛАЙД 27

Представления и контейнеры

Данные компоненты (QML-типы) используются для отображения или служат контейнером для других элементов.

Рассмотрим подробнее некоторые из основных QML-типов, относящихся к ПРЕДСТАВЛЕНИЯМ И КОНТЕЙНЕРАМ.

-**SilicaFlickable** – данный тип реализует Flickable с характерными для ОС Аврора поведением и дополнительными свойствами;

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`Flickable;`**

Наследники: **`DockedPanel`**.

Ниже приведён пример представления с простой прокруткой в стиле ОС Аврора:

```
import QtQuick 2.2
import Sailfish.Silica 1.0

Rectangle {
    width: 200; height: 100

    SilicaFlickable {
        anchors.fill: parent
        contentWidth: text.width; contentHeight: text.height

        Text {
            id: text
            text: "Hello, Sailor!"
            font.pixelSize: 100
        }
    }
}
```

-**SilicaListView** – данный тип реализует тип ListView с характерными для ОС Аврора поведением и дополнительными свойствами;

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`ListView;`**

Ниже приведён пример представления простого списка в стиле ОС Аврора:

```
import QtQuick 2.2
import Sailfish.Silica 1.0

SilicaListView {
    width: 480; height: 800
    model: ListModel {
        ListElement { fruit: "jackfruit" }
        ListElement { fruit: "orange" }
        ListElement { fruit: "lemon" }
        ListElement { fruit: "lychee" }
        ListElement { fruit: "apricots" }
    }
    delegate: Item {
        width: ListView.view.width
        height: Theme.itemSizeSmall
        Label { text: fruit }
    }
```

```
}
```

-**SilicaGridView** – данный тип реализует GridView с характерными для ОС Аврора поведением и дополнительными свойствами;

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`GridView;`**

Ниже приведён пример простой сетки в стиле ОС Аврора:

```
import QtQuick 2.2
import Sailfish.Silica 1.0

SilicaGridView {
    width: 480; height: 800
    model: ListModel {
        ListElement { fruit: "jackfruit" }
        ListElement { fruit: "orange" }
        ListElement { fruit: "lemon" }
        ListElement { fruit: "lychee" }
        ListElement { fruit: "apricots" }
    }
    delegate: Item {
        width: GridView.view.width
        height: Theme.itemSizeSmall

        Label { text: fruit }
    }
}
```

-**SilicaWebView** – данный тип реализует WebView с характерными для ОС Аврора поведением и дополнительными свойствами;

Строка импорта: **`import Sailfish.Silica 1.0;`**

Ниже приведён пример представления веб-страницы стиле ОС Аврора:

```
import QtQuick 2.2
import Sailfish.Silica 1.0

Page {
    SilicaWebView {
        id: webView

        anchors {
            top: parent.top
            left: parent.left
            right: parent.right
            bottom: urlField.top
        }
        url: "http://sailfishos.org"
    }

    TextField {
        id: urlField
        anchors {
            left: parent.left
```

```
            right: parent.right
            bottom: parent.bottom
        }
        inputMethodHints: Qt.ImhUrlCharactersOnly
        text: "http://sailfishos.org"
        label: webView.title
        EnterKey.onClicked: {
            webView.url = text
            parent.focus = true
        }
    }
}
```

Следует обратить внимание, что типы – SilicaFlickable, SilicaListView, SilicaGridView, SilicaWebView – обеспечивают поведение, характерное для ОС АВРОРА, поэтому им следует отдавать предпочтение по сравнению с аналогичными типами из модуля QtQuick: Flickable, ListView и GridView.

-**ScrollDecorator** предоставляют индикаторы прокрутки для просмотров.

Тип ScrollDecorator добавляет горизонтальный и вертикальный индикаторы прокрутки к краям соответствующим QML-типов: SilicaFlickable, SilicaListView, SilicaGridView, SilicaWebView.

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`item;`**

Для добавления индикаторов прокрутки достаточно объявить ScrollDecorator в качестве дочернего элемента Flickable, но для достижения максимальной производительности рекомендуется явно указать представление с помощью свойства flickable:

```
SilicaFlickable {
    id: flick
    contentWidth: width * 2
    contentHeight: height * 2

    ScrollDecorator { flickable: flick }
}
```

Flickable *– обеспечивает поверхность, которую можно «щелкнуть».*

Элемент Flickable размещает свои дочерние элементы на поверхности, которую можно перетаскивать и щелкать, вызывая прокрутку представления дочерних элементов. Такое поведение лежит в основе элементов, предназначенных для отображения большого количества дочерних элементов, таких как ListView и GridView.

В традиционных пользовательских интерфейсах представления можно прокручивать с помощью стандартных элементов управления, таких как полосы прокрутки и кнопки со стрелками. В некоторых ситуациях также можно перетащить вид напрямую, нажав и удерживая кнопку мыши при перемещении курсора. В сенсорных пользовательских интерфейсах это действие перетаскивания часто дополняется действием пролистывания, при котором прокрутка продолжается после того, как пользователь перестает касаться представления.

Flickable не обрезает содержимое автоматически. Если он не используется в качестве полноэкранного элемента, вам следует установить для свойства clip значение true.

ListView *– предоставляет список элементов, предоставленных моделью.*

Отображает данные из моделей, созданных на основе встроенных элементов QML, таких как ListModel и XmlListModel, или пользовательских классов моделей, определенных в C++, которые наследуются от QAbstractListModel.

ListView имеет модель, которая определяет отображаемые данные, и делегат, который определяет, как данные должны отображаться. Элементы в ListView располагаются горизонтально или вертикально. Представления списков по своей сути являются перелистываемыми, поскольку ListView наследует от Flickable.

GridView *– отображает данные из моделей, созданных на основе встроенных элементов QML, таких как ListModel и XmlListModel, или пользовательских классов моделей, определенных в C++, которые наследуются от QAbstractListModel.*

GridView имеет модель, которая определяет отображаемые данные, и делегат, который определяет, как данные должны отображаться. Элементы в GridView располагаются горизонтально или вертикально. Представления сетки по своей сути являются перелистываемыми, поскольку GridView наследует от Flickable.

item *– базовый визуальный тип QML. Является базовым типом для всех визуальных элементов в Q tQuick.*

Все визуальные элементы в Qt Quick наследуются от Item. Хотя объект Item не имеет визуального внешнего вида, он определяет все атрибуты, общие для визуальных элементов, такие как положение x и y, ширина и высота, привязка и поддержка обработки клавиш.

Основные QML-типы, относящиеся к ПРЕДСТАВЛЕНИЯМ И КОНТЕЙНЕРАМ, представлены в таблице 1.2.

Таблица 1.2 – Основные QML-типы, относящиеся к ПРЕДСТАВЛЕНИЯМ И КОНТЕЙНЕРАМ

ColumnView	Создаёт экземпляры делегата в столбце по мере необходимости
DockedPanel	Панель, которая пристыкована к краю элемента и расширяется от этого края
Drawer	Предоставляет контейнер, с помощью которого реализуется выдвигающаяся панель с ещё одной панелью снизу (под первой)
HorizontalScrollDecorator	Добавляет горизонтальный индикатор прокрутки к представлениям Silica
ScrollDecorator	Добавляет горизонтальный или вертикальный индикаторы прокрутки к представлениям Silica
SilicaFlickable	Представление Flickable в стиле ОС Аврора
SilicaGridView	Представление GridView в стиле ОС Аврора
SilicaListView	Представление ListView в стиле ОС Аврора
SilicaWebView	Представление WebView для отображения веб-страниц в стиле ОС Аврора
SlideshowView	Представление для пролистывания ряда элементов
VerticalScrollDecorator	Добавляет вертикальный индикатор прокрутки к представлениям Silica
ViewPlaceholder	Текстовая метка, отображаемая в центре элемента, которая появляется в случае недоступности содержимого страницы

СЛАЙД 28

Управление

Компоненты управления позволяют пользователям запускать действия, изменять значения и выбирать параметры.

Основные QML-типы, относящиеся к УПРАВЛЕНИЮ, представлены в таблице 1.3.

Таблица 1.3 – Основные QML-типы, относящиеся к УПРАВЛЕНИЮ

ButtonLayout	Элемент для расположения кнопок по правилам ОС Аврора
BackgroundItem	Базовый элемент в стиле ОС Аврора, который отображает нажатие на него при помощи подсветки фона
BusyIndicator	Неинтерактивный элемент, который отображается в виде вращающегося круга во время ожидания загрузки контента или завершения какого-либо процесса
BusyLabel	Неинтерактивный элемент, который отображается в виде вращающегося круга во время ожидания загрузки контента или завершения какого-либо процесса
Button	Кнопка с текстовой меткой
ColorPicker	Палитра для выбора цвета
ComboBox	Элемент графического интерфейса для выбора варианта из выпадающего списка
DatePicker	Календарь для выбора даты
GridItem	Простой элемент в стиле ОС Аврора для создания интерактивных элементов Grid
HighlightImage	Изображение с эффектом подсветки
IconButton	Кнопка с изображением
IconTextSwitch	Кнопка-переключатель с текстовой меткой и значком
Keypad	Клавиатура для набора номера
ListItem	Базовый элемент списка в стиле ОС Аврора, при нажатии на который появляется контекстное меню
PageBusyIndicator	Отображается при ожидании загрузки страницы
PagedView	Страничный просмотр элемента
PasswordField	Текстовое поле для ввода пароля
ProgressBar	Горизонтальный индикатор выполнения
Remorse	Ненадолго появляющиеся элементы интерфейса (области), которые позволяет отменить разрушающее действие (удаление)
RemorseItem	Ненадолго появляющийся элемент, который позволяет отменить разрушающее действие (удаление)
RemorsePopup	Ненадолго показывает всплывающий элемент, нажатием на который можно отменить разрушающее действие (удаление)
Separator	Горизонтальный разделитель
Slider	Горизонтальный ползунок
Switch	Кнопка-переключатель со значком
TextSwitch	Кнопка-переключатель с текстовой меткой

TimePicker	Циферблат часов для выбора времени
TouchBlocker	Элемент, который принимает все нажатия на себя
ValueButton	Интерактивный элемент управления, который отображает метку и значение

СЛАЙД 29

Отображение и ввод текста

Текстовые компоненты могут отображать текст и разрешать ввод текста.

Основные QML-типы, относящиеся к ОТОБРАЖЕНИЮ И ВВОДУ ТЕКСТА, представлены в таблице 1.4.

Таблица 1.4 – Основные QML-типы, относящиеся к ОТОБРАЖЕНИЮ И ВВОДУ ТЕКСТА

SilicaControl	Интерактивный визуальный элемент
DetailItem	Отображение метки и связанного с ней значения в стиле ОС Аврора
EnterKey	Контролирует внешний вид и поведение клавиши ввода в виртуальной клавиатуре
Icon	Монохромная иконка
SilicaItem	Визуальный элемент
Label	Текстовая метка
LinkedLabel	Текстовый абзац, в котором автоматически формируются ссылки для номеров и адресов
Palette	Цветовая палитра
PasswordField	Текстовое поле для ввода пароля
SearchField	Текстовое поле для ввода поискового запроса
SectionHeader	Заголовок текста для начала раздела на странице
TextArea	Отображает несколько строк для редактирования простого текста
TextField	Отображает одну строку для редактирования простого текста

СЛАЙД 30

Меню

Меню позволяют пользователям выбирать и запускать действие из списка опций.

Рассмотрим основные QML-типы, относящиеся к МЕНЮ.

ContexMenu – QML-тип, предоставляющий контекстное меню.

Контекстное меню – это меню, которое отображается под указанным элементом посредством вызова метода show(). Пункты меню задаются в дочерних элементах типа MenyItem.

Контекстные меню чаще всего используются для отображения пунктов меню для отдельных элементов списка.

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`item;`**

MenyItem – QML-тип, инкапсулирующий пункт меню.

Тип MenuItem реализует пункт меню для использования в меню, раскрывающихся сверху (тип PullDownMenu), снизу (тип PushUpMenu), а также в контекстных меню (тип ContextMenu). Тип MenuItem предоставляет текстовую метку и обработчик активации onClicked.

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`Label;`**

```
PushUpMenu {
    MenuItem {
        text: "Опция 1"
        onClicked: console.log("Нажата опция 1")
    }
    MenuItem {
        text: "Опция 2"
        onClicked: console.log("Нажата опция 2")
    }
}
```

MenyLabel – QML-тип, реализующий неинтерактивную текстовую метку, отображаемую в меню, раскрывающихся сверху (PullDownMeny) и снизу (PuchUpMeny).

Представляет собой статическую текстую метку, которая отображается в начале меню.

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`item;`**

PullDownMeny – QML-тип, добавляющий вытягиваемое СВЕРХУ меню в представлениях Silica.

Тип PullDownMenu предоставляет глобальные действия для представлений SilicaFlickable, SilicaListView, SilicaGridView, SilicaWebView. Раскрывающееся меню располагается над содержимым экрана. Чтобы открыть раскрывающееся меню, следует коснуться экрана ниже полосы и, не отрываясь, провести по экрану вниз. Чтобы активировать пункт такого меню, необходимо выполнить одно из следующих действий:

- не отрывая пальца от экрана, провести его вниз, остановить выбор на нужном пункте, отпустить палец от экрана;
- проведя пальцем по экрану сверху вниз, раскрыть меню целиком, затем коснуться нужного пункта.

Вытягиваемое сверху меню заполняется путём создания объектов типа MenyItem в качестве дочерних элементов PullDownMenu. Сюда же в качестве дочерних элементов можно включить объекты типа MenyLabel (неинтерактивные метки).

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`item;`**

PuchUpMeny – QML-тип, добавляющий вытягиваемое СНИЗУ меню в представлениях Silica.

Тип PullDownMenu предоставляет глобальные действия для представлений SilicaFlickable, SilicaListView, SilicaGridView, SilicaWebView. Раскрывающееся меню располагается над содержимым экрана. Чтобы открыть раскрывающееся меню, следует коснуться экрана ниже полосы и, не отрываясь, провести по экрану вниз. Чтобы активировать пункт такого меню, необходимо выполнить одно из следующих действий:

- не отрывая пальца от экрана, провести его вниз, остановить выбор на нужном пункте, отпустить палец от экрана;
- проведя пальцем по экрану сверху вниз, раскрыть меню целиком, затем коснуться нужного пункта.

Вытягиваемое сверху меню заполняется путём создания объектов типа MenyItem в качестве дочерних элементов PullDownMenu. Сюда же в качестве дочерних элементов можно включить объекты типа MenyLabel (неинтерактивные метки).

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`item;`**

СЛАЙД 31

Анимация и эффекты

Базовые анимации Silica, которые используются для плавного появления/исчезновения элементов или изменения содержимого списков, сетки и других представлений. Кроме того, эффект нарастания непрозрачности обеспечивает простой способ линейного затемнения элемента.

Рассмотрим основные QML-типы, относящиеся к АНИМАЦИЯ И ЭФФЕКТЫ.

AddAnimation – QML-тип, добавляющий элемент в стиле ОС АВРОРА.

Тип AddAnimation реализует стандартную анимацию при добавлении элементов в контейнеры типов SilicaListViewи SilicaGridView.

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`NumberAnimation;`**

FadeAnimation – QML-тип, позволяющий реализовывать анимацию появления/исчезновения в стиле ОС АВРОРА.

Тип FadeAnimation реализует стандартную анимацию затухания и появления элементов. Он часто используется внутри контейнера Behavior (**`import QtQuick 2.5`**), применённого к свойству opacity (свойство содержит непрозрачность элемента) для автоматической анимации изменения непрозрачности элемента стиле приложений ОС АВРОРА.

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`NumberAnimation;`**

FadeAnimator – QML-тип, позволяющий реализовывать анимацию появления/исчезновения в стиле ОС АВРОРА, которая выполняется вне основного потока пользовательского интерфейса приложения.

Тип FadeAnimation реализует стандартную анимацию затухания и появления элементов. Он часто используется внутри контейнера Behavior (**`import QtQuick 2.5`**),

применённого к свойству opacity (свойство содержит непрозрачность элемента) для автоматической анимации изменения непрозрачности элемента стиле приложений ОС АВРОРА.

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`OpasityAnimator;`**

OpasityRampEffect – QML-тип, представляющий собой шейдер, который применяет эффект градиента непрозрачности.

Эффект OpacityRampEffect плавно уменьшает непрозрачность элемента в указанном направлении.

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`ShaderEffect;`**

RemoveAnimation – QML-тип, позволяющий реализовывать анимацию для удаления элемента в стиле ОС АВРОРА.

Тип RemoveAnimation реализует стандартную анимацию при удалении элементов из контейнеров типов SilicaListViewи SilicaGridView.

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`SequentialAnimation;`**

СЛАЙД 32

Обложки приложения

Обложки – это визуальные представления приложений, которые работают в фоне (свёрнуты), которые отображаются на экране запущенных приложений.

Рассмотрим основные QML-типы, относящиеся к ОБЛОЖКИ АНИМАЦИИ

CoverAction – QML-тип, определяющий действие, которое будет выполнено при активации жеста Cover.

С помощью свойства *iconSource* определяется значок элемента управления, при нажатии на который вызывается обработчик сигнала onTriggered().

Элементы CoverAction определяются внутри объекта CoverActionList и позволяют пользователю управлять приложением непосредственно с обложки.

Строка импорта: **`import Sailfish.Silica 1.0;`**

CoverActionList – QML-тип, определяющий список действий для элемента Cover.

Внутри объекта CoverActionList определяются элементы CoverAction, которые позволяют пользователю управлять приложением непосредственно с обложки. При этом на обложке приложения отображаются значки, при нажатии на которые выполняются определенные действия

Строка импорта: **`import Sailfish.Silica 1.0;`**

Cover – QML-тип, реализующий активную обложку – визуальное представление приложения, которое отображается на домашнем экране ОС Аврора, когда приложение работает в фоновом режиме.

Обложки используются для отображения статуса или другой важной информации, которая передает назначение приложения. Например, приложение «ПОЧТА» может

показывать количество непрочитанных писем, а приложение «ГАЛЕРЕЯ» может показывать миниатюрные изображения случайных фотографий из коллекции пользователя. Обложка может поддерживать несколько действий, чтобы дать пользователю выполнять необходимые действия с приложением, когда оно работает в фоновом режиме.

Установить обложку можно с помощью свойства cover компонента ApplicationWindow[22].

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`item;`**

Наследники: **`CoverBackground`**.

CoverBackground – QML-тип, предоставляющий обложку с прозрачным фоном.

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`Cover;`**

CoverPlaceholder – QML-тип, предоставляющий шаблон для самой простой обложки приложения.

CoverPlaceholder предоставляет содержащий текстовую метку и изображение шаблон для самой простой обложки приложения. Обычно данный тип применяется, когда более информативное содержимое недоступно, или достаточно короткого текстового сообщения для передачи состояния приложения.

Наиболее часто используемая иконка на обложке – это иконка приложения, которая упрощает распознавание обложки.

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`item;`**

CoverTemplate – QML-тип, предоставляющий шаблон для обложки приложения.

CoverTemplate предоставляет шаблон для отображения основной информации о приложении, такой как статус, количество непрочитанных сообщений или пропущенных вызовов, а также некоторой дополнительной информации, например, времени последнего обновления.

Компонент CoverTemplate поддерживает как книжную, так и альбомную ориентацию.

Строка импорта: **`import Sailfish.Silica 1.0;`**

Наследуется от: **`item;`**

[22] Компонент верхнего уровня приложения ОС АВРОРА. Относится к QML-типу –ОСНОВНЫЕ ТИПЫ

СЛАЙД 33

ВТОРОЙ УЧЕБНЫЙ ВОПРОС.
ОБЗОР ОСНОВНЫХ ИНСТРУМЕНТОВ РАЗРАБОТКИ ПРИЛОЖЕНИЙ В ОС АВРОРА
2.1. Обзор SDK

Аббревиатура SDK расшифровывается как software development kit. SDK – это набор средств для разработки ПО под определенную платформу. Он содержит компоновочные блоки, средства отладки, а зачастую фреймворк или группу библиотек кода, например, набор подпрограмм для определенной операционной системы.

Что такое Аврора SDK?

Аврора SDK – это набор инструментов для создания, сборки, запуска и отладки приложений.

В состав Аврора SDK входят:

● Aurora IDE (IDE) – интегрированная среда разработки, основанная на Qt Creator, для разработки приложений на языках C, C++ и QML для ОС Аврора с использованием компонентов Silica. IDE предоставляет продвинутый редактор кода с интеграцией системы контроля версий, управления проектами и сборками.

● Aurora OS Emulator (эмулятор) – виртуальная машина, которая позволяет выполнять приложения в окружении ОС Аврора аналогично работе на МУ.

● Aurora OS Build Engine (среда сборки) – окружение, поставляемое как виртуальная машина или Docker-контейнер, которое обеспечивает среду для сборки приложений, не зависящую от домашней операционной системы (ОС).

● Документация по архитектуре и API.

● Примеры и шаблоны приложений.

СЛАЙД 34

На компьютере разработчика должно быть установлено следующее программное обеспечение:

●рекомендуется одна из следующих операционных систем:

-Ubuntu LTS не ниже версии 20.04;

-Альт Рабочая станция 10;

-Windows 10 (далее – Windows);

● Oracle VM VirtualBox версии не ниже 6.x (далее – VirtualBox);

● Git (только для Windows);

● Docker (только для Windows и Linux и только в online-версии установщика):

-рекомендуется использовать Docker Desktop в версии для Windows только в учётной записи с правами администратора;

-после установки Docker Desktop необходимо добавить текущего пользователя в группу «docker-users».

Если Аврора SDK устанавливается повторно, предыдущие установленные версии SDK необходимо удалить.

СЛАЙД 35

Варианты исполнения Аврора SDK

ОС Аврора существовала в двух исполнениях:

●корпоративная версия используется для коммерческих проектов, не требующих сертификации;

●сертифицированная версия используется для коммерческих проектов, требующих аттестацию по требованиям регуляторов, или в которых ведется обработка данных, подлежащих защите согласно законодательству РФ.

Версию Аврора SDK следует выбирать в соответствии с номером версии и исполнением ОС Аврора. Номер версии SDK должен совпадать с номером версии целевой ОС с точностью до первых двух чисел.

СЛАЙД 36

Аврора SDK 4.x.x

Начиная с версии 4.0.1 для корпоративного и сертифицированного исполнения используется единый Аврора SDK.

В Аврора SDK присутствуют:

●валидация приложений;

●подписание RPM-пакетов;

●подписание бинарных файлов (IMA);

●подписание модулей ядра со стороны ОС, что исключает возможность загрузки недоверенных модулей;

●контроль целостности системы;

●шифрование домашнего каталога;

●веб-браузер с поддержкой шифрования согласно ГОСТ-алгоритмам;

●сервис криптоконтейнера вместе с API;

●многопользовательский режим.

СЛАЙД 37

Новые функции в ОС Аврора 4

Улучшения затронули все составляющие мобильной инфраструктуры Аврора:

-средства обеспечения безопасности;

-пользовательский интерфейс;

-встроенные приложения;

-инструменты администрирования парка устройств;

-средства для разработки приложений.

СЛАЙД 38

ОСНОВНЫЕ ФУНКЦИИ И УЛУЧШЕНИЯ

Унификация корпоративной и сертифицированной версий

●Унифицированные механизмы безопасности в обеих версиях;

Механизмы безопасности, такие как подпись пакетов, валидация приложений, подпись модулей ядра, контроль целостности, шифрования пользовательских данных, изоляция приложений и другие, теперь доступны в любой версии Авроры.

●Переносимые приложения

Унифицированный (идентичный) исходный код приложений для корпоративной и сертифицированной версий.

●Единый SDK для обеих версий

Компиляция приложений под разные требования в едином инструменте.

СЛАЙД 39

Подпись пакетов для защиты программной среды от недоверенного и вредоносного ПО

●Все приложения подписываются;

Приложения подписываются для любой целевой версии Авроры – корпоративной и сертифицированной.

●Единый ключ разработчика;

Упрощение инфраструктуры подписи пакетов – единый ключ для подписи бинарных файлов и пакетов.

●Настройка разрешенного к установке ПО;

Спецификация разрешенных к установке приложений на основе избранных сертификатов разработчиков.

СЛАЙД 40

Многопользовательский режим

●Поддержка до 6 пользователей и администратора

Возможность реализации посменного режима работы на одном устройстве. Пользовательские данные полностью независимы.

СЛАЙД 41

Модульная система первоначальной загрузки

●Кастомизация первого старта ОС под проект;

Возможность изменения процедуры начальной настройки без внесения изменений в ОС. Быстрый ввод в эксплуатацию.

СЛАЙД 42

Улучшение пользовательского опыта

●Новая клавиатура с предиктивным вводом;

Высокая скорость набора текста. Меньше ошибок при вводе. Новый стиль, новая система подсказок, новые возможности кастомизации. Предиктивный ввод и автокоррекция.

●Новый шрифт;

Улучшенная читаемость текстов.

●Новые элементы интерфейса;

Новые кнопки, закладки и уведомления. Поддержка новых жестов и действий на экране уведомлений.

●Переработанный механизм Атмосфер;

Поддержка аппаратного ускорения обработки графики. Автоматическая адаптация стандартных интерфейсов приложений под палитру фонового изображения. Поддержка сложных эффектов сглаживания и размытия.

●Новые иконки и мелодии;

Обновлённый дизайн ОС.

●Новые возможности для разработчиков приложений

Добавлены новые функции и элементы создания пользовательского интерфейса.

СЛАЙД 43

Обновленные стандартные приложения

●Офис на движке LibreOffice;

Высокая скорость загрузки и вывода на экран документов распространённых офисных форматов. Поддержка документов со сложным форматированием.

●Обновленные приложения ФАЙЛЫ, ЗАМЕТКИ, КАЛЕНДАРЬ и др.;

Улучшенный пользовательский опыт, новые возможности.

●Поддержка сертификатов в почтовом клиенте;

Установление защищенного подключения и шифрация почтового трафика.

●Браузер на современном веб-движке Gecko 60;

Поддержка новой версии ГОСТ TLS, интеграция технологии WebRTC и аудио-видео- кодеков для коммуникаций в реальном времени (например, видеоконференцсвязи), интеграция технологии WebGL, улучшенная поддержка технологий HTML 5 и JavaScript.

СЛАЙД 44

Новые и усовершенствованные API

●NFC API;

Работа с NFC-картами и токенами. Поддержка интерфейсов pscs-lite и pksc#11.

●WebView API ;

Создание приложений, отображающих веб-контент.

●Antivirus API;

Для разработки антивирусных приложений на базе стандартных механизмов ОС.

●VPN API;

Для разработки VPN приложений на базе стандартных механизмов ОС;

●API управления жизненным циклом приложений;

Поддержка интеграции с магазином Аврора Маркет и сторонними EMM.

●API для работы с QR-кодами;

Поддержка баркодов, QR-кодов, штрихкодов штрих-кодов и т.д. Генерация QR-кодов.

●Crypto API;

Для разработки криптопровайдеров на базе QCA на базе стандартных механизмов ОС.

● Keystore API;

Для хранения чувствительной информации в защищенном хранилище.

●MDM API;

Новые политики удаленного управления парком устройств.

СЛАЙД 45

Безопасность

●Изоляция приложений

Запуск приложений в «песочницах», защита информации от сторонних процессов.

●Гранулярный доступ приложений к ресурсам

Приложения обязаны запрашивать доступ ко всем используемым ресурсам (камера, микрофон и т. д.).

●Шифрование пользовательского раздела

Защита пользовательских данных при утере устройства.

●Обновление загрузчиков (на ряде устройств)

Возможность обновления не только приложений и операционной системы, но и загрузчиков устройств.

●СледопытSSL

Обновление встроенного средства криптозащиты.

●Динамический контроль целостности программной среды

Контроль целостности программной среды осуществляется не только при загрузке ОС, но и в процессе работы.

СЛАЙД 46

Другие важные нововведения

●Новый компилятор gcc 8

С поддержкой стандарта C++ 17-й версии.

●Поддержка релизов с длительным сроком поддержки

Аврора TEE – интегрированная среда для исполнения кода в безопасном (доверенном) режиме.

Аврора СДЗ – программно-аппаратное средство доверенной загрузки с корнем доверия в кристалле.

СЛАЙД 47

2.2. Qt Creator

Qt Creator (ранее известная под кодовым названием Greenhouse) – свободная IDE для разработки на C, C++, JavaScript и QML.

Разработана Trolltech (Digia) для работы с фреймворком Qt.

Включает в себя графический интерфейс отладчика и визуальные средства разработки интерфейса как с использованием QtWidgets, так и QML.

Поддерживаемые компиляторы: GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW.

Целью Qt Creator является предоставление кроссплатформенного, полностью интегрированной среды разработки (IDE) для написания проектов на Qt. Он доступен на платформах Linux, Mac OS X и Windows.

Особенности

Qt Creator включает большое число полезных особенностей. Среди них:

● Умный редактор кода: Текстовый редактор предоставляет поддержку синтаксиса и дополнение кода.

●Мастер генерации проектов Qt4: Этот мастер позволяет пользователю генерировать проект для консольного приложения, GUI приложения или библиотеки C++.

●Интеграция справки по Qt: Можно легко получить доступ ко всей документации Qt щёлкнув на кнопку Справка.

●Интеграция с Qt Designer: Формы интерфейса пользователя могут быть спроектированы внутри Qt Creator. Просто щёлкните два раза на файле .ui в обозревателе проекта для запуска интеграции.

●Поисковик: Мощный инструмент перемещения, который позволяет пользователю найти файлы и классы с использованием минимума нажатий клавиш.

●Поддержка формата файла проекта .pro qmake: Файл проекта .pro используется в качестве файла описания проекта.

●Интерфейс отладки: Приложения можно отлаживать в Qt Creator с использованием графического интерфейса к GNU symbolic debugger (GDB) и the Microsoft Console Debugger (CDB).

СЛАЙД 48

Работа с проектами

Qt Creator поддерживает системы сборки qmake, cmake, autotools, с версии 2.7 qbs. Для проектов, созданных под другими системами, может использоваться в качестве редактора исходных кодов. Есть возможность редактирования этапов сборки проекта.

Также IDE нативно поддерживает системы контроля версии (Git), такие как Subversion, Mercurial, Git, CVS, Bazaar, Perforce. Начиная с версии 2.5, в поле комментария к правке поддерживается автодополнение.

Редактирование кода

В Qt Creator реализовано автодополнение, в том числе ключевых слов, введённых в стандарте C++11. Также есть возможность задания стиля выравнивания, отступов и постановки скобок.

Реализован ряд возможностей при работе с сигнатурами методов, а именно:

●автогенерация пустого тела метода после его обновления;

●возможность автоматически изменить сигнатуру метода в определении, если она была изменена в объявлении и наоборот;

●возможность автоматически поменять порядок следования аргументов.

При навигации по коду доступно переключение между определением и объявлением метода, переход к объявлению метода, переименование метода как в отдельном проекте, так и во всех открытых. Также есть возможность вызвать справку согласно текущему контексту.

СЛАЙД 49

Отладка кода

Среда разработки имеет графический интерфейс для следующих отладчиков: GDB, CDB и QML/JavaScript. В качестве отдельной опции реализовано отображение содержимого контейнеров, таких как QString, std::map и прочих.

Поддерживаются следующие режимы отладки:

●простой для отладки локально запущенных приложений, таких как GUI приложения на Qt;

●терминал для отладки локально запущенных процессов, которым требуется консоль, обычно это приложения без GUI;

●подключённый для отладки локальных процессов, запущенных вне Qt Creator;

●удалённый для отладки запущенных на другой машине процессов (используя gdbserver);

●ядро для отладки завершившихся аварийно процессов на Unix;

●Post-mortem для отладки завершившихся аварийно процессов на Windows;

●TRK для отладки процессов, запущенных на устройстве Symbian.

Точки остановки можно задать различными способами, а именно:

●останавливаться на заданной строчке заданного файла;

●останавливаться при вызове функции с определенным именем;

●останавливаться при обращении к данным по заданному адресу;

●останавливаться при поимке исключения;

●останавливаться при запуске или создании нового процесса;

●останавливаться при выполнении системного вызова;

●останавливаться при изменении в данных с адресами, заданными выражением.

СЛАЙД 50

2.3. Qt QML Live

Qt QML Live – инструмент, который позволяет «на лету» изменять графический интерфейс приложения при внесении правок в QML-файлы проекта без необходимости пересборки установочного пакета. Такой подход позволяет существенно ускорить процессы разработки интерфейса пользователя.

QML Live – это локальная и удаленная система перезагрузки Qt Quick в реальном времени. Это позволяет вам изменять исходный код пользовательского интерфейса QML и просматривать результат в режиме реального времени.

Часто при разработке пользовательского интерфейса (UI) необходимо отредактировать размещение каких-либо объектов или анимацию, чтобы обеспечить взаимодействие с пользователем, которое предполагает команда дизайнеров. Этот процесс проб и ошибок утомителен с классическим циклом «редактирование-сохранение-запуск-выход». QML Live делает этот цикл более эффективным: при каждом изменении файла сцена автоматически перезагружается.

QML Live имеет модульную структуру, чтобы соответствовать различным требованиям использования.

На ранней стадии проекта, как правило, можно использовать QML Live Bench, в котором есть все, что есть в типичном настольном приложении.

Позже в проекте вы можете протестировать свой код пользовательского интерфейса на устройстве. Для этого можно комбинировать QML Live Bench с QML Live Runtime.

Эта комбинация предоставляет средство визуализации QML по умолчанию для запуска на устройстве и небольшое удаленное приложение на рабочем столе для управления им.

СЛАЙД 51

QML Live Bench

QML Live Bench – это комплексный инструмент для перезагрузки QML в реальном времени, который позволяет выбирать рабочую область для наблюдения и предоставляет среду выполнения Prime QML Live для выбранного документа QML.

«Prime QML Live Runtime» – это особый тип среды выполнения, который следует за текущим выбранным .qml файлом.

Напротив, QML Live Bench позволяет просматривать несколько файлов одновременно.

СЛАЙД 52

Интеграция с Аврора IDE

Вы можете интегрировать QML Live в Аврора IDE в качестве внешнего инструмента (**рис. 2.1**).

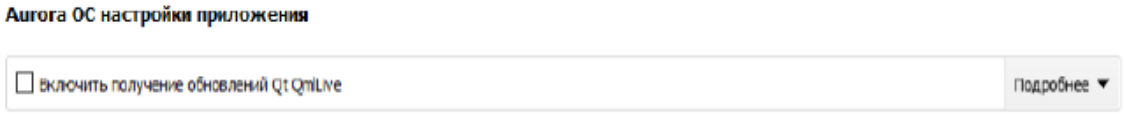

Рисунок 2.1 – Начало интеграции QML Live в Аврора IDE

Для этого необходимо:

ШАГ 1. В АВРОРА IDE открыть проект;

ШАГ 2. Выбрать вкладку проект (**рис. 2.2**):

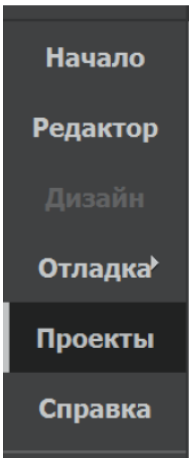

Рисунок 2.2 – Интеграция QML Live в Аврора IDE продолжается

ШАГ 3. Выбрать тип сборки и нажать «ЗАПУСК», как показано на **рис. 2.3**.

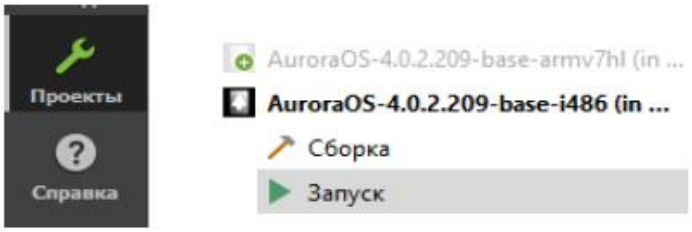

Рисунок 2.3 – Интеграция QML Live в Аврора IDE. Следующий шаг.

ШАГ 4. Найти раздел Aurora ОС настройки приложения (**рис. 2.4.**)

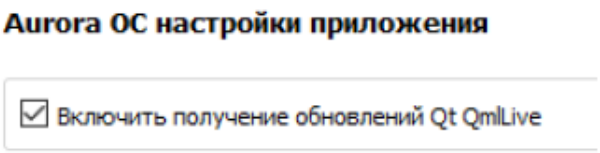

Рисунок 2.4 – Интеграция QML Live в Аврора IDE. Следующий шаг.

В результате будет предложено запустить QML Live Bench, как показано на **рис. 2.5**

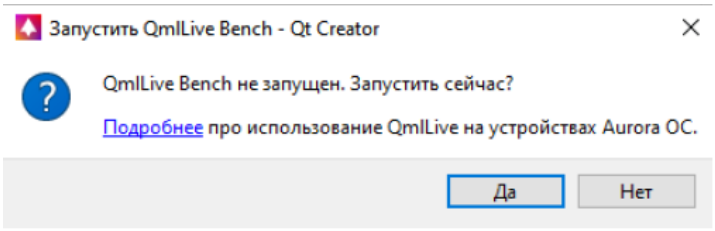

Рисунок 2.5 – Интеграция QML Live в Аврора IDE. Следующий шаг.

QML Live Bench выглядит так, как показано на **рис. 2.6**

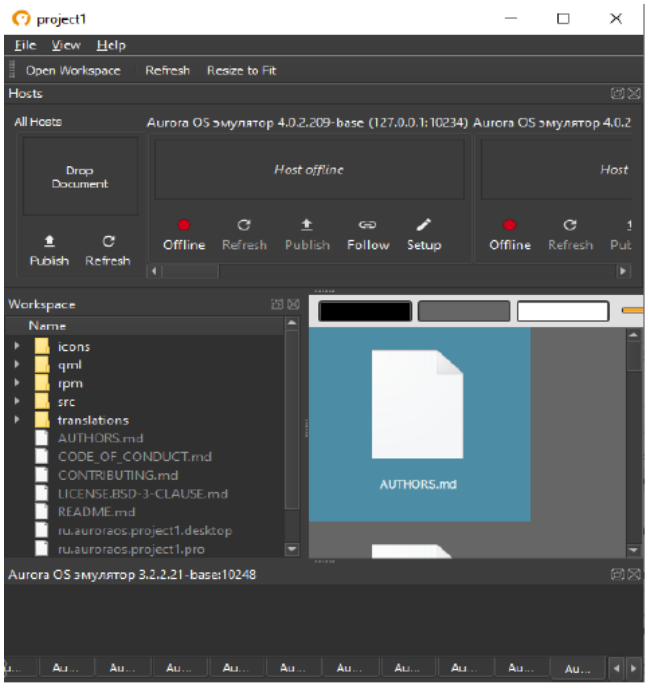

Рисунок 2.5 – Интеграция QML Live в Аврора IDE. Интегрировали

СЛАЙД 53

Среда выполнения QML в реальном времени

Инструмент QML Live Runtime предоставляет среду выполнения по умолчанию со средством просмотра QML по умолчанию и прослушивает заданный порт для вызовов IPC с удаленного компьютера. Этот инструмент идеален для разработки на целевом устройстве, когда не требуется дополнительный код C++.

Пользовательская среда выполнения

Вы можете создать свою собственную среду выполнения с функциями QML Live. Таким образом, вы можете использовать настройку представления QML с дополнительным кодом C++ вместе с системой QML Live.

Для этого вам нужно использовать класс LiveNodeEngine, чтобы иметь

возможность получать изменения рабочей области и активные обновления документов. По умолчанию IPC прослушивает порт 49156.

СЛАЙД 54

2.4. VirtualBox

VirtualBox - виртуальная машина Oracle.

Oracle VM VirtualBox, самое популярное в мире кроссплатформенное программное обеспечение для виртуализации с открытым исходным кодом, позволяет разработчикам быстрее доставлять код, запуская несколько операционных систем на одном устройстве. ИТ-отделы и поставщики решений используют VirtualBox для снижения операционных затрат и сокращения времени, необходимого для безопасного развертывания приложений локально и в облаке.

SDK Аврора использует виртуальную машину для визуализации эмулятора.

Возможности виртуальной машины Oracle VirtualBox:

●Упрощает операции

-Снижает затраты на ИТ

Кроссплатформенное программное обеспечение для виртуализации настольных компьютеров с открытым исходным кодом и низкими издержками снижает эксплуатационные расходы ИТ-специалистов за счет сокращения количества требуемых конфигураций настольных компьютеров и серверов.

-Работает на любом рабочем столе

ИТ-отделы могут упростить среду разработки, запустив одно и то же решение на любой операционной системе (ОС) хоста x86 и поддерживая широкий спектр версий ОС на виртуальных машинах (VM). Поддерживаемые операционные системы хоста включают Windows, Linux и macOS.

-Простота внедрения

Простой в использовании графический пользовательский интерфейс (GUI) и мощный интерфейс командной строки упрощают разработчикам работу с несколькими операционными системами в одной системе. Группы разработчиков могут консолидировать рабочие нагрузки с помощью VirtualBox для поддержки больших рабочих нагрузок до 32 виртуальных процессоров.

СЛАЙД 55

●Автоматизирует развертывание в облаке

-Простое и быстрое развертывание

Используя Vagrant boxes с VirtualBox, ИТ-отделы могут быстро обеспечить виртуальные машины для разработки предварительно настроенным программным обеспечением Oracle и автоматизировать выпуск в производство.

-Нажатие одной кнопки для перехода в Oracle Cloud

Встроенный графический интерфейс позволяет разработчикам легко импортировать и экспортировать виртуальные машины в стандартном формате OVF, локально или в облаке. Для Oracle Cloud Infrastructure один щелчок позволяет разработчикам загружать или загружать виртуальную машину.

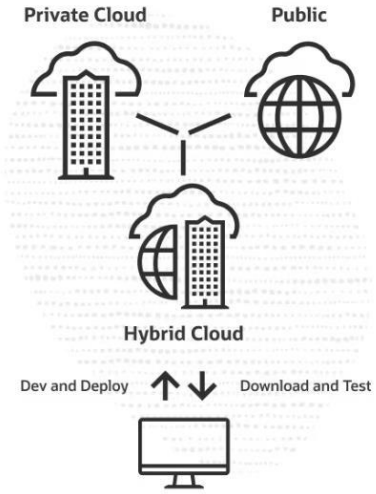

Рисунок 2.6 – Автоматизация развёртывания в облаке

СЛАЙД 56

●Быстрый контроль качества, тестирование и демонстрации

-Оптимизация тестирования

Группы обеспечения качества программного обеспечения могут упростить свою среду и сократить ресурсы, используя одно физическое

устройство для тестирования программного обеспечения на нескольких платформах и версиях ОС.

-Улучшение поддержки клиентов

Группы поддержки могут быстро решать проблемы, легко воссоздавая различные клиентские среды в одной системе.

-Запуск устаревших приложений на новом оборудовании

ИТ-отделы могут продлить срок службы устаревших приложений, используя VirtualBox для их запуска на современном оборудовании.

-Создание многоуровневых демонстрационных версий на одном устройстве

Отделы продаж могут легко продемонстрировать сложные многоуровневые решения потенциальным клиентам, используя готовые среды с несколькими виртуальными машинами и сетевыми топологиями на своих ноутбуках.

Рисунок 2.7 – Рисунок – как комментарий

СЛАЙД 57

●Позволяет удаленным сотрудникам безопасно получать доступ к приложениям

-Обеспечивает доступ к приложениям с ограниченным доступом

VirtualBox облегчает распространение ИТ-менеджерами образов критически важных приложений с ограниченным доступом для удаленных сотрудников, когда VPN-соединение считается недостаточным. Это помогает организациям повысить безопасность с помощью ограничений на основе ролей для наборов данных в этих приложениях.

-Обеспечивает безопасные и зашифрованные рабочие пространства

VirtualBox защищает удаленные подключения к приложениям с ограниченным доступом с помощью 256-разрядных ключей шифрования и гарантирует, что пользователи не смогут загружать или хранить данные на удаленных устройствах. ИТ-менеджеры могут избежать затрат и времени на перестройку ограниченных приложений, обеспечивая соответствие требованиям для удаленного доступа к приложениям.

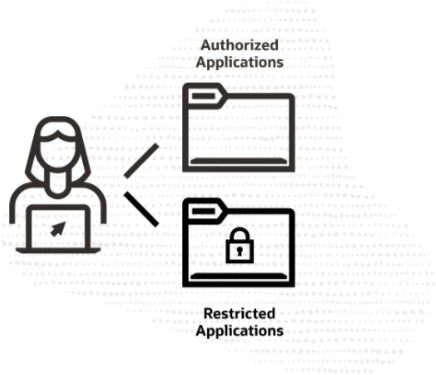

Рисунок 2.8 – Рисунок – как комментарий

СЛАЙД 58

●Лицензирование и поддержка

-Базовый пакет

Базовый пакет, лицензированный по GPL v2, позволяет разработчикам легко разрабатывать и тестировать кроссплатформенные приложения, используя основные функции Oracle VM VirtualBox, в том числе:

- ➢Кроссплатформенные гости и хосты
- ➢Оперативная миграция виртуальных машин между хостами
- ➢До 32 виртуальных процессоров
- ➢Поддержка формата OVF

-Пакет расширений

Пакет расширений, лицензированный по лицензии VirtualBox Personal Use and Evaluation License (PUEL), позволяет ИТ-подразделениям консолидировать больше рабочих нагрузок с помощью дополнительных функций, таких как:

- ➢Виртуальные USB-устройства
- ➢Поддержка протокола удаленного рабочего стола VirtualBox (VRDP)
- ➢Сквозная передача веб-камеры хоста и сквозная передача PCI
- ➢Загрузочное ПЗУ Intel PXE
- ➢Шифрование образа диска

➢Интеграция с облачной инфраструктурой Oracle

➢Мы будем использовать базовый пакет.

СЛАЙД 59

Процесс установки VirtualBox может быть различным в зависимости от операционной системы.

Установочные пакеты для различных операционных систем доступны на сайте VirtualBox.

Однако для дистрибутивов Linux рекомендуется использовать пакет из репозиториев, настроенных в операционной системе.

В ходе установки Аврора SDK будут добавлены две виртуальные машины:

●Aurora Build Engine (в случае, если выбрана установка среды сборки в виде виртуальной машины);

●Aurora Emulator (наличие VirtualBox обязательно даже для версии с Docker Build Engine).

Среда сборки обеспечивает сборку приложений, не зависящую от системы разработчика, эмулятор позволяет выполнять приложения в окружении Аврора ОС аналогично работе на мобильных устройствах.

СЛАЙД 60

2.5. Git

Git - это распределенная система управления версиями, используемая для отслеживания изменений в исходном коде программного обеспечения и совместной работы над ним.

Основные функции Git:

1. Хранение исходного кода: Git используется для хранения и управления исходным кодом программного обеспечения.

2. Управление версиями: Git отслеживает изменения в коде, позволяет вернуться к предыдущим версиям, создавать разветвленные версии и сливать их в одну.

3. Совместная работа: Git обеспечивает легкий доступ к исходному коду и позволяет нескольким разработчикам работать над одной версией кода.

4. Контроль интеграции: Git позволяет контролировать интеграцию изменений в код, а также автоматически сливать изменения в один код.

5. Быстрый доступ: Git позволяет быстро получать доступ к исходному коду и работать с ним даже тогда, когда нет доступа к Интернету.

В целом Git обеспечивает удобство работы с кодом, уменьшает количество ошибок и облегчает совместную работу над проектом.

СЛАЙД 61

Репозиторий – часть системы Git, которая позволяет программистам совместно работать над проектами. Этот инструмент облегчает жизнь IT-специалистам: с ним можно безопасно вносить изменения в программный код.

Репозиторий – это хранилище всех версий кода.

Он бывает трех видов:

●Локальный – расположен на одном компьютере, и работать с ним может только один человек.

●Централизованный – расположен на сервере, куда имеют доступ сразу несколько программистов.

●Распределенный – самый удобный вариант с облачным хранилищем. Главный репозиторий хранится в облаке, а его локальные копии – у разработчиков на компьютерах. Когда программист вносит правки в локальную версию, ее можно синхронизировать с удаленной. Получается, что в облаке всегда актуальный код.

Для работы с распределенными репозиториями нужен удобный сервис. У них понятный интерфейс, в котором можно управлять проектом, добавлять новые объекты и искать общедоступные репозитории.

СЛАЙД 62

Git-сервисы позволяют переключаться между ветками кода и просматривать коммиты.

Коммит - это фиксация изменений в репозитории Git. Каждый коммит содержит описание того, что было изменено, а также указатель на предыдущий коммит (или коммиты), образуя цепочку изменений. Коммит также содержит дату, автора и уникальный идентификатор.

Ветка - это параллельная линия развития в Git, которая позволяет работать с изменениями в проекте изолированно от основной ветки. Ветки позволяют нескольким разработчикам работать над одним проектом без возможности повреждения основной ветки. Каждая ветка может содержать свою цепочку коммитов.

Ветки полезны, когда нужно добавить новую функциональность в проект, но не хотите вносить изменения в основную ветку. Ветка может быть создана с помощью команды `git branch <branch name>. Чтобы переключиться на созданную ветку, используйте команду `git checkout <branch name>. После этого вы сможете создавать коммиты и работать в этой ветке изолированно от основной ветки. Когда работа над изменениями завершена, ветка может быть объединена с основной веткой с помощью команды `git merge <branch name>.

СЛАЙД 63

2.6. SSH

SSH (Secure Shell) - это безопасный протокол для удаленного управления компьютерами по сети. Он позволяет вам подключаться к серверу удаленно и работать с ним через командную строку.

SSH-Key - это пара криптографических ключей, которые используются для аутентификации при подключении к удаленному серверу по SSH.

В паре SSH-ключей есть приватный ключ и открытый ключ.

Приватный ключ хранится на вашем компьютере, и только вы имеете к нему доступ.

Открытый ключ копируется на сервер, к которому вы хотите подключиться. При подключении к серверу SSH использует приватный ключ, чтобы создать подпись для

сообщения, которое отправляется на сервер. Сервер проверяет эту подпись, используя открытый ключ, чтобы убедиться, что подключение идет от доверенного источника.

Существует несколько типов SSH-ключей, но самые распространенные - это RSA и ECDSA.

RSA-ключи являются стандартными и могут иметь различную длину, от 768 до 16384 бит.

ECDSA-ключи используют эллиптические кривые и считаются более безопасными, чем RSA.

Использование SSH и SSH-ключей - это более безопасный способ подключения к удаленным серверам, т.к. все данные передаются в зашифрованной форме.

Использование SSH-ключей также позволяет не вводить пароли каждый раз при подключении к серверу, что ускоряет процесс работы.

СЛАЙД 64

Где используется SSH

SSH-протокол применяется:

- в удаленном системном администрировании локальных сетей;
- в управлении работой почтовых служб (защите данных);
- для скрытой передачи файлов большого размера внутри сети;
- для переноса проектов между серверами хостинг-провайдеров;
- для подключения в сетевых многопользовательских играх и т. д.

СЛАЙД 65

Как работать по SSH

Чтобы установить SSH-соединение, необходимы два компонента: SSH-сервер и SSH-клиент.

Сервер прослушивает определенный порт (по умолчанию это порт 22) и при успешной аутентификации дает доступ пользователю.

Все команды, которые используются на SSH-клиенте, отправляются через защищенный канал связи на SSH-сервер, на котором они выполняются и откуда отправляют результат работы обратно клиенту.

На рис. 2.9 схематично изображён процесс установления SSH-соединения.

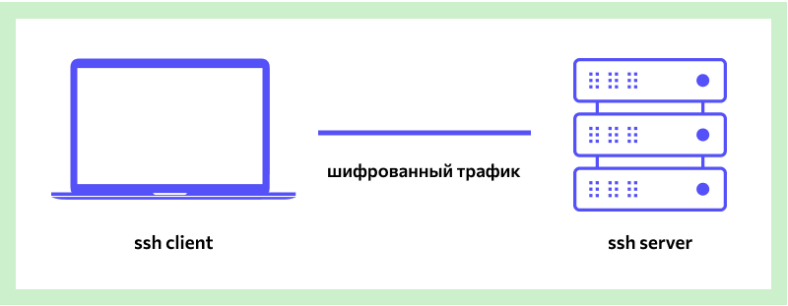

Рисунок 2.9 – SSH-соединение

СЛАЙД 66

SSH-сервер

Устанавливается на управляемой операционной системе и принимает входящие подключения от клиентских машин, проверяя соединение одним из способов:

●по IP-адресу клиента, что не слишком безопасно из-за риска подмены;

●по публичному ключу клиента и имени пользователя. Нужно создать приватный (закрытый) и публичный (открытый) ключ. Зашифровав информацию одним ключом, можно расшифровать ее только другим;

●по паролю клиента, который передается в зашифрованном виде. Это один из наиболее распространенных вариантов. Вводить его нужно при каждом подключении.

Платные и бесплатные SSH-серверы есть для всех распространенных ОС:

●BSD–OpenSSH;

●Linux–dropbear, lsh-server, openssh-server;

●Windows–freeSSHd, copssh, WinSSHD, OpenSSH и т. д.

СЛАЙД 67

SSH-клиент

Используется для входа на удаленный сервер и выполнения различных команд. Через клиента выполняется управление удаленным компьютером, в том числе:

●управление файлами и директориями;

●просмотр и редактирование файлов;

●контроль рабочих процессов;

●управление архивами, базами данных MySQL и т. д.

SSH-клиенты разработаны для всех десктопных и мобильных ОС. Имеют платные и бесплатные версии:

●для Linux/BSD - openssh-client, putty, ssh, Vinagre;

●MS Windows - PuTTY, SecureCRT, ShellGuard;

●Android - connectBot;

●Linux и MacOS имеют встроенный SSH-клиент, дополнительная настройка не требуется.

СЛАЙД 68

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

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

ШАГ 1. Перейти в **«Настройки» → «Средства разработчика» → пункт «Удалённое соединение»;**

ШАГ 2. Задать SSH-пароль и нажать кнопку **«Сохранить»**.

Результат выполнения действий представлен на **рис. 2.10**.

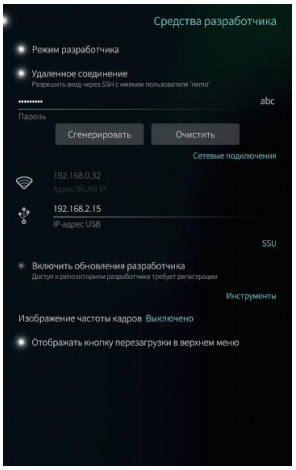

Рисунок 2.10 – Настройка подключения по SSH перед добавлением МУ.

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

ШАГ 1. В IDE открыть вкладку **«Инструменты» → «Параметры» → «Устройства» (рис. 2.11)**.

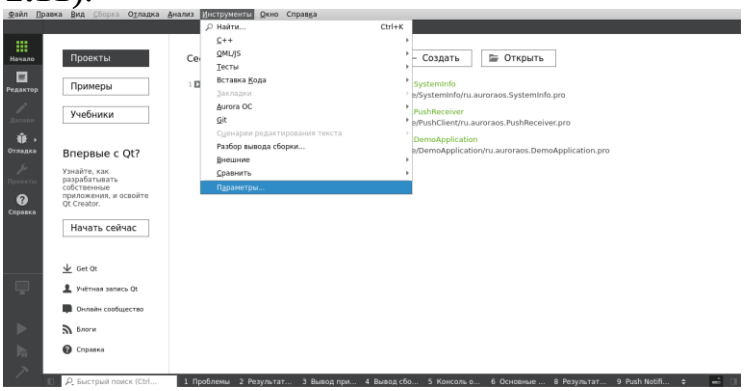

Рисунок 2.11 – Подключение МУ к Аврора IDE. Шаг 1

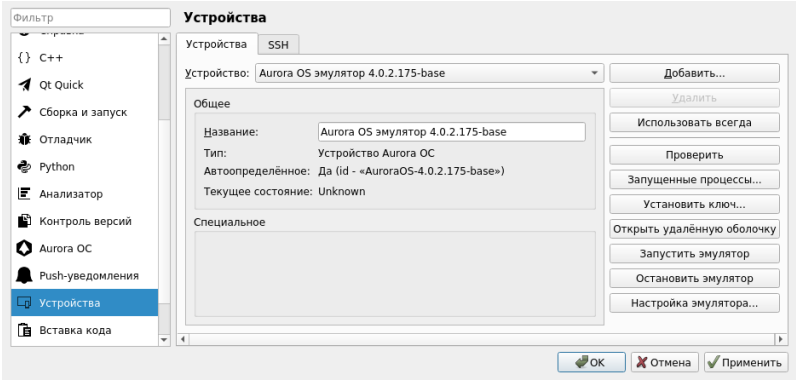

Рисунок 2.11 – Подключение МУ к Аврора IDE. Шаг 1

СЛАЙД 69

ШАГ 2. Нажать кнопку **«Добавить устройство»**.

ШАГ 3. Выбрать пункт **«Устройство Аврора ОС»** и нажать кнопку **«Запустить мастера»**.

Результат выполнения второго и третьего ШАГОВ подключения МУ к Аврора IDE представлен на **рис. 2.12**.

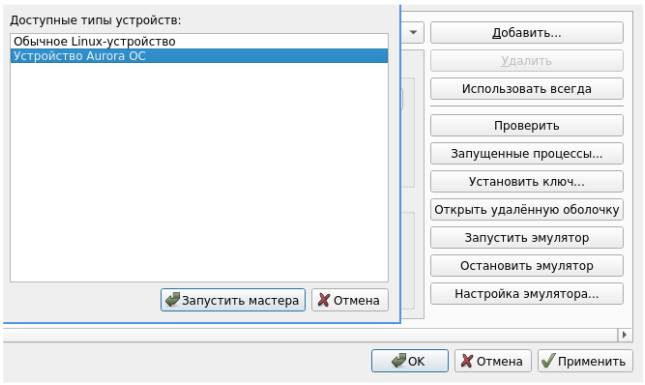

Рисунок 2.12 – Подключение МУ к Аврора IDE. Шаг 2 и 3

ШАГ 4. Ввести имя пользователя и IP-адрес МУ (по умолчанию IP-адрес SSH-соединения и пользователь defaultuser).

Результат выполнения четвёртого ШАГА подключение МУ к Аврора IDE представлен на **рис. 2.13**.

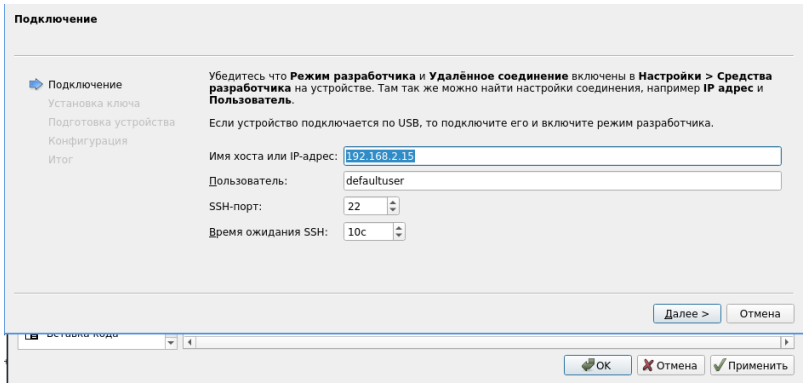

Рисунок 2.13 – Подключение МУ к Аврора IDE. Шаг 4

СЛАЙД 70

ШАГ 5. Создать пару ключей или выбрать существующие.

Результат выполнения пятого ШАГА подключение МУ к Аврора IDE представлен на **рис. 2.14** и **рис. 2.15.**

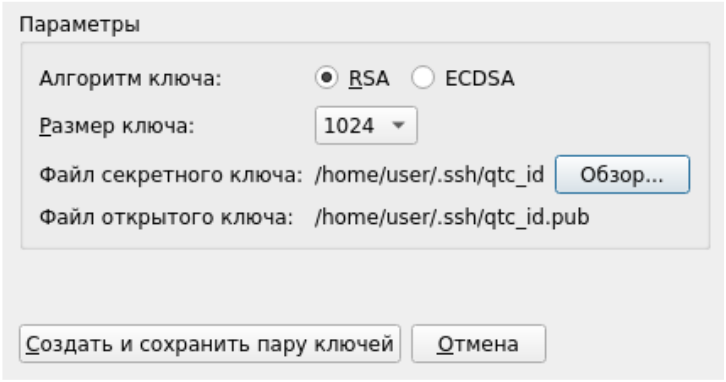

Рисунок 2.14 – Подключение МУ к Аврора IDE. Шаг 5

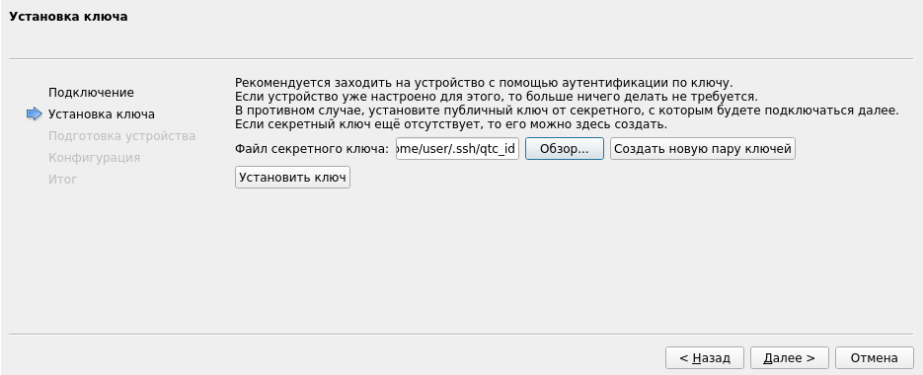

Рисунок 2.15 – Подключение МУ к Аврора IDE. Шаг 5

СЛАЙД 71

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

Результат выполнения шестого ШАГА подключение МУ к Аврора IDE представлен на **рис. 2.16**.

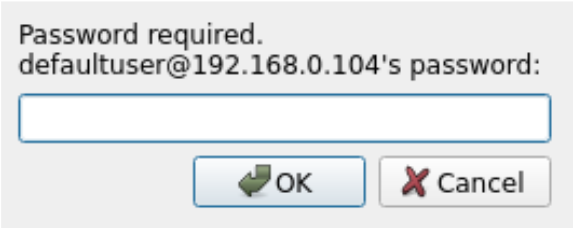

Рисунок 2.16 – Подключение МУ к Аврора IDE. Шаг 6

Если пароль введён верно, ключ будет успешно установлен (**рис. 2.17**).

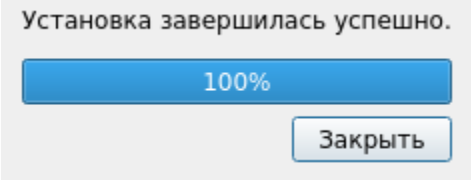

Рисунок 2.17 – Подключение МУ к Аврора IDE. Шаг 6 – ключ установлен успешно

ШАГ 7. После успешной установки ключа рядом с кнопкой должна появиться зелёная галочка.

Результат выполнения седьмого ШАГА подключение МУ к Аврора IDE представлен на **рис. 2.18** и **рис. 2.19.**

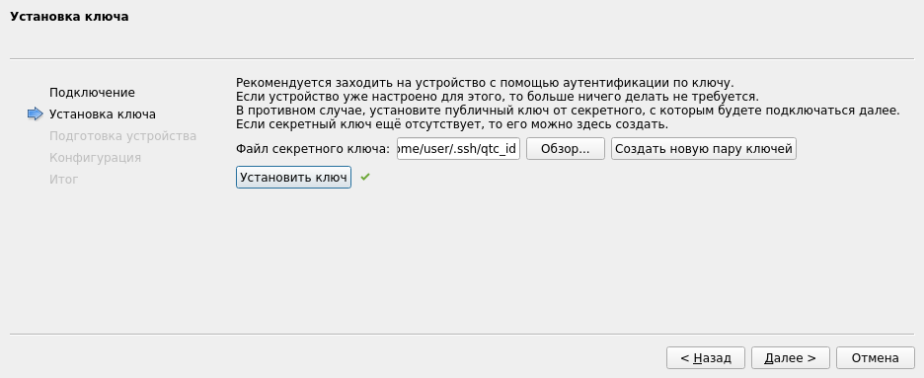

Рисунок 2.18 – Подключение МУ к Аврора IDE. Шаг 7

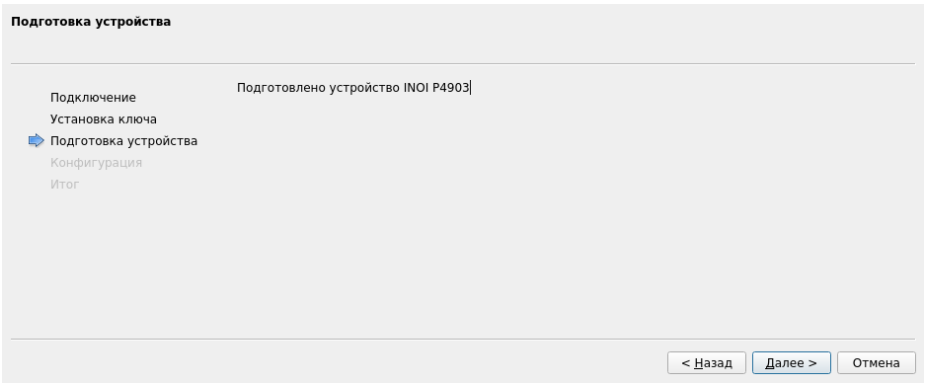

Рисунок 2.19 – Подключение МУ к Аврора IDE. Шаг 7

СЛАЙД 72

ШАГ 8. Нажать кнопку **«Далее»**, после чего МУ будет подготовлено к соединению. Когда подготовка завершится, снова нужно нажать кнопку «Далее».

ШАГ 9. При необходимости на вкладке КОНФИГУРАЦИЯ ввести название новой конфигурации и диапазон свободных портов, а затем нажать кнопку «Далее». По умолчанию в названии конфигурации будет записана модель МУ и тип архитектуры. Количество свободных портов должно быть не менее двух, это необходимо для отладки.

Результат выполнения девятого ШАГА подключение МУ к Аврора IDE представлен на **рис. 2.20.**

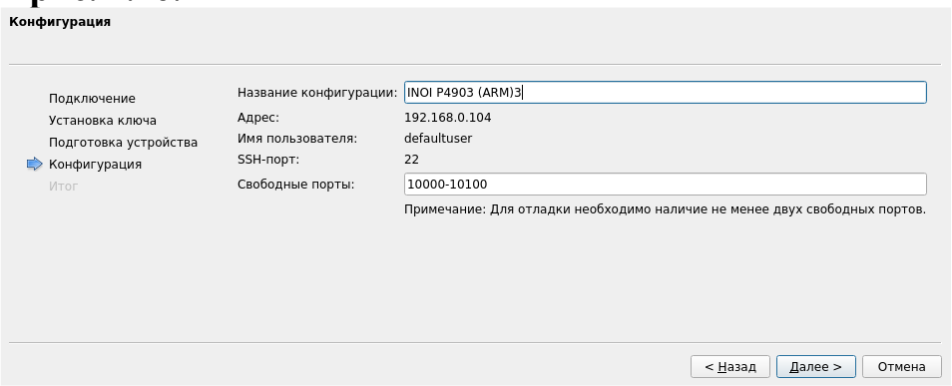

Рисунок 2.20 – Подключение МУ к Аврора IDE. Шаг 9

ШАГ 10. На вкладке **Итог** нажать кнопку **«Завершить»**. Подключение МУ будет завершено.

Результат выполнения десятого ШАГА подключение МУ к Аврора IDE представлен на **рис. 2.21** и **рис. 2.22.**

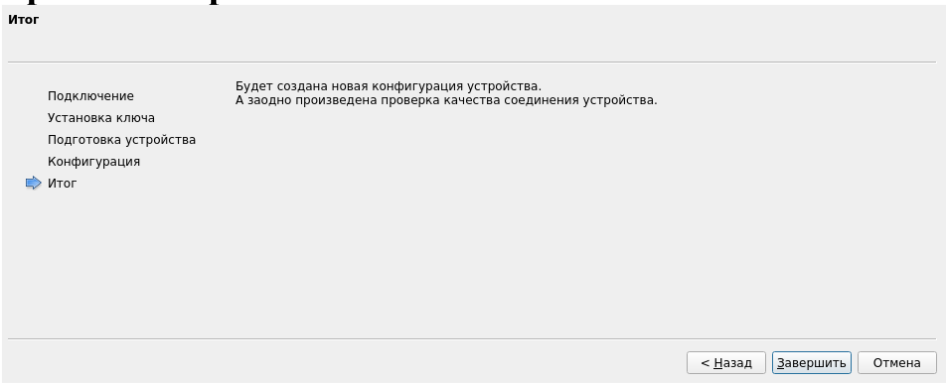

Рисунок 2.21 – Подключение МУ к Аврора IDE. Шаг 10

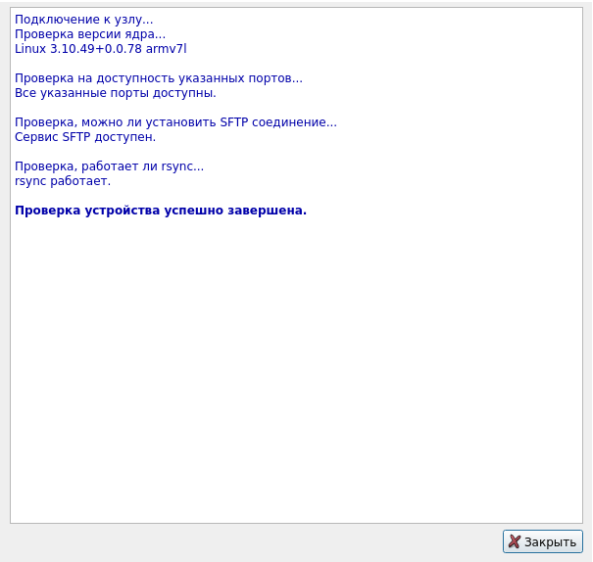

Рисунок 2.22 – Подключение МУ к Аврора IDE. Шаг 10

СЛАЙД 73

ШАГ 11. На вкладке **«Устройства»** появится новое МУ.

Результат выполнения десятого ШАГА подключение МУ к Аврора IDE представлен на **рис. 2.23.**

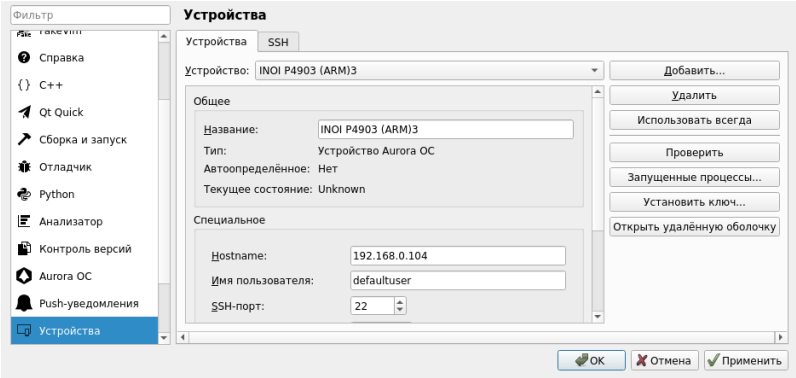

Рисунок 2.22 – Подключение МУ к Аврора IDE. Шаг 11. Многое получилось

СЛАЙД 74

2.7. Эмулятор

Aurora OS Emulator (эмулятор) – виртуальная машина, которая позволяет выполнять приложения в окружении ОС Аврора аналогично работе на мобильных устройствах.

Интерфейс эмулятора АВРОРЫ представлен на **рис. 2.23, 2.24** и **2.25**.

Рисунок 2.23 – Эмулятор АВРОРЫ

Рисунок 2.24 – Эмулятор АВРОРЫ

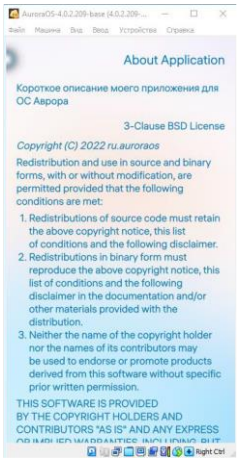

Рисунок 2.25 – Эмулятор АВРОРЫ

СЛАЙД 75

2.8. Создание проекта приложение по шаблону.

2.9. Сборка и запуск приложения в эмуляторе.

2.9. Запуск приложения с использованием Qt QmlLive

2.10. Сборка RPM-файла приложения для ОС Аврора

Практика №2. ВИДЕО. Продолжительностью 9 минут 50 секунд

СЛАЙД 76

ТРЕТИЙ УЧЕБНЫЙ ВОПРОС.
СТРУКТУРА ПРОЕКТА: ОСОБЕННОСТИ И ОБЩИЕ СВЕДЕНИЯ

3.1. Обзор файлов, входящих в проект, их назначение и использование при сборке

Дерево проекта

Начнем сразу с общей структуры всего проекта с последовательным разбором его элементов (**рис. 3.1**):

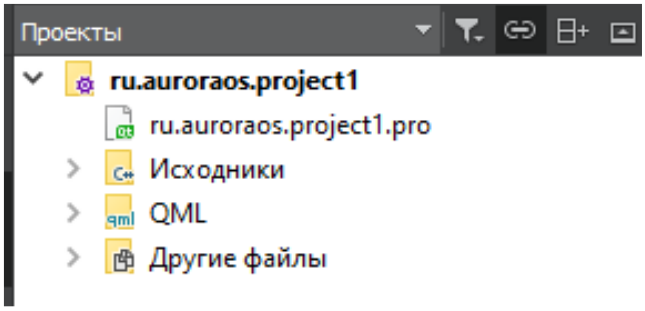

Рисунок 3.1 – Структура проекта. Дерево проекта

В проекте первый файл с расширением .pro - основной сборочный файл.

Папка Исходники: в ней находятся файлы с расширением .cpp

Папка QML: в ней находятся файлы с расширением. qml

В других файлах находятся иконки программы, изображения, используемые в программе, аудио, видеофрагменты и прочие ресурсы.

●**Каталог Исходники (рис. 3.2)**

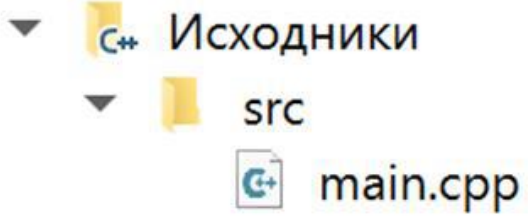

Рисунок 3.2– Структура проекта. Каталог ИСХОДНИКИ

Здесь находится еще одна папка src, в которой находится файл main.cpp. (**рис. 3.3**).

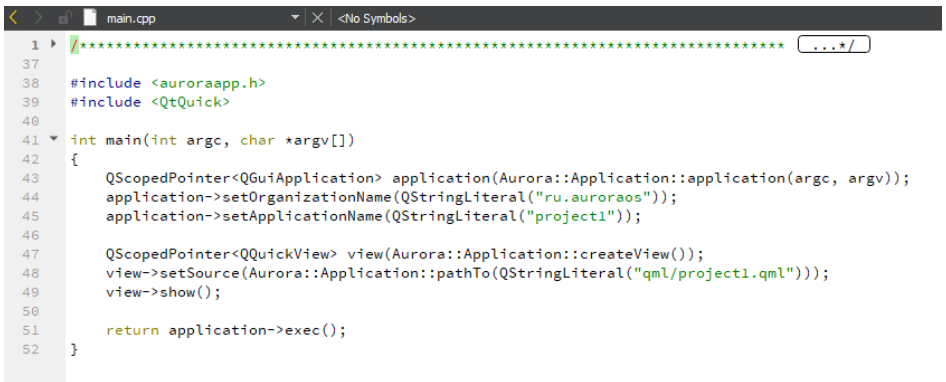

Рисунок 3.3– Структура проекта. Каталог ИСХОДНИКИ. Папка src

СЛАЙД 77

●**Каталог QML (рис. 3.4)**

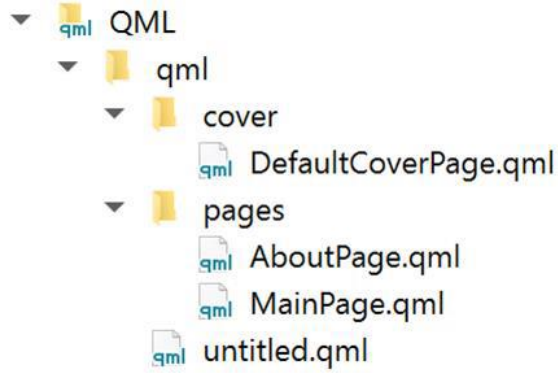

Рисунок 3.4– Структура проекта. Каталог QML

Здесь находятся: папка для обложки приложения (cover), файлы которой генерируют отображение запущенной программы в свернутом виде, папка для страниц приложения (pages) и ОСНОВНОЙ ФАЙЛ ПРИЛОЖЕНИЯ (**project1.qml**), содержание которого представлено на **рис. 3.5**.

Основной файл приложения **project1.qml**

```
import QtQuick 2.0
import Sailfish.Silica 1.0

ApplicationWindow {
    objectName: "applicationWindow"
    initialPage: Qt.resolvedUrl("pages/MainPage.qml")
    cover: Qt.resolvedUrl("cover/DefaultCoverPage.qml")
    allowedOrientations: defaultAllowedOrientations
}
```

Рисунок 3.5– Структура проекта. Каталог QML.

-***objectName*** - название приложения;

-***initialPage*** - прописывается главная страница приложения;

-***cover*** - прописывается обложка приложения. То, как будет выглядеть свернутое приложение;

-***allowedOrientations*** – какие ориентации приложения разрешены. По умолчанию разрешены все ориентации.

СЛАЙД 78

Папка Pages

Содержимое папки **Pages** представлено на **рис. 3.6**.

Рисунок 3.6 – Структура проекта. Каталог QML. Папка Pages

В шаблонном приложении по умолчанию есть файл основной страницы приложения (MainPage.qml) и добавлена еще одна страница - о приложении (AboutPage.qml). В дальнейшем можно добавлять сюда и новые страницы приложения.

Папка Cover

Содержимое папки **Cover** представлено на **рис. 3.7**.

Рисунок 3.7 – Структура проекта. Каталог QML. Папка Cover

Файл, который содержит информацию об обложке по умолчанию. Можно его изменить или добавить новый.

СЛАЙД 79

●**Каталог Другие файлы (рис. 3.8).**

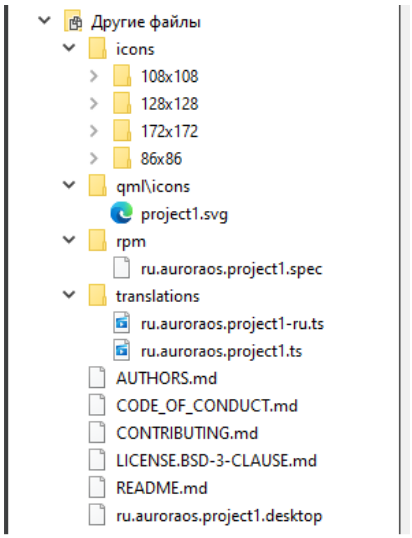

Рисунок 3.8 – Структура проекта. Каталог Другие файлы

Сейчас довольно сложно написать более или менее серьезное приложение, в котором не задействованы никакие сторонние библиотеки.

Конечно, всегда есть вариант разместить эти зависимости в системных папках и настроить к ним пути, но это решение имеет множество недостатков.

Поэтому в данной директории находятся различные другие папки и файлы, которые используются в приложении.

СЛАЙД 80

Рассмотрим главные конфигурационные файлы: pro, pro.user, spec, desktop.

Файл pro

Файл с расширением pro – это текстовый файл, который используется для конфигурирования qmake при сборке проекта. Он генерируется автоматически. Наиболее часто вручную в этот файл вносятся дополнения в секцию CONFIG, в которой указываются дополнительные модули и библиотеки QT, которые не

подключаются по умолчанию. Файл конфигурирует весь проект и более подробно будет рассмотрен позже.

Файл pro.user

Это текстовый файл, который содержит в себе настройки проекта, связанные с конкретным устройством, на котором ведется разработка проекта. Он создается автоматически и обычно не требует вмешательства программиста. При изменении пути проекта этот файл желательно удалить, чтобы избежать ошибок при сборке проекта.

Файл spec

В директории rpm находится файл с расширением spec. Он используется для конфигурирования rpm пакета. В нем указываются: лицензия, имя файла устанавливаемой программы в виде пакета и команды для сборки, установки и удаления программы.

Файл desktop

Файл Desktop это текстовый файл, который используется как ярлык в ОС Аврора для быстрого запуска приложения. Например, в Windows для этой цели используются ярлыки на рабочем столе.

СЛАЙД 81

3.2. Сборка, деплой и запуск проекта

Запуск и отладка проекта

Приложения для ОС Аврора пишутся на C++/Qt с использованием QML для описания интерфейса пользователя. Создание приложения осуществляется в Аврора IDE, основанной на Qt Creator, и практически совпадает с процессами создания приложений для множества настольных и мобильных платформ. Отличия связаны с тем, что сборка происходит в среде сборки, а запуск — в эмуляторе или на внешнем устройстве с–ОС Аврора.

Создание или открытие проекта

Приступить к работе над проектом можно одним из следующих способов:

- создать новый проект из шаблона;
- создать новый проект из примера;
- открыть существующий проект.

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...».

3. В открывшемся окне «Новый файл или проект» выбрать вкладку «Проекты» → «Приложение» и отметить «ОС Аврора SDK Qt Quick Application», после чего нажать кнопку «Выбрать...».

СЛАЙД 82

4. В появившемся окне «Введение и размещение проекта» указать имя проекта, директорию и нажать кнопку «Далее». Важно иметь в виду, что 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 предлагаться для следующих создаваемых проектов.

5. В следующем окне «Application Details» ввести необходимые данные о приложении и нажать кнопку «Далее».

6. В открывшемся окне «Выбор комплекта» выбрать необходимые комплекты для сборки и нажать кнопку «Далее». Комплект arm7hl используется для мобильных устройств, i486 — для эмулятора. Позже набор комплектов можно изменить в настройках проекта.

7. В появившемся окне «Управление проектом» выбрать необходимые данные и нажать кнопку «Завершить». Данное окно позволяет настроить взаимное положение проекта относительно других и подключить одну из систем контроля версий, доступных в операционной системе.

8. В открывшемся редакторе исходного кода можно приступить к работе над проектом.

СЛАЙД 83

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 открывшейся галерее доступных примеров приложений выбрать интересующий пример и кликнуть правой кнопкой мыши по нему. При наведении курсором мыши на миниатюру примера будет показано его описание.

4. В появившемся окне «Copy Project to writable Location» указать директорию и нажать кнопку «Ок». Важно иметь в виду, что 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.

5. В следующем окне выбрать необходимые комплекты для сборки и нажать кнопку «Настроить проект». Комплект arm7hl используется для мобильных устройств, i486 – для эмулятора. Позже набор комплектов можно изменить в настройках проекта.

6. В открывшемся редакторе исходного кода можно приступить к работе над проектом.

СЛАЙД 84

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. Важно иметь в виду, что *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».

4. Если файл с расширением ***.user** отсутствует, или он некорректен, в окне Аврора IDE перейти в режим **«Проекты»** и выбрать необходимые комплекты для сборки. Комплект arm7hl используется для мобильных

устройств, i486 – для эмулятора. Позже набор комплектов можно изменить в настройках проекта.

5. В окне Аврора IDE в режиме **«Редактор»** можно приступить к работе над проектом.

СЛАЙД 85

Сборка проекта

На этапе сборки предполагается, что в Аврора IDE существует открытый проект. Для сборки проекта используется среда сборки, поэтому независимо от операционной системы 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 SDK, а также все вложенные в них директории.

Для сборки проекта необходимо выполнить следующие действия:

ШАГ 1. Запустить среду сборки. Запуск, если требуется, происходит автоматически при начале сборки. Для управления виртуальной машиной

в ручном режиме необходимо выполнить следующее:

●для запуска на панели слева нажать кнопку ***Запуск «Aurora Build Engine»***,

и дождаться, пока она не примет вид ***Остановка «Aurora Build Engine»***

●для остановки нажать кнопку ***Остановка «Aurora Build Engine»***

ШАГ 2. На панели слева нажать кнопку ***Опции сборки***

и выбрать комплекты сборки.

Для эмулятора необходимо выбрать AuroraOS-i486, для мобильных устройств – AuroraOS-armv7hl.

Здесь же можно выбрать способ сборки:

●«Выпуск» — завершающая сборка пакета для передачи заказчику или пользователю программы;

●«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).

ШАГ 3. После завершения настроек нажать кнопку ***Сборка проекта***

для запуска проекта.

Для того, чтобы отладка и профилирование проекта были доступны, необходимо у настройки Проекты → Сборка → Отладка и профилирование QML установить значение Enable.

СЛАЙД 86

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 следующее:

ШАГ 1. На панели слева нажать кнопку ***Опции сборки***

и выбрать комплект AuroraOS-i486

ШАГ 2. Запустить эмулятор нажатием кнопки ***Запуск «Aurora Build Emulator»***

и дождаться, пока она не примет вид ***Остановка «Aurora Build Emulator»***

Откроется окно VirtualBox, загрузится эмулятор.

Если нажать кнопку ***Остановка «Aurora Build Emulator»***,

эмулятор остановится, окно VirtualBox закроется.

СЛАЙД 87

ШАГ 3. Для запуска нажать кнопку ***«Запустить»***

Для отладки нажать кнопку ***«Отладка»***

Чтобы кнопки стали активны, необходимо выбрать «Deploy As RPM Package» или «Deploy by Copying Binaries» в панели выбора комплектов и способов сборки. На экране эмулятора появится диалог подтверждения установки приложения **(рис. 3.9)**.

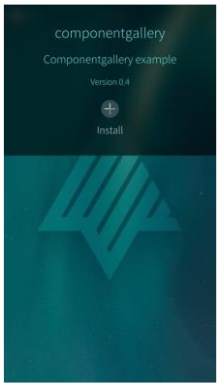

Рисунок 3.9 – Запуск приложения – появление диалога подтверждения установки приложения – выполнение ШАГА 3

СЛАЙД 88

ШАГ 4. Для установки приложения коснуться значка «Install» на экране эмулятора. Приложение установится, и откроется его стартовая страница **(рис. 3.10)**.

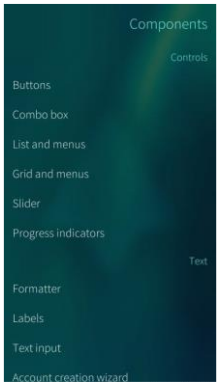

Рисунок 3.10 – Запуск приложения –выполнение ШАГА 4

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.

СЛАЙД 89

3.3. Основы qmake/cmake

qmake

Инструмент qmake распространяется вместе с набором библиотек Qt начиная с версии 3.0 (2001 г.). Изначально он являлся переписанной на C++ версией инструмента tmake – сценария на языке Perl, разработка которого велась с 1996 г. Основной причиной, вынудившей разработчиков Qt создать собственный инструмент автоматизации построения, была недостаточная гибкость набора инструментов Autotools.

Так же, как и Autotools, qmake не является системой построения в строгом понимании этого слова, он всего лишь генерирует файлы описания проектов для других систем.

На вход программе qmake подаётся описание проекта на высокоуровневом языке (один или несколько файлов с расширением «.pro»), в результате инструмент создаёт файл для системы make (Makefile). Также возможно генерирование файлов проектов и решений для интегрированных сред разработки Visual Studio и XCode16.

Интегрированная среда Qt Creator использует файлы «*.pro» в качестве файлов описания проекта. Однако в отличие от других сред разработки, эта среда поддерживает только визуальное редактирование списка файлов проекта, но не его настроек. Для изменения настроек необходимо отредактировать файл проекта в текстовом редакторе (например, в самой среде). В диалоговых окнах среды Qt Creator можно только устанавливать настройки, не указываемые в файлах проекта: директория построения, параметры командной строки при вызове qmake и т. д. Для осуществления построения среда Qt Creator сначала запускает инструмент qmake, затем make.

Утилита qmake записывает генерируемые файлы в заданную директорию (по умолчанию в текущей). При этом входные файлы добавляются в генерируемые проекты по относительным путям, а выходные и промежуточные файлы позже, на этапе построения, записываются в поддиректории относительно директории с генерируемым проектом.

Таким образом, можно легко организовать построение вне директории проекта, причём расположение директории с промежуточными и выходными файлами определяется запуском утилиты qmake. То есть расположение директории построения можно легко менять, не меняя файла описания (*.pro).

Язык qmake поддерживает специальные переменные, определяющие тип проекта (приложение, библиотека и т. д.), списки файлов исходных кодов, заголовочных

файлов, файлов ресурсов и т. д. При необходимости автоматически генерируются правила для подключения библиотек Qt, вызова инструментов обработки исходных файлов из состава Qt, хотя qmake можно использовать и для проектов, не использующих Qt. Также языком поддерживаются условные конструкции и функции (встроенные и определяемые пользователем). В условных конструкциях можно проверять параметры конфигурации (например, целевую платформу), вызывать встроенные и пользовательские логические функции, при помощи которых можно организовывать циклы, выводить диагностические сообщения и т. д. Эти и другие средства языка позволяют расширять возможности инструмента qmake, добавляя поддержку новых компиляторов, языков программирования, инструментов, платформ, библиотек и т. д.

СЛАЙД 90

Таким образом, можно выделить следующие достоинства инструмента qmake:

●Простой высокоуровневый язык описания проектов.

●Переносимость: поддержка большого количества целевых платформ и компиляторов (Intel C Compiler, Microsoft Visual C++ и т. д.).

Требования к системе разработчика включают наличие инструмента qmake, т. е. подойдёт любая система, на которую портирован набор библиотек Qt.

●Возможность работы над проектами в средах Visual Studio, XCode, Qt Creator. Если желательно автоматизировать построение, можно сгенерировать обычные make-файлы.

●Поддержка в генерируемых проектах библиотек Qt, добавление которых вручную является слишком сложным.

●Простая в использовании поддержка построения вне каталога проекта.

●Расширяемость.

СЛАЙД 92

У инструмента qmake нет существенных недостатков, но всё же можно выделить несколько проблем, препятствующих его широкому распространению:

●Ориентированность в первую очередь на набор библиотек Qt. Хотя при помощи расширений возможно научить qmake работать с другими инструментами, в составе утилиты эти расширения отсутствуют. Что касается библиотек, инструменту qmake известны расположения только заголовочных файлов, библиотечных модулей и т. д. из состава Qt. Пути к файлам других библиотек нужно указывать явно.

●В отличие от инструментов Autotools, система qmake не предусматривает возможности запуска серии тестов для определения особенностей среды построения.

●Также не предусмотрены средства для генерирования заголовочных и прочих файлов.

СЛАЙД 91

Основные команды qmake

Рассмотрим основные команды qmake, в частности, те, которые появляются в pro-файле Аврора-проекта по умолчанию (**рис. 3.11**).

```
TARGET = ru.auroraos.project1

CONFIG += \
    auroraapp

PKGCONFIG += \

SOURCES += \
    src/main.cpp \

HEADERS += \

DISTFILES += \
    rpm/ru.auroraos.project1.spec \
    AUTHORS.md \
    CODE_OF_CONDUCT.md \
    CONTRIBUTING.md \
    LICENSE.BSD-3-CLAUSE.md \
    README.md \

AURORAAPP_ICONS = 86x86 108x108 128x128 172x172

CONFIG += auroraapp_i18n

TRANSLATIONS += \
    translations/ru.auroraos.project1.ts \
    translations/ru.auroraos.project1-ru.ts \
```

Рисунок 3.11 – ru.auroraos.project1.pro

●TARGET – Указывает имя целевого файла. По умолчанию содержит базовое имя файла проекта.

●CONFIG – Определяет конфигурацию проекта и параметры компилятора. Значения распознаются внутри qmake и имеют особый смысл.

●PKGCONFIG – Модуль pkgconfig используется для тонкой настройки поведения инструмента pkg-config, который потенциально может быть использован при поиске зависимостей.

●SOURCES – Указывает имена всех исходных файлов в проекте.

●HEADERS – Определяет заголовочные файлы для проекта. qmake автоматически определяет, требуется ли moc для классов в заголовках, и добавляет в проект соответствующие зависимости и файлы для генерации и компоновки файлов moc.

● DISTFILES – Определяет список файлов, которые должны быть включены в цель dist. Эта функция поддерживается только в спецификациях UnixMake.

● TRANSLATIONS – Указывает список файлов перевода (.ts), которые содержат переводы текста пользовательского интерфейса на неродные языки.

СЛАЙД 92

CMake

CMake является свободным инструментом с открытым исходным кодом, основным разработчиком которого выступает компания Kitware. Название системы расшифровывается как «cross-platform make». Разработка инструмента ведётся с 1999 г., в качестве прототипа была использована утилита pcmaker, написанная в 1997 г. одним из авторов CMake. В настоящее время инструмент внедряется в процесс разработки многих программных продуктов, в качестве примеров широко известных

проектов с открытым кодом можно привести KDE, MySQL, Blender20, LLVM + clang21 и многие другие.

Принцип работы инструмента CMake аналогичен принципу работы qmake: из каталога исходных кодов считывается файл CMakeLists.txt с описанием проекта, на выходе инструмент генерирует файлы проекта для одной из множества конечных систем построения.

Требования для сборки самого CMake включают наличие утилиты make и интерпретатора сценариев на языке bash либо скомпилированного инструмента CMake одной из предыдущих версий, а также компилятора C++. При этом в исходных кодах CMake преднамеренно используются только возможности языка и стандартной библиотеки, поддерживаемые достаточно старыми версиями компиляторов. Таким образом, CMake переносим на большое количество платформ.

Следует отметить, что современные версии интегрированной среды Qt Creator в дополнение к описаниям проектов на языке qmake также поддерживают язык CMake.

Система CMake управляется при помощи универсального процедурного языка. В то время как инструмент qmake позволяет в более простой и компактной форме выполнять описание проектов, использующих набор библиотек и инструментов Qt, при помощи CMake легче описывать проекты, которые используют другие библиотеки и инструменты, а также решать нестандартные задачи, которые возникают при организации процесса построения.

СЛАЙД 93

Основные возможности CMake, отсутствующие в qmake, включают в себя следующее:

●Простой интерфейс для подключения библиотек, являющихся результатами построения одних целей, к другим. Например, проект может включать цель библиотеки и использующего её приложения. В CMake можно легко установить зависимость между такими целями, при этом к правилам построения приложения будут автоматически добавлены все необходимые настройки компилятора и компоновщика для подключения библиотеки. Те же правила в qmake придётся определять на более низком уровне. Кроме этого, в описании цели библиотеки можно определять дополнительные настройки, которые будут использованы при построении всех её клиентов, что избавляет от их повтора для каждого из них.

●Команды и сценарии для поиска в системе наборов библиотек и/или инструментов (пакетов в терминологии CMake). В состав CMake входит большое количество сценариев (модулей поиска) для наиболее популярных и распространённых в среде разработчиков пакетов. Можно создавать собственные модули поиска на языке CMake, также можно встраивать поддержку CMake в собственные наборы библиотек.

●Средства генерирования исходных файлов и сценариев в процессе построения (аналогично системе Autotools, которая способна создавать правила для генерирования файлов config.h и Makefile).

В целом можно утверждать, что qmake больше ориентирован на использование инструментария Qt, в состав которого входит, тогда как CMake, скорее, является более универсальным решением.