АНДРЕЙ БОРОВСКИЙ

bhv®

Qt4.7+ Практическое программирование на C++

- СОЗДАНИЕ КРОССПЛАТФОРМЕННЫХ ПРИЛОЖЕНИЙ ДЛЯ WINDOWS И LINUX
- ПРОГРАММИРОВАНИЕ ГРАФИКИ, БАЗ ДАННЫХ, МНОГОПОТОЧНЫХ ПРИЛОЖЕНИЙ, ВИДЖЕТОВ И ДР.
- РАБОТА С XML
- НОВЫЕ ПОДСИСТЕМЫ Qt DECLARATIVE И Qt3/D
- БОЛЕЕ 40 ДЕМОНСТРАЦИОННЫХ ПРОГРАММ
- ПРОФЕССИОНАЛЬНЫЕ ИНСТРУМЕНТЫ ДЛЯ РАБОТЫ С БИБЛИОТЕКОЙ Qt

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Андрей Боровский

Qt4.7+ Практическое программирование на C++

Санкт-Петербург
«БХВ-Петербург»
2012

УДК 681.3.06
ББК 32.973.26-018.2
Б83

Боровский А. Н.

Б83 Qt4.7+. Практическое программирование на C++. — СПб.: БХВ-Петербург, 2012. — 496 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0757-8

Книга посвящена разработке приложений для Windows и Linux с использованием библиотеки Qt версий 4.7.x и 4.8. Подробно рассмотрено программирование трехмерной и интерактивной графики, баз данных, многопоточных приложений, создание собственных виджетов, описание принципов работы с XML, а также использование новейших подсистем Qt Declarative и Qt3/D. Дано описание классов Qt применительно к решению конкретных задач. Значительное внимание уделено основным принципам разработки сложных приложений. Рассмотрено применение различных профессиональных инструментов разработчика при работе с библиотекой Qt. На авторской странице поддержки книги расположены исходные тексты демонстрационных примеров (более 40).

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Ирина Иноземцева*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*
Зав. производством	*Николай Тверских*

Подписано в печать 31.10.11.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 39,99.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0757-8

Оглавление

Введение

Написание хорошей книги о библиотеке Qt — задача непростая. И не только потому, что написать хорошую книгу вообще непросто. Библиотека Qt давно уже стала популярным средством разработки программ на платформах Windows и Linux, а также для некоторых мобильных устройств. Это значит, что о программировании с использованием Qt уже написаны хорошие книги (некоторые из них выдержали не одно издание). Имена Марка Саммерфилда (Mark Summerfield), Джасмина Бланшетта (Jasmin Blanchette) и Макса Шлее (Max Schlee) должны быть знакомы всем, кто имеет систематический опыт программирования с использованием Qt. Вдобавок к этому библиотека Qt снабжена великолепной документацией с подробным описанием всех классов и множеством примеров. Частично эта документация переведена на русский язык.

Что такое Qt?

Самый простой ответ на этот вопрос выглядит так: Qt — кросс-платформенная библиотека классов C++. Сначала это была библиотека для создания программ с графическим интерфейсом пользователя, которые можно было бы собирать на разных системах без изменения исходных текстов. Со временем библиотека разрослась вширь и вглубь, и теперь, помимо разработки собственно графических интерфейсов, включает в себя сотни классов, охватывающих самые разные аспекты программирования — от разработки приложений баз данных до создания мультимедийных программ и от работы с динамически загружаемыми модулями до конечных автоматов и собственного интерпретатора языков сценариев. И все это работает практически одинаково на таких разных платформах как Microsoft Windows, Linux, Mac OS, многочисленные коммерческие версии UNIX, а также WinCE и Symbian. Библиотека Qt работает на 32- и 64-битных процессорах Intel, процессорах ARM старших моделей и некоторых других. Библиотеку очень легко локализовать, т. е. перевести ее текстовые ресурсы на новый язык, что уже сделано (и постоянно делается) для большинства языков мира, так что после установки в русскоязычной системе многие компоненты Qt сразу же "заговорят" по-русски. Инструментарий, необходимый для локализации приложений, как и многие другие вспомогательные инструменты, входит в состав самой Qt.

Но и это еще не все. Сравнительно недавно библиотека Qt перестала быть только библиотекой классов C++ и вспомогательных элементов и обзавелась собственной кросс-платформенной интегрированной средой разработки Qt Creator (что, однако, не мешает работать с библиотекой Qt в других популярных средах, таких как Microsoft Visual Studio и Eclipse).

Библиотека Qt используется для разработки приложений крупнейшими поставщиками программного продукта. Достаточно назвать такие компании как Nokia (которая в данный момент и владеет библиотекой Qt), Oracle и Google. Кроме того, на основе Qt разрабатывается одна из самых популярных графических оболочек для Linux — KDE, и огромное количество открытых приложений для Linux и других UNIX-систем.

Что вы найдете в этой книге?

В данной книге я старался, насколько это возможно, не повторять то, что уже написано про Qt другими авторами, и не дублировать документацию к библиотеке. Основная цель этой книги — помочь читателю в решении практических задач, в том числе задачи, возникающей перед каждым программистом, пишущим серьезные приложения, — адаптации Qt под свои нужды. Библиотека Qt весьма обширна и охватывает самые разные аспекты разработки программного обеспечения, но ни одна библиотека в мире не содержит в точности те инструменты, которые необходимы именно вам. Особенно, если вы пишете приложение, которое должно быть в чем-то уникальным и непохожим на другие. В этой книге рассмотрены основные вопросы практического программирования на Qt, но в ней также показано, как модифицировать и адаптировать различные компоненты Qt, не теряя таких преимуществ как переносимость приложения между разными платформами и способность модифицированных компонентов взаимодействовать со стандартными компонентами Qt.

Практический характер книги наложил на нее свой отпечаток. Библиотека Qt существует не в вакууме. В книге показано, как Qt взаимодействует с популярными средами разработки и инструментами программирования, как эти инструменты взаимодействуют между собой при использовании Qt. В книге также продемонстрировано, как с помощью Qt реализовать популярные концепции программирования, такие как кросс-платформенность, принцип модульности и паттерны проектирования.

Но и это еще не все. Как программисту с большим стажем работы с Qt, мне приходилось видеть немало ошибок, допущенных менее опытными программистами. На страницах этой книги я стремился предупредить читателя о наиболее распространенных ошибках и помочь ему избежать малоэффективных решений. Для этой цели использованы примеры, представляющие собой фрагменты программ, написанных профессиональными программистами (не только мной). Поскольку Qt широко применяется для разработки программного обеспечения с открытыми исходными текстами, найти исходные тексты надежных, быстрых и эффективных приложений, написанных с помощью Qt программистами высокого уровня, не так уж и трудно.

Следует сказать и о том, чем не является эта книга. Эта книга не является энциклопедией по Qt. Библиотека Qt настолько обширна, что охватить ее целиком в одной книге не представляется возможным. Некоторые аспекты программирования в Qt, например, разработка мультимедийных приложений, остались за рамками этой книги. Книга не является энциклопедией еще и в том смысле, что материал не систематизирован так, как это принято в энциклопедиях (любая попытка систематизировать материал подобным образом неизбежно привела бы к дублированию документации, которая, в любом случае, является самой лучшей энциклопедией по Qt). Из сказанного следует, что книгу лучше читать не выборочно, как справочник, а целиком.

Хотя книга охватывает Qt 4 в целом, особое внимание уделено новинкам последних на момент написания книги версий Qt — 4.7, 4.7.1, 4.7.2, 4.7.3.

Книга состоит из трех частей. Первая часть (*главы 1–4*) является своего рода "расширенным введением" в Qt. Я не стал останавливаться на самых простых понятиях Qt, поскольку они очень хорошо освещены в документации, в том числе на русском языке. Вместо этого я сразу перешел к более углубленному рассмотрению таких концепций как сигналы, слоты и события Qt, механизмы обработки событий, нетривиальные методы работы с Qt Creator и Qt Designer и автоматическое тестирование. Помимо этого первая часть содержит описание методов интеграции Qt с различными популярными инструментами разработки.

Во второй части книги (*главы 5–10*) обсуждается практика разработки серьезных проектов. В *главах 5 и 6* рассматриваются вопросы создания многопоточных приложений Qt. В связи с триумфальным шествием многоядерных процессоров эта тема весьма актуальна, но, на мой взгляд, недостаточно подробно представлена в литературе по Qt. *Глава 7* посвящена разработке приложений баз данных, причем упор сделан на адаптацию существующих компонентов Qt для решения различных нетривиальных задач, которые возникают при работе с базами данных. В *главах 8–10* подробно рассматривается то, для чего Qt изначально была создана, — работа с графикой и отформатированным текстом. Помимо прочего, обсуждается создание простой двухмерной компьютерной игры и программы, которая интерактивно проверяет орфографию, выделяя слова с ошибками, как это делает Microsoft Word, а также способы представления в программах больших изображений (например, географических карт) с минимальными затратами памяти. Кроме того, рассматриваются такие передовые технологии, как использование ресурсов вашей видеокарты для расчетов в приложениях Qt, использование аппаратной поддержки OpenGL для ускорения работы с двухмерной графикой и некоторые другие.

Третья часть (*главы 11–15*) содержит материал, который я классифицировал как дополнительный. Здесь вы узнаете, как работать с конечными автоматами Qt, как задействовать в своих программах систему сценариев Qt, как создавать и использовать динамические модули расширений Qt, а также, как применять новинку Qt — язык QML.

Все главы книги снабжены полнофункциональными примерами программ. Исходные тексты этих примеров вы найдете на моем сайте по адресу **http://symmetrica.net/qt47book/**.

Для кого предназначена эта книга?

С учетом уровня изложения материала и концепции книги от читателя требуется наличие определенных предварительных знаний. Если пробелы в знании Qt можно заполнить с помощью документации непосредственно в процессе чтения книги, то хорошее знание языка программирования C++ требуется с самого начала. Желательно хорошее понимание концепций объектно-ориентированного программирования, принципа модульности и знакомство с паттернами проектирования приложений.

Книга будет полезна тем программистам, которые обладают обширным опытом разработки приложений в таких средах как, например, Borland C++ Builder и Microsoft Visual Studio MFC, и желают перейти на Qt (желание, которое я всячески приветствую), а также тем, кто знаком с предыдущими версиями Qt и хочет обновить свои знания.

Если вы хотите глубже разобраться в таких сложных вопросах как многопоточное программирование, применение языка сценариев для расширения возможностей приложений Qt или использование ресурсов современных высокопроизводительных видеокарт для решения неграфических задач, эта книга тоже может быть вам полезна.

ЧАСТЬ I

Начальный этап разработки проектов

ГЛАВА 1

Краткое введение в Qt 4

Наверняка вы уже знаете, что Qt library — это кросс-платформенная библиотека классов C++ и вспомогательных инструментов, предназначенная для разработки самого широкого круга приложений как для персональных компьютеров, так и для мобильных устройств. Хотя я предполагаю, что вы не только хорошо знаете C++, но и знакомы, по крайней мере, с основами Qt library, в этой главе будет дан краткий обзор Qt 4.7, который окажется особенно полезен тем из вас, кто не имел серьезного опыта разработки на Qt, но пользовался другими средствами разработки, основанными на C++, например Borland C++ Builder или Microsoft MFC. Если же вы вообще незнакомы ни с чем из вышеперечисленного, то вам рекомендуется начать с одной из книг, указанных в *Списке литературы*.

Средства разработки для библиотеки Qt library

Программиста, который только начинает знакомиться с Qt, ожидает некоторое удивление: вместо одного варианта библиотеки ему предлагают несколько коммерческих и открытых вариантов, и не совсем понятно, какой из них лучше выбрать. Возможно, вы не до конца разобрались во всем этом многообразии, даже если у вас уже есть опыт программирования в Qt, тем более что количество вариантов этой библиотеки постоянно меняется. Во время написания этой книги были доступны, как минимум, три среды разработки и два вида лицензирования.

Разработчики Qt распространяют свой продукт на условиях двойного лицензирования, дающего право на бесплатное использование библиотеки в некоммерческих продуктах и более широкую свободу выбора лицензии при покупке коммерческой версии Qt.

Открытый вариант Qt распространяется на условиях GPL 2.0. Сами представители Qt именуют свой подход "Quid Pro Quo", что в вольном переводе с древней латыни означает "баш на баш". Если вы хотите пользоваться средствами Qt бесплатно, взамен вы должны предоставить сообществу свой код (открытая модель). Если вы не желаете делиться кодом, вы должны заплатить деньги (своего рода "выкуп", который, естественно, будет потрачен на дальнейшее совершенствование Qt). Интерес-

но отметить, что лицензионная политика Qt не позволяет лицензиатам применять тот же принцип двойного лицензирования к своим продуктам. Если вы ведете разработку с помощью Qt, ваш проект должен быть либо открытым на условиях GPL, либо коммерческим. То есть вам, конечно, никто не мешает раздавать ваш код бесплатно, но если это делается не на основе GPL, то для разработки этого кода необходимо использовать (и, естественно, оплатить) коммерческую версию Qt.

Коммерческий дистрибутив Qt распространяется в трех вариантах: Qt Console, который позволяет создавать только консольные приложения, Qt Desktop Lite, которая позволяет создавать приложения с графическим интерфейсом, но не включает в себя некоторые важные компоненты, которые присутствуют в Qt Console, например библиотеки для работы с сетью, и Qt Desktop — наиболее полный вариант.

Коммерческая версия Qt поддерживает ОС Windows, Mac OS X, Linux, Embedded Linux, Windows CE. Что касается открытого варианта библиотек, то набор поддерживаемых ОС здесь примерно тот же, но условия работы несколько другие. Некоммерческая версия Qt содержит все те же компоненты, что и наиболее полная коммерческая версия.

Примечание

Во время написания этой книги компания Nokia, которая ведет разработку Qt, фактически перешла под крыло Microsoft. Компания Microsoft интересуется компанией Nokia, прежде всего, как производителем смартфонов, на которые Microsoft планирует устанавливать собственную ОС Windows Phone. При этом создание версии Qt для Windows Phone в планы Nokia не входит, т. к. у Microsoft для этой платформы есть свое проприетарное средство разработки. В результате всего этого будущее Qt для смартфонов и других мобильных устройств выглядит несколько туманным. Что касается Qt для больших компьютеров, то разработчики клянутся, что с ней все будет по-прежнему, и мы им верим.

Рассмотрим теперь различные варианты Qt library с точки зрения средств разработки на платформах Windows и Linux, на которых мы, в основном, и сосредоточим наше внимание.

Qt SDK

Этот дистрибутив содержит все необходимое для того, чтобы сразу приступить к разработке приложений Qt. В него, помимо уже собранных библиотек Qt, входит интегрированная среда разработки Qt Creator (рис. 1.1), полная справочная документация и файлы примеров. Для сборки приложений Qt SDK использует инструментарий GNU. На платформе Linux используется тот инструментарий, который поставляется вместе с Linux, а на платформе Windows — инструментарий MinGW, который входит в дистрибутив.

Примечание

Следует помнить, что объектные файлы, которые генерирует MinGW, несовместимы с объектными файлами, которые создает Microsoft Visual Studio. То же относится и к разделяемым библиотекам, экспортирующим классы C++. Если вы создаете на MinGW разделяемую библиотеку, которую потом можно будет использовать в среде Microsoft Visual Studio, позаботьтесь о том, чтобы все функции экспортировались этой библиотекой в формате C (мы еще скажем об этом позже).

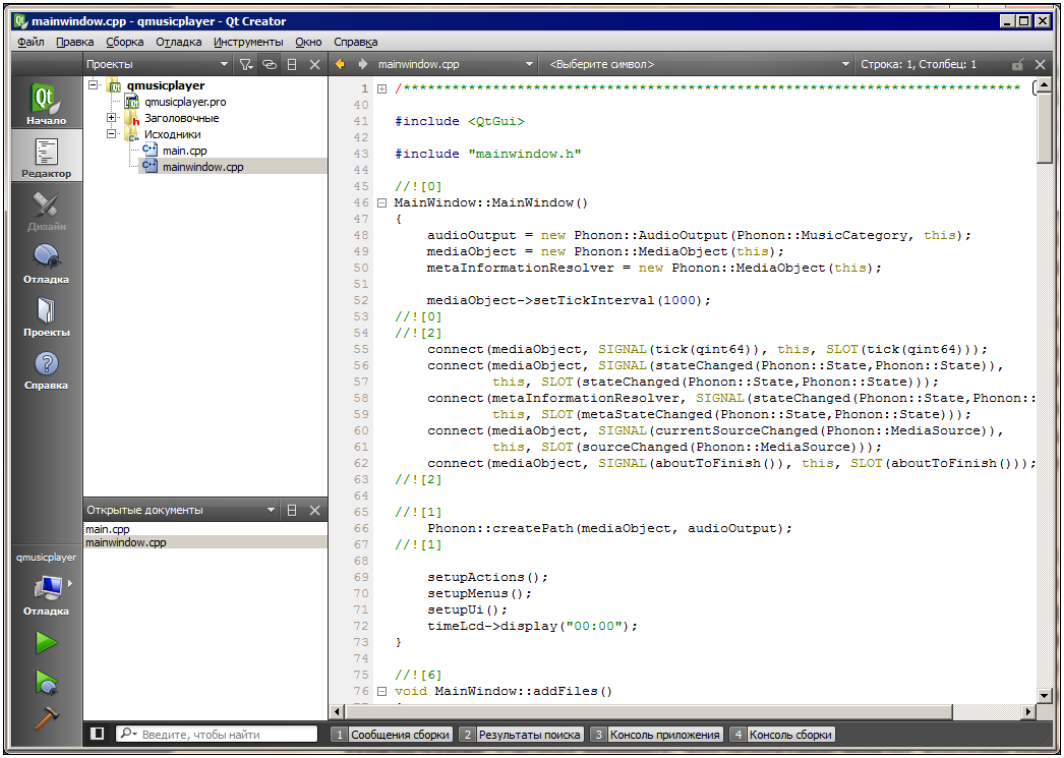

Рис. 1.1. Интегрированная среда Qt Creator

Qt для Microsoft Visual Studio

Существуют библиотеки Qt, специально собранные для Microsoft Visual Studio. Сама по себе среда Visual Studio не приспособлена для работы с Qt (создать заготовку проекта приложения Qt в Visual Studio по умолчанию нельзя). Тут можно пойти двумя путями. Первый путь заключается в том, чтобы для генерации специальных файлов Qt (файлов проектов, ресурсов и т. д.) использовать напрямую соответствующие инструменты Qt (qmake, designer, rcc и т. д.). Затем править исходные тексты, отлаживать и собирать проект в Visual Studio. Можно и вовсе не пользоваться средой Visual Studio, а собирать проект с помощью сопутствующей ей утилиты nmake. Тем более, что файлы, которые генерирует qmake в этом варианте Qt library, предназначены именно для nmake.

Более удобное решение — использовать средство интеграции Qt и Visual Studio, которое позволяет работать с инструментами Qt так, как если бы они были частью интегрированной среды разработки Microsoft (рис. 1.2). Этот инструмент нужно скачать и установить отдельно.

Для компиляции приложения в этом случае используется компилятор Microsoft, который, как считается, генерирует несколько более компактный и быстродействующий код, нежели компилятор GNU. Разумеется, этот вариант доступен только под ОС Windows.

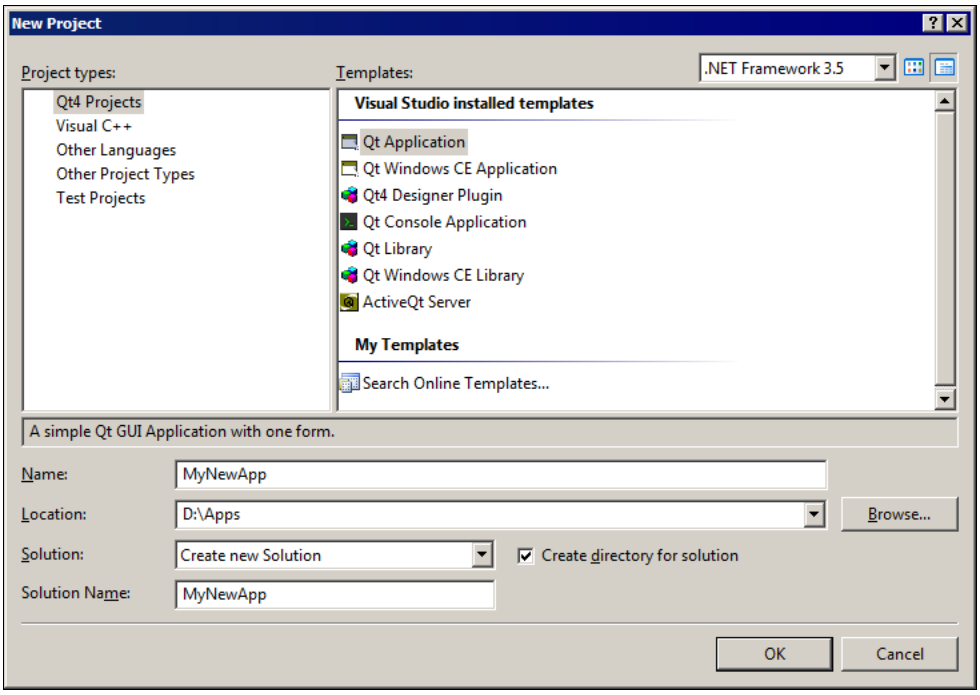

Рис. 1.2. Интеграция Qt и Microsoft Visual Studio (окно выбора типа проекта)

Примечание

При работе с Qt для Microsoft Visual Studio необходимо, чтобы переменные среды окружения были настроены на работу с Qt. Неправильная настройка переменных среды окружения может стать источником неочевидных ошибок в процессе сборки приложения. При установке пакета создаются два ярлыка. Один открывает окно командной строки Qt Command Prompt с уже настроенными переменными среды, второй запускает Microsoft Visual Studio с теми же настройками.

Qt Framework

Этот вариант не содержит никаких компиляторов и интегрированных сред. В него входят только сами библиотеки Qt, вспомогательные инструменты Qt, примеры и документация. Этот вариант Qt можно скачать не только в виде собранных библиотек для различных платформ, но и в виде исходных текстов для самостоятельной сборки (причем собирается все, включая такие программы как qmake). Вариант со сборкой из исходных текстов лучше выбрать тогда, когда вы собираетесь использовать Qt в такой конфигурации, для которой официальной сборки не существует. Кроме того, новейшие релизы Qt зачастую доступны только в таком варианте. Обычно номер доступной версии Qt Framework опережает номер версии Qt в доступной версии Qt SDK. Тем не менее Qt SDK можно настроить на работу с новейшей версией Qt Framework. Чтобы настроить Qt Creator на работу с версией Qt, отличной от той, что входит в состав SDK, необходимо перейти в группу **Проекты** Qt Creator и открыть вкладку **Настройки сборки** (рис. 1.3). Грамотный выбор пара-

метров на этой вкладке позволит настроить Qt Creator на работу с любым вариантом библиотек Qt, который поддерживает ваша платформа, и с любым компилятором. Но подробное описание этого процесса выходит за рамки данной книги. Лично я предпочитаю сочетание Qt Creator, компилятора GNU и новейшей версии Qt Framework. Кстати, если вам нужен только Qt Creator, не обязательно скачивать весь SDK.

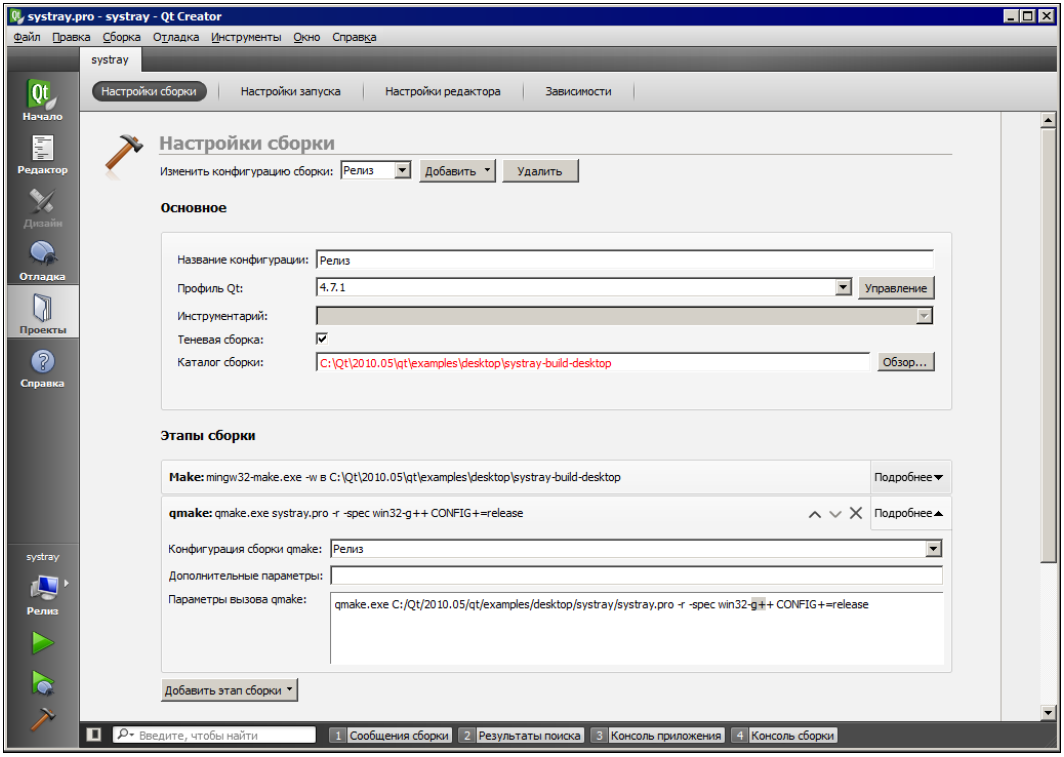

Рис. 1.3. Настройка Qt Creator для работы с Qt Framework другой версии

Qt и Eclipse

Если для разработки вы выбрали версию Qt, использующую инструментарий GNU, но не хотите пользоваться Qt Creator, имеет смысл установить среду интегрированной разработки Eclipse. Для того чтобы среда Eclipse могла работать с C++ (изначально она ориентирована на Java), необходимо установить пакет расширений Eclipse CDT и пакет интеграции Qt и Eclipse (его можно скачать с сайта Qt). Установка пакета интеграции Qt и Eclipse выполняется очень просто: все, что вам придется сделать, — это распаковать архив пакета в ту же директорию, которая содержит директорию Eclipse. Уровень интеграции Qt и Eclipse — примерно такой же, как и уровень интеграции Qt с Microsoft Visual Studio (рис. 1.4). Перед сборкой первого приложения Qt в Eclipse вам потребуется указать версию библиотеки Qt и директорию, в которой расположены такие инструменты Qt как qmake, moc и т. д.

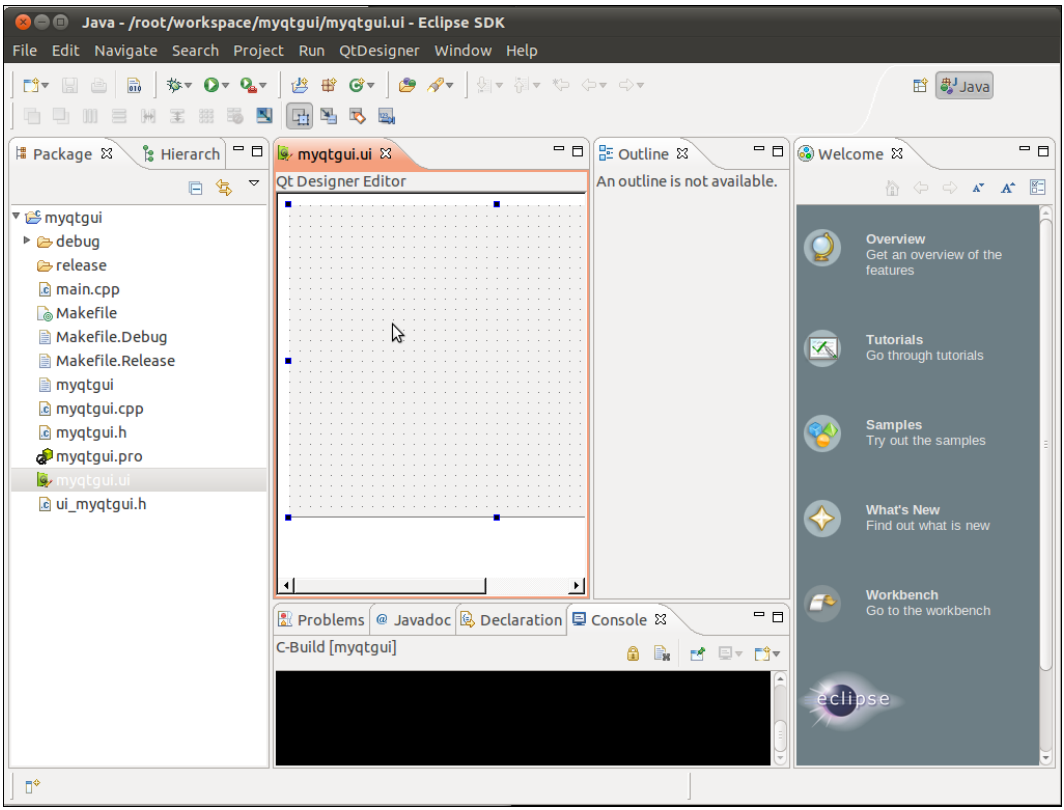

Рис. 1.4. Интеграция Qt и Eclipse

Примечание

Пусть вас не смущает, что пакет интеграции Qt и Eclipse собран не с той версией Qt, которую вы собираетесь использовать. По моим наблюдениям, это никак не влияет на работу с Qt. Например, в моей системе пакет интеграции Qt и Eclipse собран с Qt 4.6.2, а для сборки приложений с его помощью я использую новейшую на данный момент версию Qt 4.7.2. Впрочем, вы так же можете скачать исходные тексты пакета интеграции и собрать его самостоятельно под свою версию Qt.

Qt и CMake

Традиционно программисты, работающие в Linux (и других UNIX-системах), используют для сборки программ инструментарий GNU build system (состоящий из утилит autoconf, automake, libtool, gnulib). Именно с помощью GNU build system создаются знакомые всем нам файлы configure, с помощью которых мы генерируем make-файлы для сборки приложения. Система GNU build system настолько тесно связана с историей Linux и других открытых проектов, что своеобразное заклинание "configure–make–make install" рассматривается некоторыми UNIX-программистами как единственно верный способ установки ПО. К недостаткам GNU build system можно отнести ее ориентацию на инструментарий разработчика GNU (GNU

make, GCC и т. п.), который очень популярен на открытых UNIX-платформах, но мало распространен в других средах.

Стремясь заполнить этот пробел, разработчики из компании Kitware создали свой собственный вариант кросс-платформенного генератора сборочных файлов — CMake (cross-platform make).

Каковы же достоинства CMake? Прежде всего, CMake является по-настоящему кросс-платформенным генератором проектов, позволяющим создавать единые описания проектов для Linux и других UNIX-систем (включая Mac OS X) и Windows. Остановимся на этом подробнее. Важное отличие Windows от Linux (с точки зрения разработчика) заключается в том, что на платформе Windows нет единого стандарта сборочных файлов. MS Visual Studio использует собственные файлы проектов, у C++ Builder они другие, а у MinGW — третьи. Преимущество CMake в том, что эта система способна генерировать "родные" сборочные файлы для всех перечисленных средств разработки (как и для многих других). Кроме того, CMake стремится максимально использовать собственные средства генерации сборочных файлов — например, для генерации сборочных файлов проекта Qt используется qmake. И вообще, разработчики CMake уделили Qt много внимания.

Помимо прочего, CMake обладает интеллектуальной системой поиска инструментов сборки и библиотек на конкретной платформе (интроспекцией) и автоматического конфигурирования. Благодаря этому система CMake сама устанавливает многие параметры сборочных файлов, которые в других системах управления сборкой приходится устанавливать вручную.

> ***Примечание***
>
> Для тех, кто совсем незнаком с CMake, дам некоторые пояснения. Схематично работу CMake можно описать следующим образом: для сборки приложения создается файл CMakeLists.txt, в котором описываются параметры сборки (расположение файлов исходных текстов, требуемые внешние модули, цели сборки). Далее этот файл передается утилите cmake. Результатом работы cmake является файл, содержащий инструкции сборки приложения для конкретной платформы (make-файл GNU make, файл проекта Visual Studio и т. д.). Суть идеи заключается в том, что описание процесса сборки в файле CMakeLists.txt абстрагировано как от конкретных особенностей отдельных систем (расположение файлов, возможности компиляторов), так и целых платформ. Читая общее описание процесса сборки из файла CMakeLists.txt, программа cmake создает файл инструкций сборки, учитывающий специфику конкретной системы.

CMake — консольная утилита (для которой, правда, есть графическая оболочка). Если проект Qt поставляется с файлом CMakeLists.txt, то перед сборкой этого проекта необходимо перейти в его директорию и выдать команду:

```
Cmake ./
```

В результате будет создан файл сборки проекта, соответствующий вашей платформе и используемому средству разработки. Дальше, в зависимости от того, какой именно файл был сгенерирован, нужно либо открыть его в интегрированной среде разработки, либо запустить утилиту make или nmake.

Qt Creator умеет работать с проектами, использующими в процессе сборки CMake. Если проект Qt поставляется с файлом CMakeLists.txt, то всю информацию о входящих в проект файлах, параметрах их сборки и компоновки Qt Creator экспортирует из файла CMakeLists.txt. Вы можете редактировать файл CMakeLists.txt в окне редактора Qt Creator, при этом, когда вы захотите сохранить измененный файл, Qt Creator предложит вам выполнить утилиту cmake. Соглашайтесь. Во-первых, это поможет вам сразу убедиться, что отредактированный файл CMakeLists.txt не содержит ошибок (или устранить обнаруженные), а во-вторых, в результате выполнения cmake будут созданы файлы проекта и make-файлы, на основе которых Qt Creator и выполнит сборку приложения в соответствии с вашим замыслом. Microsoft Visual Studio игнорирует файлы CMake, зато CMake не игнорирует Visual Studio и всегда готов сгенерировать файл проекта для этой среды разработки.

Что касается Eclipse, то у него есть даже специальный пакет расширений для работы с CMake, и эта среда интегрируется с CMake так же хорошо, как и Qt Creator.

Если вы намереваетесь распространять свое приложение в виде исходных текстов, которые пользователь должен скомпилировать самостоятельно, использование CMake в проекте Qt является, на мой взгляд, наилучшим выбором.

Общая структура приложений Qt

Для того чтобы понять, что делают различные инструменты Qt, необходимо представлять себе общую структуру приложения Qt. Как и все программы на C++, приложение Qt должно содержать главную функцию `main()`. Обычно функция `main()` приложения Qt делает немного: создает объект класса `QApplication`, создает объекты, управляющие локализацией приложения, создает главный визуальный элемент приложения (обычно это главное окно программы) и запускает цикл обработки сообщений.

Вот, например, типичная функция `main()` настоящего, а не учебного приложения Qt (листинг 1.1).

Листинг 1.1. Функция `main()` типичной программы Qt

```
#include <QApplication>
#include <QTranslator>
#include <QLocale>
#include "mainwindow.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QTranslator translator;
        if (!translator.load("yagf_" + QLocale::system().name(),
                "/usr/local/share/yagf/translations"))
            translator.load("yagf_" + QLocale::system().name(),
                "/usr/share/yagf/translations");
```

```
    app.installTranslator(&translator);
        QTranslator translator2;
        translator2.load("qt_" + QLocale::system().name(),
            "/usr/share/qt4/translations");
        app.installTranslator(&translator2);
    MainWindow mainWindow;
    Window.show();
    return app.exec();
}
```

Класс `QApplication` играет центральную роль в работе приложения Qt. У класса `QApplication` несколько конструкторов, мы выбираем из них конструктор с двумя параметрами, аналогичными параметрам функции `main`. Получив переменные `argc` и `argv`, конструктор извлекает из них ключи, которые относятся к конфигурации приложения на данной платформе. Например, если приложение создано для платформы Linux/X Window, то в качестве дополнительных ключей ему могут быть переданы дисплей для вывода и геометрия окна. Таким образом, используя объект класса `QApplication`, мы "бесплатно" получаем приложение, которое ведет себя в соответствии со стандартами X Window. Далее следует блок, связанный с локализацией приложения. *См. разд. "Инструменты интернационализации и локализации"* далее в этой главе.

Затем мы создаем объект `mainWindow` класса `MainWindow`. Это и есть главное окно нашей программы. Соответствующий класс объявлен в файле mainwindow.h, а определен в файле mainwindow.cpp. Таким образом, данное приложение включает в себя, как минимум, три файла: main.cpp, в котором реализована функция `main()`, mainwindow.h и mainwindow.cpp, в которых объявлен и реализован класс `MainWindow`. На самом деле, в проекте этого приложения гораздо больше файлов, но перечисленное выше составляет необходимый минимум. Обратите внимание, `mainWindow` объявлена как локальная переменная функции `main()`. Это логично, поскольку выход из функции `main()` в любом случае приведет к завершению программы.

Последнее, что мы делаем в нашей программе, — запускаем цикл обработки сообщений с помощью метода `exec()` объекта `app`. Цикл будет выполняться до тех пор, пока приложение не получит команду завершить работу. Цикл обработки сообщений запускается с помощью метода `exec()` объекта `app` класса `Application`. Когда приложение завершится, метод `exec()` вернет значение, которое будет передано оператору `return` функции `main()`. Нетрудно догадаться, что это значение является кодом завершения программы.

Инструменты Qt

В этом разделе мы перечислим те инструменты, которые распространяются со всеми дистрибутивами Qt и без которых нельзя серьезно заниматься программированием с помощью Qt.

Утилита qmake

Если говорить коротко, утилита qmake предназначена для генерации файлов проектов и файлов, управляющих сборкой приложений. Допустим, что у вас в некоторой директории есть набор исходных файлов программы Qt (файлы с расширениями cpp, h, ui, qrc и т. д.). Важно, чтобы это действительно были файлы программы Qt, а не чего-то другого. Перейдите в эту директорию и выдайте команду:

```
qmake –project
```

В результате в директории появится файл с расширением pro. Это заготовка файла проекта вашего приложения, автоматически созданная утилитой qmake. Если вы откроете этот файл, то увидите в нем набор переменных, которым присваиваются некие значения. Как минимум, файл проекта содержит переменные `CONFIG`, `HEADERS`, `SOURCES`, `FORMS` и `TARGET`. Переменная `CONFIG` содержит информацию, необходимую для системы сборки, ее значениями могут быть наименования конфигураций сборки: `debug`, `release` и др. Переменные `HEADERS` и `SOURCES` содержат, соответственно, перечни заголовочных файлов приложения и файлов с расширением cpp. Переменная `FORMS` содержит перечень файлов с расширением ui (подробнее об этих файлах будет рассказано в разделе, посвященном Qt Designer). Переменная `TARGET` определяет, что именно является результатом сборки приложения. Файл проекта, созданный программой qmake, почти всегда включает в себя и другие переменные и даже более сложные конструкции. Рассмотрим типичный файл проекта, созданный программой qmake с нуля (листинг 1.2).

Листинг 1.2. Пример файла проекта

```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
HEADERS += mainwindow.h
FORMS += mainwindow.ui
SOURCES += main.cpp mainwindow.cpp
```

Откуда qmake "узнал", что исходные тексты программы содержатся в файлах main.cpp и mainwindow.cpp, что описание внешнего вида главного окна содержится в файле mainwindow.ui, а объявление класса — в файле mainwindow.h? Никакой магии здесь нет. Программа просто рассортировала файлы, которые она нашла в той директории, где была запущена. Сортировка выполнялась по именам файлов, а не по их содержимому, так что qmake могла и ошибиться.

Если теперь мы просто выдадим команду:

```
qmake
```

то в нашей директории появятся один или несколько make-файлов, с помощью которых уже можно собрать приложение. Какие это будут make-файлы, зависит от

конфигурации Qt, которую мы используем. Если наши библиотеки ориентированы на Microsoft Visual Studio, то это будут файлы для утилиты nmake. Если наша версия Qt использует инструментарий GNU, то это будут make-файлы для GNU make.

Утилита qmake умеет создавать не только make-файлы. Если при работе с версией Qt, предназначенной для Microsoft Visual Studio, мы выдадим команду:

```
qmake -tp vc -o untitled.vcproj
```

то результатом станет файл проекта Microsoft Visual Studio.

Будет ли сборка приложения успешной, если мы выполним созданные таким образом make-файлы или попробуем собрать проект Microsoft Visual Studio? В случае простейшего приложения, скорее всего, да. В случае более сложном, скорее всего, нет. Дело в том, что, как отмечалось ранее, утилита qmake создает заготовку проекта приложения по формальным признакам. В случае сложного приложения эта заготовка, скорее всего, будет включать в себя далеко не все, что нужно приложению. Соответственно и созданные с помощью qmake файлы, управляющие сборкой, будут неполноценными. Интегрированные среды разработки с подключенными модулями интеграции Qt стараются помочь вам сформировать правильный файл проекта путем подсказок. Например, вы можете выбрать, какие именно подсистемы Qt (OpenGL, сетевую, связи с базами данных) будет использовать ваше приложение. Все эти сведения попадут в файл проекта, который интегрированная среда разработки создаст с помощью той же утилиты qmake. Однако и этого может оказаться недостаточно, и тогда файл проекта придется редактировать вручную.

Не забывайте, что вызов

```
qmake –project
```

всегда создает новый файл проекта (и затирает все, что вы внесли туда вручную), тогда как вызовы qmake без ключа `-project` не меняют содержимое файла с расширением pro. Они, как правило, генерируют файлы, непосредственно управляющие сборкой приложения в данной системе.

Если вы собираетесь выполнять сборку приложения на нескольких разных платформах (или с использованием разных компиляторов на одной платформе), вы можете создать один кросс-платформенный файл с расширением pro и использовать его на всех целевых платформах. Правда, при создании такого файла вам не удастся избежать его ручного редактирования (и потребуется хорошее знание синтаксиса файлов с расширением pro). В дальнейших примерах к этой книге мы будем использовать кросс-платформенные файлы проектов, но подробные инструкции по их составлению опустим. Все это хорошо описано в документации по Qt, доступной, в том числе, и на русском языке. Вы можете спросить, зачем нам нужны специальные кросс-платформенные файлы проектов, если сама библиотека Qt уже является кросс-платформенной. Вопрос справедливый. Для многих приложений, в том числе довольно сложных, файл с расширением pro будет выглядеть одинаково на всех платформах именно благодаря кросс-платформенности Qt. Однако в некоторых случаях в файлах проектов приходится описывать механизмы взаимодействия приложения с системой, которые зависят от системы, а не от Qt. И тогда в файле про-

екта приходится принимать специальные меры для обеспечения кросс-платформенности.

После всего этого у читателя может возникнуть вопрос, зачем нам описанная выше система CMake, если у нас есть qmake. Ответ прост. Хотя утилиты cmake и qmake очень похожи (и та, и другая используют файлы мета-проектов для генерации инструкций для конкретной системы сборки), qmake более ориентирована на разработчиков конкретного приложения (а не на пользователей, которым придется собирать это приложение из исходных текстов). Вспомните, что неправильный вызов qmake может привести к потере всего содержимого файла с расширением pro. Кроме того, CMake — универсальная система управления сборкой, которая используется не только для Qt. У сборщика вашей программы может не быть опыта работы с qmake, но наверняка есть опыт работы с CMake.

Если же вы не собираетесь распространять исходные тексты программы, то система CMake вам, скорее всего, не нужна. Впрочем, не исключено, что нужна, ведь она, помимо прочего, является прекрасным средством обеспечения кросс-платформенности вашего проекта.

Инструменты интернационализации и локализации

Когда величайшие мудрецы человечества учили свои народы грамоте, они, к сожалению, не предвидели, что в далеком будущем текст будут вводить с клавиатуры. Мало того, что в одних языках пишут слева направо, а в других — справа налево, начертание символов в некоторых языках (например, арабском) зависит от соседних символов. Все это нужно учитывать, если вы хотите выполнить грамотную интернационализацию своей программы. Для демонстрации возможностей интернационализации мы напишем максимально простую программу (листинг 1.3).

Листинг 1.3. Программа, подготовленная для интернационализации

```
#include <QApplication>
#include <QPushButton>
#include <QTranslator>
#include <QLocale>
int main(int argc, char *argv[])
{
QApplication I18nApp(argc, argv);
    QTranslator Translator(0);
    Translator.load(QString(QApplication::applicationDirPath() +
        QString("/I18nApp_") + QLocale::system().name()) , ".");
    I18nApp.installTranslator(&Translator);
    QPushButton Button(QObject::tr("Push me!"), NULL);
    Button.show();
    return I18nApp.exec();
}
```

Исходные тексты этой программы вы найдете в папке Ch1/I18n на сайте **http://symmetrica.net/qt47book/**). Пояснять все, что происходит в этой программе, я не буду. Если вы хотя бы немного знакомы с Qt и читали предыдущие разделы, вы все поймете. Программа выводит на экран кнопку с надписью "Push me!" ("Нажми меня!"). Наша задача — сделать так, чтобы текст интерфейса программы (состоящий из одной строчки!) можно было перевести на любой язык, не вмешиваясь в исходные тексты самой программы.

Откройте файл I18nApp.pro (листинг 1.4).

Листинг 1.4. Файл I18nApp.pro

```
SOURCES += \
    main.cpp
TRANSLATIONS += I18nApp_ru.ts
```

Этот файл проекта даже проще, чем файл проекта, рассмотренный нами ранее. Но в нем появилась переменная, с которой мы раньше не встречались. Переменная `TRANSLATIONS` содержит перечень имен файлов, в которых находятся исходные тексты для перевода интерфейса программы. Собственно говоря, расширение файла ts как раз и является сокращением от translation source. Имена файлов, содержащих исходные тексты перевода, могут быть любыми, но далее мы убедимся, что очень удобно, когда они содержат в качестве суффикса или префикса название соответствующей локали (ru — для русской, fr — французской, de — немецкой и т. д.). Если вы захотите перевести интерфейс программы и на другие языки, вам понадобятся дополнительные файлы с именами, соответствующими этой же схеме.

Файла I18nApp_ru.ts в директории I18nApp нет. Вы должны создать его сами, это несложно. Просто в окне консоли выдайте команду:

```
lupdate I18nApp.pro
```

При правильном выполнении команды в окне консоли вы увидите примерно следующее:

```
Updating 'I18nApp_ru.ts'...
    Found 1 source text(s) (1 new and 0 already existing)
```

Утилита lupdate является частью инструментария Qt (поэтому приведенная выше команда сработает только в том случае, если среде окружения известно, где она находится). В рассматриваемом примере программа lupdate делает следующее: из файла I18nApp.pro она извлекает имена файлов исходных текстов (в нашем примере — файл main.cpp), а из этих файлов извлекает все текстовые строки, помеченные особым образом. Далее, из того же файла I18nApp.pro извлекает имена файлов исходных текстов перевода (в нашем примере — файл I18nApp_ru.ts) и записывает в них строки, которые нужно перевести.

Теперь, когда у нас появились заготовки файлов перевода, мы можем приступить к самому переводу. Для этой цели нам понадобится графическая утилита Qt Linguist (рис. 1.5). Интерфейс этой утилиты очевиден. Она показывает нам строку кода, из

которой был извлечен текст для перевода, сам текст и статус перевода. Запустив Qt Linguist, мы последовательно открываем созданные нами файлы с расширением ts, выбираем в левом окне объект создаваемой программы, предназначенный для перевода. При этом в главном окне утилиты отображаются все связанные с объектом строковые ресурсы. Обратите внимание, что Qt Linguist "знает", что мы переводим именно на русский. Все дело в суффиксе ru, который мы добавили к имени файла, содержащего исходный текст перевода. Переведя очередную фразу, необходимо пометить перевод как законченный. При этом слева от исходной фразы появится зеленая галочка.

Между прочим, файлы с расширением ts — это текстовые файлы в формате XML, так что если вы разберетесь в их структуре, то сможете добавлять переводы в

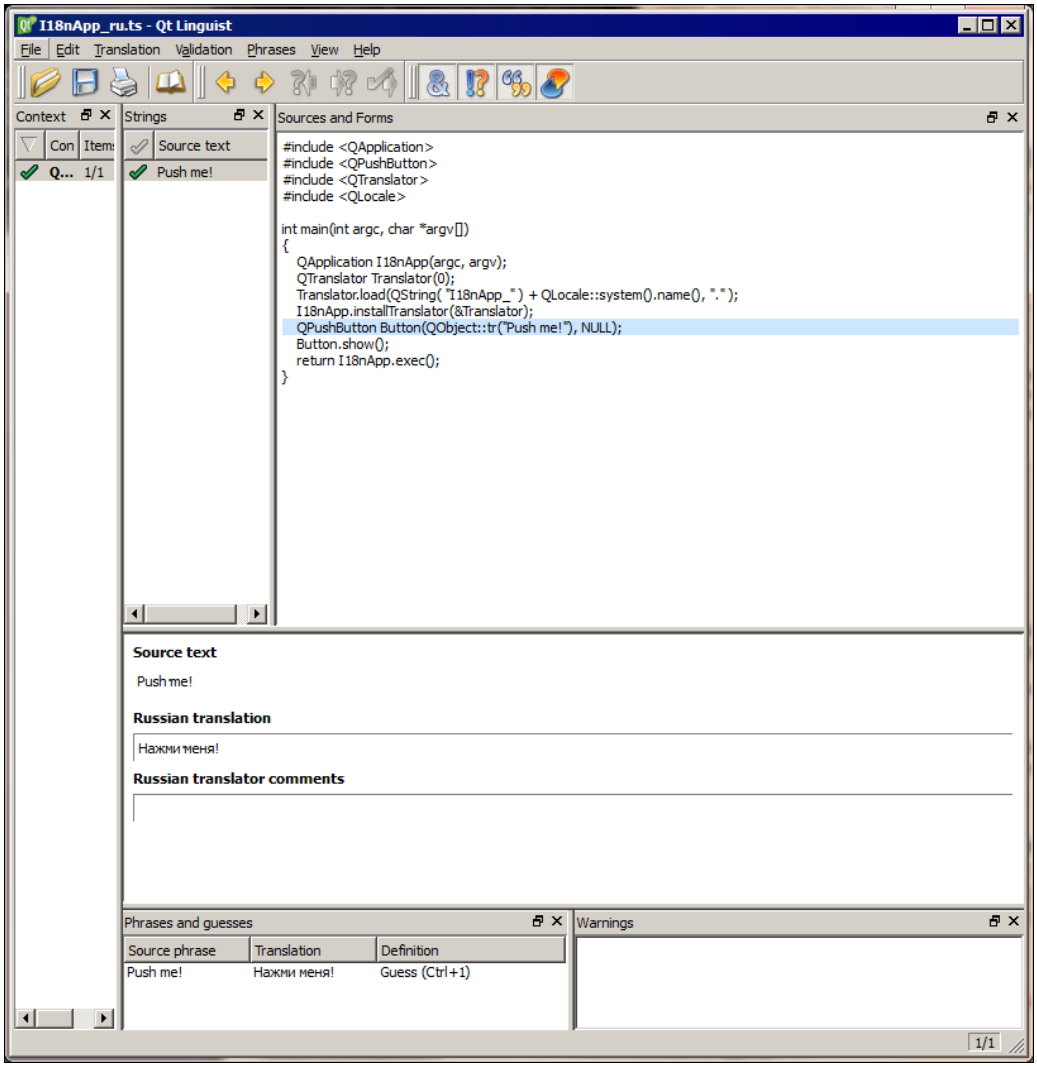

Рис. 1.5. Qt Linguist

обычном текстовом редакторе, не пользуясь Qt Linguist, хотя, на мой взгляд, это неудобно.

Итак, у нас есть файл, содержащий перевод интерфейса нашей программы на русский язык, но работа на этом далеко не закончена. Созданные файлы можно рассматривать как исходные тексты файлов ресурсов перевода. Их еще нужно скомпилировать с помощью утилиты lrelease. В окне консоли выдадим команду:

```
lrelease I18nApp.pro
```

В результате появится файл I18nApp_ru.qm. Это и есть готовый ресурс для перевода интерфейса программы на русский язык. Скопируйте его в ту же директорию, где находится исполнимый файл программы (если его там еще нет). Если вы теперь запустите исполнимый файл программы, то увидите, что она "заговорила" по-русски (рис. 1.6).

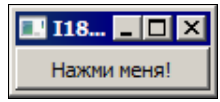

Рис. 1.6. Русифицированная программа

Примечание

Теперь у меня есть еще один довод в пользу применения CMake. Посмотрите, сколько инструментов нам пришлось использовать для того, чтобы собрать файлы перевода в готовый ресурс перевода. CMake бы сделал все это автоматически.

Теперь вернемся к исходному тексту нашей программы. Класс `QTranslator` загружает файлы ресурсов интернационализации во время выполнения программы и управляет переводом интерфейса на соответствующий язык. Как определить, какой файл ресурсов должен быть загружен? Метод `QLocale::system().name()` возвращает имя текущей локали. Мы добавляем это имя в качестве суффикса к имени файла ресурса. Здесь возникает одна интересная ситуация. Метод `QLocale::system().name()` возвращает строку наподобие `"ru_RU"` (`"язык_страна"`), так что сгенерированное нами имя файла будет выглядеть как I18nApp_ru_RU. Наш модуль ресурсов локализации называется I18nApp_ru.qm. Как же объекту класса `QTranslator` удается загрузить правильный файл? Дело в том, что этот объект учитывает, что переданный ему шаблон имени файла — всего лишь подсказка. Сначала он пытается найти файл с точно таким именем и расширением qm. Затем, если такой файл не найден, он отрезает расширение qm и снова пытается найти файл. В случае неудачи снова добавляет расширение qm, но "откусывает" от имени шаблона суффикс. В качестве разделителя суффиксов по умолчанию используется символ "_". То есть объект отрежет суффикс `"_RU"` и с третьей попытки загрузит "правильный" файл I18nApp_ru.qm. Обратите внимание также на то, что Qt не делает разницы между прямым и обратным слэшами при разделении директорий, так что их можно даже смешивать в одной ссылке на файл.

Еще интереснее обстоит дело с директориями. В нашем примере файл ресурса локализации находится в той же директории, что и исполнимый файл программы. Это

логично, но невсегда соответствует правилам. Например, в UNIX-системах файлы программ и файлы ресурсов, как правило, хранятся раздельно. Например, исполнимый файл приложения может храниться в директории /usr/bin/ (это только один из возможных вариантов), а файлы ресурсов перевода — в директории /usr/shre/*имя_приложения*/translations/.

> ***Примечание***
>
> Для UNIX-систем существует также спецификация XDG Base Directory Specification, которая определяет, в каких именно директориях следует хранить различные файлы приложения. Если вы планируете разрабатывать приложения для Unix-систем, я рекомендую вам ознакомиться с этим документом (он доступен в Интернете).

В Windows традиционным местом хранения исполнимого файла программы и файлов ресурсов является директория C:\Program Files*имя_приложения*\, причем внутри этой директории приложение может создавать любые иерархии. Папка C:\Program Files*имя_приложения*\ не подходит для хранения локальных ресурсов отдельных пользователей системы, т. к. не у всех пользователей может быть разрешение на запись в эту папку (в грамотно настроенной системе так и должно быть). Мы подробнее поговорим об этом позже.

Вернемся к нашему коду. Метод `installTranslator()` класса `QApplication` устанавливает в приложении объект, выполняющий перевод интерфейса. Теперь для перевода любой строки, содержащей эквивалент в файлах ресурсов, можно воспользоваться статическим методом `tr()` класса `QObject`. Метод `tr()` указывает, что переданная ему строка должна быть переведена. Этот же метод ищет утилита lupdate, когда извлекает из файла исходных текстов строки для перевода. У метода `tr()` есть аналог — метод `trUtf8()`, он делает то же, что и `tr()`, только предполагает, что текст передан ему в кодировке UTF-8. Каким методом пользоваться? Это, прежде всего, зависит от того, в какой кодировке вы пишете исходные тексты программы. Ведь исходная строка для перевода берется именно из файла исходного текста. Под Windows вы, скорее всего, будете использовать локальную восьмибитную кодировку, и для нее подойдет метод `tr()`. Под Linux сейчас практически везде используется кодировка UTF-8, и здесь понадобится метод `trUtf8()`. Вы так же можете явным образом определить, в какой кодировке представлены строки, предназначенные для перевода. Это можно сделать в файле проекта вашего приложения с помощью переменной `CODECFORTR`, например:

```
CODECFORTR = UTF-8
```

или программно, в функции `main()`, с помощью класса `QTextCodec` (листинг 1.5).

Листинг 1.5. Файл I18nApp.pro

```
QTextCodec::setCodecForLocale(QTextCodec::codecForName("UTF-8"));
QTextCodec::setCodecForTr(QTextCodec::codecForName("UTF-8"));
QTextCodec::setCodecForCStrings(TextCodec::codecForName("UTF-8"));
```

В этом листинге мы указали, что для всех строк нашей программы используется кодировка UTF-8.

Разделение контекстов

Иногда одно и то же английское слово необходимо перевести по-разному. Например, слово build может, в зависимости от контекста, означать либо "собрать", либо "сборка", либо еще что-нибудь. Система перевода Qt предусматривает различные переводы одного и того же слова для разных контекстов. Например, вы можете указать:

```
fileMenu = menuBar()->addMenu(tr("&Build"), "action");
```

Здесь строка `"action"` указывает контекст для перевода. Указание контекста попадет в файл в расширением ts и будет видно вам в программе Qt Linguist.

Склонение существительных, следующих после числительных

Думаю, при работе за компьютером всем приходилось видеть сообщения типа "сохранено 23 записей". Как писать такие строки правильно? Я имею в виду, как писать такие строки так, чтобы при переводе на другие языки имена существительные, следующие за числительными или цифрами, склонялись так, как этого требуют правила соответствующего языка? Можно, конечно, прибегнуть к сухому складскому стилю: "сохраненных записей: 23" Но можно научить программу говорить и более человеческим языком.

Для примера добавьте в функцию `main()`, перед вызовом метода `exec()`, такие строки:

```
int n =3;
QMessageBox::information(0, QObject::tr("info"),
    QObject::tr("%n plane(s) in the sky", "", n));
```

И не забудьте включить в файл main.cpp заголовочный файл `<QMessageBox>`.

Далее вызовите утилиту lupdate, как было описано ранее. Запустите Qt Linguist. Теперь поле для перевода выглядит иначе, чем в предыдущем случае (рис. 1.7). Когда утилите Qt Linguist попадается для перевода конструкция `tr("%n plane(s) in the sky", "", n)`, утилита предлагает нам три варианта перевода: единственное число, "двойное" число и множественное число. Qt Linguist "знает", что мы переводим на русский (ранее я объяснил, почему). Она также "знает", что в русском языке существует три формы склонения существительных, следующих после числительных. Одна форма — для числительных, заканчивающихся на 1 (эта форма названа единственным числом, хотя она применима к любому числу, заканчивающемуся на 1), другая форма для числительных, заканчивающихся на 2, 3, 4 (это то, что мы назвали "двойственным числом"), и третья форма — для числительных, заканчивающихся на 5, 6, 7, 8, 9, 0. Заполните в Qt Linguist все три формы, как это сделал я, скомпилируйте ресурс перевода с помощью lrelease и скопируйте его в директорию с исполнимым файлом. Экспериментируя с различными значениями `n`, вы можете убедиться, что перевод всегда выполняется в соответствии с правилами русского языка (4 самолета в небе, 5 самолетов в небе, 21 самолет в небе и т. д.).

> ***Примечание***
>
> Как бы "умна" не была система перевода Qt, она не подскажет нам, как склоняется существительное кочерга с числительным 5, 6 и другими. Впрочем, я надеюсь, что вам не придется писать программу для учета этих самых... объектов "кочерга".