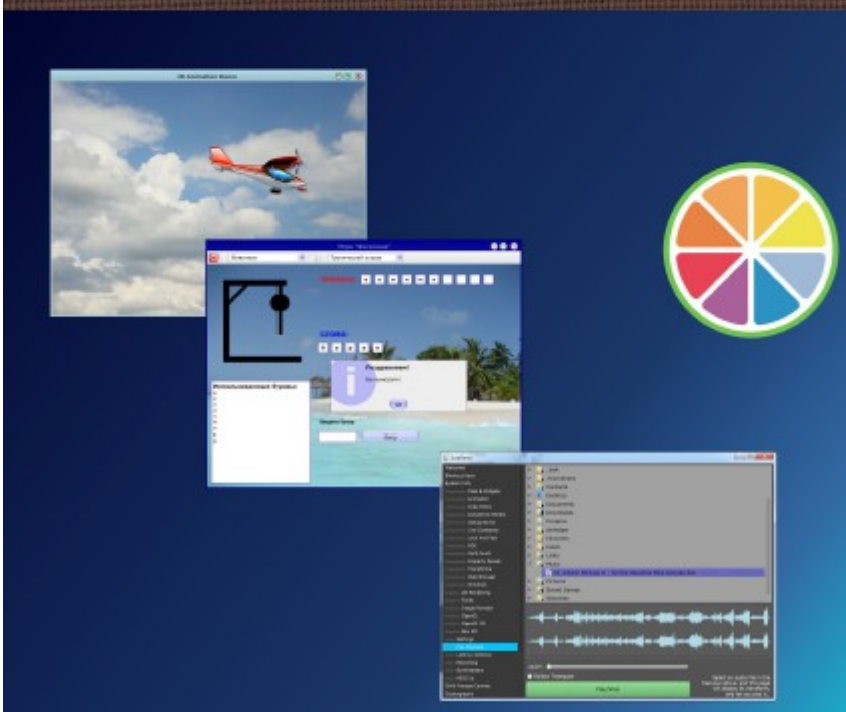
Программирование на C++ с JUCE 4.2.x

Андрей Кузнецов



Программирование на C++ с JUCE 4.2.x:

Создание кроссплатформенных мультимедийных приложений с использованием библиотеки JUCE на простых примерах

Андрей Николаевич Кузнецов

Linmedsoft
2016

Информация о книге

Кузнецов А. Н. Программирование на C++ с JUCE 4.2.x: Создание кроссплатформенных мультимедийных приложений с использованием библиотеки JUCE на простых примерах. − Алматы: Linmedsoft, 2016. − 384 с.: илл.

Книга посвящена разработке приложений для Linux, Windows, Mac OS X и iOS на языке C++ с использованием кроссплатформенной библиотеки JUCE версии 4.2.x. Подробно рассмотрены возможности, предоставляемые этой библиотекой, а также практическое применение классов, входящих в её состав, на большом количестве простых, подробно прокомментированных примеров. Книга содержит пошаговую исчерпывающую информацию по созданию приложений JUCE различной степени сложности от простейших до мультимедийных.

Первая публикация: 2011, издательство «Интуит»

Linmedsoft, Алматы
E-mail: linmedsoft@gmail.com

Предисловие

Со времени публикации моего онлайн курса «Разработка кроссплатформенных приложений с использованием JUCE», где рассматривались версии 1.5 и 2.0 библиотеки, было внесено значительное число изменений как в код её классов и методов, так и в инструменты, используемые для создания проектов.

Автор JUCE, Julian Storer, наконец осуществил своё намерение объединить Introjucer и Jucer в единую среду разработки с возможностью визуального проектирования интерфейса.

К сожалению, это привело к тому, что код, написанный для второй версии JUCE, не всегда может компилироваться с третьей версией библиотеки.

В этой связи в этой книге были переписаны примеры, использованные в курсе-предшественнике, и добавлено описание работы с обновлённой средой Introjucer / Projucer.

Настоящая книга также была дополнена практическими примерами по написанию настольных игр, включающих главное меню, панель инструментов и панель состояния, которые могут послужить основой для создания многочисленного класса подобных программ.

Несмотря на то, что JUCE широко используется для написания аудио-плагинов кроссплатформенных приложений, графический интерфейс последних строится обычно на основе иных библиотек. В этой связи автор настоящей книги особое внимание уделяет использованию многочисленных классов JUCE для написания простых примеров оконных приложений с постепенным нарастанием сложности.

Научиться программировать можно, лишь начав программировать! Поэтому скорее загружайте библиотеку JUCE и приступайте к экспериментам с ней! И пусть на этом пути вас ждёт удача!

Автор — Андрей Кузнецов,

г. Алматы, Казахстан

Для кого эта книга

Здравствуй, читатель!

Ты начал просматривать эту книгу, а значит, задаёшься вопросом, о чём она и нужна ли именно тебе.

Что ж, постараюсь ответить. Это книга о кроссплатформенном программировании с помощью библиотеки JUCE и языка разработки C++.

В наши дни уже очевидно, что писать программы для одной и только одной платформы, Windows, — это искусственно ограничивать число их пользователей. Компьютеры Apple с операционной системой Mac OS X никогда не теряли популярности, а интерес пользователей к UNIX-системам и, в частности, Linux вырос настолько, что последняя составила реальную конкуренцию продуктам от Microsoft. В моду вошли iPhone и iPad, а значит, приходится думать о разработке для мобильной операционной системы iOS.

Не потеряться в этом многообразии позволит кроссплатформенность разрабатываемых приложений, если не на бинарном уровне, но на уровне исходных текстов (принцип «написал программу — компилируй её в любом месте»).

Ответом на потребность в написании портируемых приложений стало появление ряда руководств, посвящённых технологии разработки программных продуктов с помощью кроссплатформенных библиотек. Это довольно большое число руководств по Qt, несколько меньшее — по GTK и уж совсем небольшое — по wxWidgets. Эти книги давно нашли своего читателя. Так зачем же нужна ещё одна, о JUCE?

Изначально JUCE создавалась как часть аудиосеквенсера для Mac OS X и Windows, Tracktion, из-за чего включает в себя примечательное число аудиофункций. По сути, на момент написания книги JUCE стала стандартом де-факто при написании кроссплатформенных программ обработки звука и аудио-плагинов. Эта библиотека, распространяемая под двумя типами лицензий, свободной и коммерческой, используется для создания программного обеспечения для профессиональных музыкантов многими компаниями, среди которых такие гиганты медиа-индустрии как Korg, TC Electronics, Mackie, M-Audio, PreSonus, SaneWave и другие. JUCE широко представлена как основа для создания OpenSource приложений; достаточно упомянуть программу для алгоритмической композиции Common Music Grace, такие библиотеки для обработки звука как UGen++ и CSL (Create Signal Library). Даже те открытые проекты, которые используют для создания графического интерфейса пользователя

другие библиотеки, зачастую включают VST-плагины, написанные с помощью JUCE. К таким программам, например, относится MIDI-секвенсер Qtractor, графический интерфейс пользователя которого построен на основе Qt.

К несомненным достоинствам JUCE относятся также ясные имена классов и методов, описывающие их назначение, исчерпывающая онлайн документация, оригинальность интерфейса и возможность гибкой настройки отображаемых компонентов, а также удобные инструменты для создания проектов под различные платформы и визуального проектирования интерфейса пользователя.

Несмотря на это, сложилась парадоксальная ситуация: JUCE широко используется как для создания проприетарных, так и свободных программ, библиотека является предметом для изучения в таких учебных заведениях, как Unversity of Chicago и University of the West of England, но в то же время по ней отсутствуют исчерпывающие руководства, способные послужить точкой старта для новичков. Для того, чтобы восполнить этот пробел, и написана настоящая книга.

Итак, вам стоит читать её, если:

- вы кое-что знаете о C++ и о разработке приложений для Windows, но хотите сделать первый шаг к программированию и для других платформ;
- вы желаете создавать одинаково выглядящие приложения с красочным интерфейсом как для настольных, так и мобильных операционных систем;
- вы разработчик мультимедийных приложений, гораздо более опытный в программировании звука, чем я, но ищущий отправную точку для перехода к кроссплатформенности создаваемых продуктов.

Эта книга ориентирована на программиста, который уже написал по крайней мере пару программ, и предполагает определённые знания языка C++. Это руководство по разработке с использованием JUCE, а не учебник программирования.

При работе над книгой перед автором стояла довольно сложная задача: написать пошаговое руководство JUCE, в котором основные классы библиотеки рассматриваются на простых примерах и которое, однако, не должно повторить печальную судьбу большинства других учебников — быть отложенным в сторону после прочтения. По замыслу автора его книга должна стать также и справочником, «поваренной

книгой» (how-to coockbook) по библиотеке. Поэтому в первых главах автор всё время забегает вперёд, а в последующих отсылает читателя к ранее изученному материалу. И именно поэтому порядок чтения книги совершенно произвольный (хотя вначале стоит прочесть главу о сборке программ, использующих JUCE). Вы можете либо читать её подряд, переходя по главам (сложность материала нарастает постепенно), либо открывать книгу, где угодно, на любой заинтересовавшей теме, используя как дополнение к онлайн документации (http://learn.juce.com/learn-home).

Что нужно для работы с этой книгой

Для работы с этой книгой достаточно иметь компьютер с установленным компилятором (список поддерживаемых компиляторов и операционных систем приводится в первой главе книги). Все приведённые примеры были протестированы под управлением следующих операционных систем: Linux (дистрибутив Kubuntu 14.04), MS Windows 7 и MS Windows 8.1, Mac OS X (Snow Leopard).

Сообщайте об ошибках

Что ж, эта книга — первый опыт написания пошагового руководства по библиотеке JUCE, а значит, просто не может быть свободной от недочётов. Разумеется, автор приложил все силы к тому, чтобы избежать ошибок, но... не ошибается лишь тот, кто ничего не делает. В том случае, если вы, дорогие читатели, заметили какую-то ошибку, то, пожалуйста, сообщите о ней с помощью формы обратной связи (https://leanpub.com/juce4x_ru/feedback).

Часть I. Основы JUCE

Глава 1. Общие сведения о JUCE. Получение библиотеки и её установка

Как известно, C++ — язык кроссплатформенный на уровне компиляции, т. е. программа для какой-либо операционной системы, написанная на нём, может быть откомпилирована и запущена в другой без каких-либо модификаций (либо с минимальными изменениями) исходных текстов. Однако вышесказанное относится лишь к консольным программам. Функции интерфейсов прикладного программирования (application programming interface, API) для создания среды рабочего стола разнятся для различных операционных систем. В то же время графический интерфейс пользователя (graphic user interface, GUI) уже давно стал стандартом де-факто по крайней мере для настольных приложений.

Поэтому программист, планирующий создавать свои продукты для различных платформ, должен определиться с инструментом для разработки переносимого графического интерфейса. В настоящее время библиотек, предоставляющих такую возможность, существует довольно много: Qt, GTK, Motiff, Tk, U++ и другие. К их числу относится и JUCE (Jules' Utility Class Extensions) — кроссплатформенная библиотека для создания приложений для Linux, Windows, Mac OS X, iOS и Android.

Подобно многим другим кроссплатформенным библиотекам (Qt, GTK, U++ и др.), JUCE является универсальной, т. е. предоставляет не только средства для разработки графического интерфейса пользователя (GUI toolkit), но и набор классов для различных нужд (работа с графикой, звуком, сетью, XML и т.п.). За счёт этого разработчики могут создавать с помощью JUCE приложения различной направленности без использования дополнительных библиотек. Именно это и является основной целью JUCE и определяет ряд особенностей, выделяющих её в ряду других универсальных GUI toolkit'ов. В отличие от тех же Qt и GTK, кроссплатформенность JUCE достигается за счёт прорисовки собственных оригинальных элементов пользовательского интерфейса с применением низкоуровневых системных функций вместо использования «родных» компонентов / виджетов для каждой платформы. Даже заголовок и рамку окна JUCE по умолчанию рисует самостоятельно, хотя можно программно переключиться на стандартное обрамление окон (рисунок 1.1).

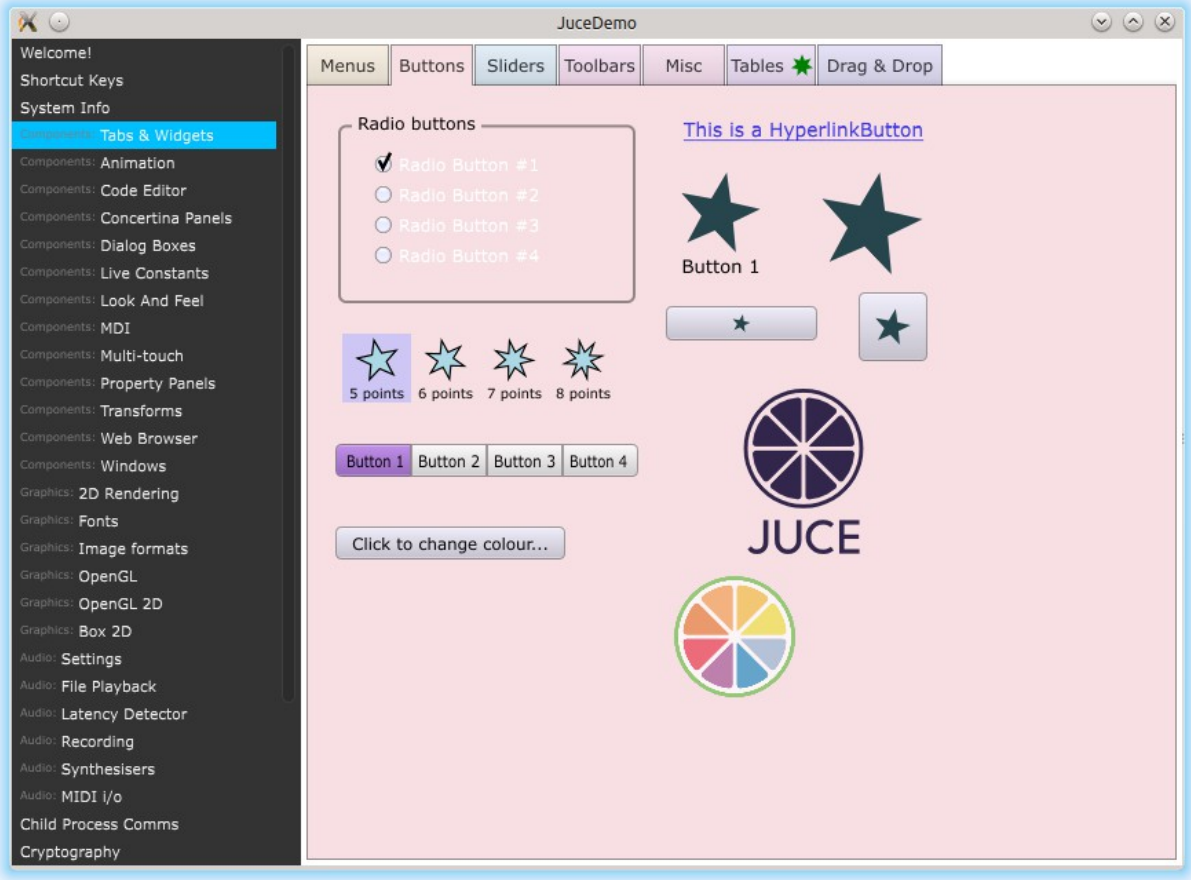


Рисунок 1.1. Оригинальный внешний вид виджетов в демонстрационной программе Juce Demo, входящей в поставку библиотеки (строка заголовка окна стандартная для графического окружения KDE / Plasma 4)

Подобный подход можно рассматривать и как достоинство, и как недостаток библиотеки. С одной стороны, этим достигается попиксельная идентичность интерфейсов приложений на всех поддерживаемых платформах, а для Linux — решение проблемы множественной зависимости. С другой стороны, JUCE не слишком подходит для приложений, интегрированных в ту или иную графическую среду, поскольку принципиально не может эмулировать внешний вид системных компонентов / виджетов. Тем самым, основная цель JUCE — это создание программ с оригинальным внешним видом, не зависящих (почти) от внешних библиотек.

Первоначально JUCE была разработана как часть кроссплатформенного аудио-редактора и MIDI-секвенсера Tracktion, выпущенного Raw Material Software в 2002 году, и лишь в 2004 году была

опубликована как самостоятельный инструмент разработки. До сих пор основная ниша JUCE — написание приложений для работы со звуком. Библиотека включает в себя поддержку воспроизведения звука через аудио и MIDI интерфейсы, полифонические синтезаторы, понимает файлы распространённых аудиоформатов (таких как MP3, WAV, AIFF, FLAC и Vorbis).

Однако JUCE содержит достаточно средств для разработки и иных программ любой степени сложности, и автор надеется, что эта книга вызовет заслуженный интерес к этой замечательной библиотеке.

1.1. Поддерживаемые компиляторы

На момент написания книги были выпущены различные версии библиотеки JUCE: 3.2.0, 4.0.x, 4.1.x и 4.2.x.

Для версий JUCE 3.2.0 и 4.0.x официально подтверждена правильная работа со следующими компиляторами:

1. для операционной системы Linux это g++, входящий в состав GCC, начиная с версии 3.3;
2. для операционных систем линейки Windows это Microsoft Visual Studio (начиная с версии Visual Studio 2005 и до 2015; поддержка Visual C++ 6 в JUCE убрана, начиная с третьей версии библиотеки). Библиотека JUCE обеспечивает взаимную совместимость программ, собранных для Windows различных версий;
3. для Mac OS X это GCC, входящая в состав инструмента разработки XCode (для OS X 10.4 или более поздней);
4. для создания нативных приложений iPhone и iPad также используется XCode;
5. для разработки под Android необходима среда Android Studio (также поддерживаются проекты Ant).

Кроме того, начиная с третьей версии, в JUCE добавлена поддержка проектов Code::blocks / MinGW для Windows и Code::blocks / g++ для Linux.

Однако для использования последних версий JUCE (4.1.x, 4.2.x) необходимы компиляторы, поддерживающие стандарт C++11. Автор рекомендует использовать при работе с актуальными релизами JUCE следующие компиляторы: в Linux это g++, начиная с версии 4.4; а в Windows – начиная с Microsoft Visual Studio 2013.

1.2. Получение и типы лицензий

Права на библиотеку JUCE до 2014 года принадлежали компании

Raw Material Software, расположенной в Великобритании, хотя по сути инструмент разработки поддерживался одним её сотрудником, программистом Julian Storer (рисунок 1.2). В ноябре 2014 года JUCE и компания Raw Material Software были приобретены компанией ROLI.

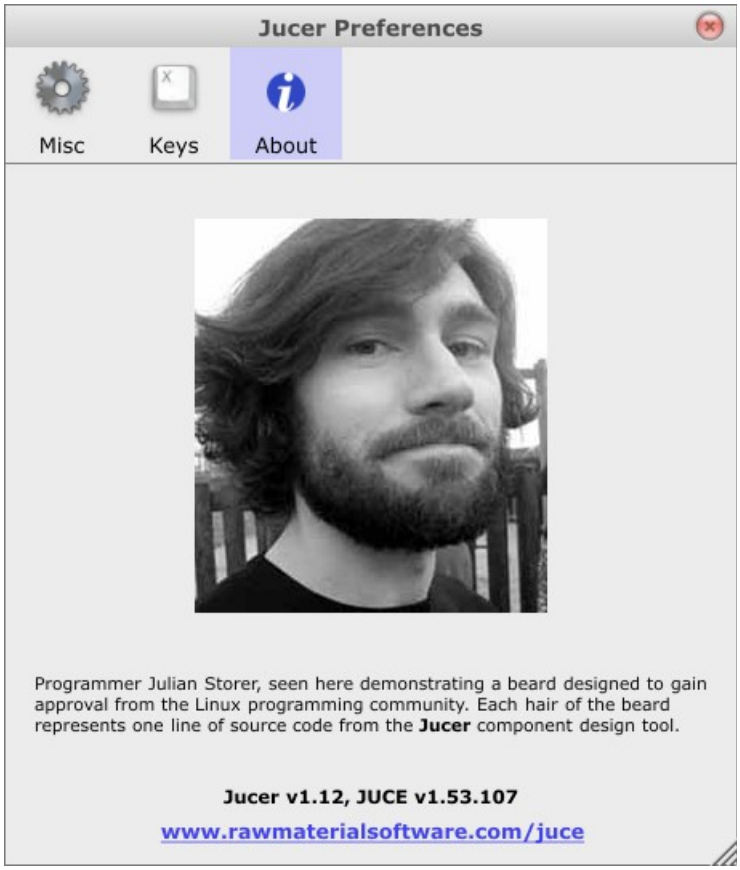


Рисунок 1.2. Julian Storer, разработчик библиотеки JUCE. Шутливая подпись под фотографией сообщает, что каждый волосок в бороде соответствует одной строчке кода визуального редактора компонентов библиотеки, Jucer

В настоящее время JUCE выпускается под тремя типами лицензий, которые можно разделить на две группы: свободная и коммерческая.

К первой относится лицензионный план Community, который даёт право разработчикам бесплатно получать и распространять библиотеку, используя последнюю для создания исключительно приложений с открытым исходным кодом под GNU Public License (GPL).

Коммерческие лицензии дают право создавать приложения, основанные на JUCE, с закрытым исходным кодом. К группе коммерческих лицензий относятся лицензионные планы Indie и Pro (Professional). На момент написания книги стоимость одной лицензии колебалась от 49 $ до 999 $.

Начиная со 2-й версии, исходный код JUCE разбит на функциональные фрагменты (модули). Большинство модулей распространяются под лицензиями GPL 2 и 3 версий, а также AGPL 3.

Один из модулей (juce_core) распространяется под лицензией ISC.

Полную информацию о коммерческих лицензиях можно найти по следующим ссылкам: https://www.juce.com/get-juce/indie и https://www.juce.com/get-juce/pro.

Далее мы будем работать с Open-Source версией JUCE. Распространяется она в виде исходных текстов; это относится как её модулям, так и демонстрационным приложениям и инструментам (BinaryBuilder, Introjucer / Projucer). Как следствие, размер архива небольшой: 11,1 Мб для Linux и 9,7 Мб для Windows. Сравните его с размером установщика другой кроссплатформенной библиотеки, Qt, который, в зависимости от целевой операционной системы, может «весить» от 500 до 900 Мб.

Загрузить актуальную версию JUCE можно по следующей ссылке: https://www.juce.com/get-juce.

Для бесплатной загрузки JUCE понадобится ввод Вашего адреса электронной почты (будет произведена регистрация в сообществе JUCE с предоставлением доступа к форуму разработчиков).

После регистрации вы увидите три кнопки для загрузки ZIP архива библиотеки с откомпилированной средой управления проектами JUCE, Projucer, для Linux, Windows и (Mac) OS X (рисунок 1.3).

Подробнее почитать о работе с Projucer вы можете, перейдя по следующей ссылке: https://www.juce.com/releases/projucer-juce-4.

Кроме того, актуальную версию JUCE можно загрузить с GitHub.

Это можно сделать двумя способами. Во-первых, можно загрузить архив мастер-ветки репозитория JUCE по ссылке, которая находится ниже кнопок загрузки (см. рисунок 1.3).

Во-вторых, клонировать репозиторий с помощью git, что «идеологически» более правильно.

Большинство распространённых дистрибутивов Linux включают git в официальные репозитории. Проверить, наличие git в системе можно командой

```
git --version
```

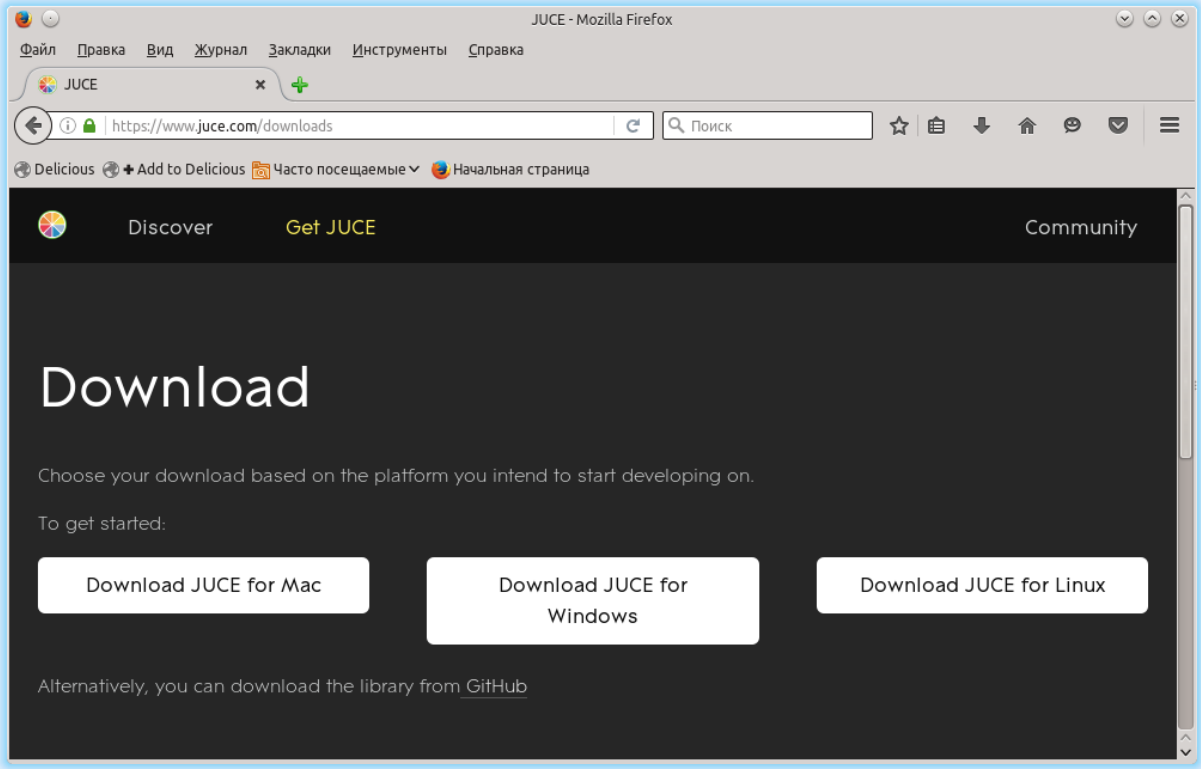


Рисунок 1.3. Загрузка JUCE с заранее откомпилированным Projucer

Установить git в *Ubuntu можно командой

```
sudo apt-get install git
```

Получить последнюю версию установщика git для Windows можно здесь: https://git-scm.com/download/win.

Его инсталляция проходит привычным образом и можно принять все опции по умолчанию, за исключением одного: в окне «Adjusting your PATH environment» отметьте радиокнопку «Use Git from the Windows Command Prompt» (рисунок 1.4): это позволит вам запускать git не из Git Bash, а из командной строки Windows, что гораздо удобнее.

После установки git клонировать мастер-ветку репозитория JUCE можно командой

```
git clone https://github.com/julianstorer/JUCE.git
```

При этом в текущем каталоге будет создана папка JUCE с локальной копией файлов и папок мастер-ветки, находящейся в удалённом

репозитории на GitHub.

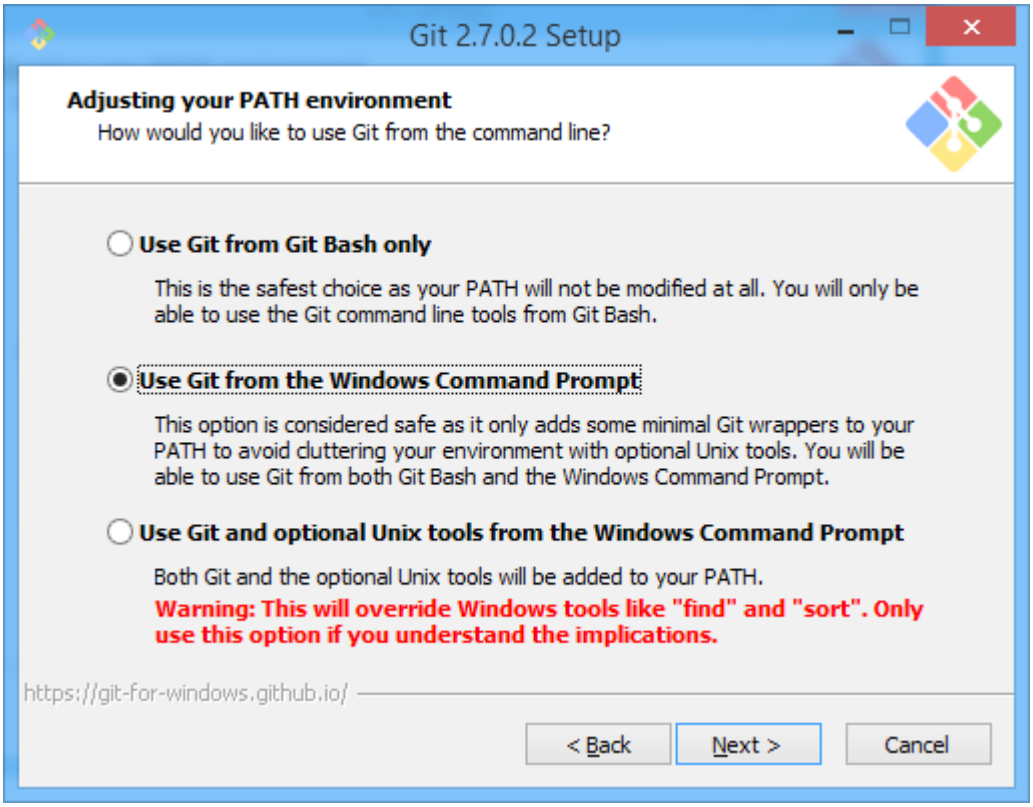


Рисунок 1.4. Установка git для Windows

По умолчанию консоль запускается в домашней папке пользователя. Для Windows это «C:\Users\Имя_пользователя». Для того, чтобы облегчить доступ к каталогу JUCE, до начала клонирования целесообразно перейти в папку «Документы» (рисунок 1.5).

В ряде случаев (например, если вы хотите использовать для сборки проектов компиляторы, не поддерживающие стандарт C++11) целесообразно загрузить старые версии JUCE.

Различные релизы JUCE версий 4.x можно найти по ссылке https://github.com/julianstorer/JUCE/releases.

Версию 3.2.0 можно найти на GitHub автора этой книги: https://github.com/ankuznetsov/JUCE-3.

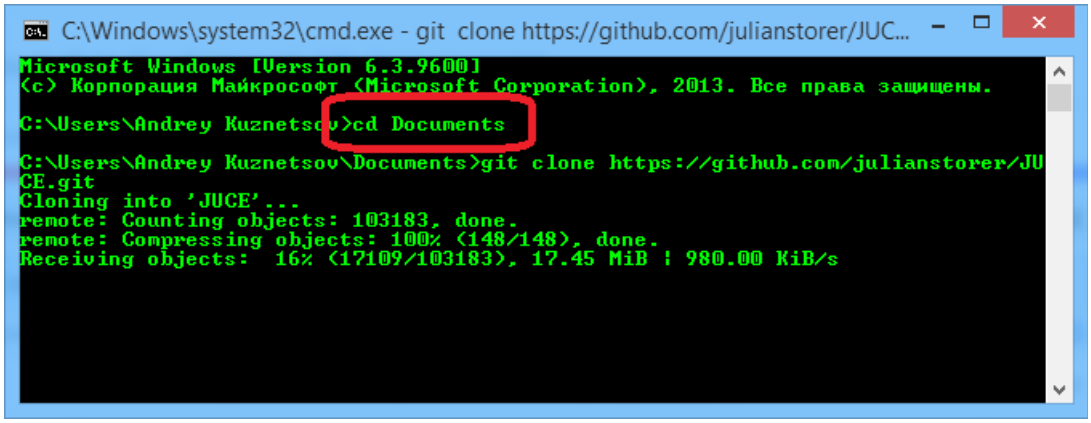


Рисунок 1.5. Клонирование мастер-ветки JUCE в Windows

1.3. Установка библиотеки

«Установка» JUCE заключается в том, чтобы распаковать загруженный архив и поместить дерево каталогов JUCE в какую-нибудь папку (например, в папку JUCE). В этой же папке целесообразно создать подпапку «projects», где будут находиться каталоги с файлами проектов ваших программ.

Для дальнейшей работы вам потребуется среда Introjucer / Projucer, которая предназначена для создания проектов JUCE, а также для визуальной компоновки и редактирования компонентов с последующей генерацией C++ кода). В устаревших версиях библиотеки (1.5, 2.0) этим целям служили две программы, которые назывались Jucer и the jucer.

Старые версии JUCE (3.x, 4.1.x) включали бесплатную среду управления проектами Introjucer, а Projucer была доступна лишь для разработчиков, купивших коммерческую лицензию.

Начиная с версии JUCE 4.2.0 Projucer стала открыта для всех пользователей, а среда Introjucer была исключена из поставки библиотеки.

Заметим, что открытая (некоммерческая) версия Projucer ни по внешнему виду, ни по функциональности не отличается от Introjucer, поэтому в данной книге среду управления проектами мы будем обозначать как Introjucer / Projucer.

Исходные тексты данной среды находятся в папке extras какталога установке JUCE. Самостоятельная сборка Introjucer / Projucer может понадобиться в случае, если вы клонировали JUCE с помощью git или откомпилированная среда по каким-либо причинам не запустилась на вашем компьютере.

Здесь мы рассмотрим сборку Introjucer / Projucer для Linux и Windows.

1.4. Сборка Introjucer / Projucer для Linux

Будьте готовы, что для корректной компиляции инструментов и демонстрационных программ JUCE вам понадобятся дополнительные пакеты разработки (dev-пакеты). В них содержатся файлы, необходимые компилятору, например, заголовочные файлы C / C++.

Так, в *Ubuntu 14.0.4 должны быть установлены:

```
libasound2-dev
libfreetype6-dev
libgl1-mesa-dev
libxcursor-dev
libxinerama-dev
libxrandr-dev
mesa-common-dev
libcurl4-openssl-dev
```

Проверить наличие этих пакетов в системе можно командой:

`dpkg -s libasound2-dev libfreetype6-dev libgl1-mesa-dev libxcursor-dev libxinerama-dev libxrandr-dev mesa-common-dev libcurl4-openssl-dev | grep Status` (рисунок 1.6).

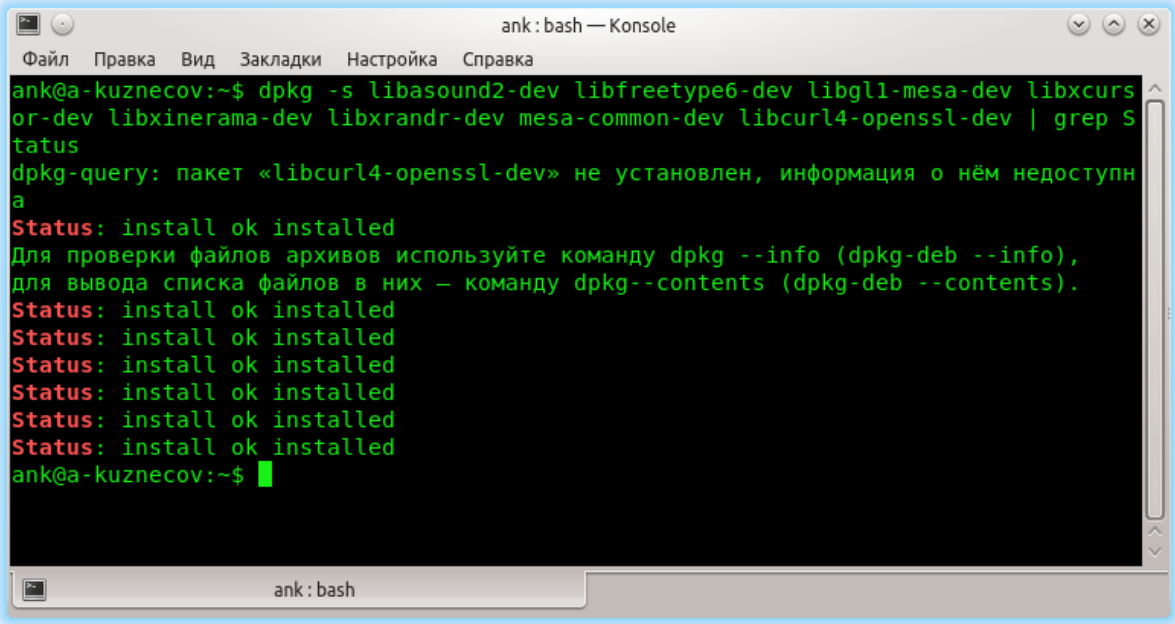


Рисунок 1.6. Проверка наличия необходимых пакетов разработки в Kubuntu. Отсутствует пакет libcurl4-openssl-dev

Для сборки Introjucer / Projucer откройте консоль и перейдите в папку

<каталог JUCE>/extras/<environment>/Builds/Linux, где <environment> — это папка с названием Introjucer в случае JUCE версий 3.2.0 или 4.1.x и Projucer – в случае, если вы скачали последнюю версию библиотеки.

Наберите в консоли make CONFIG=Release для того, чтобы собрать программу. Как правило этого достаточно. В некоторых дистрибутивах Linux (например, Fedora) может появиться ошибка линковщика. В этом случае необходимо отредактировать Makefile вручную, добавив в секцию LDFLAGS соответствующие флаги связывания (например, в Fedora мне понадобилось добавить -ldl и -lXext).

Для облегчения работы имеет смысл перенести программу Introjucer / Projucer из каталога, где она была собрана, в корневой каталог JUCE (рисунок 1.7).

Для знакомства с возможностями JUCE будет нелишним собрать демонстрационную программу Juce Demo (рисунок 1.8).

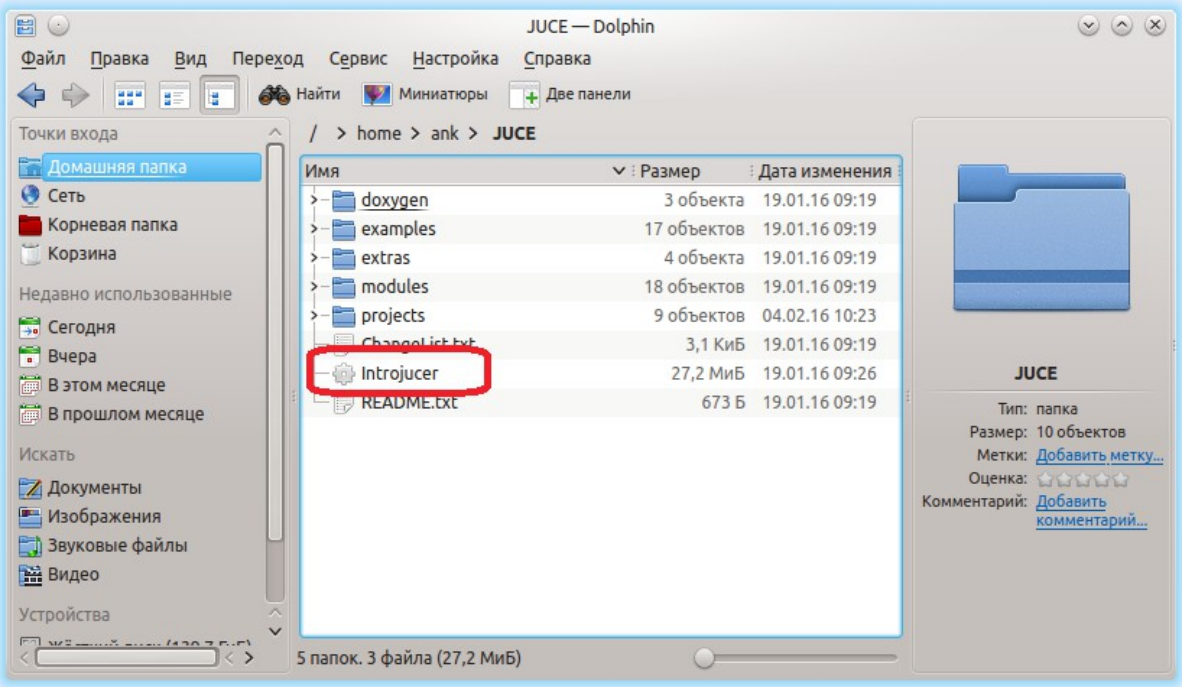


Рисунок 1.7. Дерево каталогов JUCE

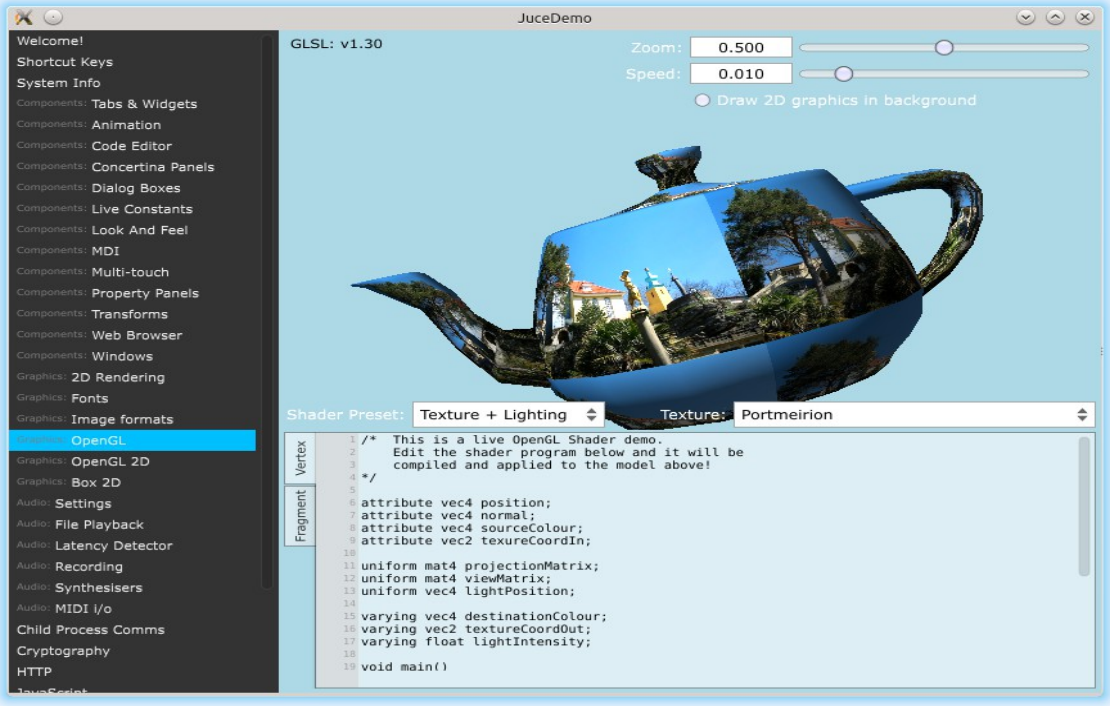


Рисунок 1.8. Работа демонстрационного приложения JUCE, Juce Demo

Откройте консоль и перейдите в папку <каталог JUCE>/examples/Demo/Builds/Linux. Наберите в консоли make. В случае ошибки связывания отредактируйте файл Makefile вручную, добавив в секцию LDFLAGS флаги -ldl и -lXext.

1.5. Сборка Introjucer / Projucer для Windows

В случае, если вы используете старые версии JUCE (3.2.0, 4.1.x), то собрать Introjucer под Windows вы можете, используя ряд платформ-зависимых проектов: это проекты Visual Studio (от 2005 до 2015), а также Code::Blocks.

В случае использования старых сред разработки (например, Visual Studio 2008), в процессе сборки программы вы можете получить ошибку

```
Cannot open include file: 'd2d1.h'
```

В этом случае вам необходимо установить DirectX SDK. Скачать этот комплект средств разработки можно по следующей ссылке: https://www.microsoft.com/en-us/download/details.aspx?id=6812.

Для современных компиляторов (например, Visual Studio 2013) это не требуется.

В случае, если вы хотите собрать Projucer с использованием новых версий JUCE (4.2.0, 4.2.1 или 4.2.2), то выбор сред разработки ограничен компиляторами, поддерживающими стандарт C++11. Также вы не сможете использовать Code::Blocks.

Для сборки Introjucer / Projucer откройте консоль и перейдите в папку <каталог JUCE>/extras/<environment>/Builds/VisualStudio2013, где <environment> — это папка с названием Introjucer в случае JUCE версий 3.2.0 или 4.1.x и Projucer – в случае, если вы скачали последнюю версию библиотеки. Подразумевается, что в качестве среды разработки используется MS Visual C++ 2013; в противном случае перейдите в соответствующую папку в каталоге Builds. Откройте решение «The Introjucer.sln», выберите в выпадающем списке «Конфигурация решений» стандартной панели инструментов значение Release и нажмите клавишу F7, чтобы собрать программу.

Для сборки Introjucer в Code::Blocks выполните аналогичные шаги, открыв проект «The Introjucer.cbp» из папки CodeBlocks. Для сборки проекта нажмите кнопку «Build» на инструментальной панели (рисунок 1.9).

Для знакомства с возможностями JUCE будет нелишним собрать демонстрационную программу Juce Demo (рисунок 1.8). Перейдите в

папку <каталог Juce>examples/Demo/Builds/VisualStudio2013, откройте проект и нажмите клавишу F7.

Для облегчения работы имеет смысл перенести программу Introjucer / Projucer из каталога, где она была собрана, в корневой каталог JUCE.

Теперь у вас есть все необходимые инструменты для написания программ, использующих JUCE.

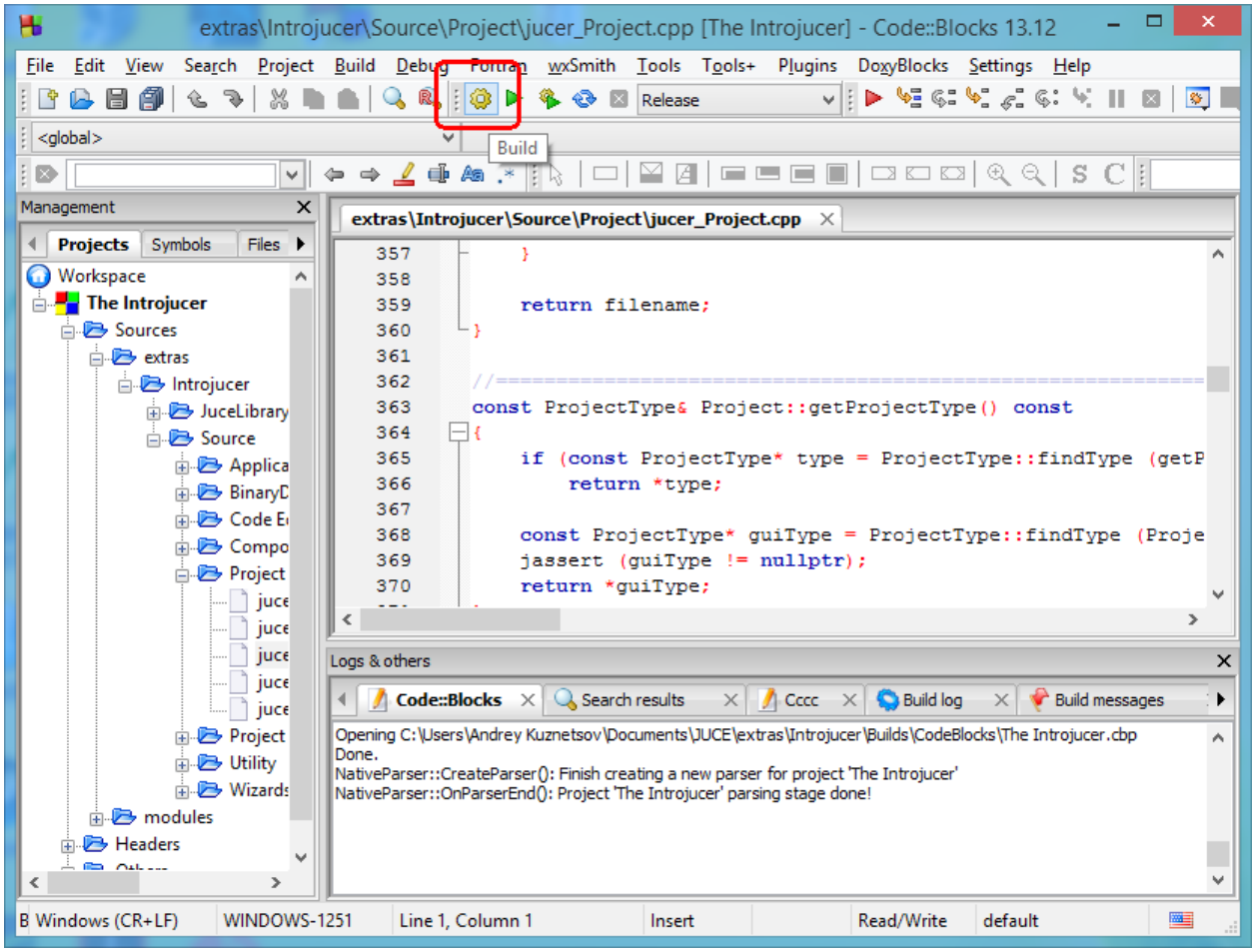


Рисунок 1.9. Сборка Introjucer в Code::Blocks для Windows

Глава 2. Начало работы с JUCE. Создание приложений. Главное окно. Первая программа с использованием JUCE

2.1. Создание и сборка проекта JUCE

В JUCE, как и в других кроссплатформенных библиотеках, существует два типа проектов: платформ-независимый (имеет расширение *.jucer и представляет собой XML-файл) и платформ-зависимые (для Linux это Makefile, для Windows — проекты Visual Studio и т.п.), которые генерируются программой Introjucer / Projucer (см. главу 1) на основе платформ-независимого проекта. Последний также создаётся в программе Introjucer / Projucer.

Рассмотрим создание проектов и сборку приложения, использующего JUCE, на примере простого оконного приложения.

Запустите на выполнение программу Introjucer или Projucer (напомним, что ни по внешнему виду, ни по функциональности бесплатная версия Projucer не отличается от Introjucer). При первом запуске будет показано окно «Create New Project» (рисунок 2.1).

Вызвать его также можно, выбрав в меню Introjucer / Projucer пункты **File – New Project...**

Окно содержит ряд кнопок, нажатие на которые вызывает тот или иной тип проекта (GUI Application, Animated Application и т. п.), который можно либо откомпилировать без изменений для изучения работы Juce, либо использовать как основу для создания собственных приложений того же типа.

Кроме того, в нижней части окна находятся кнопки для создания пустого проекта («Create Blank Project»): в этом случае исходные тексты разработчик добавляет вручную; открытия пользовательского («Open Existing Project») или демонстрационного проекта («Open Example Project») из числа находящихся в папке «examples» библиотеки.

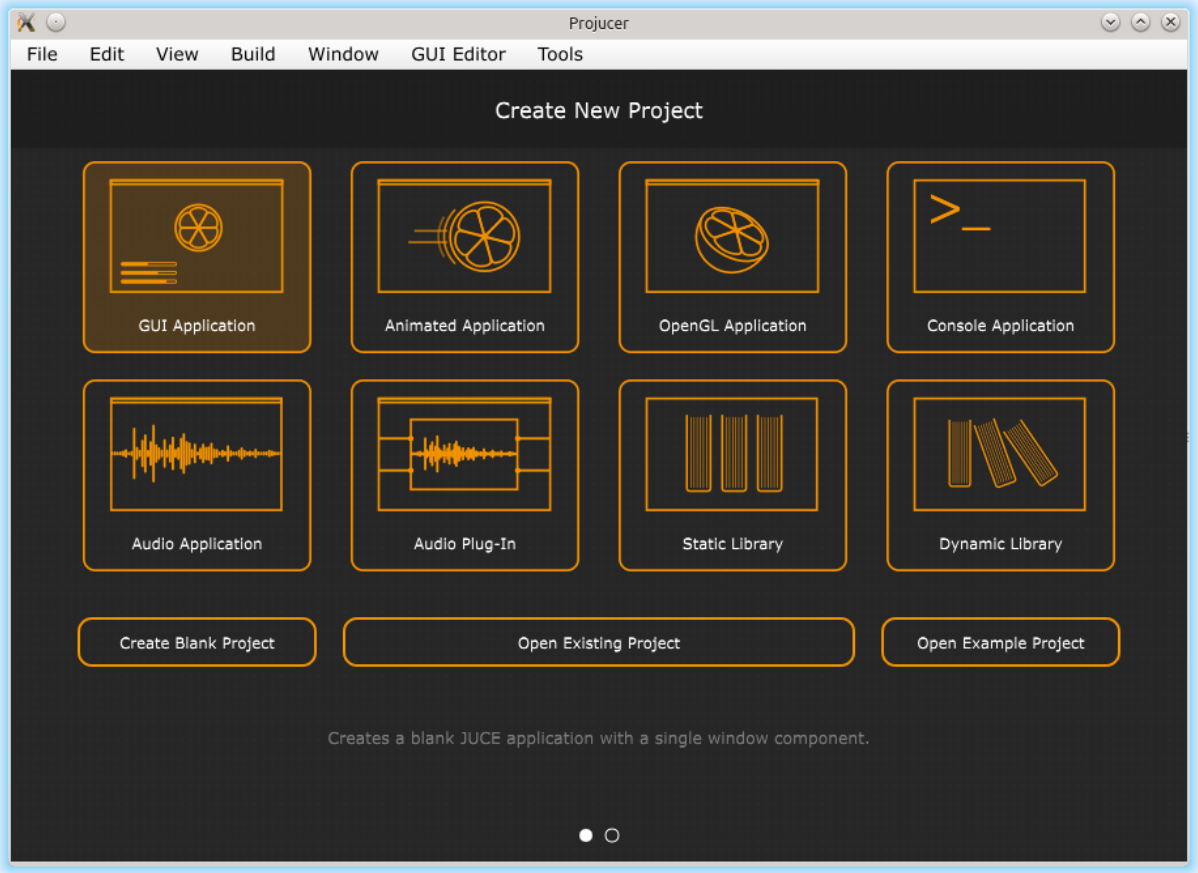


Рисунок 2.1. Projucer: окно выбора проекта

Для того, чтобы запустить проект приложения с графическим интерфейсом пользователя (оконное приложение) нажмите первую слева кнопку в верхнем ряду окна Introjucer, “GUI Application” (рисунок 2.1).

Появится окно настроек проекта Introjucer / Projucer (рисунок 2.2).

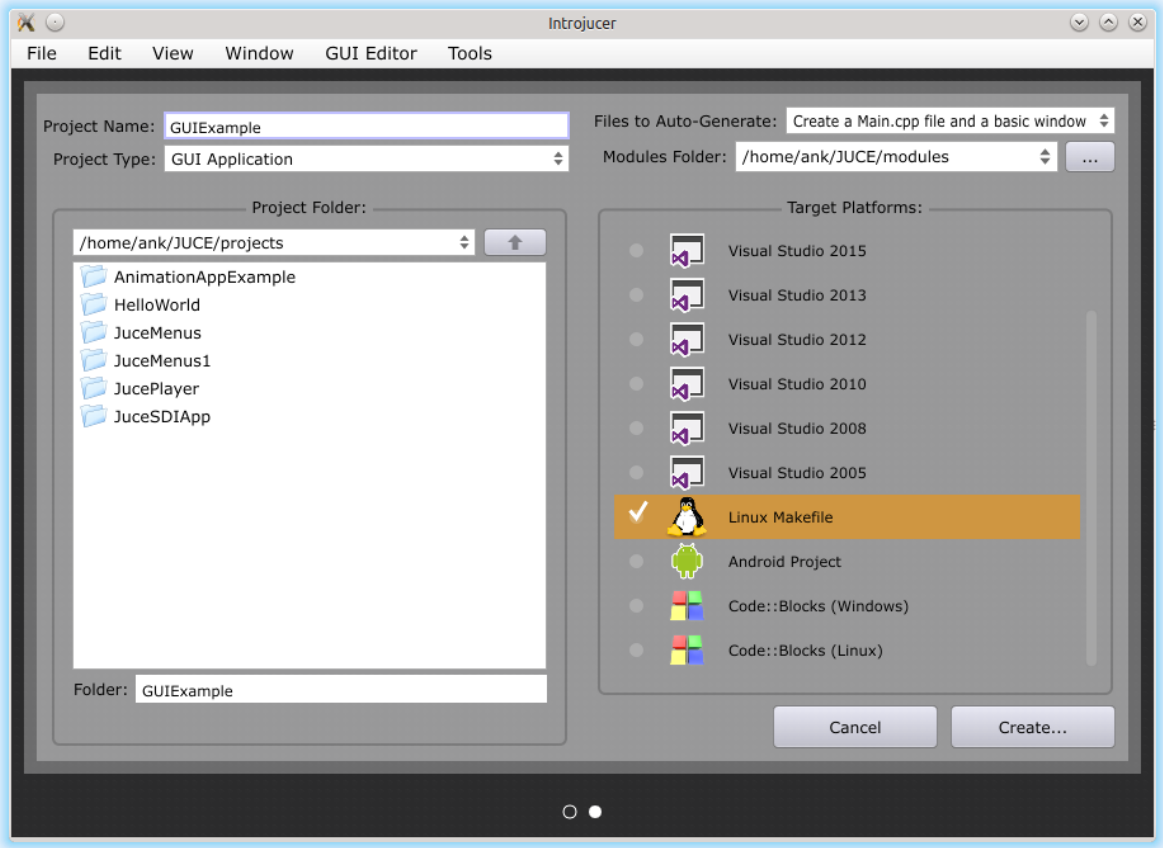


Рисунок 2.2. Настройка проекта Introjucer / Projucer

В левой части окна отображаются опции названия и размещения проекта, а в правой — его сборки.

В списке «Project Folder:» выберите каталог, в котором будет расположена папка проекта (например, папка «projects»).

Название папки проекта должно совпадать с его именем. В поле ввода «Project Name:» замените название по умолчанию «NewProject» на «GUIExample» (рисунок 2.2). При этом название папки в поле «Folder:» изменится автоматически на соответствующее.

Для того, чтобы компиляция проекта прошла без ошибок, важно убедиться, что путь к модулям JUCE в выпадающем списке «Modules Folder:» указан верно.

В группе «Target Platforms:» приведён список доступных экспортёров для целевых платформ. По умолчанию отмечен тот из них, который соответствует операционной системе, в которой запущен Introjucer / Projucer (рисунок 2.2). Если вы планируете собирать свою программу и для других операционных систем, вы можете отметить и другие

экспортёры; в этом случае соответствующие **платформ-зависимые** проекты будут сгенерированы после того, как вы сохраните проект Introjucer / Projucer.

Нажмите кнопку «Create...» для того, чтобы перейти к окончательной конфигурации проекта (рисунок 2.3).

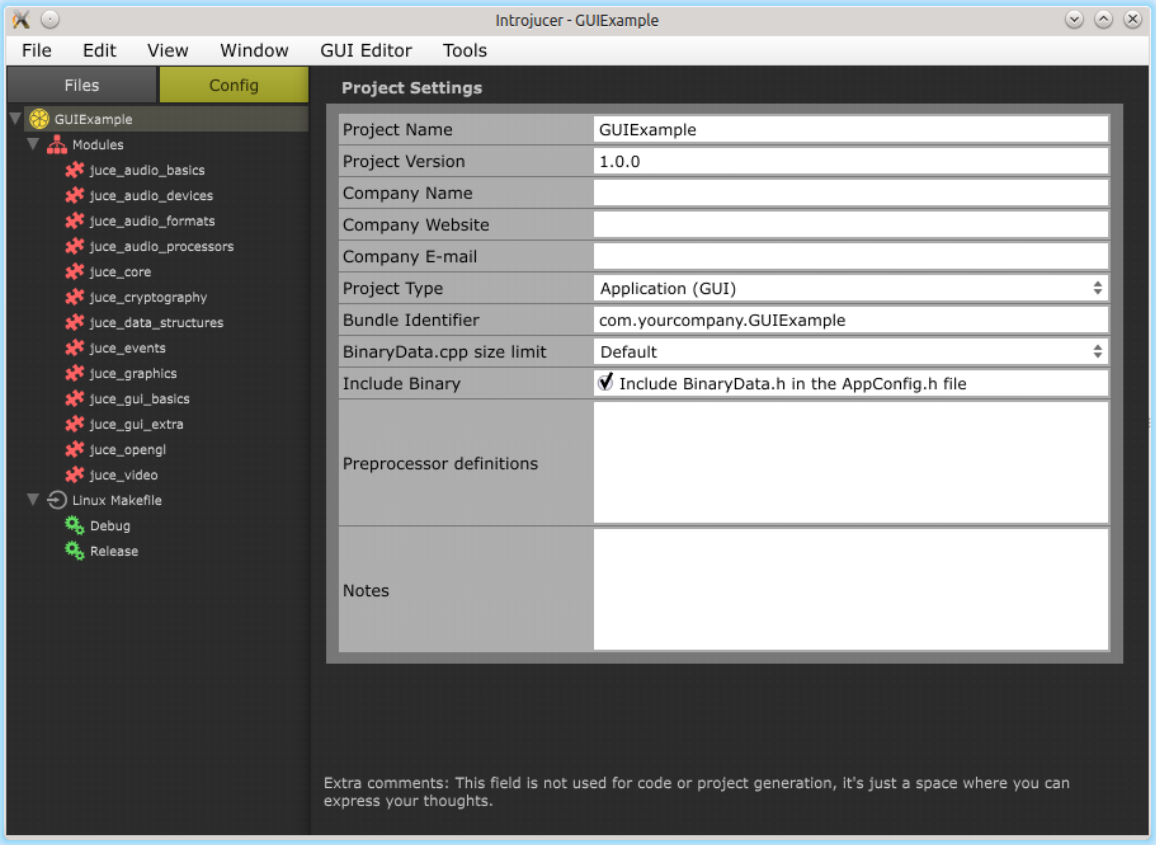


Рисунок 2.3. Настройка проекта Introjucer / Projucer

На вкладке Project Settings будут отображены основные свойства проекта: Project Name (название проекта), Project Version (его версия), Company Name (название компании-разработчика), Company Website (сайт компании), Company E-mail (и её адрес электронной почты), Project Type (тип проекта; в нашем случае это приложение с графическим интерфейсом пользователя – Application (GUI)), Bundle Identifier (строка – идентификатор приложения). Последняя необходима для однозначной характеристики создаваемого приложения в Mac OS X и iPhone. Строка должна соответствовать нотации Reverse-DNS и состоять из двух частей: названия компании разработчика (по умолчанию стоит com.yourcompany,

которое следует заменить на реальное) и названия программы (рисунок 2.3). Следует помнить, что строка не должна содержать символа «*».

В левой части настроек проекта Introjucer / Projucer, в группе «Modules» отображается список модулей библиотеки JUCE, включённых в проект. Однако не все они требуются для компиляции простого оконного приложения. Для уменьшения размеров исполняемого файла и сокращения времени компиляции исключим из проекта следующие модули:

- juce_audio_basics;
- juce_audio_devices;
- juce_audio_formats;
- juce_audio_processors;
- juce_cryptography;
- juce_opengl;
- juce_video.

Для этого щёлкните правой кнопкой мыши по названию модуля и выберите в контекстном меню «Remove this module» (рисунок 2.4).

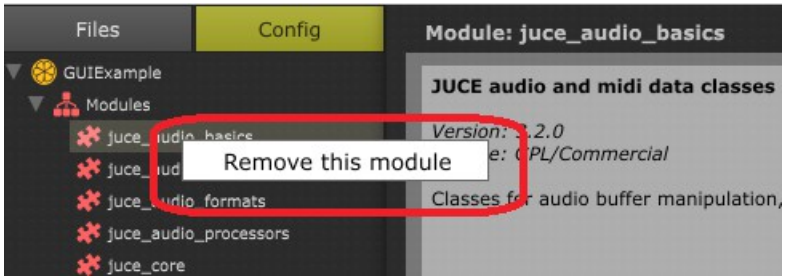


Рисунок 2.4. Удаление лишних модулей из проекта

Если вы хотите добавить к проекту отсутствующий экспортёр (сгенерировать дополнительный **платформ-зависимый** проект для той или иной операционной системы), щёлкните **правой** кнопкой мыши по названию проекта и выберите нужный пункт из контекстного меню (рисунок 2.5).

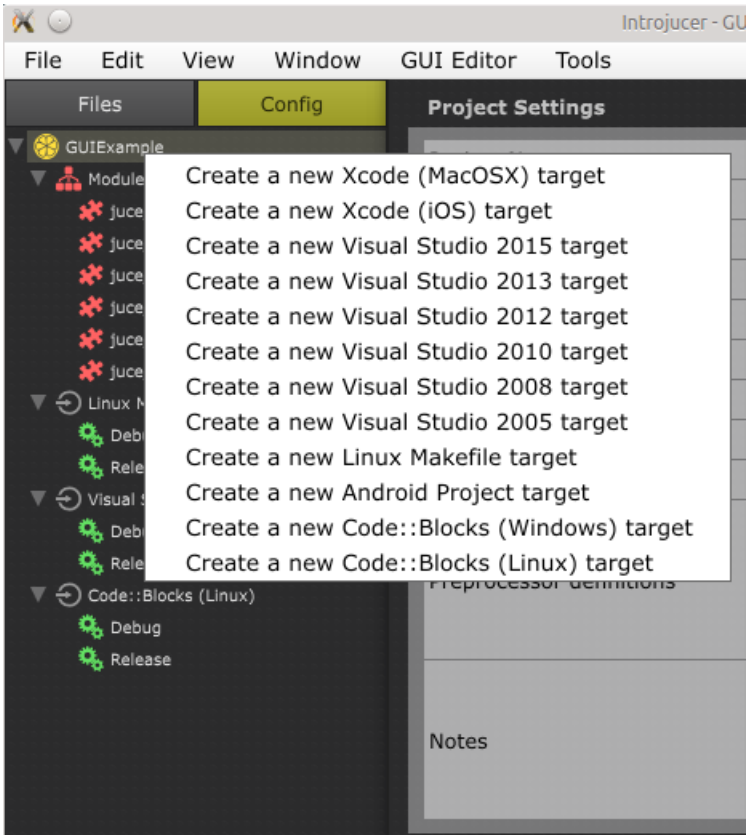


Рисунок 2.5. Добавление нового экспортёра в проект Introjucer

К сожалению, при добавлении нового экспортёра зачастую путь к модулям Juce определяется неправильно, что приводит к ошибкам компиляции. Поэтому этот путь необходимо исправить на корректный вручную **для каждого модуля** (рисунок 2.6).

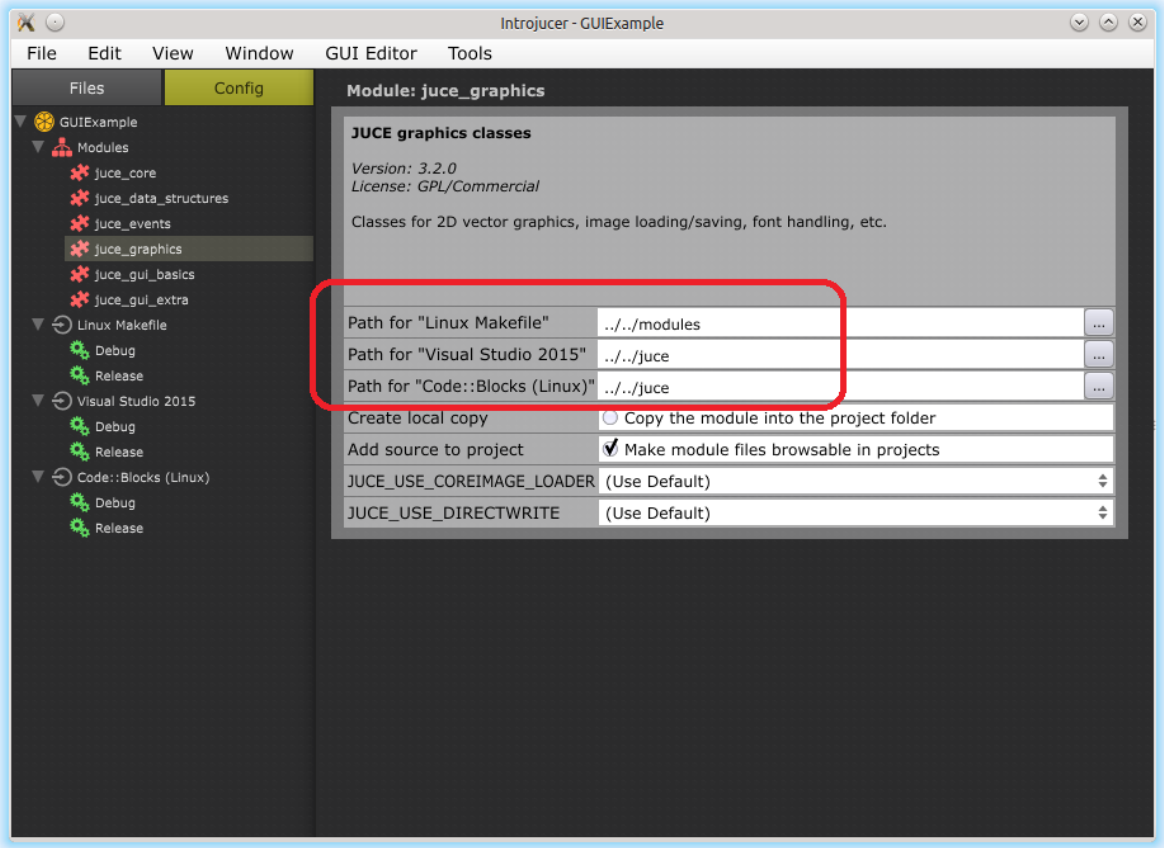


Рисунок 2.6. Пути к модулям JUCE для различных экспортёров не совпадают

Помимо экспортёров, Introjucer / Projucer генерирует исходные тексты программы, в которые впоследующем программист может вносить изменения. Для того, чтобы их увидеть, переключитесь с вкладки «Config» на вкладку «Files» (рисунок 2.7).

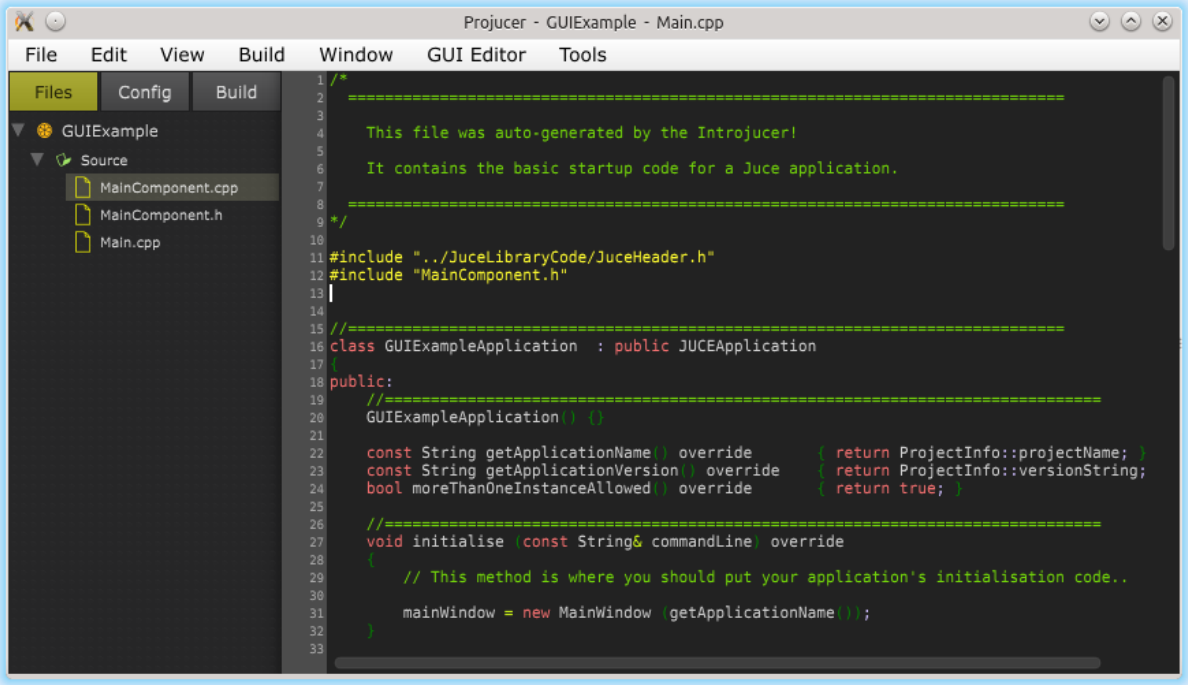


Рисунок 2.7. Проект Projucer (открыты исходные тексты программы)

В проекте Introjucer / Projucer файлы, независимо от их физического расположения, организуются в **группы**. Группы обозначены значком в виде зелёной папки.

Обязательной группой проекта, в которой находятся исходные тексты программы, является «Source».

Также довольно часто в проекты добавляют группу, содержащую **бинарные файлы**, например файлы изображений. Названия других групп, помимо «Source», может быть произвольным.

Новую группу можно добавить, находясь на вкладке «Files» как собственно в проект (для этого необходимо щёлкнуть правой кнопкой мыши по названию приложения), так и в другую группу (для этого необходимо щёлкнуть правой кнопкой мыши по названию группы). Через это же меню в проект можно добавить новый файл, компонент и т. п. (рисунок 2.8). При добавлении нового файла в группу / проект открывается диалог его сохранения на жёстком диске.

После завершения настройки проекта сохраните его (выберите в меню Introjucer / Projucer пункты File → Save Project).

В папке GUIExample будет сохранён Xml-файл платформ-независимого проекта GUIExample.jucer (в последующем его можно

редактировать в Introjucer / Projucer, выбрав в меню File → Open...), папка JuceLibraryCode, в которой содержится необходимый для работы приложения код (его необходимо включать в исходные тексты программ директивой #include "../JuceLibraryCode/JuceHeader.h"), а также папка Builds, которая содержит подпапки платформ-зависимых проектов.

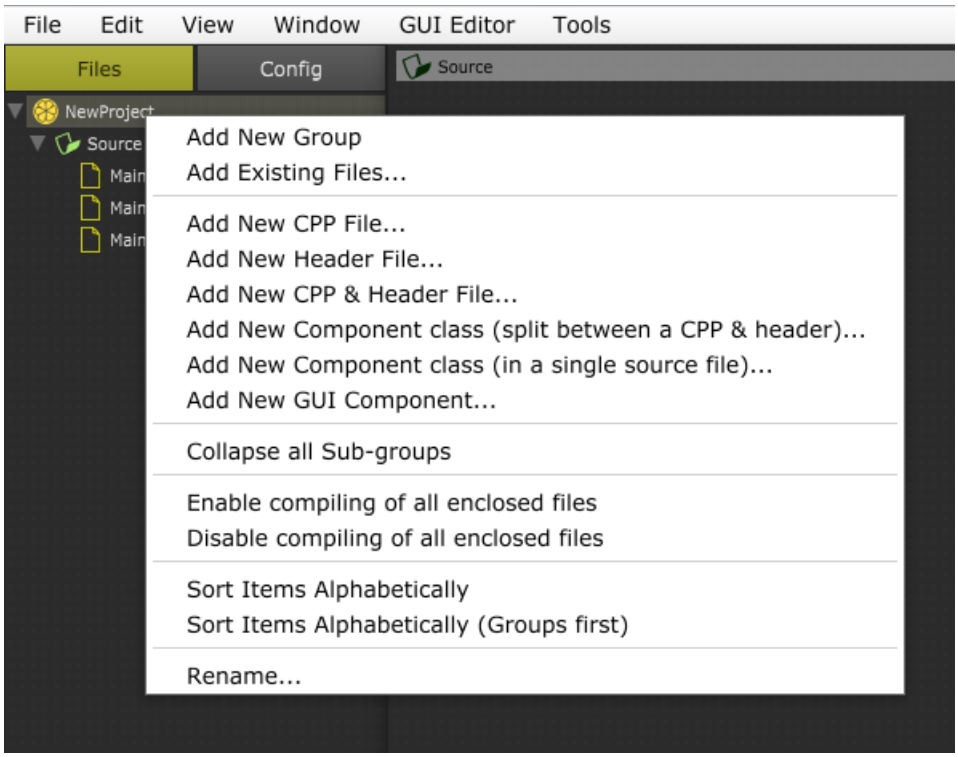


Рисунок 2.8. Добавление новых групп и файлов в проект Introjucer / Projucer

После генерации платформ-зависимых проектов исполняемый файл программы компилируется согласно правилам среды разработки целевой операционной системы, поэтому проект Introjucer / Projucer на этом этапе можно закрыть.

Теперь можно собрать программу для нашей целевой платформы. Платформ-зависимые проекты создаются при сохранении проекта Introjucer / Projucer в папке «Builds», находящейся в каталоге вашей программы.

По умолчанию, если Introjucer / Projucer была запущена под Linux, то создаётся Makefile, а если под Windows, то создаётся проект Visual Studio 2010 (рисунок 2.9).

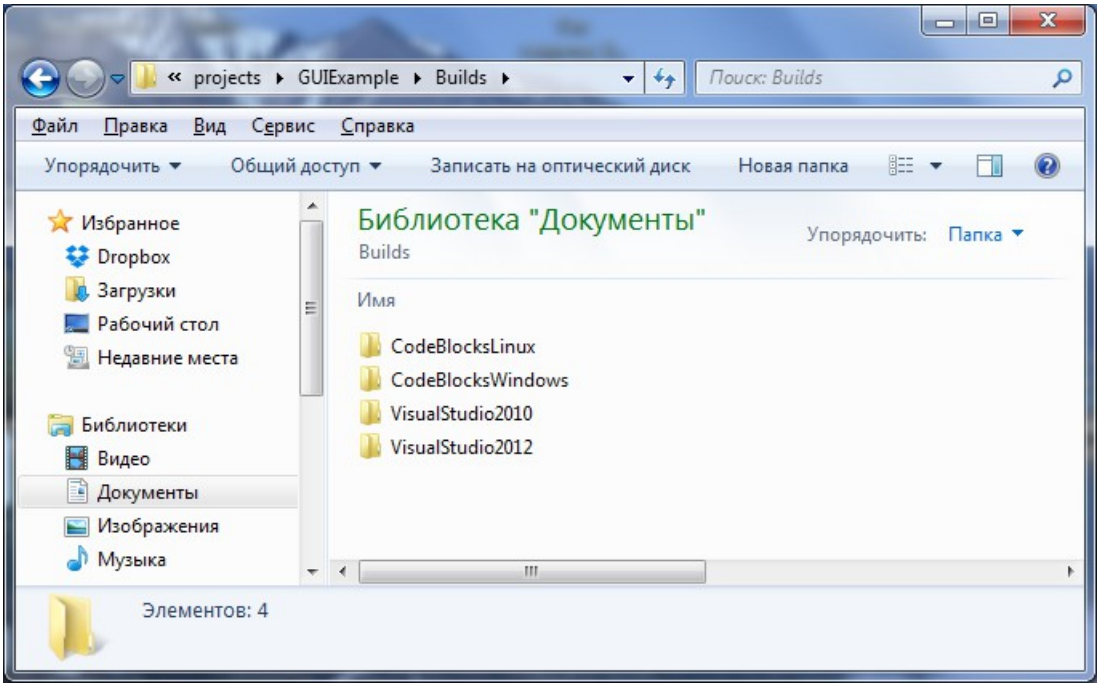


Рисунок 2.9. Платформ-зависимые проекты JUCE. К экспортёру, созданному по умолчанию, добавлены Code::Blocks для Linux и Windows, а также Visual Studio 2012

Сборка программы осуществляется по правилам конкретной среды разработки. Например, для Visual Studio необходимо открыть файл «GUIExample.sln» и нажать F7. В случае необходимости внесения правок в код программы, это можно делать в среде разработки целевой операционной системы, не открывая исходные тексты приложения в Introjucer / Projucer.

В Linux довольно удобно использовать Code::Blocks вместо традиционного Makefile. Если вы добавили этот экспортёр в проект Introjucer, откройте в Code::Blocks файл «GUIExample.cbp», соберите программу, нажав клавиши Ctrl+F9 и запустите её на выполнение клавишей F8 (рисунок 2.10).

К сожалению, компилятор MinGW, входящий в поставку Code::Blocks для Windows, часто выдаёт ошибки компиляции при работе с Juce, поэтому в данной операционной системе лучше использовать Visual Studio.

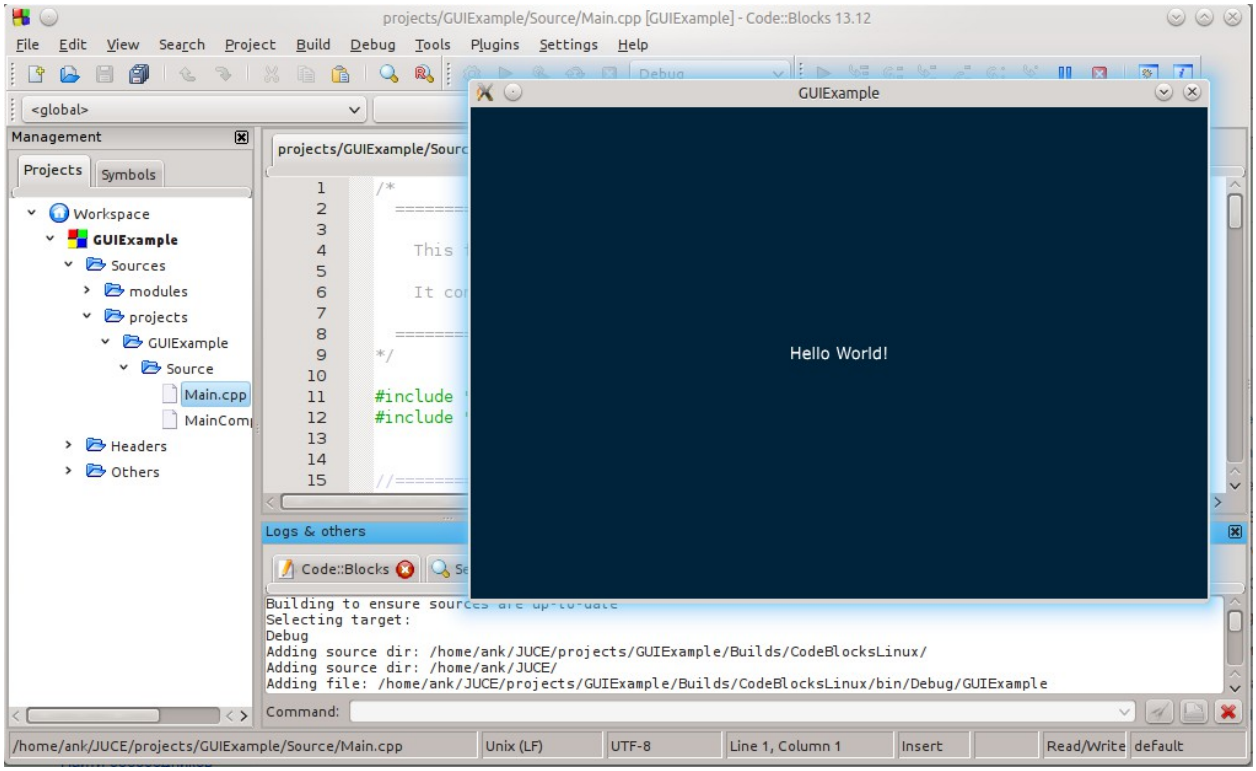


Рисунок 2.10. Работа простого оконного приложения JUCE (собрано в Code::Blocks для Linux)

2.2. Структура программы, использующей библиотеку JUCE

Файлы исходных текстов программы, рассмотренной выше, были автоматически сгенерированы Introjucer / Projucer. Однако мы можем создать их и самостоятельно, а затем подключить к пустому проекту. Поскольку, на взгляд автора, писать код самостоятельно — это лучший способ изучить особенности библиотеки, в этой книге мы будем использовать именно его.

Создайте в папке ваших проектов каталог JBWindow, а в нём — подпапку Source. В последней сохраните файлы main.cpp (листинг 2.1), TApplication.h (листинг 2.2), TApplication.cpp (листинг 2.3), TMainForm.h (листинг 2.5) и TMainForm.cpp (листинг 2.6).

Запустите программу Introjucer или Projucer (рисунок 2.11). Если в окне Introjucer / Projucer открыт какой-то проект, закройте его (File → Close Project). Создайте новый проект так, как мы это делали раньше (File → New Project...).

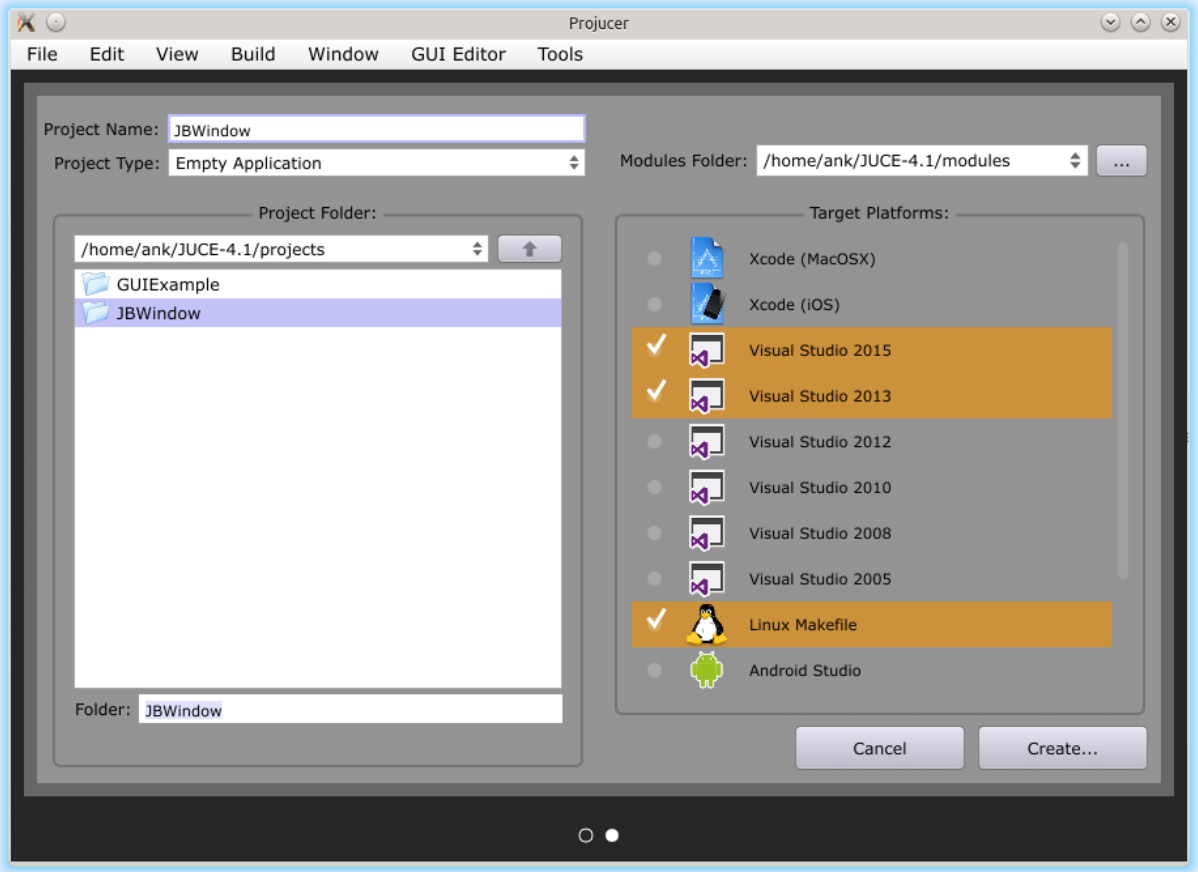


Рисунок 2.11. Создание пустого проекта Projucer

В левом нижнем углу программы нажмите кнопку «Create Blank Project».

В группе «Project Folder:» выберите название созданной папки, JuceWindow. Оно автоматически отобразится в поле ввода «Folder:». Скопируйте оттуда это название и замените им имя по умолчанию в поле «Project Name» (рисунок 2.11).

Отметьте нужные экспортёры в списке «Target Platforms» и нажмите кнопку «Create...»

В следующем окне Introjucer / Projucer выполните настройку проекта так, как это описано выше, после чего перейдите на вкладку «Files».

Выделите группу «Source» и щёлкните по ней правой кнопкой мыши. В контекстном меню выберите пункт Add Existing Files... В появившемся диалоговом окне Add Files to Jucer Project выберите файлы из папки «Source» (используйте клавишу <SHIFT>) и нажмите кнопку «Open». На левой панели Introjucer / Projucer будет отображён список файлов,

включённых в проект. При необходимости исходные тексты программы можно просматривать и редактировать непосредственно в Introjucer / Projucer (рисунок 2.12).

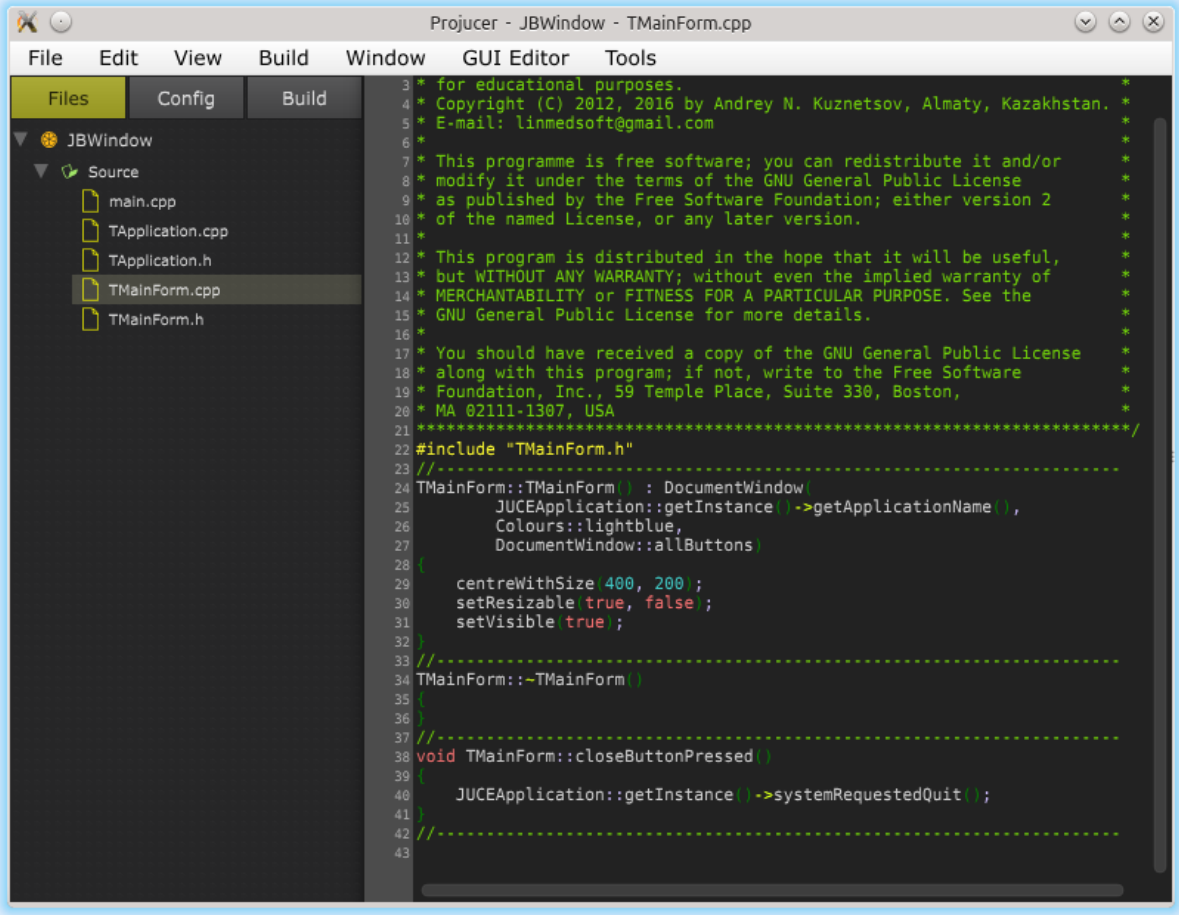


Рисунок 2.12. Просмотр исходных текстов программы в Projucer

Запуск программы, написанной с использованием JUCE, осуществляет макрос START_JUCE_APPLICATION, который принимает в качестве аргумента имя класса приложения, наследуемого от базового класса JUCEApplication, и генерирует платформ-зависимую функцию main / WinMain (листинг 2.1).

Листинг 2.1. Файл main.cpp

```
// TApplication — класс, унаследованный от JUCEApplication
```

```
#include "TApplication.h"
//--------------------------------------------------------------------------
// Этот макрос заменяет функцию main
START_JUCE_APPLICATION(TApplication)
//--------------------------------------------------------------------------
```

Класс `JUCEApplication` содержит чистые виртуальные методы, которые обязательно должны быть переопределены в классе-наследнике. Объект класса, наследуемого от `JUCEApplication`, ответственен за инициализацию (как правило создание и отображение главного окна) и корректное завершение кода программы. В связи со спецификой дизайна библиотеки в конструкторе и деструкторе класса приложения ничего писать нельзя, поэтому прежде всего должны быть переопределены его методы инициализации и очистки:

- `virtual void JUCEApplication::initialise(const String& commandLineParameters);`
- `virtual void JUCEApplication::shutdown().`

Кроме того, переопределяются следующие методы:

- `virtual void JUCEApplication::systemRequestedQuit()` — вызывается в момент, когда операционная система пытается завершить работу приложения и совершает действия, определённые программистом, которые должна выполнить программа перед закрытием;
- `virtual const String JUCEApplication::getApplicationName()` — возвращает имя приложения;
- `virtual const String JUCEApplication::getApplicationVersion()` — возвращает версию приложения;
- `virtual bool JUCEApplication::moreThanOneInstanceAllowed()` — определяет, возможен ли запуск более чем одного экземпляра приложения.

В листингах 2.2 и 2.3 показан простейший пример класса `TApplication`, объект которого отображает окно класса `TMainForm`:

Листинг 2.2. Файл TApplication.h

```
#ifndef _TApplication_h_
#define _TApplication_h_
//--------------------------------------------------------------------------
// Автоматически генерируемый файл, содержащий объявления классов Juce
#include "../JuceLibraryCode/JuceHeader.h"
//--------------------------------------------------------------------------
class TMainForm; // Класс главного окна
//--------------------------------------------------------------------------
class TApplication : public JUCEApplication
{
```

```
public:
    TApplication();
    ~TApplication();

    void initialise(const String&);
    void shutdown();
    void systemRequestedQuit();
    const String getApplicationName();
    const String getApplicationVersion();
    bool moreThanOneInstanceAllowed();

private:
    TMainForm* pMainWindow; // Главное окно приложения
};
//------------------------------------------------------------------------
#endif
```

Листинг 2.3. Файл TApplication.cpp

```
#include "TApplication.h"
#include "TMainForm.h"
//------------------------------------------------------------------------
TApplication::TApplication() : pMainWindow(0)
{
}
//------------------------------------------------------------------------
TApplication::~TApplication()
{
}
//------------------------------------------------------------------------
void TApplication::initialise(const String& /*sCommandLine*/)
{
    pMainWindow = new TMainForm(); // Создаём главное окно программы
}
//------------------------------------------------------------------------
void TApplication::shutdown()
{
    if(pMainWindow != 0) delete pMainWindow;
}
//------------------------------------------------------------------------
void TApplication::systemRequestedQuit()
{
    quit(); // Завершаем работу приложения
}
//------------------------------------------------------------------------
const String TApplication::getApplicationName()
{
    // Задаём имя приложения
    // (будет отображаться в заголовке главного окна)
    return "JB Window Demo";
}
//------------------------------------------------------------------------
const String TApplication::getApplicationVersion()
{
    // Получаем версию приложения из опций проекта
```

```
    return ProjectInfo::versionString;
}
//------------------------------------------------------------------------
bool TApplication::moreThanOneInstanceAllowed()
{
    // Запрещаем запуск более одного экземпляра приложения
    return false;
}
//------------------------------------------------------------------------
```

Метод `initialise()` вызывается при запуске приложения и, после выполнения действий, определённых пользователем (в нашем случае это создание главного окна), запускает цикл событий приложения (event-dispatch loop), который отслеживает их вплоть до тех пор, пока не будет вызван метод `static void JUCEApplication::quit()`, который убивает цикл и завершает работу программы.

Как видно из листинга 2.3, окно программы создаётся динамически, и программист должен сам позаботиться о его уничтожении в методе `shutdown()`.

Как упоминалось выше, метод `systemRequestedQuit()` вызывается, когда поступает запрос на завершение работы приложения (например, при закрытии главного окна). При этом в методе можно либо определить действия, которые должна выполнить программа до завершения своей работы (вывод сообщений, сохранение документов, настроек и т.п.) с последующим вызовом метода `quit()`, либо просто ограничиться последним, как в нашем случае (листинг 2.3).

Метод `getApplicationName()` возвращает строку — имя приложения. Именно эта строка отображается в заголовке главного окна.

Метод `getApplicationVersion()` возвращает строку, содержащую номер версии приложения, значение которой может быть либо задано вручную, либо взято из опций проекта Introjucer / Projucer (листинги 2.3, 2.4).

Листинг 2.4. Реализация метода getApplicationVersion()

```
const String TApplication::getApplicationVersion()
{
    return "1.0";
}
```

В том случае, если метод `moreThanOneInstanceAllowed()` возвращает `true`, при повторном щелчке по значку программы запустится ещё один её экземпляр; в случае, если метод возвращает `false`, может быть запущен только один экземпляр приложения.

Основным классом JUCE для создания окна с изменяемыми размерами, строкой заголовка и кнопками «свернуть окно», «распахнуть окно» и «закрыть окно» является `DocumentWindow`. Унаследуем от него собственный класс `TMainForm` (листинги 2.5 и 2.6).

Листинг 2.5. Файл TMainForm.h

```
#ifndef _TMainForm_h_
#define _TMainForm_h_
//------------------------------------------------------------------------
#include "../JuceLibraryCode/JuceHeader.h"
//------------------------------------------------------------------------
class TMainForm : public DocumentWindow
{
public:
    TMainForm();
    ~TMainForm();

    void closeButtonPressed();
};
//------------------------------------------------------------------------
#endif
```

Листинг 2.6. Файл TMainForm.cpp

```
#include "TMainForm.h"
//------------------------------------------------------------------------
TMainForm::TMainForm() : DocumentWindow(
        JUCEApplication::getInstance()->getApplicationName(),
        Colours::lightblue,
        DocumentWindow::allButtons)
{
    centreWithSize(400, 200);
    setResizable(true, false);
    setVisible(true);
}
//------------------------------------------------------------------------
TMainForm::~TMainForm()
{
}
//------------------------------------------------------------------------
void TMainForm::closeButtonPressed()
{
    JUCEApplication::getInstance()->systemRequestedQuit();
}
//------------------------------------------------------------------------
```

Внешний вид такого окна представлен на рисунке 2.13.

Рисунок 2.13. Окно с изменяемыми размерами (в строку заголовка включены кнопки «свернуть окно», «распахнуть окно» и «закрыть окно»)

В качестве первого параметра конструктор класса `DocumentWindow` принимает следующие параметры: строку — название приложения, информацию о цвете фона окна, флаги кнопок заголовка окна и булеву переменную добавления на рабочий стол.

В нашем классе (`TMainForm` — листинг 2.6) с помощью статического метода `static JUCEApplication* JUCE_CALLTYPE JUCEApplication::getInstance() noexcept` мы получаем ссылку на экземпляр приложения и вызываем метод его `getApplicationName()`, возвращающий строку — название программы.

В качестве второго параметра принимается объект класса `Colour` (передаваемый по ссылке), который определяет цвет, которым будет залит фон окна. Дабы не создавать объект класса `Colour`, можно воспользоваться набором предопределённых цветов `Colours` (см. приложение 1).

Третий параметр конструктора класса `DocumentWindow` — целочисленная переменная, флаг включения кнопок заголовка окна. Класс `DocumentWindow` включает перечислимый тип `TitleBarButtons`, указывающий какие именно кнопки, должны быть включены в строку заголовка окна (листинг 2.7):

Листинг 2.7. Перечислимый тип TitleBarButtons

```
enum TitleBarButtons
{
    minimiseButton = 1,
    maximiseButton = 2,
    closeButton = 4,
    allButtons = 7
}
```

Как видно из листинга 2.6, мы включили в строку заголовка нашего класса окна все кнопки. В том случае, если нам требуется окно неизменяемого размера, мы можем переписать конструктор нашего класса окна следующим образом (листинг 2.8)

Листинг 2.8. Конструктор класса TMainForm (окно с неизменяемыми размерами)

```
#include "TMainForm.h"
//--------------------------------------------------------------------------
TMainForm::TMainForm() : DocumentWindow(
        JUCEApplication::getInstance()->getApplicationName(),
        Colours::lightblue,
        DocumentWindow::minimiseButton | DocumentWindow::closeButton)
{
    centreWithSize(400, 200);
    // Запрещаем изменение размеров окна посредством растягивания мышью
    setResizable(false, false);
    setVisible(true);
}
//--------------------------------------------------------------------------
TMainForm::~TMainForm()
{
}
//--------------------------------------------------------------------------
void TMainForm::closeButtonPressed()
{
    JUCEApplication::getInstance()->systemRequestedQuit();
}
//--------------------------------------------------------------------------
```

Внешний вид такого окна представлен на рисунке 2.14.

Рисунок 2.14. Окно с неизменяемыми размерами (в строку заголовка включены кнопки «свернуть окно» и «закрыть окно»)

Заметим, что число кнопок заголовка окна (а также их положение) можно изменить вызовом метода `void DocumentWindow::setTitleBarButtonsRequired(int requiredButtons, bool positionTitleBarButtonsOnLeft)`. Если последний аргумент функции принимает значение `true`, кнопки будут располагаться не у правого края заголовка окна, а у левого (листинг 2.9, рисунок 2.15).

Листинг 2.9. Конструктор класса TMainForm (окно с неизменяемыми размерами, кнопки расположены у левого края заголовка)

```
#include "TMainForm.h"
//--------------------------------------------------
TMainForm::TMainForm() : DocumentWindow(
        JUCEApplication::getInstance()->getApplicationName(),
        Colours::lightblue,
        DocumentWindow::allButtons)
{
    centreWithSize(400, 200);
    // Запрещаем изменение размеров окна посредством растягивания мышью
    setResizable(false, false);
    setTitleBarButtonsRequired(minimiseButton |
DocumentWindow::closeButton,
                               true);
    setVisible(true);
}
//--------------------------------------------------
TMainForm::~TMainForm()
{
}
//--------------------------------------------------
void TMainForm::closeButtonPressed()
{
    JUCEApplication::getInstance()->systemRequestedQuit();
}
//--------------------------------------------------
```

В качестве последнего параметра конструктор класса принимает булеву переменную `bool addToDesktop`. В случае, если переменная принимает значение true, окно автоматически добавляется на рабочий стол, в противном случае вы можете использовать окно в качестве дочернего. По умолчанию переменная принимает значение `true`.

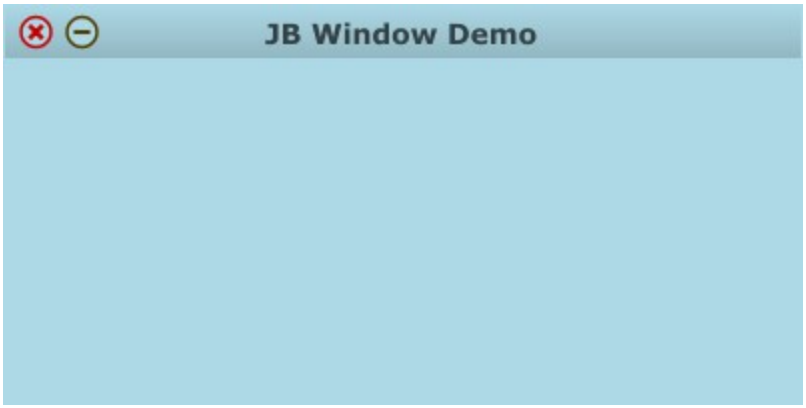


Рисунок 2.15. Окно с атипичным расположением кнопок заголовка

Класс `DocumentWindow` наследует классы компонента (`Component`, см. главу 3) и окна с изменяемыми размерами (`ResizableWindow`). В конструкторе класса `TMainForm` (листинг 2.6) мы вызываем следующие методы, унаследованные от класса `DocumentWindow`:

- `void Component::centreWithSize(int width, int height)` — задаёт линейные размеры компонента (ширина и длина) и помещает его в центре родительского компонента. В том случае, когда родительского компонента нет (наш случай), окно помещается в центр рабочего стола.
- `void ResizableWindow::setResizable(bool shouldBeResizable, bool useBottomRightCornerResizer)` — в зависимости от принимаемых параметров показывает, может ли пользователь изменять размеры окна потягиванием мыши (да, если `shouldBeResizable` принимает значение `true`) и следует ли отображать виджет-ресайзер в правом нижнем углу окна (если виджет отображается, то пользователь может изменять размеры окна посредством потягивания мышью за него, но не за границу окна).
- `virtual void Component::setVisible(bool shouldBeVisible)` — в случае, если аргумент принимает значение true, метод делает окно видимым и доступным для манипуляций пользователя.

Вид окна, в котором все параметры принимают значения `true`, показан на рисунке 2.16.

В JUCE возможно управление не только положением и количеством кнопок заголовка окна, но и его текстом (надписью). По умолчанию текст заголовка окна ориентирован по центру. В том случае, если мы хотим, чтобы он был выровнен по левому краю, необходимо в конструкторе

класса окна вызвать метод

```
void DocumentWindow::setTitleBarTextCentred(bool textShouldBeCentred),
```

присвоив параметру функции значение `false` (рисунок 2.17).

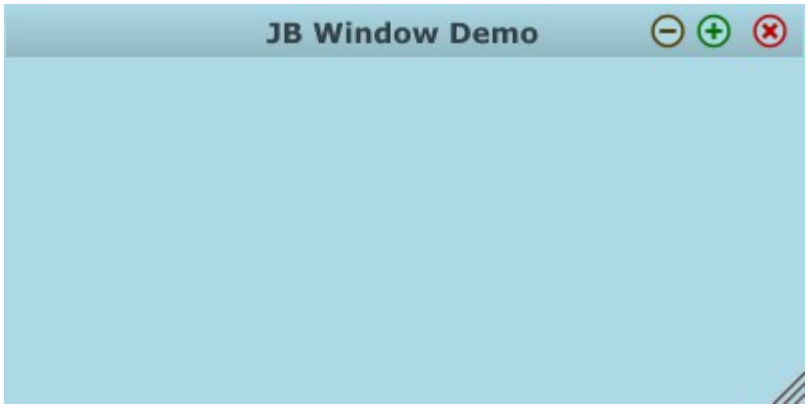


Рисунок 2.16. Окно с изменяемыми размерами (размеры изменяются с помощью специального виджета в его правом нижнем углу)

Рисунок 2.17. Приложение JUCE с выравниванием заголовка окна по левому краю

Отметим, что JUCE позволяет использовать не только собственный заголовок окна, но и заголовок окна (а также рамку) целевой операционной системы. Для этого используется метод `void TopLevelWindow::setUsingNativeTitleBar(bool useNativeTitleBar)` (листинг 2.10, рисунок 2.18).

Листинг 2.10. Конструктор класса TMainForm (окно с нативным заголовком окна целевой операционной системы)

```
#include "TMainForm.h"
//--------------------------------------------------------------------------
TMainForm::TMainForm() : DocumentWindow(
        JUCEApplication::getInstance()->getApplicationName(),
        Colours::lightblue,
        DocumentWindow::allButtons)
{
    // Позволяет использовать нативную (характерную для целевой ОС)
    // строку заголовка
    setUsingNativeTitleBar(true);
    centreWithSize(400, 200);
    setResizable(true, false);
    setVisible(true);
}
//--------------------------------------------------------------------------
TMainForm::~TMainForm()
{
}
//--------------------------------------------------------------------------
void TMainForm::closeButtonPressed()
{
    JUCEApplication::getInstance()->systemRequestedQuit();
}
//--------------------------------------------------------------------------
```

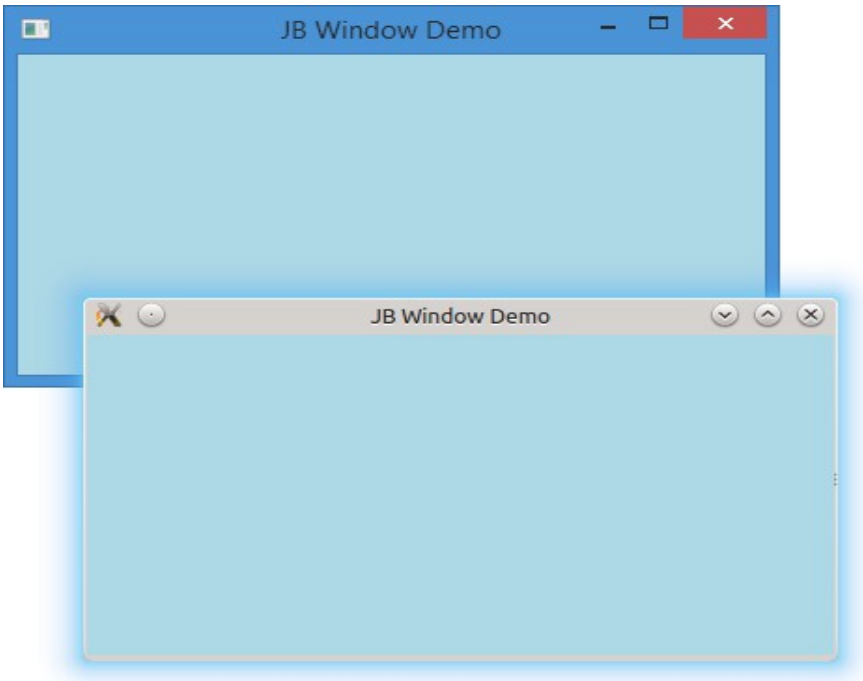


Рисунок 2.18. Приложения JUCE с заголовком и рамкой окна, характерными для Windows 8.1 и KDE 4

Внешний вид элементов управления приложений JUCE, включая кнопки заголовка окна, может быть настроен с использованием класса `LookAndFeel`. Всего в JUCE существует три стандартных варианта отображения контролей (`LookAndFeel_V1`, `LookAndFeel_V2` и `LookAndFeel_V3`). По умолчанию в 4-й версии JUCE используется стиль `LookAndFeel_V3`, для которого характерны плоские кнопки (см., например, рисунок 2.17). Автору настоящей книги больше нравятся выпуклые кнопки старого стиля (`LookAndFeel_V2`). Для того, чтобы переключиться на старый стиль отображения управляющих элементов, необходимо объявить объект класса `LookAndFeel_V1` или `LookAndFeel_V2` в качестве члена класса главного окна (листинг 2.11).

Листинг 2.11. Файл TMainForm.h

```
#ifndef _TMainForm_h_
#define _TMainForm_h_
//--------------------------------------------------------------------------
#include "../JuceLibraryCode/JuceHeader.h"
//--------------------------------------------------------------------------
class TMainForm : public DocumentWindow
{
public:
    TMainForm();
    ~TMainForm();

    void closeButtonPressed();

private:
    // Объект, отвечающий за стиль отображения элементов окна
    LookAndFeel_V2 Look2;
    //LookAndFeel_V1 Look1;
};
//--------------------------------------------------------------------------
#endif
```

Установить новый внешний вид окна можно с помощью метода `void Component::setLookAndFeel(LookAndFeel* newLookAndFeel)` (листинг 2.12).

Листинг 2.12. Файл TMainForm.cpp

```
#include "TMainForm.h"
//--------------------------------------------------------------------------
TMainForm::TMainForm() : DocumentWindow(
        JUCEApplication::getInstance()->getApplicationName(),
        Colours::lightblue,
        DocumentWindow::allButtons)
{
    centreWithSize(400, 200);
```

```
    setResizable(true, false);
    setVisible(true);

    // Я люблю выпуклые кнопки старого стиля! ;)
    setLookAndFeel(& Look2);
    //setLookAndFeel(& Look1);
}
//------------------------------------------------------------------------
TMainForm::~TMainForm()
{
}
//------------------------------------------------------------------------
void TMainForm::closeButtonPressed()
{
    JUCEApplication::getInstance()->systemRequestedQuit();
}
//------------------------------------------------------------------------
```

Внешний вид приложения JUCE, использующего стиль `LookAndFeel_V2`, показан на рисунке 2.19.

Рисунок 2.19. Окно с кнопками заголовка в стиле LookAndFeel_V2

Внешний вид приложения JUCE, использующего стиль `LookAndFeel_V1`, показан на рисунке 2.20.

Рисунок 2.20. Окно с кнопками заголовка в стиле LookAndFeel_V1

Глава 3. Компонентная модель JUCE. Общие свойства компонентов

JUCE написана с помощью современной реализации языка C++ без включения устаревших C-подобных конструкций и основывается на объектно-ориентированной компонентной модели. Это значит, что программа, использующая JUCE, состоит из компонентов — независимых модулей программного кода, предназначенных для выполнения определённых функций и повторного использования, подключаемых динамически на этапе выполнения программы. Взаимодействие компонентов JUCE осуществляется за счёт слушателей (listeners), которые отслеживают события компонентов (изменение размера, щелчок пользователя в пределах границ компонента и т. п.). В соответствии с парадигмой компонентной модели расширение функциональности приложения достигается путём создания новых компонентов на основе существующих. При этом новые компоненты наследуют лишь интерфейсы, но не реализацию ранее созданных. Мы с этим столкнулись в главе 2, когда создавали класс главного окна нашей программы (`TMainForm`), унаследовав его от класса `DocumentWindow`, который, в свою очередь, унаследован от `ResizableWindow` и `Component`.

Также, в соответствии с компонентной моделью, в JUCE широко используется множественное наследование, когда класс-потомок имеет более одного класса-родителя.

В предыдущей главе мы создали простейшее оконное приложение с изменяемыми размерами и кнопками строки заголовка. Несмотря на то, что практическая ценность этой программы нулевая, она способна реагировать на действия пользователя за счёт класса главного окна `TMainForm`. Если проследить иерархию нашего класса, то она будет следующей: `Component` → `TopLevelWindow` → `ResizableWindow` → `DocumentWindow` → `TMainForm`. Таким образом, в основе класса главного окна лежит класс `Component`. То же самое можно сказать про любой элемент управления и отображения информации JUCE-программы.

Любой компонент может выступать в качестве контейнера для других компонентов. Включение в состав компонента-контейнера дочернего компонента осуществляется с помощью функции `void Component::addChildComponent(Component* child, int zOrder = -1)`, где `child` — добавляемый компонент, а `zOrder` — индекс в списке дочерних компонентов, в который будет внедрён новый компонент. В том случае, если индекс равен -1, компонент будет внедрён в начало списка, а если 0, то в его конец.

Заметим, что после добавления в список контейнера дочерний компонент остаётся невидимым, пока не будет вызван его метод `virtual void Component::setVisible(bool shouldBeVisible)`. Поэтому удобнее использовать другую функцию, `void Component::addAndMakeVisible(Component* child, int zOrder = -1)`, которая не только добавляет в контейнер дочерний компонент, но и делает последний видимым.

Также заметим, что главное окно приложения обычно включает так называемый главный компонент или компонент содержимого (content component). Его включают в класс главного окна способом, несколько отличным от вышеописанного, посредством вызова метода

```
void ResizableWindow::setContentOwned(
                                Component* newContentComponent,
                                bool resizeToFitWhenContentChangesSize);
```

Здесь `newContentComponent` — указатель на добавляемый в окно компонент содержимого. Если аргумент `resizeToFitWhenContentChangesSize` принимает значение `true`, то окно будет сохранять свои размеры таким образом, чтобы соответствовать размерам компонента содержимого. В противном случае новый компонент будет растягиваться таким образом, чтобы занять всё доступное пространство окна.

Ранее использовавшийся для добавления компонента содержимого в главное окно метод

```
void ResizableWindow::setContentComponent(Component* newContentComponent,
                                          bool deleteOldOne = true,
                                          bool resizeToFit = false)
```

в текущей версии библиотеки является устаревшим (deprecated).

Рассмотрим работу с компонентами на простом примере. В качестве основы возьмём пример из главы 2 и добавим в класс нашего главного окна `TMainForm` компонент содержимого — ярлык (класс `Label`), который будет отображать надпись «Привет, мир!» (листинги 3.1 и 3.2).

Листинг 3.1. Файл TMainForm.h

```
#ifndef _TMainForm_h_
#define _TMainForm_h_
//------------------------------------------------------------------------
#include "../JuceLibraryCode/JuceHeader.h"
//------------------------------------------------------------------------
class TMainForm : public DocumentWindow
```

```
{
public:
    TMainForm();
    ~TMainForm();

    void closeButtonPressed();

private:
    // Компонент содержимого
    Label* pHelloLabel;
};
//------------------------------------------------------------------------
#endif
```

Листинг 3.2. Файл TMainForm.cpp

```
#include "TMainForm.h"
//------------------------------------------------------------------------
#define tr(s) String::fromUTF8(s)
//------------------------------------------------------------------------
TMainForm::TMainForm() : DocumentWindow(
        JUCEApplication::getInstance()->getApplicationName(),
        Colours::lightblue,
        DocumentWindow::allButtons)
{
    setResizable(true, false);
    setVisible(true);

    // Динамически создаём ярлык
    pHelloLabel = new Label("Hello Label", tr("Привет, мир!"));
    // Задаём размер ярлыка, равный размеру главного окна
    pHelloLabel->setSize(400, 200);
    // Устанавливаем ярлык в качестве компонента содержимого
    setContentOwned(pHelloLabel, true);
    // Устанавливаем размер и гарнитуру шрифта
    pHelloLabel->setFont(Font(38.0000f, Font::bold));
    // Устанавливаем выравнивание текста
    pHelloLabel->setJustificationType(Justification::centred);
    // Запрещаем редактирование текста ярлыка
    pHelloLabel->setEditable(false, false, false);
    // Устанавливаем цвет фона ярлыка
    pHelloLabel->setColour(Label::backgroundColourId, Colours::lightblue);
    // Устанавливаем цвет текста надписи
    pHelloLabel->setColour(Label::textColourId, Colours::black);

    centreWithSize(getWidth(), getHeight());
}
//------------------------------------------------------------------------
TMainForm::~TMainForm()
{
}
//------------------------------------------------------------------------
void TMainForm::closeButtonPressed()
{
    JUCEApplication::getInstance()->systemRequestedQuit();
```

```
}
//--------------------------------------------------------------------
```

В заголовочном файле TMainForm.h в закрытой части класса главного окна мы объявили указатель на ярлык (`Label* pHelloLabel` — см. листинг 3.1), который впоследствии установили в качестве главного компонента окна (листинг 3.2). Внешний вид приложения показан на рисунке 3.1.

Обратите внимание, что для корректного вывода текста в программе, строка должна передаваться в определённый нами макрос tr(s), который позволяет откомпилировать её в кодировке UTF-8 вне зависимости от текущей системной кодировки.

Использование готовых компонентов позволяет создавать лишь довольно примитивные программы. Для разработки сложных, функциональных приложений необходимо создавать собственные компоненты, что и будет рассмотрено в следующей главе.

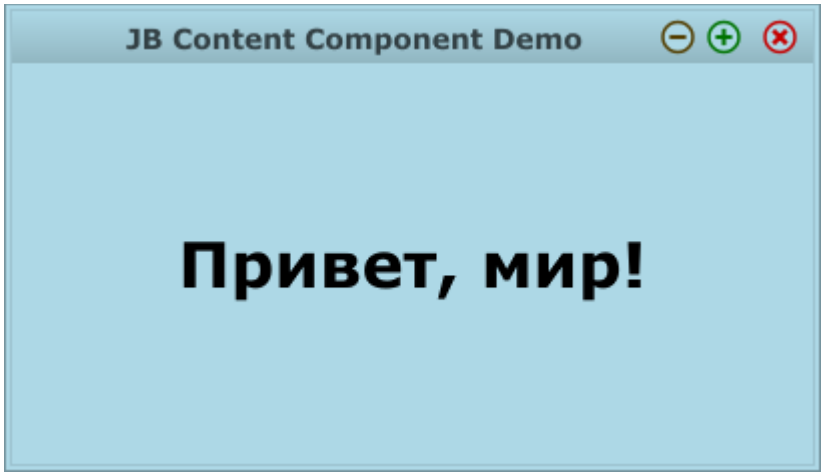


Рисунок 3.1. Приложение JUCE, использующее в качестве компонента содержимого главного окна ярлык с надписью

Глава 4. Разработка собственных компонентов

Графический интерфейс пользователя давно стал стандартом de facto при разработке прикладных программ для различных операционных систем. Компоненты — это основа для его создания. Они делятся на компоненты ввода-вывода текстовой, цифровой и иерархической информации и управляющие. С одним компонентов отображения текстовой информации, ярлыком (`Label`), вы уже познакомились в предыдущей главе.

Как правило, компонент интерфейса программы — это не просто область на экране, отображающая текст и / или изображение; это элемент, реагирующий на события системы и действия пользователя тем или иным образом (генерация сообщений другим компонентам, изменение внешнего вида и размера, выполнение вычислений и т. д. и т. п.). Взаимодействие подобных компонентов основывается на работе передатчиков событий (event broadcaster) и приёмников или слушателей событий (event listener).

Элементы интерфейса, имеющие стандартный внешний вид и выполняющие стандартные действия (кнопки, выпадающие меню и списки, флажки, переключатели и т.п.) называются виджетами — термин, пришедший из Linux. Конечно, «стандартный» внешний вид виджетов JUCE — понятие относительное, поскольку, как упоминалось выше, библиотека не использует элементы управления целевой платформы, а рисует все элементы пользовательского интерфейса самостоятельно с помощью системных функций низкого уровня.

Слово «widget» является шуточным сокращением фразы «which it, gadget» — «эта, как её, штуковина». В Windows системах более привычным является название «элемент управления» (control). Виджеты включаются в компоненты-контейнеры, которые должны «слушать» события, генерируемые первыми, т. е. являться приёмниками (listener). Это достигается наследованием класса компонента-контейнера от нужного класса слушателя (`ButtonListener`, `ComboBoxListener`, `SliderListener` и т. п.). Классы слушателей являются абстрактными и содержат чистые виртуальные (pure virtual) функции, которые в классе компонента-контейнера должны быть переопределены.

Как правило, в качестве компонента-контейнера выступает упоминавшийся в предыдущей главе компонент содержимого (content component), который предоставляет доступ к клиентской части окна приложения. Класс компонента содержимого наследуется от класса `Component` и класса (классов) слушателей.

Вообще, множественное наследование весьма характерно для JUCE,

поскольку довольно большое число классов библиотеки являются абстрактными. Так, например, в отличие от библиотеки Qt, в JUCE невозможно создать таймер (объект класса `Timer`), поэтому для того, чтобы воспользоваться функциями класса, необходимо создать компонент, унаследовав его класс от классов `Component` и `Timer`.

Класс `Component` является основой для всех виджетов, включая окна и диалоги. Сам класс `Component` не является абстрактным, поэтому создание экземпляра данного класса вполне возможно. Однако в «чистом» виде объект класса `Component` бесполезен. В случае, если мы добавим в него какие-либо виджеты с помощью функции `void Component::addAndMakeVisible(Component* child, int zOrder = -1)`, код будет откомпилирован, однако на этапе выполнения возникнет ошибка, связанная с тем, что в JUCE программист должен сам позаботиться об уничтожении дочерних виджетов при завершении работы программы, переопределив деструктор класса компонента-контейнера. Уничтожение дочерних виджетов достигается вызовом функции `void Component::SafePointer<ComponentType>::deleteAndZero()`, которая удаляет из контейнера дочерний компонент и обнуляет указатель на него. Однако для того, чтобы не вызывать эту функцию для каждого из дочерних виджетов, проще воспользоваться функцией `void Component::deleteAllChildren()`, которая удаляет из контейнера все включённые в него компоненты.

Рассмотрим всё вышеизложенное на простом примере. Создадим собственный компонент, который будет включать в себя два дочерних виджета: ярлык (`Label`) и кнопку с текстом (`TextButton`). Мы унаследуем класс нашего компонента от `ButtonListener` для того, чтобы отслеживать щелчок пользователя по кнопке, и переопределим функцию `virtual void Button::Listener::buttonClicked(Button*)`, которая будет вызывать метод класса приложения `static void JUCEApplication::quit()`, завершающий работу программы (листинги 4.1 и 4.2).

Листинг 4.1. Файл TCentralComponent.h

```
#ifndef _TCentralComponent_h_
#define _TCentralComponent_h_
//------------------------------------------------------------------------
#include "../JuceLibraryCode/JuceHeader.h"
//------------------------------------------------------------------------
// Класс компонента содержимого.
// Наследует класс слушателя кнопок
class TCentralComponent  : public Component,
                           public ButtonListener
{
public:
```

```
    TCentralComponent();
    ~TCentralComponent();

    void paint(Graphics&);
    void resized();
    // Функция, отслеживающая щелчки по кнопке
    void buttonClicked(Button*);

private:
    Label* pHelloLabel;
    TextButton* pCloseButton;

    // Предотвращает создание копии конструктора и оператора =
    TCentralComponent(const TCentralComponent&);
    const TCentralComponent& operator= (const TCentralComponent&);
};
//------------------------------------------------------------------------
#endif
```

Листинг 4.2. Файл TCentralComponent.cpp

```
#include "TCentralComponent.h"
//------------------------------------------------------------------------
#define tr(s) String::fromUTF8(s)
//------------------------------------------------------------------------
TCentralComponent::TCentralComponent() : Component("Central Component")
{
    pHelloLabel = new Label("Hello Label", tr("Привет, мир!"));
    addAndMakeVisible(pHelloLabel);
    pHelloLabel->setFont(Font(38.0000f, Font::bold));
    // Ориентация текста ярлыка - по центру виджета
    pHelloLabel->setJustificationType(Justification::centred);
    // Запрет на редактирование содержимого ярлыка
    pHelloLabel->setEditable(false, false, false);
    // Синий цвет шрифта
    pHelloLabel->setColour(Label::textColourId, Colours::blue);
    pHelloLabel->setColour(TextEditor::backgroundColourId,
Colours::azure);

    pCloseButton = new TextButton("Close Button");
    addAndMakeVisible(pCloseButton);
    pCloseButton->setButtonText(tr("Закрыть"));
    // Устанавливаем в качестве слушателя кнопки
    // сам компонент-контейнер
    pCloseButton->addListener(this);

    setSize (400, 200);
}
//------------------------------------------------------------------------
TCentralComponent::~TCentralComponent()
{
    // Удаляем дочерние виджеты
    // и обнуляем их указатели
    deleteAndZero(pHelloLabel);
```

```
    deleteAndZero(pCloseButton);
}
//------------------------------------------------------------------------
void TCentralComponent::paint(Graphics& Canvas)
{
    Canvas.fillAll(Colours::lightblue);
}
//------------------------------------------------------------------------
void TCentralComponent::resized()
{
    pHelloLabel->setBounds(0, 0, proportionOfWidth(1.0000f),
proportionOfHeight(0.9000f));
    pCloseButton->setBounds(getWidth() - 20 - 100, getHeight() - 35, 100,
25);
}
//------------------------------------------------------------------------
void TCentralComponent::buttonClicked(Button* pButton)
{
    // Если нажата кнопка "Закрыть"...
    if(pButton == pCloseButton)
    {
        // выходим из программы
        JUCEApplication::quit();
    }
}
//------------------------------------------------------------------------
```

Любой компонент (виджет) представляет собой прямоугольную область на экране компьютера, характеризующийся координатами верхне-левого угла, а также длиной и шириной. При изменении размеров главного окна программы (`DocumentWindow`) автоматически изменяются и размеры компонента содержимого, поэтому заботиться об указании его координат нет необходимости. Однако размеры дочерних виджетов компонента содержимого приходится указывать вручную, переопределяя функцию `virtual void Component::resized()` (листинг 4.2). Координаты виджетов задаются функцией `void Component::setBounds(int x, int y, int width, int height)`, где `x` и `y` — координаты верхне-левого угла дочернего компонента относительно верхне-левого угла компонента-родителя, а `width` и `height` — ширина и высота виджета.

Однако задание абсолютных размеров виджетов не всегда удобно, поскольку они оказываются «привязанными» к определённому месту компонента-контейнера, что не всегда выглядит красиво при распахивании окна программы или при растягивании его мышью.

Для того, чтобы задать размеры и позицию виджета относительно компонента родителя в JUCE используются функции-члены класса `Rectangle`: `getWidth()` и `getHeight()`, возвращающие значения ширины и высоты родительского компонента.

Также можно использовать функции `int Component::proportionOfWidth(float proportion) const` и `int Component::proportionOfHeight(float proportion) const`, которые возвращают значения ширины и высоты пропорционально таковым родительского компонента. Так, `proportionOfWidth(1.0000f)` (листинг 4.2) означает, что ширина дочернего виджета будет соответствовать таковой родительского компонента (аналогично использованию функции `getWidth()`), а `proportionOfHeight(0.9000f)` означает, что высота виджета должна составлять 90% высоты родительского компонента.

Теперь добавим экземпляр класса нашего компонента (`TCentralComponent`) в качестве закрытого члена класса главного окна (листинг 4.3).

Листинг 4.3. Файл TMainForm.h

```
#ifndef _TMainForm_h_
#define _TMainForm_h_
//------------------------------------------------------------------------
#include "../JuceLibraryCode/JuceHeader.h"
//------------------------------------------------------------------------
class TCentralComponent;
//------------------------------------------------------------------------
class TMainForm : public DocumentWindow
{
public:
    TMainForm();
    ~TMainForm();

    void closeButtonPressed();

private:
    // Объект, отвечающий за стиль прорисовки виджетов
    LookAndFeel_V2 Look2;

    // Компонент содержимого
    TCentralComponent* pCentralComponent;
};
//------------------------------------------------------------------------
#endif
```

В реализации класса главного окна установим наш компонент в качестве компонента содержимого (листинг 4.4).

Листинг 4.4. Файл TMainForm.cpp

```
#include "TMainForm.h"
```

```
//------------------------------------------------------------------------
#include "TCentralComponent.h"
//------------------------------------------------------------------------
TMainForm::TMainForm() : DocumentWindow(
        JUCEApplication::getInstance()->getApplicationName(),
        Colours::lightblue,
        DocumentWindow::allButtons)
{
    // Динамически создаём экземпляр класса нашего компонента
    pCentralComponent = new TCentralComponent();
    // Устанавливаем его в качестве компонента содержимого
    setContentOwned(pCentralComponent, true);

    // Да, да! Я не люблю плоские кнопки!
    // Закомментируйте строку ниже, чтобы вернуться
    // к стилю версии 3
    setLookAndFeel(& Look2);

    centreWithSize(getWidth(), getHeight());
    setResizable(true, false);
    setVisible(true);
}
//------------------------------------------------------------------------
TMainForm::~TMainForm()
{
}
//------------------------------------------------------------------------
void TMainForm::closeButtonPressed()
{
    JUCEApplication::getInstance()->systemRequestedQuit();
}
//------------------------------------------------------------------------
```

Внешний вид программы представлен на рисунке 4.1.

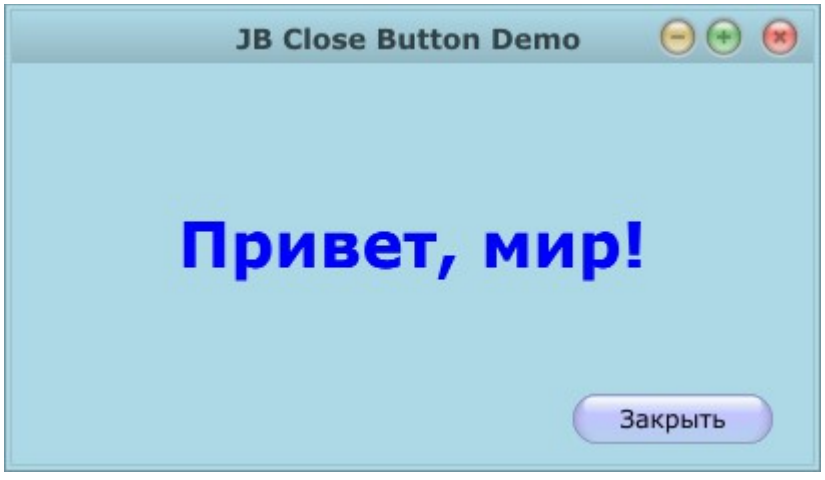


Рисунок 4.1. Приложение JUCE, использующее в качестве компонента содержимого компонент-контейнер, содержащий ярлык и кнопку

Поскольку компонент содержимого, как правило, является основным модулем, отвечающим за функциональность приложения, в дальнейшем в этой книге мы будем приводить код только этого класса.

Глава 5. Визуальное проектирование компонентов

Задавать размеры и положение дочерних виджетов относительно родительского компонента — занятие не из лёгких. Для облегчения этой задачи в JUCE включён специальный инструмент, включённый в состав среды Introjucer / Projucer, дающий возможность визуального проектирования создаваемого компонента и включённых в него дочерних виджетов с последующей генерацией C++ кода.

Существует несколько способов для того, чтобы включить в проект Introjucer / Projucer компонент с возможностью визуального проектирования положения, формы и цвета его субкомпонентов. Здесь мы рассмотрим пример простой программы, требующей минимального написания кода.

Запустите Introjucer или Projucer, закройте старый проект, если он был открыт, и выберите в меню **File → New Project...**

Нажмите кнопку «GUI Application» в правом верхнем углу окна Introjucer / Projucer (см. рисунок 2.1).

Создайте проект «GUIExample1» в папке проектов JUCE так, как это было описано ранее. Проследите, чтобы в выпадающем списке «Files to Auto-Generate:» было выбрано значение «Create a Main.cpp file and a basic window» (рисунок 5.1) для того, чтобы был сгенерирован код главного окна и компонента содержимого.

Перейдите на вкладку «Files» вновь созданного проекта. В группе «Source» вы увидите три автоматически созданных файла:

- Main.cpp, который содержит код классов приложения и главного окна, а также макрос, запускающий программу;
- MainComponent.h — заголовочный файл, содержащий объявление класса компонента содержимого (ссылка на него уже включена в объявление класса главного окна);
- MainComponent.cpp — реализация класса компонента содержимого.

К сожалению, сгенерированный код не поддерживает визуальное проектирование, т. к. не содержат специальной разметки. Однако можно заменить исходные тексты класса компонента содержимого другими с тем же именем, но с включённой поддержкой визуального проектирования компонентов.

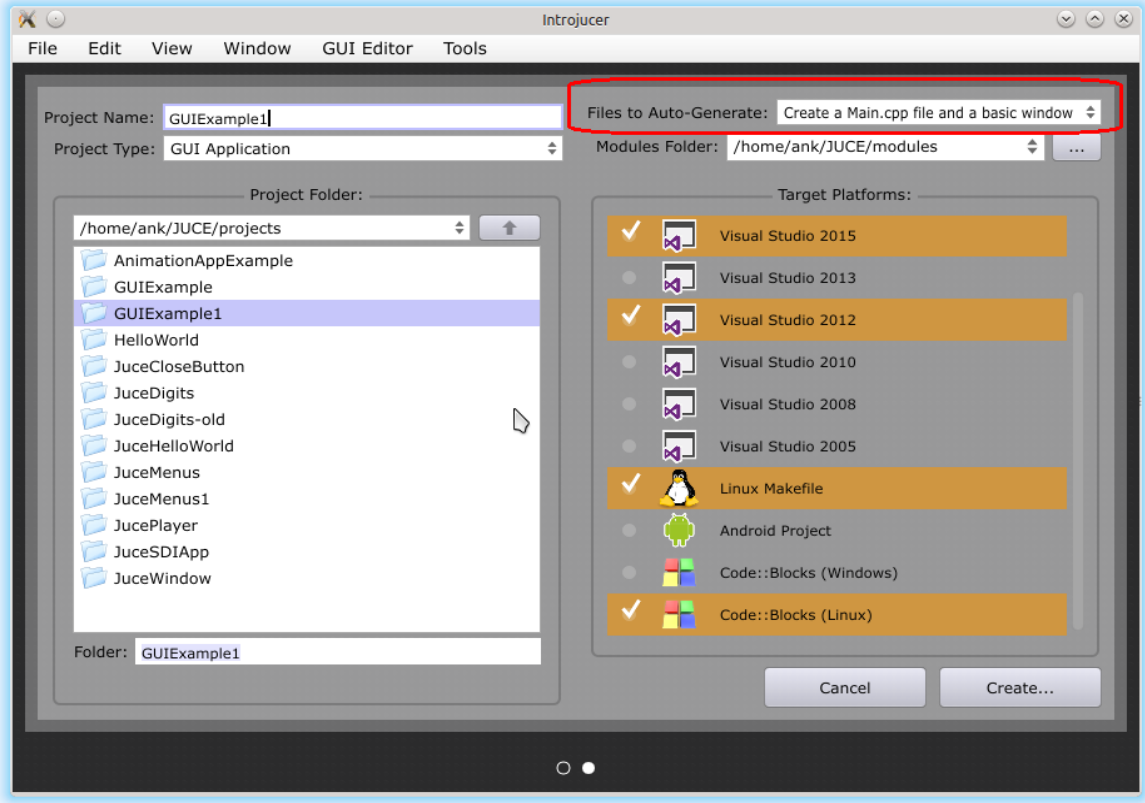


Рисунок 5.1. Создание проекта Introjucer с автоматической генерацией кода приложения с графическим интерфейсом пользователя

Для этого щёлкните правой кнопкой мыши по группе «Source» (или по любому из файлов в её составе) и выберите в контекстном меню пункт «Add New GUI Component...» (рисунок 5.2).

Сохраните исходные тексты создаваемого компонента под именами «MainComponent.h» и «MainComponent.cpp», заменив ими ранее сгенерированные файлы.

Выберите мышью имя файла «MainComponent.cpp» в группе «Source».

В правой части Introjucer / Projucer появится виджет с вкладками, отображающий информацию о создаваемом компоненте. На вкладке Class в списке General class settings будет отображены: имя класса компонента (Class name), название компонента (Component name), если оно будет отличаться от названия класса, родительский класс / классы (Parent classes), параметры конструктора (Constructor params), начальные размеры виджета (Initial width и Initial height), равные по умолчанию 600 и 400.

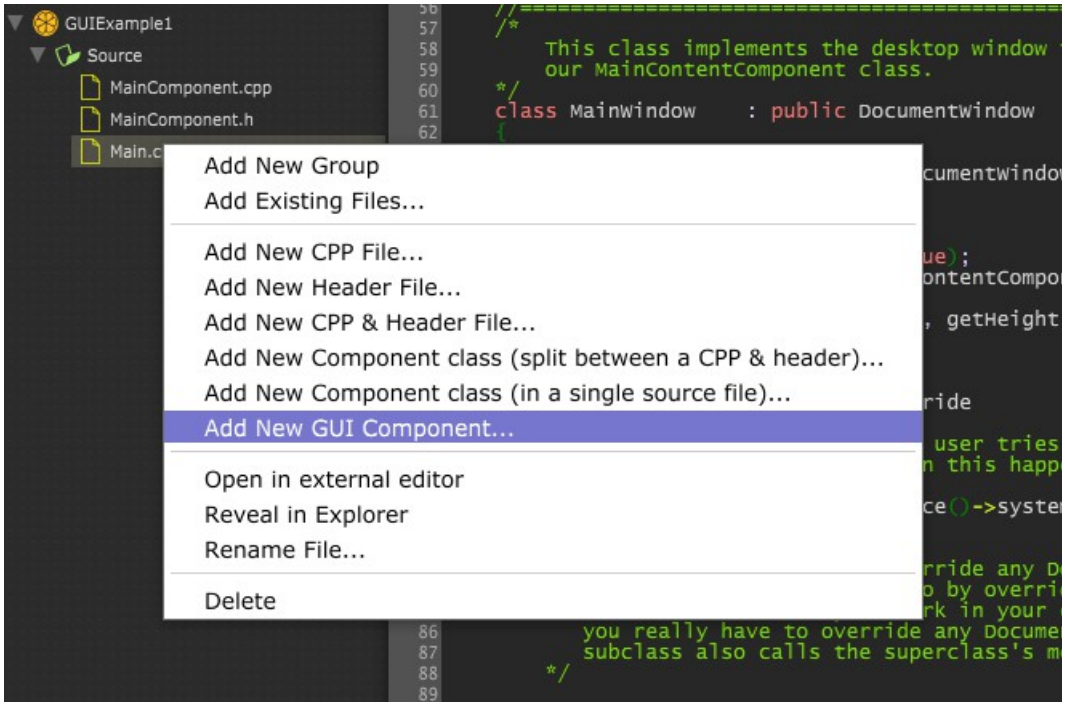


Рисунок 5.2. Добавление компонента с возможностью визуального проектирования интерфейса в проект Introjucer / Projucer

В выпадающем списке Fixed size, можно выбрать, будут ли размеры виджета подстраиваться под размеры рабочего пространства главного окна Introjucer / Projucer (Resize component to fit workspace) или же создаваемый компонент будет с неизменяемыми размерами (Keep component size fixed — рисунок 5.3).

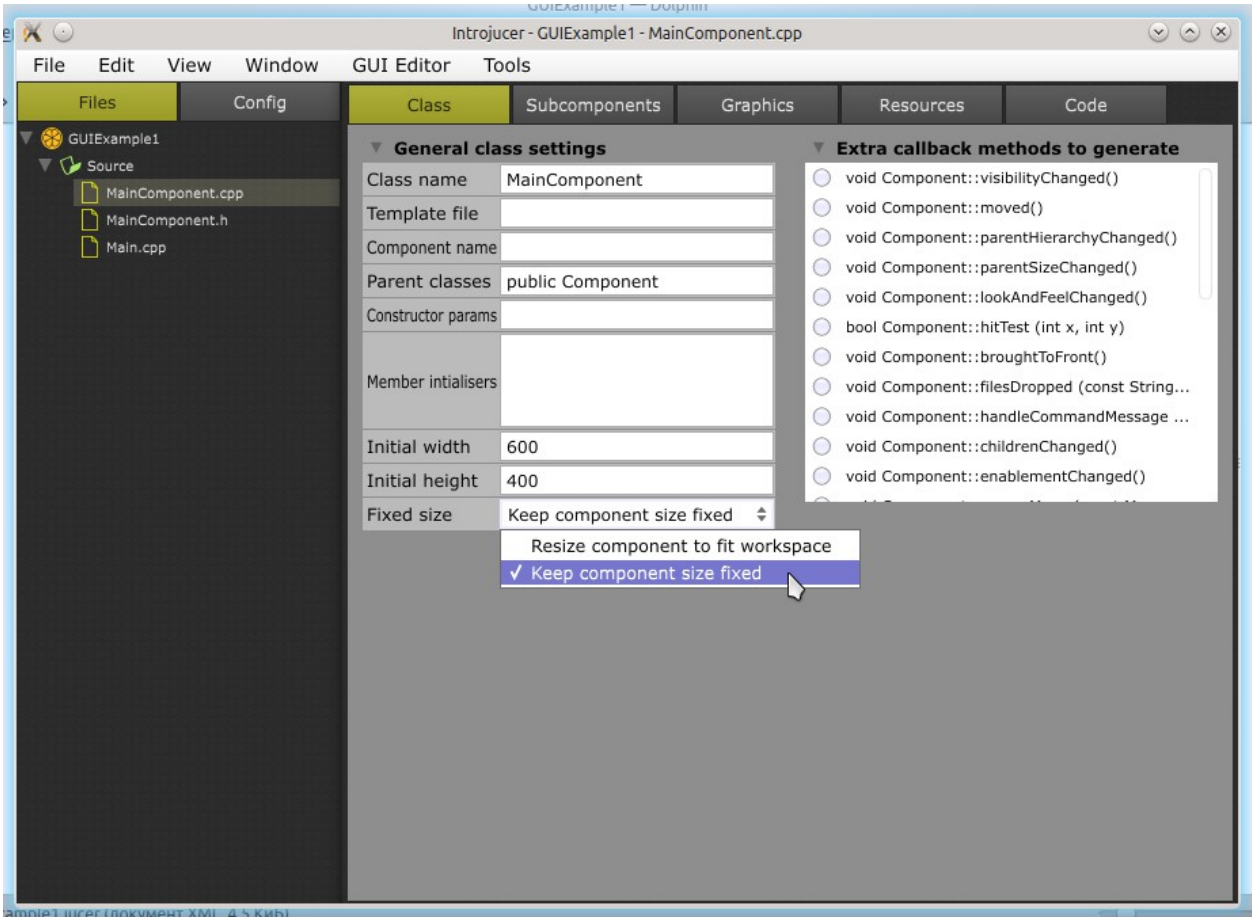


Рисунок 5.3. Визуальный редактор среды Introjucer. Редактирование свойств класса компонента

На вкладке Subcomponents мы, собственно, можем спроектировать внешний вид нашего компонента, размещая на его поверхности различные виджеты и редактируя их свойства в списке справа (для этого субкомпонент необходимо выделить мышью — рисунок 5.4).

Белое поле с координатной сеткой — это и есть заготовка нашего будущего компонента, на который можно добавлять стандартные или пользовательские виджеты посредством контекстного меню. В нём доступны следующие стандартные компоненты: Text Button (кнопка с текстом), Toggle Button (радиокнопка), Slider (ползунок), Label (ярлык), Text Editor (поле ввода), Combo Box (выпадающий список), Group Box (групповой блок), Hyperlink Button («кнопка» - гиперссылка), Viewport (поле просмотра), Tabbed Component (компонент с вкладками), Tree View (древовидный список), Image Button (кнопка с изображением).

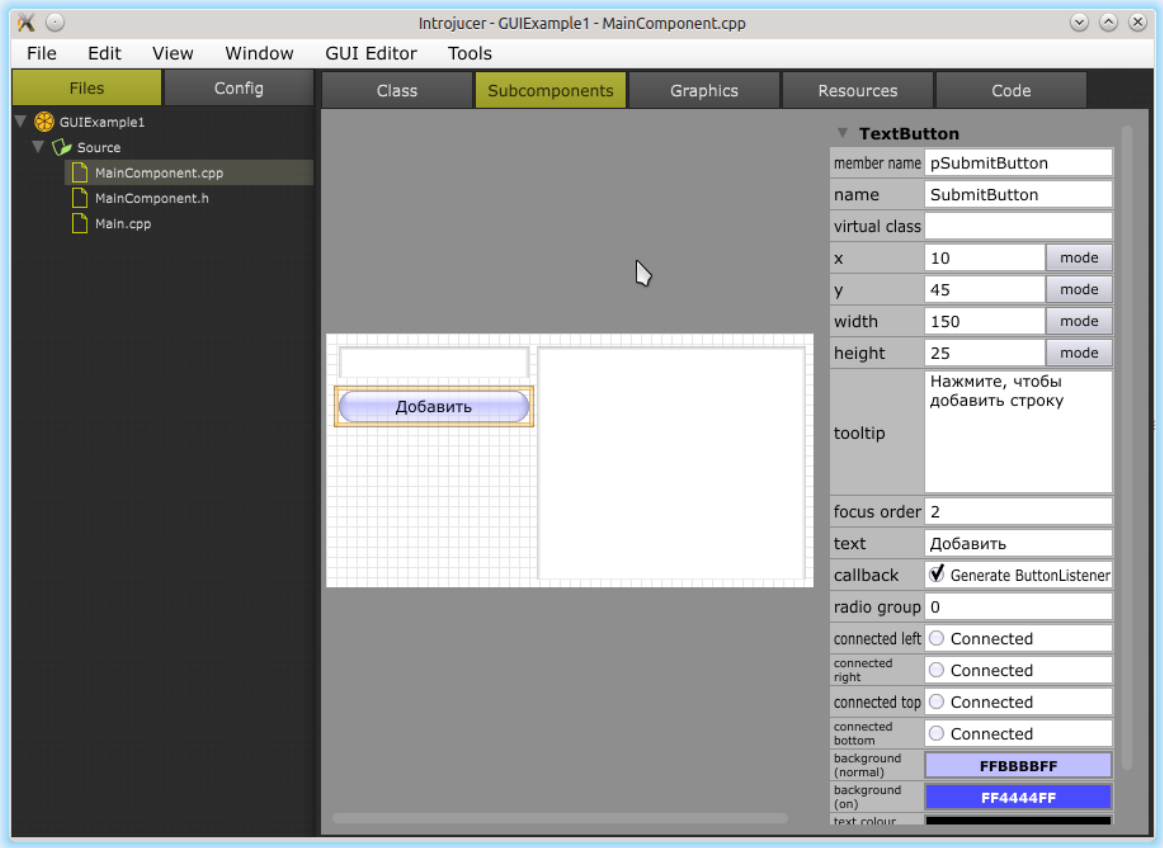


Рисунок 5.4. Редактирование свойств дочерних (стандартных) компонентов проектируемого компонента-контейнера в визуальном редакторе среды Introjucer

Редактируемые свойства добавляемого дочернего компонента (виджета) становятся доступны в поле справа при выделении компонента мышью (рисунок 5.4). Большинство этих свойств интуитивно понятны, и для человека, хотя бы в небольшой степени владеющего английским языком, не составит труда в них разобраться.

Основным свойством дочерних компонентов является «member name» - имя объекта класса виджета. Свойство «name» предназначено для нужд отладки и может быть оставлено пустым.

Геометрия виджета задаётся свойствами его координат: x и y — расстояние в пикселях от верхне-левого угла компонента-контейнера, width — ширина и height — длина дочернего компонента. По умолчанию все координаты задаются в абсолютных цифрах. Задавать геометрию дочерних виджетов подобным образом имеет смысл в том случае, если размеры родителя во время работы программы остаются неизменными. В

противном случае размеры субкомпонентов должны быть относительны размеров контейнера и друг друга.

Рассмотрим это на примере нашего приложения, внешний вид которого представлен на рисунке 5.4. Добавим на поле проектирования компонента 2 поля ввода (Text Editor) и одну кнопку с надписью (Text Button).

Назовём их `pStringEdit`, `pStringsList` и `pSubmitButton`.

Логика работы приложения будет следующей: при нажатии на кнопку `pSubmitButton` текст, содержащийся в поле `pStringEdit` (в том числе пустая строка) будет добавляться в список строк поля `pSubmitButton`.

Для выравнивания субкомпонентов (поле ввода `pStringEdit` и кнопка `pSubmitButton`) по левому краю родителя установите значение их свойства `x` в одинаковое абсолютное значение (например, 10). Значение свойства y будет прирастать на размер самих компонентов и выбранного нами размера между ними.

Если координаты верхне-левого угла, длина и ширина предыдущих компонентов были статичными, т. е. были выражены абсолютными числами и не менялись при изменении размеров родительского компонента, то координаты поля ввода `pStringsList` в этом случае будут относительны размера компонента-контейнера, т. к. должны «подстраиваться» под родителя.

Для того, чтобы задать режим вычисления любой из координат (x, y), а также длины и ширины, следует нажать кнопку mode рядом с полем свойства компонента (рисунок 5.4). После нажатия на кнопку появляется меню, в котором можно выбрать, как именно будет вычисляться та или иная величина, описывающая размер и положение компонента в данный момент времени.

«Привяжем» верхне-левый угол поля `pStringsList` к левому краю компонента-контейнера. Для координаты `x` в меню отметьте два пункта: «Absolute distance from the left of parent» и «Anchored at left of component».

Для того, чтобы длина и ширина поля ввода `pStringsList` изменялись при растягивании компонента-контейнера, нужно изменить свойства `width` и `height`. Это возможно сделать различными способами. Например, для первого свойства выберем в меню «Percentage of width of parent», а для второго, соответственно, «Percentage of height of parent». Это приведёт к тому, что дочерний виджет будет всегда занимать определённый процент площади родителя.

Можно также сделать так, чтобы длина и ширина дочернего компонента равнялась таковым родителя за вычетом некого числа. Для этого в контекстном меню свойств «width» и «height» поля ввода

`pStringsList` выберите соответственно «Subtracted of width of parent» и «Subtracted of height of parent». Мышью установите желаемый размер компонента.

Выбрав в меню Introjucer / Projucer «GUI Editor — Test component...», можем увидеть нашу компоновку в действии и убедиться, что всё работает, как было задумано (родительский компонент должен быть выбран мышью).

Перейдя на вкладку «Code» визуального редактора среды Introjucer / Projucer, можно посмотреть сгенерированный программой код.

Часть кода необходимо откорректировать вручную. Перейдите к реализации класса «MainWindow». Исправьте строку

```
setContentOwned (new MainContentComponent(), true);
```

на

```
setContentOwned (new MainComponent(), true);
```

Хотя по умолчанию заголовок главного окна нашей программы имеет все кнопки, включая кнопку «развернуть», изменить размер окна протягиванием мышью невозможно, в чём можно убедиться, откомпилировав приложение.

Чтобы исправить это, добавьте в реализацию класса «MainWindow» строку

```
setResizable(true, false);
```

Сохраните изменения в файле «Main.cpp» и проекте.

Файл реализации созданного компонента можно открыть для редактирования в Introjucer повторно; при этом во вкладке Subcomponents откроется ранее созданная компоновка. Это достигается включением в код реализации специальных комментариев, содержащих необходимые метаданные, которые распознаются the jucer. Эти комментарии имеют следующий вид:

```
//[UserButtonCode_textButton] -- add your button handler code here..
//[/UserButtonCode_textButton]#
```

Собственный код программист может (и должен) вставлять между этими комментариями. Любые изменения, сделанные в другом месте, будут потеряны при следующем сохранении компонента в Introjucer / Projucer.

Помимо добавления новых виджетов, контекстное меню, появляющееся при щелчке правой кнопкой мыши по поверхности компонента-контейнера в режиме его визуального проектирования, позволяет редактировать цвет фона. Выберите в контекстном меню пункт «Edit background graphics». Будет совершён автоматический переход на вкладку «Graphics». Ранее добавленные виджеты станут недоступны для редактирования, а справа отобразится группа «Class Properties» с единственным свойством, background. Щёлкните по полю ввода (по умолчанию в нём отображается код белого цвета, FFFFFFFF), а затем выберите в диалоговом окне нужный цвет фона (рисунок 34). Чтобы закрыть диалог, щёлкните мышью по свободному пространству панели «Class Properties». Для завершения выбора цвета фона и возвращения к редактированию свойств компонентов перейдите на вкладку «Subcomponents» (рисунок 5.5).

Кроме того, Introucer / Projucer позволяет легко добавлять в код программы бинарные ресурсы. Чаще всего это изображения.

Для использования какого-либо графического файла в создаваемом компоненте необходимо выделить мышью файл реализации класса компонента (в нашем примере это MainComponent.cpp), перейти на вкладку Resources в правой части визуального редактора Introjucer / Projucer и нажать на кнопку «Add new resource...» (рисунок 5.6).

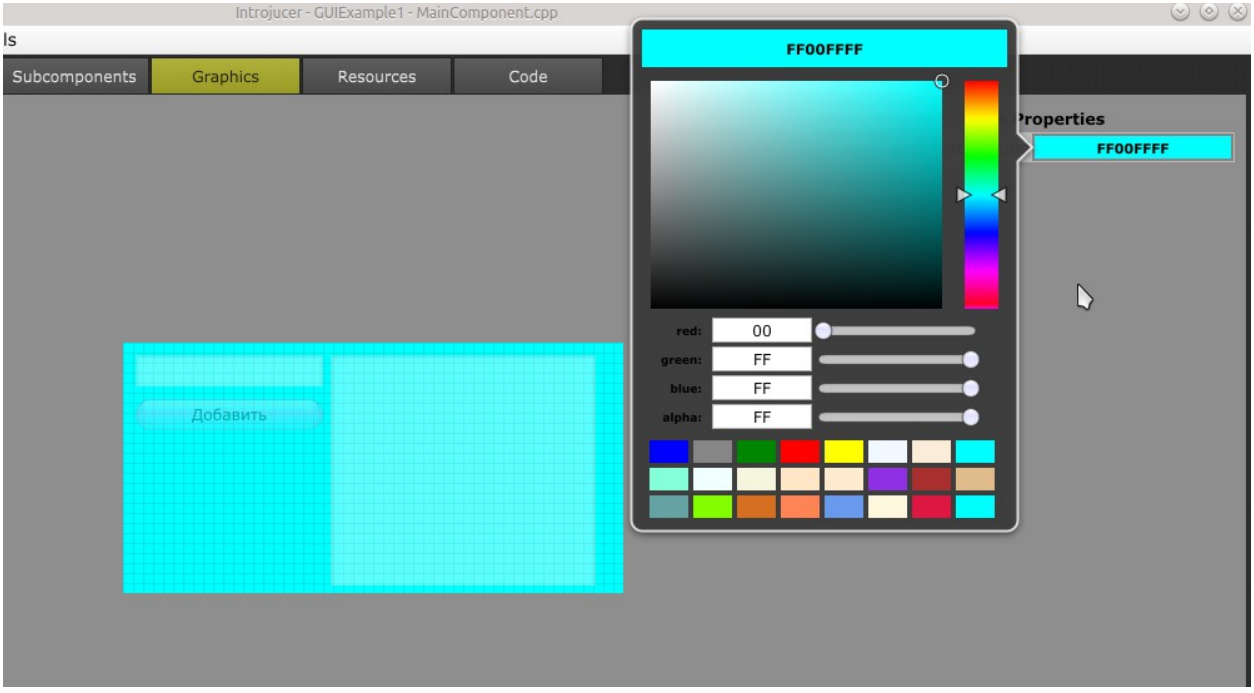


Рисунок 5.5. Выбор цвета фона компонента

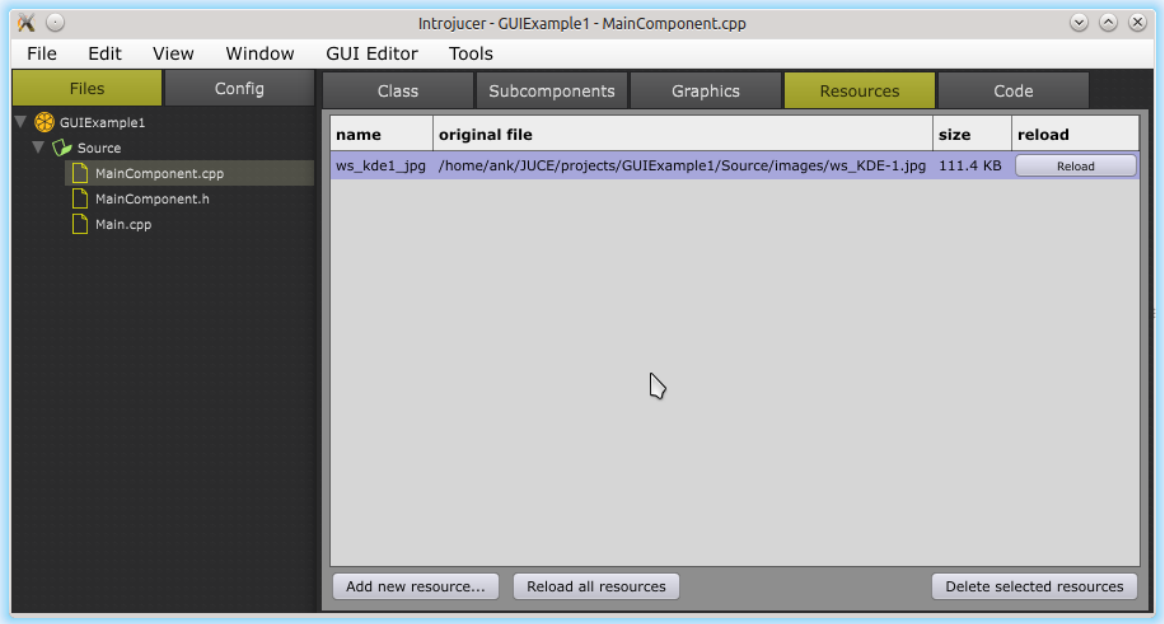


Рисунок 5.6. Добавление изображения в виде бинарного ресурса создаваемого компонента

В диалоговом окне выбираем необходимое изображение. Файл добавляется в список (при необходимости его можно перезагрузить, нажав кнопку «Reload»; при сохранении компонента бинарные данные из него переводятся в форму обычного массива C++ (static const unsigned char) и вставляются в код. В классе компонента автоматически создаётся переменная класса `Image*`, которая инициализируется в конструкторе данными из этого массива. Эту переменную можно использовать в коде программы для доступа к данным ресурса.

Добавим в ресурсы нашего приложения фоновое изображение, загруженное с сайта Wallpaper Seek (http://www.wallpaperseek.com/kde-wallpapers_w4788.html, рисунок 5.6).

Имея набор графических файлов, загруженных в виде ресурсов, можно их использовать для изменения фона создаваемого компонента. Это делается в редакторе фоновой графики Introjucer / Projucer. Ранее мы уже его использовали для изменения цвета фона компонента. Редактор можно вызвать, как мы это делали раньше, через контекстное меню либо (что проще) перейти на вкладку Graphics визуального редактора среды Introjucer / Projucer.

Добавим фоновое изображение, которое будет отображаться под нашими виджетами. Выберем из контекстного меню пункт «New Image».

Зададим размеры объекта класса Image следующим образом: x — 0, y — 0, width — Percentage of width of parent, 100%, height — Percentage of height of parent, 100%. Тем самым изображение займёт всю доступную площадь компонента. В выпадающем списке «image source» отображаются все графические файлы, добавленные ранее в компонент в качестве ресурсов. Если этого не было сделано, то можно добавить новый, выбрав в выпадающем списке пункт «-- create a new image resource --».

После выбора необходимого ресурса изображение отображается на компоненте (рисунок 5.7).

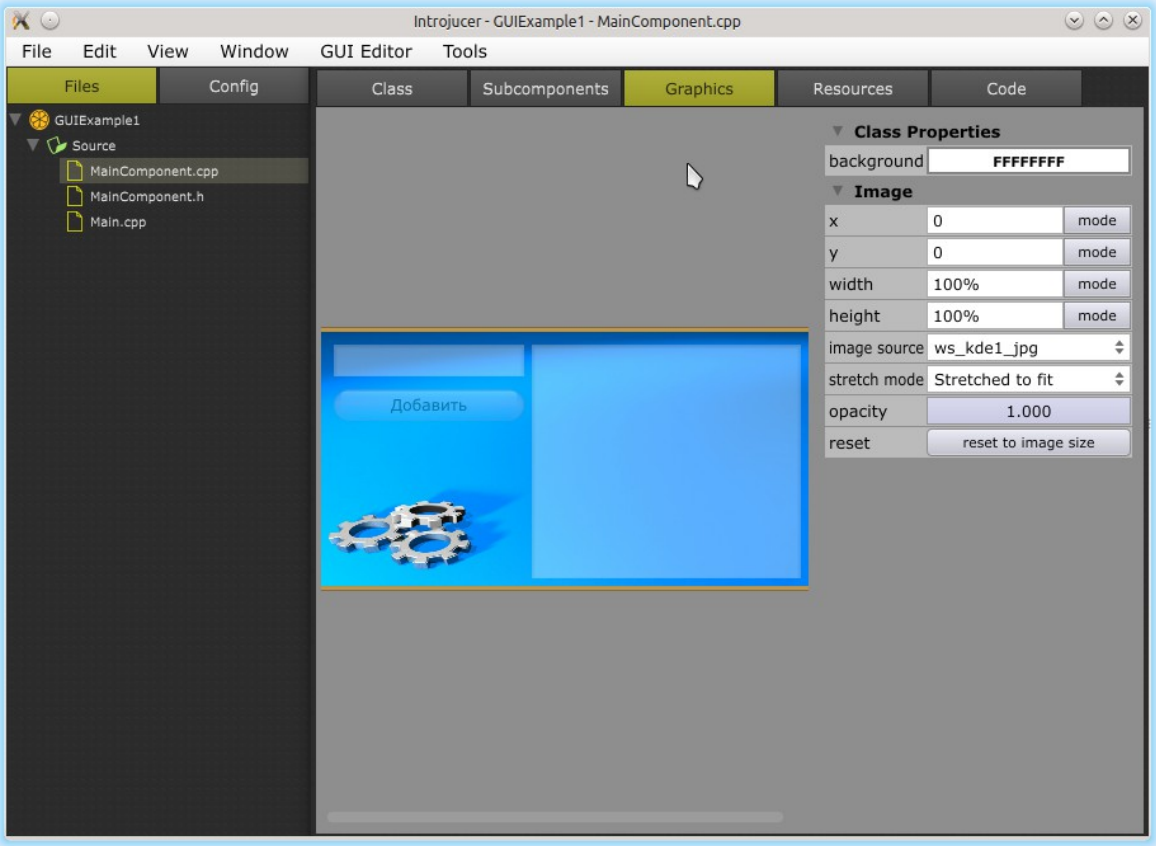


Рисунок 5.7. Добавление фонового изображения в компонент, редактируемый в Introjucer

Завершающим этапом работы над нашим приложением является добавление его функциональности. Эту работу нам облегчит то, что Introjucer / Projucer уже добавила заготовку функции-обработчика щелчка по кнопке `pSubmitButton`:

```
void MainComponent::buttonClicked (Button* buttonThatWasClicked)
```

Следует помнить, что если мы хотим, чтобы наш код не пропал при сохранении изменений в файле MainComponent.cpp, мы должны записать его непосредственно после комментария

```
//[UserButtonCode_pSubmitButton] -- add your button handler code here..
```

Итак, мы добиваемся того, чтобы при нажатии на кнопку `pSubmitButton` текст из поля `pStringEdit` добавлялся в поле `pStringsList`, а первое поле при этом очищалось (листинг 16).

Листинг 5.1. Обработчик события нажатия на кнопку pSubmitButton

```
void MainComponent::buttonClicked (Button* buttonThatWasClicked)
{
    //[UserbuttonClicked_Pre]
    //[/UserbuttonClicked_Pre]

    if (buttonThatWasClicked == pSubmitButton)
    {
        //[UserButtonCode_pSubmitButton] -- add your button handler code
here..
        if(pStringEdit->getText() == String::empty) return;

        pStringsList->insertTextAtCaret(pStringEdit->getText() += "\n");
        pStringsList->setCaretPosition(pStringsList->getCaretPosition() +
1);
        pStringEdit->clear();
        //[/UserButtonCode_pSubmitButton]
    }

    //[UserbuttonClicked_Post]
    //[/UserbuttonClicked_Post]
}
```

Внешний вид работающего приложения показан на рисунке 5.8.

Помимо изображений, в контекстном меню редактора фоновой графики можно выбрать различные графические примитивы: прямоугольник (Rectangle), прямоугольник с закруглёнными углами (Rounded Rectangle), эллипс (Ellipse), градиент (Path), а также текст (Text). Для каждого элемента можно задавать прозрачность, заливку цветом либо в виде линейного или радиального градиента, а также заполнять фигуры изображениями (свойство fill mode → Image Brush; в выпадающем списке fill image выбирается соответствующее изображение из ресурсов). Контрольные точки градиентов можно перетаскивать мышью.

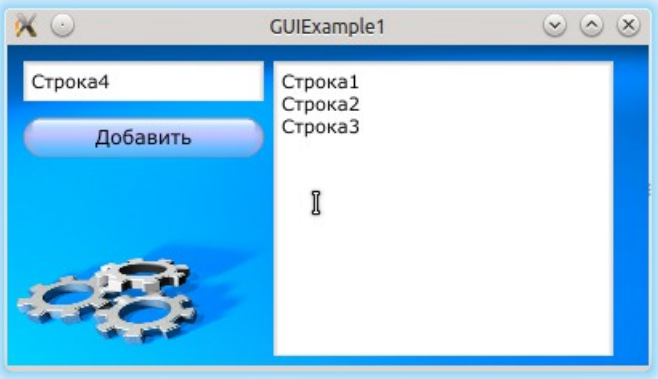


Рисунок 5.8. Работа приложения с визуально спроектированным компонентом содержимого

Генерируемый код рисования фона компонента помещается в метод `paint()` его класса. Изучив его, легко понять логику операций рисования в JUCE.

Конец ознакомительного фрагмента

Об авторе

Андрей Николаевич Кувшинов - кандидат медицинских наук, пространственный эпидемиолог, C++ и R разработчик, технический писатель. Эксперт и администратор консультации по C / C++ на портале специалистов RFPro (http://rfpro.ru).

Книга посвящена разработке приложений для Linux, Windows, Mac OS X и iOS на языке C++ с использованием кроссплатформенной библиотеки JUCE версии 4.2.x. Подробно рассмотрены возможности, предоставляемые этой библиотекой, а также практическое применение классов, входящих в её состав, на большом количестве простых, подробно прокомментированных примеров. Книга содержит пошаговую исчерпывающую информацию по созданию приложений JUCE различной степени сложности от простейших до мультимедийных.

ISBN 978-1-365-18247-1
90000
9 781365 182471
ID: 18961367
www.lulu.com