



А В Р О Р А

Модуль 1. Основы Qt Quick

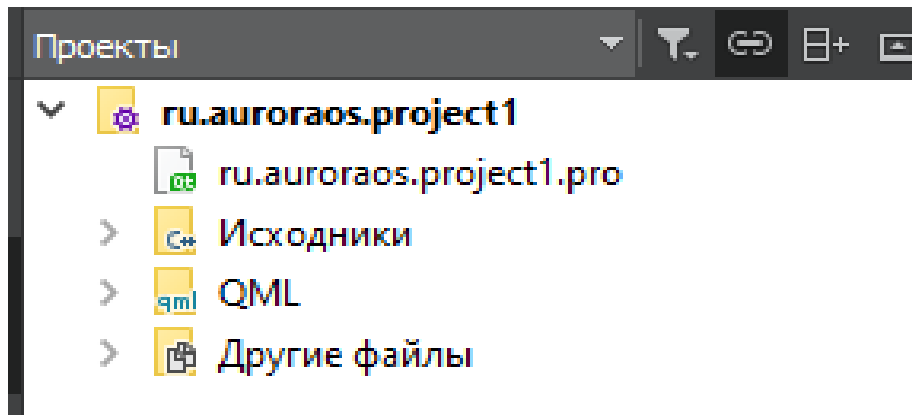
Тема 1.3. Структура проекта

1. Обзор файлов, входящих в проект, их назначение и использование при сборке	2
2. Сборка, деплой и запуск проекта	6
3. Основы qmake/cmake	12
Источники.....	18

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:



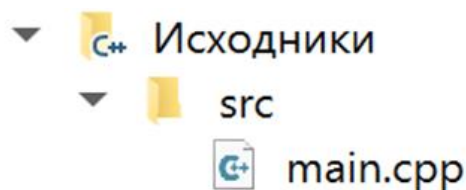
В проекте первый файл с расширением .pro - основной сборочный файл.

Папка Исходники: в ней находятся файлы с расширением .cpp

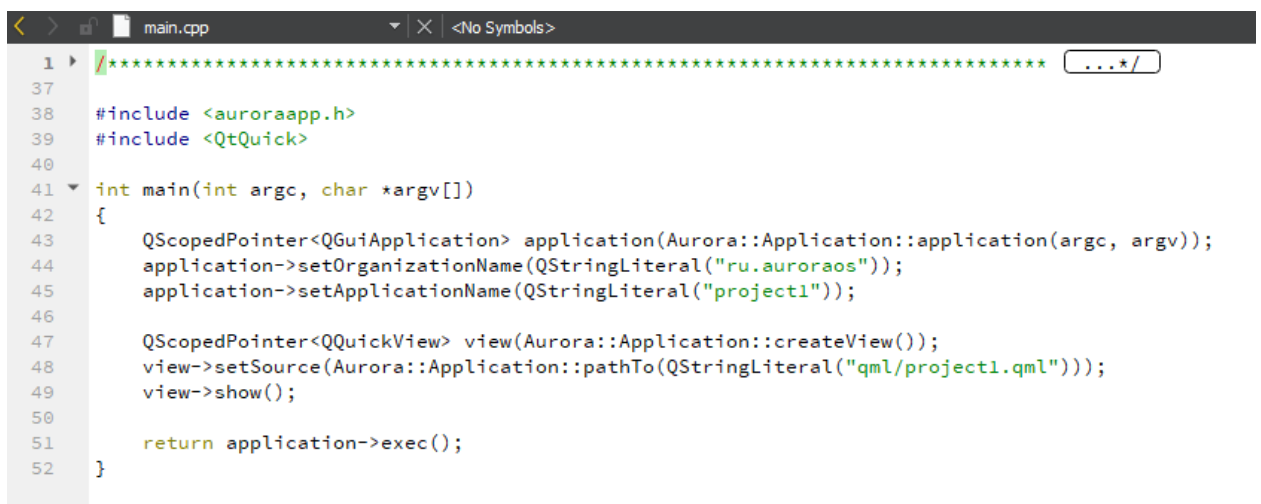
Папка QML: в ней находятся файлы с расширением .qml

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

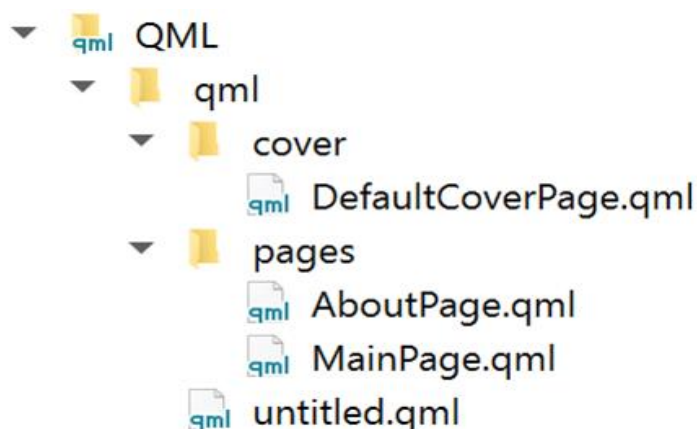
• Каталог Исходники



Здесь находится еще одна папка src, в которой находится файл main.cpp.



- 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**).

project1.qml

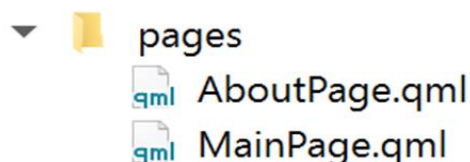
```
import QtQuick 2.0
import Sailfish.Silica 1.0

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

- 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.

Pages



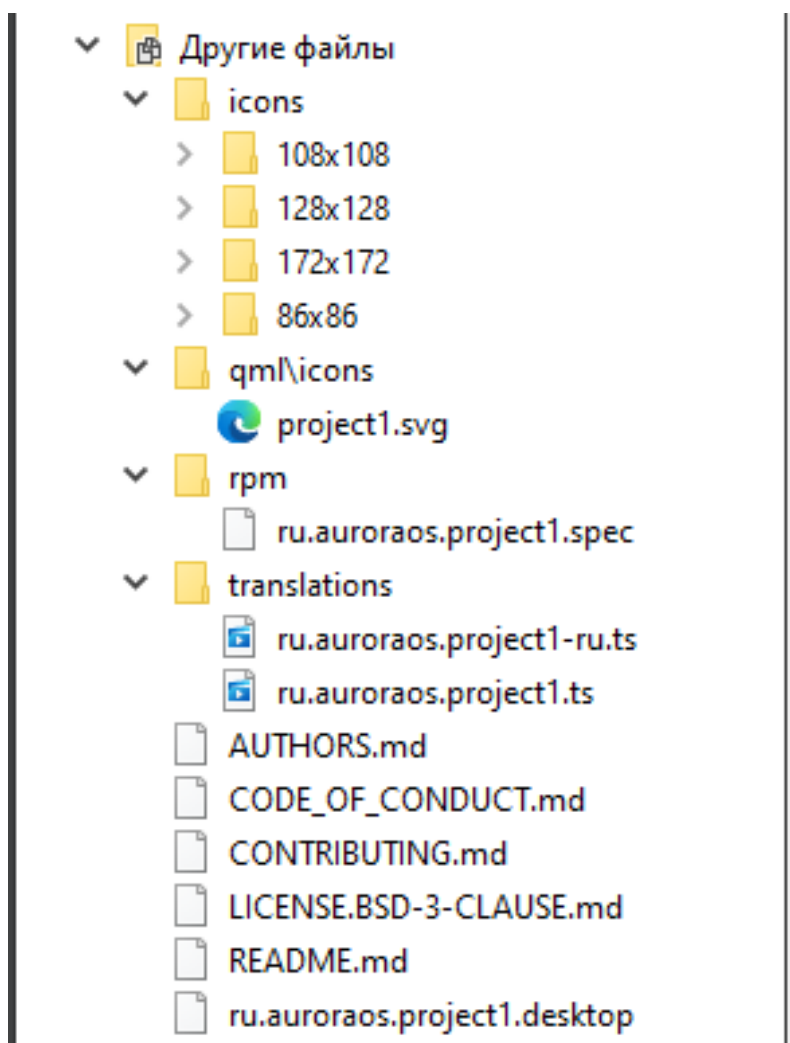
В шаблонном приложении по умолчанию есть файл основной страницы приложения (MainPage.qml) и добавлена еще одна страница - о приложении (AboutPage.qml). В дальнейшем можно добавлять сюда и новые страницы приложения.

Cover



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

• 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.

Также 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 `grm` находится файл с расширением `spec`. Он используется для конфигурирования `grm` 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.

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...».
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.

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.

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.

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:


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»*.

2. На панели слева нажать кнопку  *Опции сборки* и выбрать комплекты и способы 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).


3. После завершения настроек нажать кнопку  *Сборка проекта* для запуска сборки проекта.




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



Запуск приложения

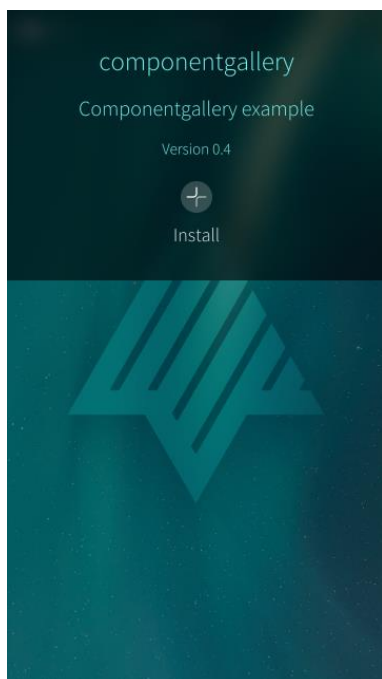
Приложение может быть запущено как на внешнем устройстве, работающем под управлением ОС Аврора, так и в эмуляторе, который устанавливается при инсталляции Аврора SDK.

Для запуска приложения на эмуляторе необходимо выполнить следующее:

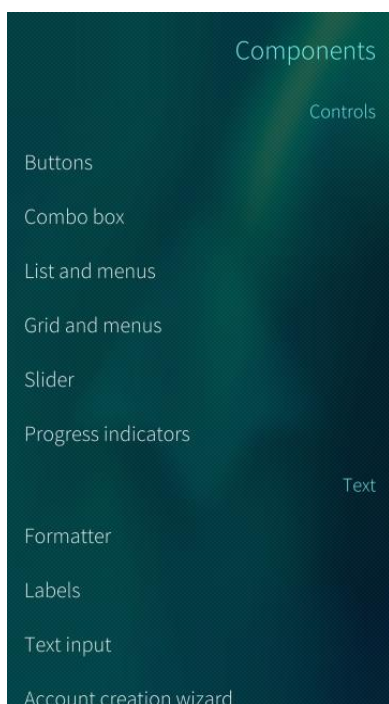
1. На панели слева нажать кнопку  *Опции запуска* и выбрать комплект AuroraOS-i486.

2. Запустить эмулятор нажатием кнопки  *Запуск «Aurora Emulator»* и дождаться, пока она не примет вид  *Остановка «Aurora Emulator»*. Откроется новое окно VirtualBox, и загрузится эмулятор. Если нажать кнопку  *Остановка «Aurora Emulator»*, эмулятор остановится, и окно VirtualBox закроется.

3. Для запуска приложения нажать кнопку  *«Запустить»*, для отладки —  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.



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



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

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.

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.

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.

Основные команды qmake

Рассмотрим основные команды qmake, в частности, те, которые появляются в рго-файле Аврора-проекта по умолчанию

```

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 \

```

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.

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 `рsmaker`, 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. 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.

Источники

1. [Структура Qt-проекта](#)
2. [Запуск и отладка проекта | Портал разработчиков ОС Аврора](#)
(портал разработчиков ОС Аврора)
3. [Qt 5.6](#)
4. Дубров, Д.В. Система построения проектов CMake: учебник / Д. В. Дубров; Южный федеральный университет. — Ростов-на-Дону: Издательство Южного федерального университета, 2015. — 419 с.