Практика 1.

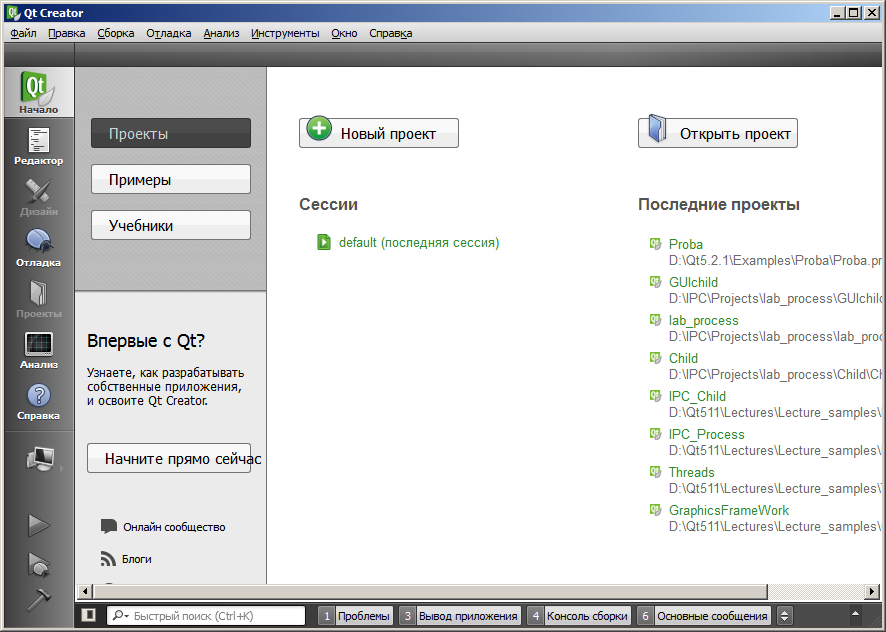
# IDE QtCreator. Окно. Класс QWidget. Вид главного окна. Состояние главного окна. Интернационализация приложения - QtLinguist. Фон окна. Курсор. Иконка. Размеры окна. Задание размеров и положения окна. Диалоговые окна (QMessageBox, QFileDialog, QColorDialog).

## Создание каркаса приложения

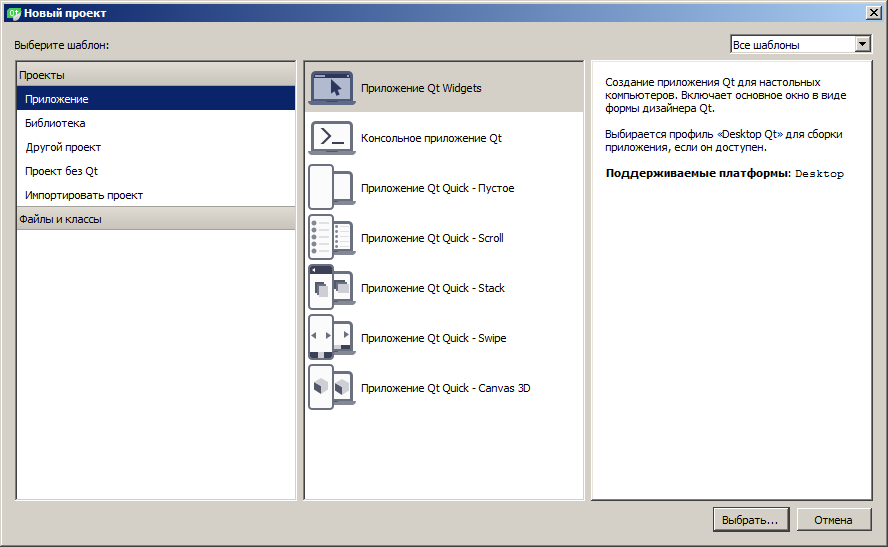
Чтобы понять устройство Qt-приложения, для начала минимизируем использование возможностей, которые предоставляет QtCreator для автоматизации создания кода. На первом этапе делегируем QtCreator-у только:

* + создание и модификацию файла проекта .pro (добавление новых файлов в проект)
  + создание простейшей заготовки Qt-приложения
  + создание заготовок классов

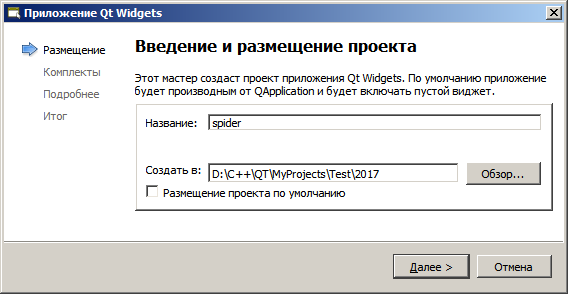
1. Запустите IDE Qt Creator
2. Нажмите кнопку «Новый проект»:



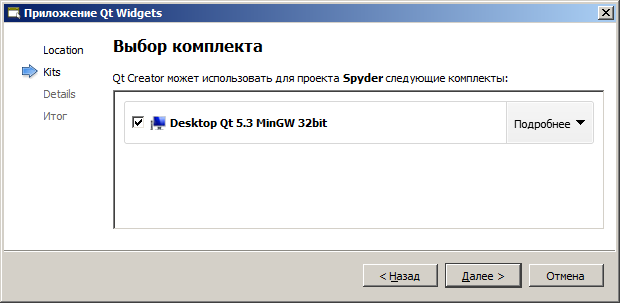
1. Выберите тип проекта и “Приложение Qt Widgets”. Нажмите кнопку «Выбрать»



1. Введите название проекта и путь. Нажмите кнопку «Далее».

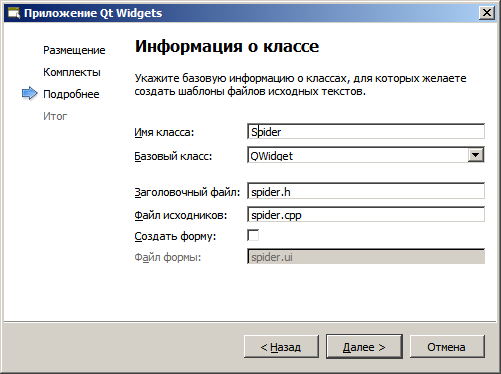


1. Подтвердите выбор инструментария. Нажмите кнопку «Далее»



1. **Откажитесь от создания QtCreator-ом формы** (а опосредованно и от автоматизации создания графического интерфейса пользователя – user interface):

* Задайте имя класса, например “Spider”
* Задайте базовый класс **QWidget**
* Уберите флажок “Создать форму”



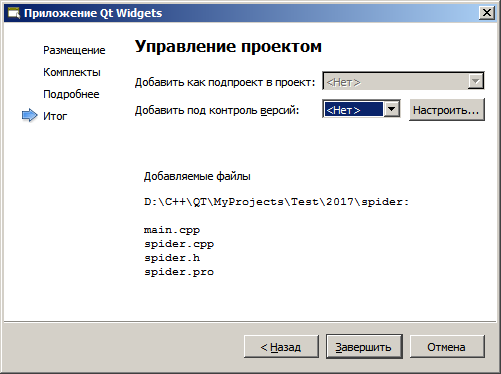
Класс Spider будет поддерживать главное окно приложения. Чтобы получить всю базовую функциональность виджета, унаследуем наш класс от класса Qt – **QWidget.**

Замечание: виджет (или по-простому окно) – выполняет две основных роли:

* Визуализирует функциональность приложения или его части
* Осуществляет взаимодействие с пользователем

Нажмите кнопку «Далее».

1. Перед генерацией каркаса приложения QtCreator выводит перечень тех файлов, которые он создаст и подключит в проект:



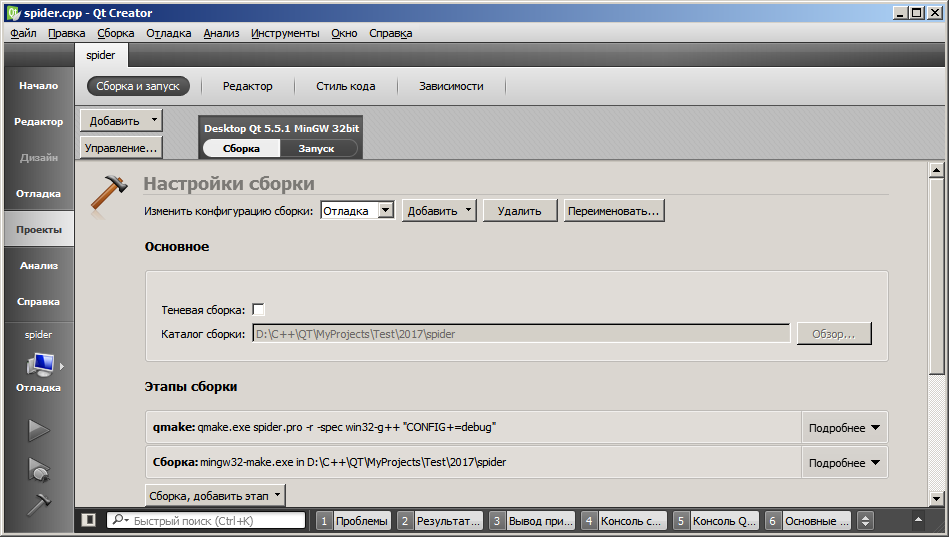
Нажмите кнопку «Завершить».

## Теневая сборка

Теневая сборка означает сборку проекта в отдельном каталоге, ***каталоге сборки***. Каталог сборки отличается от каталога исходных кодов.

Один из плюсов теневой сборки в том, что исходные (не зависящие от конкретного компилятора, среды и платформы) файлы хранятся в одной папке, а сгенерированные MOC – в другой папке, что позволяет легко получать разные конфигурации сборки для одного набора исходников.

Для нашего первого учебного проекта теневую сборку лучше отключить:



## Сгенерированные файлы проекта

QtCreator создал следующие файлы:

Замечание: эти файлы можно открыть/посмотреть:

Редактор/Заголовочные файлы и Исходники. Предполагается, что в эти файлы Вы будете добавлять свой код.

main.cpp

int main(int argc,char\* argv[]) //уже знакомая функция main

{

QApplication a(argc,argv); //создание объекта, инкапсулирующего функциональность приложения.

Spider w; //создание объекта для поддержки главного окна приложения.

В конструкторе можно осуществить все действия для инициализации главного окна

w.show(); //отобразить главное окно, иначе по умолчанию оно останется невидимым

return a.exec(); //запуск цикла обработки сообщений, выход из которого осуществляется только по желанию пользователя.

Цикл обработки сообщений извлекает из очереди сообщений приложения очередное событие (нажатие кнопки на клавиатуре, движение мышки, событие от таймера…) и отправляет его на обработку соответствующей функции – обработчику события (получателем может быть окно, объект QObject,…)

}

//------------------------------------------------------------------------

spider.h

class Spider:public QWidget //наследует от класса Qt - QWidget

{

Q\_OBJECT//макрос, необходимый для поддержки механизма сигналов/слотов (в частности в данной работе позволяет использовать функцию **tr**(), а в дальнейшем и connect())

public:

Spider(QWidget\*parent=0); //у главного окна приложения родителя нет (по умолчанию родительским окном является **desktop**)

~Spider ();

};

//------------------------------------------------------------------------

spider.cpp – заготовки для реализации конструктора и деструктора

#include "spider.h"

Spider::Spider(QWidget\*parent)

:QWidget(parent)

{

}

Spider::~Spider()

{

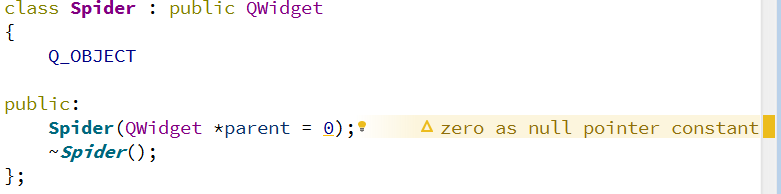
}

Заготовку приложения можно запустить и полюбоваться на пустое окно.

Замечание:

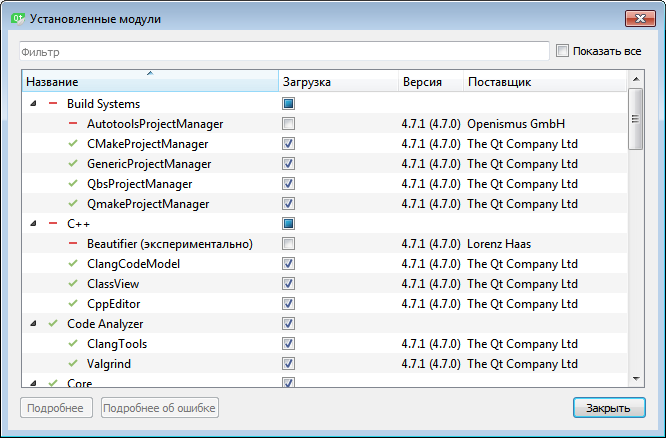
Модель кода является частью среды IDE и служит для статического анализа кода и выявления возможных проблем.

В **Qt Creator 4.7.0** по умолчанию задействован новый построитель модели кода для языка С++, основанный на Clang. По сравнению со старым встроенным построителем в Clang более оперативно отражаются все изменения в спецификациях C++, предоставляется более полная информация о возможных проблемах в коде без необходимости компиляции кода.



Для возврата к старому построителю при отключении в настойках плагина **ClangCodeModel** требуется:

1. Отключить плагин **ClangCodeModel** (Справка-> О модулях->Установленные модули). Т.е. снять флажок **ClangCodeModel**



1. Перезагрузить QtCreator

# Класс QWidget

Класс QWidget поддерживает любые окна и предоставляет базовую функциональность для всех виджетов (окон и специализированных окошечек - элементов управления), например: изменение размеров, местоположения, обработки событий и др.

Терминология: виджеты, у которых нет предка, называются виджетами верхнего уровня (top-level widgets) или окнами.

Замечание: после создания виджета верхнего уровня, чтобы отобразить его на экране, нужно вызвать метод ***show ()****,* иначе и виджет верхнего уровня, и его виджеты-потомки останутся невидимыми.

Класс **QWidget** имеет конструктор с двумя параметрами:

QWidget :: QWidget(**QWidget\* pwgt = 0**, //указатель на виджет-предок. Если этот параметр равен нулю, то будет создан виджет верхнего уровня

**Qt::WindowFlags f = 0**) //задаются свойства окна (внешний вид и особенности отображения **главного окна**)

В нашем приложении будем работать с классом Spider - наследником класса QWidget.

## 3адание свойств окна (вида и поведения)

Внешний вид окна определяется значением флагов **Qt::WindowFlags** , которые можно задать:

* с помощью **графического редактора** (пока мы **не** используем графический редактор!) при создании формы.
* с помощью кода **в тексте программы**. Вопрос в том, в каком месте программы следует это делать.

Настройка внешнего вида виджета с помощью кода.

1 вариант:

Можно в функции *main* сформировать значение флагов и передать его в конструктор, но конструктор класса *Spider* не имеет параметра такого типа, поэтому придется изменять сигнатуру конструктора.

2 вариант:

Можно значение флагов сформировать в конструкторе класса *Spider* и для их установки применить метод *setWindowFlags().*

|  |  |
| --- | --- |
| Синтаксис | Пример |
| void QWidget::setWindowFlags(Qt::WindowFlags type); | Qt::WindowFlags f = <значение>  QWidget w;  w.setWindowFlags(f); |

получить текущий набор флагов можно посредством метода *windowFlags()* :

|  |  |
| --- | --- |
| Синтаксис | Пример |
| Qt::WindowFlags QWidget::windowFlags(); | QWidget w;  Qt::WindowFlags f=w.windowFlags(); |

**Qt::WindowFlags** является битовой маской и содержит две группы флагов:

**type -** отвечает за тип окна и задается единственным значением

**hint** – задает особенности отображения окна. Здесь может быть задано несколько значений, комбинируемых побитовым **ИЛИ.**

Обе группы поддерживаются именованными константами.

Настройка типа окна

* по умолчанию **type** имеет значение [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::Widget (с остальными значениями этой группы в нашем простеньком приложении экспериментировать неудобно).

|  |  |  |
| --- | --- | --- |
| Тип окна(type) | значение | описание |
| [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::Widget (умолчание) | 0x00000000 | Виджет является дочерним, если при создании задан родитель. Если родитель не задан, то он является независимым окном |
| [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::Window | 0x00000001 | виджет является окном с рамкой и строкой заголовка, независимо от того, есть у него родитель или нет |
| [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::Dialog | 0x00000002 | Window | Это значение по умолчанию для объектов типа QDialog. Обычно у диалоговых окон отсутствуют кнопки maximize/minimize. |
| [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::Popup | 0x00000008 | Window |  |
| [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::Tool | Popup | Dialog |  |
| Qt::WindowType\_Mask | 0x000000ff | Маска для извлечения типа окна, часть оконных флагов |
| . . . |  |  |

Иногда требуется определить текущий тип окна. Это можно сделать так:

// Получение типа окна (Способ № 1)

Qt::WindowType type1 = windowType();

// Получение типа окна (Способ № 2)

Qt::WindowFlags type2 = windowFlags() & Qt::WindowType\_Mask);

Настройка стиля окна

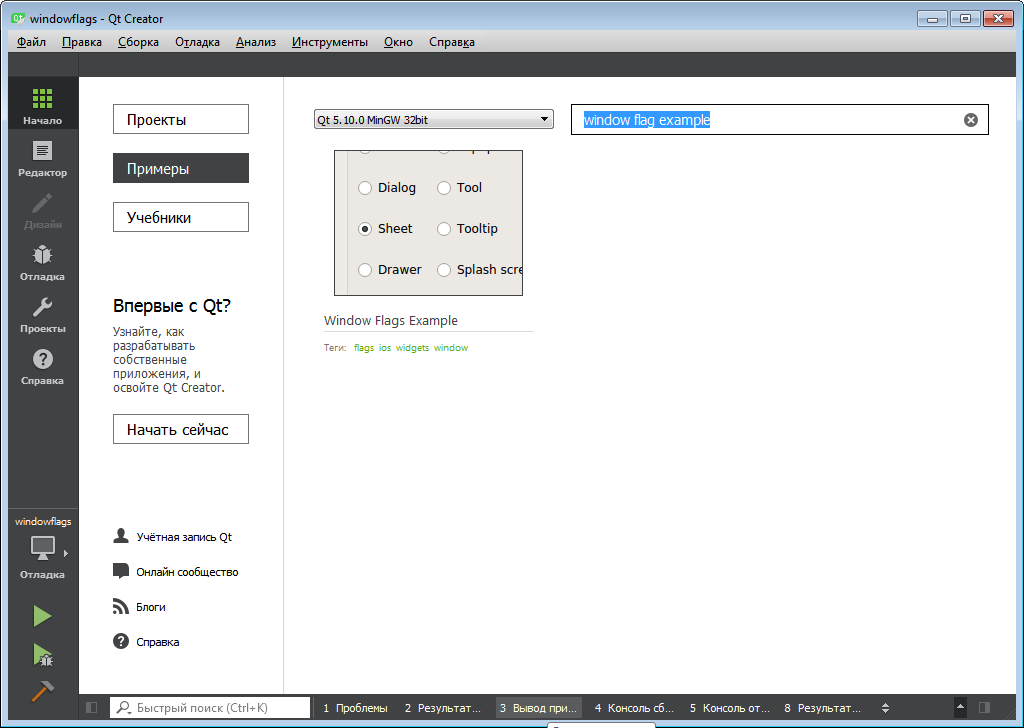
* группа **hint** влияет на внешний вид и поведение **только** окон, т.е. top-level виджетов.
* Наше окно по умолчанию имеет строку заголовка, системное меню и кнопки: минимизировать, восстановить, закрыть.
* Для того, чтобы изменить значения, заданные по умолчанию, следует прежде всего добавить стиль Qt::CustomizeWindowHint, который отменяет умолчания.

|  |  |  |
| --- | --- | --- |
| Стиль окна(hint) | значение | описание |
| Qt::CustomizeWindowHint | 0x02000000 | Выключает значения флагов, заданных по умолчанию. |
| [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::WindowTitleHint | 0x00001000 | Задает заголовок окна. |
| [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::WindowSystemMenuHint | 0x00002000 | Добавляет системное меню окна, и, возможно, закрывающую кнопку. |
| [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::WindowMinimizeButtonHint | 0x00004000 | Добавляет кнопку сворачивания. На некоторых платформах для того, чтобы это работало, требуется добавить Qt::WindowSystemMenuHint. |
| [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::WindowMaximizeButtonHint | 0x00008000 | Добавляет кнопку разворачивания. На некоторых платформах для того, чтобы это работало, требуется добавить Qt::WindowSystemMenuHint. |
| [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::WindowCloseButtonHint | 0x08000000 | Добавляет закрывающую кнопку. |
| [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::WindowStaysOnTopHint | 0x00040000 | Информирует оконную систему, что окно должно находиться поверх всех. |
| … |  |  |

Замечание:

Для того, чтобы лучше разобраться с тем, как использовать те или иные флаги можно запустить приложение из комплекта примеров, предоставляемых Qt. Для этого надо :

1. Нажать кнопку «Начало»
2. В поле поиска набрать «window flag example»
3. Нажать на пиктограмму примера
4. Пример будет загружен в QtCreator



1

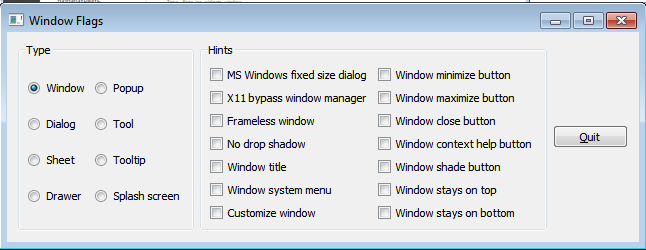
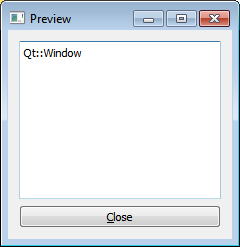
2

3

5

1. Для запуска приложения нажать на кнопку  или набрать

Ctrl +R



Поэкспериментируйте с флагами группы hint.

**Задание 1**

1) Получите текущее значение флагов, заданное по умолчанию.

2) Создайте переменную типа Qt::WindowFlags

3) задайте тип окна (имеет смысл использовать значения [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::Window, [Qt](qthelp://org.qt-project.qtwidgets.501/qtcore/qt.html)::Dialog )

4) с помощью флагов группы hint, объединенных побитовой операцией **или**, попытайтесь задать различные стили окна.

4) установите на виджете полученное значение флагов

5) проверьте, что получилось

Подсказки:

1) удобнее всего наблюдать эффект, установив значение **Qt::WindowStaysOnTopHint.**

2) стили Qt::WindowTitleHint, Qt::WindowSystemMenuHint, Qt::WindowMinimizeButtonHint, Qt::WindowMaximizeButtonHint, Qt::WindowCloseButtonHint учитываются только в том случае, когда одновременно установлен **Qt::CustomizeWindowHint.**

## Управляем состоянием главного окна приложения

Окно может находиться в следующих состояниях [Qt::WindowStates](file:///D:\Marina\Qt\qt.html#WindowState-enum):

|  |  |  |
| --- | --- | --- |
| Qt::WindowNoState | 0x00000000 | Нормальное |
| Qt::WindowMinimized | 0x00000001 | Минимизированное (свернутое в панель задач) |
| Qt::WindowMaximized | 0x00000002 | Максимизированное с рамкой |
| Qt::WindowFullScreen | 0x00000004 | На весь экран без рамки. |
| Qt::WindowActive | 0x00000008 | Окно имеет фокус ввода (принимает ввод с клавиатуры). |

**Получение текущего состояния окна:**

|  |  |
| --- | --- |
| Синтаксис | Пример |
| [Qt::WindowStates](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qt.html#WindowState-enum) QWidget::windowState(); | QWidget w;  [Qt::WindowStates](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qt.html#WindowState-enum) state =w.windowState(); |

**Для задания состояния окна:**

|  |  |
| --- | --- |
| Синтаксис | Пример |
| void QWidget::setWindowState([Qt::WindowStates](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qt.html#WindowState-enum) windowState); | QWidget w;  [Qt::WindowStates](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qt.html#WindowState-enum) st=  <значение>; w.windowState(st); |

При этом для видимого окна установки применяются немедленно, для невидимого – при отрисовке.

**Другие методы для манипуляций c окном:**

|  |  |
| --- | --- |
| Синтаксис | Пример |
| void QWidget::[showMaximized](file:///D:\Marina\Qt\qwidget.html#showMaximized)(); | QWidget w;  w.[showMaximized](file:///D:\Marina\Qt\qwidget.html#showMaximized)(); |
| void QWidget::[showMinimized](file:///D:\Marina\Qt\qwidget.html#showMaximized)(); | QWidget w;  w.[showMinimized](file:///D:\Marina\Qt\qwidget.html#showMaximized)(); |
| void QWidget::[show](file:///D:\Marina\Qt\qwidget.html#showMaximized)Normal(); | QWidget w;  w.[show](file:///D:\Marina\Qt\qwidget.html#showMaximized)Normal(); |

**Задание 2**

Поэкспериментируйте с состоянием главного окна. Заставьте его появляться на экране в максимизированном (минимизированном) виде.

## Изменяем заголовок главного окна

По умолчанию заголовок главного окна формируется, исходя из **имени исполняемого файла**.

Заголовок является свойством (property) класса QWidget:

QString QWidget::windowTitle

**Для получения заголовка и установки заголовка на виджете**:

|  |  |
| --- | --- |
| Синтаксис | Пример |
| QString QWidget::windowTitle() const; | QWidget w;  QString title =w.windowTitle(); |
| void QWidget::setWindowTitle(const QString &); | QWidget w;  QString title(“My first window”);  w.setWindowTitle(title);  w.setWindowTitle(“Hello!”); |

Замечание: заголовок имеет смысл только для окон верхнего уровня (top-level).

**Как и где можно изменить заголовок**

* в режиме дизайнера заголовок можно задавать непосредственно в окне свойств графического редактора, но **пока** бОльшую часть работы мы делаем «вручную» =>
* в тексте программы изменить заголовок можно с помощью  
  **void QWidget::setWindowTitle (constQString &);**
  + в конструкторе главного окна (это логично)
  + или в функции main() после создания соответствующего объекта и перед отображением главного окна на экране
  + в любой момент выполнения программы

**Задание 3**

Установите заголовок окна

### Учимся говорить по-русски

1. Если приложение будет общаться с пользователем только по-английски или по-русски, то проще всего преобразовывать традиционные (однобайтовые нуль-терминированные) строковые литералы в требуемый тип QString посредством макроса QStringLiteral (появился только в Qt 5):

QString s=**QStringLiteral** ("Ура!!!Русский!!!");

2. Если же существует вероятность того, что

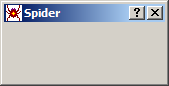
* Ваше приложение будет востребовано не только отечественным Заказчиком
* или требуется обеспечить независимость Вашего приложения от среды, то

стоит задуматься о возможности «интернационализировать» приложение. То есть (не внося изменения в текст самой программы), позволить пользователю видеть любой текст (не только заголовок главного окна) на своем родном языке.

**QtLinguist** позволяет написать универсальное приложение, которое может во время выполнения загружать строки на разных языках. Это мы научимся делать в конце этой лабораторной работы.

## Иконка приложения

По умолчанию у главного окна установлена стандартная иконка приложения ([QApplication::windowIcon](file:///D:\C++\QT\NEW%20COURSE\BASE_QT%20%20%20%20%20%20DEV-Qt10\LAB%201\Lab1_Qt10_задание\Lab1_Qt10\qapplication.html#windowIcon-prop)(),но обычно у каждого приложения есть своя специфическая иконка. Иконка отображается:

* в левом углу строки заголовка главного окна,
* в панели задач,
* на рабочем столе,
* а также в утилитах типа Explorer рядом с именем исполняемого файла.

Иконка является свойством (property) класса QWidget.

QIcon QWidget::windowIcon

**Для получения иконки и установки иконки на виджете**:

|  |  |
| --- | --- |
| Синтаксис | Пример |
| QIcon QWidget::windowIcon () const; | QWidget w;  QIcon icon =w.windowIcon(); |
| void QWidget::setWindowIcon(const QIcon & icon); | QWidget w;  QIcon icon(“MyIcon.ico”);  w.setWindowIcon(icon); |

Отображение своей специфической иконки в заголовке приложения и на панели задач.

Последовательность действий:

* В любом графическом редакторе (в частности, под Windows можно использовать графический редактор IDEVS) нарисовать иконку (обычно 32\*32, 24цвета)  
  или **позаимствовать** уже существующую понравившуюся иконку.
* Сохранить файл с изображением в папке проекта (обычно расширение такого файла .ico)
* #include <QIcon>
* Создать объект типа **QIcon** и проинициализировать посредством конструктора  
  QIcon::QIcon( const [QString](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qstring.html) & fileName ),где *filename*–это спецификация файла
* Проассоциировать иконку с главным окном посредством метода   
  void [QWidget::setWindowIcon](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qwidget.html#windowIcon-prop)(const QIcon&) //для главного окна или

void QApplication[::setWindowIcon](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qwidget.html#windowIcon-prop)(const QIcon&);

Замечание:

это не лучший вариант задания иконки приложения, так как

* Для выполнения приложения на другом компьютере требуется перенести туда не только собственно исполняемый файл, но и файл иконки;
* Иконка не будет появляться на рабочем столе или в Explorer-е (для этого, она должна храниться в исполняемом файле в качестве ресурса)

Поэтому обычно иконку и другие ресурсы приложения хранят непосредственно в исполняемом файле. Такой способ будет рассмотрен позже (в теме система ресурсов Qt).

**Задание 4.1:**

1) Для экономии времени на занятии позаимствуйте мою иконку (перепишите файл .ico в папку Вашего проекта или лучше перепишите целиком папку Images\_lab1)

2) проассоциируйте ее с Вашим приложением (она должна появиться в левом верхнем углу строки заголовка).

А уже дома можете создать свою специфическую иконку.

Установка пиктограммы приложения в Windows:



Чтобы в Windows увидеть иконку рядом с именем приложения или на рабочем столе выполните следующее :

в файл проекта (.pro) добавьте директиву:  
RC\_ICONS= <имя\_файла\_иконки>

**Задание 4.2:**

Установите пиктограмму приложения в Windows

## Управляем фоном виджета

Предварительное замечание:

На различных платформах внешний вид виджетов отличается. Для того, чтобы вид Вашего приложения не отличался от стиля, принятого в ОС, НЕ рекомендуется жестко кодировать цвета виджетов. Встроенные виджеты Qt используют класс QStyle для выполнения почти всех операций рисования. Кроме того, Qt содержит набор подклассов QStyle, которые эмулируют стили различных платформ, поддерживаемых Qt (QWindowsStyle, QMacStyle). Мы будем устанавливать жестко цвет фона окна исключительно в учебных целях.

Именно потому, что Qt старается отображать виджеты приложения в едином стиле, не отличающемся от стиля, принятого на определенной платформе процесс установки фона на виджете может показаться усложненным.

В большинстве случаев разработчик оконного приложения оставляет по умолчанию фон окна светло-серым. Но иногда хочется:

* закрашивать фон одним из предопределенных в системе цветом (черным, красным, …)
* закрашивать фон произвольным цветом
* по образцу («в полосочку», …)
* в качестве фона выводить изображение

Базовый для всех окон класс QWidget, содержит свойство (property) QPalette palette:

QPalette QWidget::palette

**Для получения палитры и установки палитры на виджете**

|  |  |
| --- | --- |
| Синтаксис | Пример |
| const QPalette & QWidget::palette() const; | QWidget w;  QPalette palette =w.palette(); |
| void QWidget::setPalette(const QPalette & p); | QWidget w;  QPalette palette ;  <настройка палитры>  w.setPalette (palette); |

Палитра используется классом QStyle и его подклассами при перерисовке виджета. Метод QStyle::[standardPalette()](http://doc.crossplatform.ru/qt/4.8.x/html-qt/qstyle.html#standardPalette) возвращает палитру для стиля.

Виджет может находиться в разных состояниях (активном, неактивном, недоступном). И для разных состояний виджета ( [QPalette::ColorGroup](http://doc.crossplatform.ru/qt/4.8.x/html-qt/qpalette.html#ColorGroup-enum)) имеется свой набор цветов.

А каждый набор содержит цвет для определенных ролей, задаваемых перечислением [QPalette::ColorRole](http://doc.crossplatform.ru/qt/4.8.x/html-qt/qpalette.html#ColorRole-enum). Роли описывают в каких ситуациях цвета будут использованы (для отрисовки фона виджета, текста или кнопок).

**Цветовые группы**

enum QPalette::ColorGroup

|  |  |  |
| --- | --- | --- |
| QPalette::Disabled | 1 | Виджет недоступен |
| QPalette::Active | 0 | Активное состояние виджета |
| QPalette::Inactive | 2 | Неактивное состояние виджета |
| QPalette::Normal | 0 | Активное состояние виджета |

Замечания:

* окна верхнего уровня не могут быть Disabled.
* в большинстве случаев виджет в активном и неактивном состоянии выглядит одинаково.

**Цветовые роли**

enum QPalette::ColorRole (некоторые значения)

|  |  |  |
| --- | --- | --- |
| QPalette::Window | 10 | Основной цвет фона. По умолчанию- светло-серый |
| QPalette::WindowText | 0 | Основной цвет для пера. По умолчанию- черный |
| QPalette::Button | 1 | Цвет кнопки |
| QPalette::ButtonText | 8 | Цвет текста на кнопке |
| QPalette::Highlight | 12 | Цвет выделения элементов. По умолч. темно-голубой |
| QPalette::HighlightedText | 13 | Цвет текста выделенных элементов. |

**Для настройки палитры имеются методы:**

|  |  |
| --- | --- |
| Синтаксис | Пример |
| void QPalette::setColor(QPalette::ColorGroup group,  QPalette::ColorRole role,  const QColor & color);  void QPalette::setColor(QPalette::ColorRole role,  const QColor & color); | QPalette pal;  pal.setColor(QPalette::Activ,  QPalette::Window,  QColor(255,0,0));  pal.setColor(QPalette::Window,  Qt::red); |
| void QPalette::setBrush(QPalette::ColorGroup group,  QPalette::ColorRole role,  QBrush & brush);  void QPalette::setBrush(QPalette::ColorRole role,  QBrush & brush); | QPalette pal;  pal.setBrush(QPalette::Inactive,  QPalette::Window,  QBrush(Qt::green));  QBrush brush (Qt::red,  Qt::CrossPattern);  pal.setBrush(QPalette::Window,  brush); |

При создании любого виджета палитра ( QPalette) формируется «по умолчанию». Умолчание предполагает, что цвет фона окна – светло-серый, цвет текста – черный, … Если умолчания не устраивают, можно «настроить» цвета любого виджета специфическим образом.

Последовательность действий:

* #include<QPalette>
* Создаем и инициализируем посредством default-конструктора объект типа QPalette  
  или получаем копию уже используемой виджетом палитры посредством   
  const QPalette &QWidget::palette();
* Нам требуется настроить фон виджета, а значит в этом случае для цвета следует выбрать роль QPalette::Window из перечисления QPalette::ColorRole.
* Задаем цвет фона посредством методов:

//только для указанной группы  
voidQPalette::[setColor](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qpalette.html#setColor) (QPalette::ColorGroup *group*, QPalette::ColorRole *role*,

const QColor& *color* ) ;

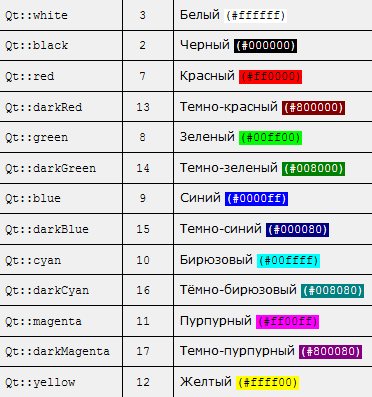
voidQPalette::[set](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qpalette.html#setColor)Brush (QPalette::ColorGroup *group*, QPalette::ColorRole *role*,

const QBrush& *brush* );

// или для всех групп

voidQPalette::[setColor](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qpalette.html#setColor-2) (QPalette::ColorRole *role*, const QColor& *color* ) ;

voidQPalette::[set](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qpalette.html#setColor)Brush (QPalette::ColorRole *role*, const QBrush& *brush* );

* Ассоциируем созданную палитру с виджетом посредством  
  voidQWidget::setPalette ( const QPalette & )

Замечание:

* цвет можно задать посредством «глобальных» предопределенных констант : Qt::red, Qt::green, ...
* а можно сформировать произвольно с помощью конструкторов класса QColor

**Задание 5.1:**

сделайте фон окна разным в активном состоянии и в неактивном

### Изображение в качестве фона окна

В качестве фона можно задать изображение. Для хранения изображений предназначен класс QPixmap. Класс QBitmap предназначен для монохромных изображений. Загрузить изображение в QPixmap можно:

* с помощью конструктора . В качестве параметра достаточно задать только имя файла.
* или создать объект типа QPixmap с помощью default конструктора, а потом воспользоваться методом load(), которому также достаточно задать только имя файла.

|  |  |
| --- | --- |
| Синтаксис | Пример |
| QPixmap::QPixmap (const QString &*fileName*,  const char \**format* = 0,  Qt::ImageConversionFlags*flags*=Qt::AutoColor); | QPixmap pix (“MyPicture.bmp”); |
| bool QPixmap::load(const QString &*fileName*,  const char \**format* = 0,  Qt::ImageConversionFlags*flags*=Qt::AutoColor); | QPixmap pix ;  bool b =pix.load(“MyPicture.bmp”); |

Последовательность действий:

* Позаимствовать изображение в любом поддерживаемом Qt формате (.bmp, .jpg, .png…), например, в папке Images\_lab1
* Создать объект типа **QPixmap** с требуемым изображением (#include<QPixmap>)  
  QPixmap::QPixmap( const [QString](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qstring.html) & fileName, constchar \* format = 0,

[Qt::ImageConversionFlags](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qt.html#ImageConversionFlag-enum) flags = Qt::AutoColor );

* Создать объект типа **QBrush** (#include<QBrush>) таким образом, чтобы кисть закрашивала поверхность посредством изображения  
  QBrush::QBrush( const [QPixmap](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qpixmap.html) & pixmap );
* Создать объект типа QPalette (#include<QPalette>) с помощью default конструктора
* Сформировать палитру посредством вызова метода:  
  void QPalette::setBrush( [ColorRole](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qpalette.html#ColorRole-enum) role, const [QBrush](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qbrush.html) & brush ) или  
  void QPalette::setBrush ( [ColorGroup](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qpalette.html#ColorGroup-enum) group, [ColorRole](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qpalette.html#ColorRole-enum) role, const [QBrush](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qbrush.html) & brush );

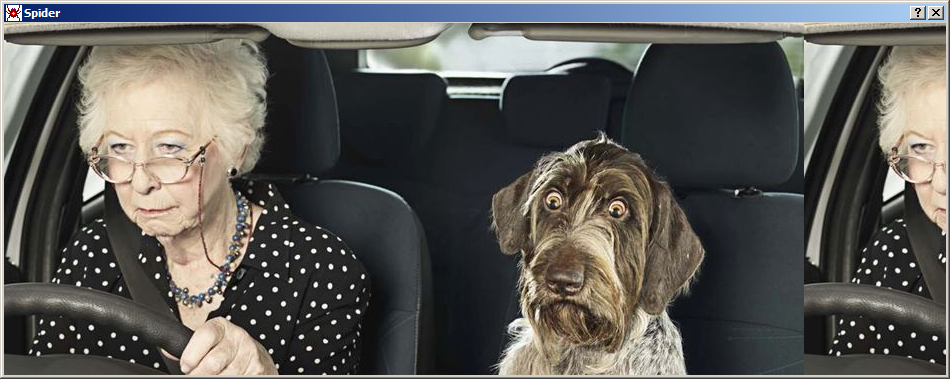
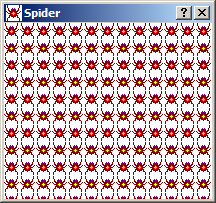


**Задание 5.2**

задайте картинку в качестве фона окна (лучше для неактивного состояния).

Изображение можете для экспериментов позаимствовать у меня (а на досуге найти/создать свою картинку)

Если размер картинки меньше размера окна, то она «размножается» (изображением заполняется поверхность окна).

### Можно задать любую степень прозрачности окна посредством:

void setWindowOpacity(qreal level); // level<=1 l

**Задание 5.3**

сделайте фон окна полупрозрачным

## Задаем размеры и расположение окна

Иногда необходимо/желательно уметь задавать размеры, первоначальное расположение, не позволять окну увеличиваться/уменьшаться до определенных размеров…

Обычно расположение окна (левый верхний угол) задается относительно родительского окна. У главного окна приложения формально родителя нет, но фактически родителем является **desktop**, поэтому для окон верхнего уровня расположение задается относительно верхнего левого угла экрана.

Класс QWidget содержит свойства, определяющие размеры и расположение виджета, и предоставляет аксессоры для этих свойств. А также дополнительные методы, связанные с размерами и положением окна. Часть функций описывают только клиентскую область виджета, часть учитывают размеры рамки.

Замечание:

разница в использовании таких функций проявляется только для окон верхнего уровня (то есть для дочерних окон, кнопок, полей редактирования… разницы нет).

**Функции получения размеров и положения (относительно родительского виджета):**

* Функции, учитывающие рамку

|  |  |
| --- | --- |
| Синтаксис | Пример |
| int QWidget::x()const;  int QWidget::y()const;  QPoint QWidget::pos()const;  QRect QWidget::[frameGeometry()](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qwidget.html#frameGeometry-prop) const;  QSize QWidget::frameSize() const; | QWidget w;  QRect rect=w. frameGeometry();  QPoint p = w.pos(); |

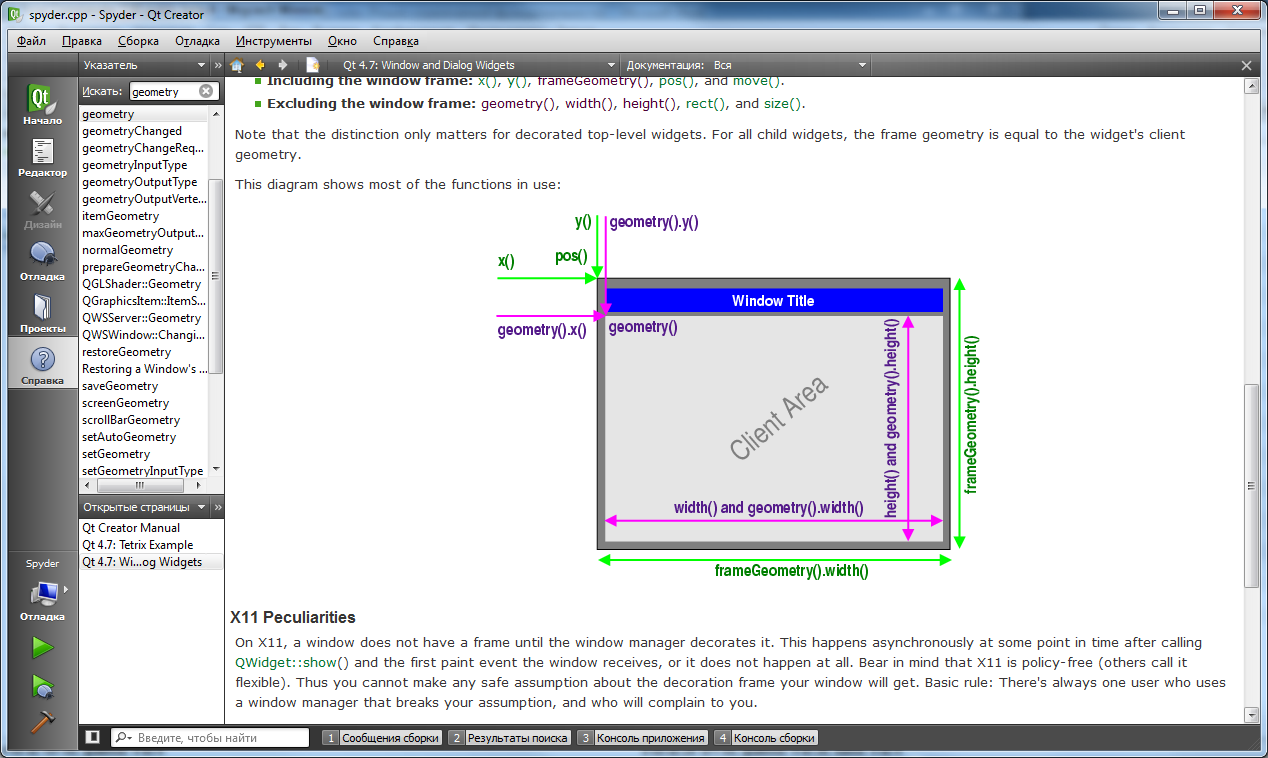
* Функции, НЕ учитывающие рамку

|  |  |
| --- | --- |
| Синтаксис | Пример |
| const QRect & QWidget::[geometry()](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qwidget.html#frameGeometry-prop) const;  QSize QWidget::size() const;  int QWidget::width() const;  int QWidget::height() const;  QRect QWidget::rect[()](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qwidget.html#frameGeometry-prop) const; | QWidget w;  QRect rect=w. rect(); |

**Функции задания размеров и положения:**

* Функции, учитывающие рамку: [move()](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qwidget.html#pos-prop)
* Функции, НЕ учитывающие рамку: setG[eometry()](file:///D:\Marina\Qt\qwidget.html#geometry-prop), [resize()](file:///D:\Marina\Qt\qwidget.html#size-prop)
* Методы resize() и move() вызывают соответствующие обработчики событий. События обсуждаются в следующей лабораторной работе.

|  |  |
| --- | --- |
| Синтаксис | Пример |
| void QWidget::move (int x, int y);  void QWidget::move (const QPoint& p); | QWidget w;  w.move(100,200); |
| void QWidget::setGeometry(int x, int y,  int w, int h);  void QWidget::setGeometry(const QRect& p); | QWidget w;  w.setGeometry(10,20,100,100); |
| void QWidget::resize(int w, int h);  void QWidget::resize(const QSize& s); | QWidget w;  w.resize(100,100); |



Замечания:

* Так как resize() вызывает обработчик события об изменении размеров виджета ([resizeEvent](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qwidget.html#resizeEvent)()), не следует вызывать resize() или [setGeometry](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qwidget.html#geometry-prop)() внутри этого обработчика – это может привести к бесконечной рекурсии.
* аналогично не следует вызывать move() в обработчике события о перемещении виджета - [moveEvent](file:///D:\Marina\Qt\qwidget.html#moveEvent)()
* если для виджета были установлены минимальный/максимальный размеры (посредством setMinimumSize() или  setMaximumSize()), при задании размера виджета делаются соответствующие поправки.
* при задании нулевых размеров окно становится невидимым
* Если виджет является видимым при изменении его размеров, виджет получает событие [resizeEvent](file:///D:\Marina\Qt\qwidget.html#resizeEvent)(). Если изменяется размер временно невидимого виджета, он получает это событие непосредственно перед появлением на экране.
* При изменении положения (если виджет видимый), виджет получает событие [moveEvent](file:///D:\Marina\Qt\qwidget.html#moveEvent)() немедленно. Если виджет невидим на момент изменений, он получит это событие непосредственно перед появлением на экране.
* Позиция виджетов-потомков внутри виджета-предка может изменяться методом setGeometry () "вручную" или автоматически, с помощью специальных классов размещений (layouts)

**Задание 6.1**

В нашей заготовке размеры и позиция окна задаются по умолчанию ОС. Задайте эти значения самостоятельно посредством перечисленных выше методов.

Задавать произвольные значения не всегда целесообразно, так как при переносе приложения на другой компьютер или при изменении разрешения дисплея, размеры экрана в пикселях могут быть другими. Лучше иметь возможность подстраивать размер и позицию окна под текущий размер экрана. Кроме того, Вы можете подключить несколько мониторов с разными метриками.

Чтобы получить указатель на desktop:

* Подключить файл для QApplication (#include<QApplication>)
* Для получения указателя на виджет экрана использовать статический метод

QDesktopWidget \*QApplication::desktop(); [**static**]

Чтобы узнать размеры экрана, можно использовать методы класса QDesktopWidget:

* Подключить файл для QDesktopWidget (#include<QDesktopWidget>)
* Используя полученный указатель на виджет экрана получить прямоугольник метод :

const [QRect](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qrect.html) QDesktopWidget::screenGeometry () const;

**Задание 6.2**

расположите окно в любой четверти экрана

Иногда полезно не давать виджету увеличиваться больше или уменьшаться меньше заданных размеров. Для ввода таких ограничений можно использовать методы класса QWidget:

void QWidget::setMaximumSize();

void QWidget::setMinimumSize();

Или с помощью соответствующих методов установить только минимальную/максимальную высоту/ширину.

Замечание: установленные ограничения будут использоваться только в обработчике resizeEvent(), поэтому на начальный размер окна они не влияют.

**Задание 6.3**

задайте минимальный/максимальный размер окна

Бывают приложения, в которых пользователю вообще не стоит предоставлять возможность изменять размеры окна (либо только ширину/высоту). Для таких задач используются методы класса QWidget:

void QWidget::setFixedHeight(int h);

void QWidget::setFixedSize(const QSize& s);

void QWidget::setFixedSize(int w,int h);

void QWidget::setFixedWidth(int w);

**Задание 6.4**

задайте фиксированный размер окна (если для фона используется изображение, то логично зафиксировать размер окна, чтобы картинка «не обрезалась» и «не множилась»)

Размер окна лучше изменить в соответствии с размерами изображения посредством  
[QSize](file:///D:\Marina\Qt\qsize.html) QPixmap::size () const;   
Логично, чтобы размер окна не превышал размеры картинки, поэтому стоит ограничить максимальный размер окна:  
void QWidget::setMaximumSize ( const QSize& )

## Курсор

Базовый для всех окон класс QWidget, содержит свойство (property) QCursor cursor;

**Для получения курсора и установки курсора на виджете**:

QCursor **cursor**() const;

void **setCursor**(const QCursor &)

По умолчанию курсор имеет форму стрелки.

Изменить вид курсора и восстановить умолчание можно посредством:

|  |  |
| --- | --- |
| Синтаксис | Пример |
| void QWidget::setCursor(const QCursor & );  void QWidget::unsetCursor(); | QWidget w;  QCursor cursor= Qt::CrossCursor;  w.setCursor(cursor);  w.unsetCursor(); |

Замечание:   
курсор принимает заданную форму только в пределах **клиентской** области того виджета, для которого был вызван метод. Это означает, что для нашего главного окна приложения в «системных» областях окна (меню, рамка…) курсор будет иметь форму стрелки.

Qt предоставляет некоторые наиболее используемые курсоры посредством перечисления enum Qt::CursorShape. Любую из приведенных в справке констант можно использовать в качестве параметра функции setCursor().

Примеры констант: Qt::CrossCursor, Qt::WaitCursor, Qt::PointingHandCursor,…

**Задание 7.1**

замените стандартную стрелку на любой из предоставляемых Qt курсоров

Иногда желательно создать специфический курсор. В этом случае:

* Нарисовать в каком-нибудь графическом редакторе или позаимствовать (у меня) черно-белое изображение. Изображение должно быть квадратным, предпочтительный размер - 32\*32 пиксела. Сохранить его в формате .bmp

**Формирование монохромного курсора**

|  |  |
| --- | --- |
| Синтаксис | Пример |
| QCursor::QCursor(const [QBitmap](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qbitmap.html) & bitmap,  const [QBitmap](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qbitmap.html) & mask,  int hotX = -1, int hotY = -1); | QBitmap bmp(“cursor.bmp”);  QCursor cur(bmp, bmp); |
| где *bitmap* и *mask*– это два изображения .bmp одинакового размера, при помощи которых попиксельно формируется изображение курсора ( (b==1 &&m==1)==black, (b==0 &&m==1)==white, (b==0 &&m==0) ==tranparent, hotX и hotY – это коорд. изображения, которые будут пересчитываться в позицию курсора) | |

**Формирование цветного курсора**

1. Создать QPixmap pix (“спецификация файла с изображением”);
2. Установить маску – pix.setMask(QBitmap("спецификация файла с изображением "));
3. Создать объект типа QCursor с помощью конструктора:   
   QCursor::QCursor(const [QPixmap](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qpixmap.html) & pixmap, int hotX = -1,int hotY = -1);

|  |  |
| --- | --- |
| Синтаксис | Пример |
| QCursor::QCursor(const [QPixmap](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qpixmap.html) & pixmap,  int hotX = -1,int hotY = -1); | QPixmap pix(“cursor.bmp”);  pix.setMask(QBitmap(“cursor.bmp”));  QCursor cur(pix); |

**Задание 7.2**

Измените вид курсора.

Подсказка:

в классе главного окна заведите переменную типа QCursor, например  
QCursor\*spCursor;  
а в конструкторе предусмотрите формирование курсора посредством картинки.

## Стандартные диалоги

**Для выбора нужного цвета**  можно использовать стандартный диалог. В классе QColorDialog имеется статический метод **getColor**:

QColor QColorDialog:: getColor (

const QColor & initial = Qt::white, // цвет фона

QWidget \* parent = 0, //родительский объект

const QString & title = QString(),  //заголовок окна

ColorDialogOptions options = 0); [**static**]

Диалог возвращает значение типа QColor. Чтобы определить, какой кнопкой было закрыто окно, необходимо для возвращенного значения вызвать метод:

bool QColor::isValid() const;

*true* - была нажата кнопка **Ok**,

*false* - **Cancel** (Отмена).

**Выбор файла для открытия**

Для того, чтобы загрузить файл, требуется указать его спецификацию. Удобно для выбора имени файла использовать стандартные диалоги.

Класс **QFileDialog** содержит диалоги:

* выбора файла для открытия,
* выбора файла для сохранения,
* выбора директории.

В классе QFileDialog имеется статический метод **getOpenFileName**:

QString QFileDialog::getOpenFileName(

QWidget \* parent = 0, //родительский объект

const QString & caption = QString(), //заголовок окна

const QString & dir = QString(), //каталог

const QString & filter = QString(),  //шаблон для фильтрации

QString \* selectedFilter = 0,

Options options = 0);

Диалог возвращает значение типа QString. Чтобы определить, было ли выбрано имя файла, можно для возвращенного значения вызвать метод QString::isEmpty() или QString::isNull();

Например,

QString str=QFileDialog::getOpenFileName(0,"Open Dialog", ".", "\*.cpp \*.h" );

if (!str.isNull()) // Проверка, что выбор сделан

{ . . .}

else{} //нажата кнопка Cancel

Замечание 1:

Обратите внимание на то, что в методах, вызывающих диалоги, все параметры заданы по умолчанию и если они Вас устраивают, то можно вызывать диалоги, не указывая параметров.

Замечание 2:

Для того, чтобы лучше разобраться с тем, какие имеются стандартные диалоги и как они выглядят можно запустить приложение из комплекта примеров, предоставляемых Qt. Для этого надо :

1. Нажать кнопку «Начало»
2. В поле поиска набрать «standard dialogs example»
3. Нажать на пиктограмму примера
4. Пример будет загружен в QtCreator

**Задание 8.1**

В задании 5.1 требовалось задать цвет для фона.

Используйте диалог выбора цвета фона.

**Задание 8.1**

В задании 5.2 требовалось загрузить картинку для фона.

Используйте диалог выбора файла для определения имени файла картинки. Проверьте работу.

## Интернационализация приложения

Так как интернационализация, осуществляемая посредством QtLinguist, является механизмом этапа выполнения, то **время выполнения программы немного увеличивается**. Но если затратами времени можно пренебречь, то стоит предпочесть универсальность эффективности.

Специфика:

* все строки, предназначенные для интернационализации, нужно подвергать обработке посредством статического метода **QObject::tr()**  
  QStringQObject::**tr**(const char \**sourceText*, const char \**disamb* = 0, int*n* = -1) [static]
* Обычно в качестве *sourceText* параметра метода tr() используется **строка на английском языке**
* Второй параметр метода tr() - *disamb*– комментарий для переводчика, который можно опустить
* Метод следует использовать не только для перевода строк, но и для обозначения валюты, акселераторов и др. (локализация приложения)

Например, tr(“Spider”);

Последовательность действий:

### Модификация файла проекта

Добавляем в файл проекта (.pro) директиву TRANSLATIONS («вручную»):

**TRANSLATIONS+=spider\_ru.ts**

Это предписание утилите **lupdate**

* проанализировать исходные файлы проекта и
* создать для них файл перевода на русский язык

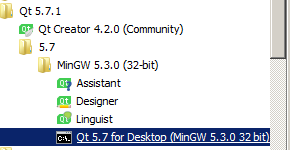
### Утилита lupdate

Создание файла переводов осуществляется утилитой **lupdate**. Утилита lupdate ищет в тексте программы вызов метода **tr()** – это указание, что текст нуждается в переводе. Из строк, найденных таким образом утилитой, создаются специальные TS-файлы (TranslationSource), которые будут использованы во время выполнения приложения.

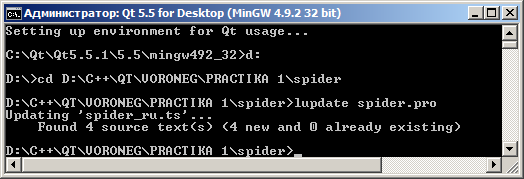
1 cпособ:

Запустите утилиту **lupdate** «вручную» из командной строки:

1. Запустить терминал командной строки из меню «Пуск»



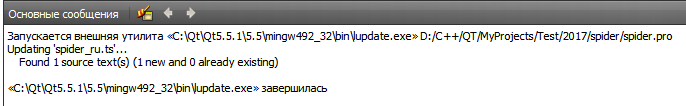
1. Перейти в рабочий каталог
2. Запустить утилиту lupdate: lupdate.exe **spider.pro**



2 cпособ:

Запустите утилиту **lupdate** из QtCreator (Меню/Инструменты/Внешние/Linguist/ lupdate)

В результате работы утилиты будет выдано сообщение:



В результате в директории проекта будет создан файл **spider\_ru**.ts в XML-формате с примерно следующим содержимым:

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE TS>

<TS version="2.1" language="ru\_RU">

<context>

<name>Spyder</name>

<message>

<source>**Spider**</source>

<translation type="**unfinished**"></translation>

</message>

</context>

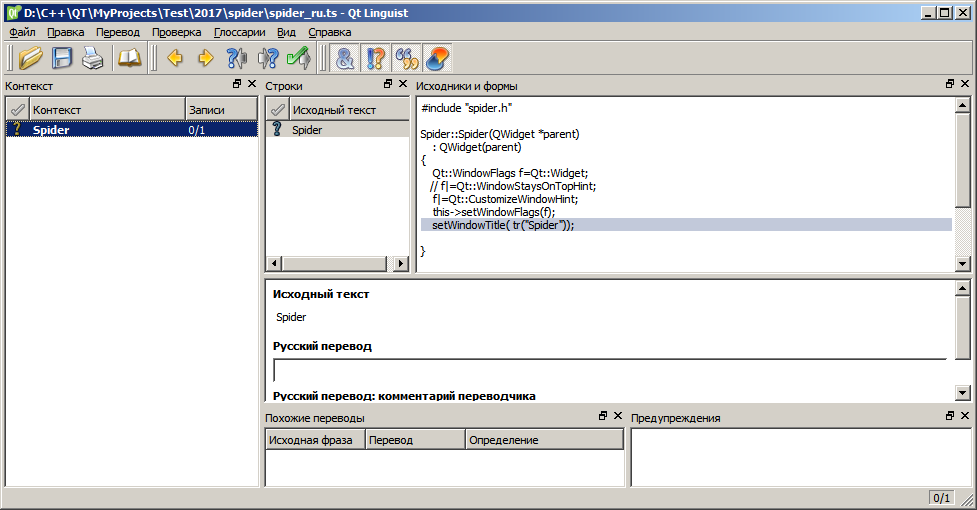
</TS>

Можно увидеть подлежащий переводу “spider” (у меня отмечено красным) и незавершенную секцию перевода (синим).

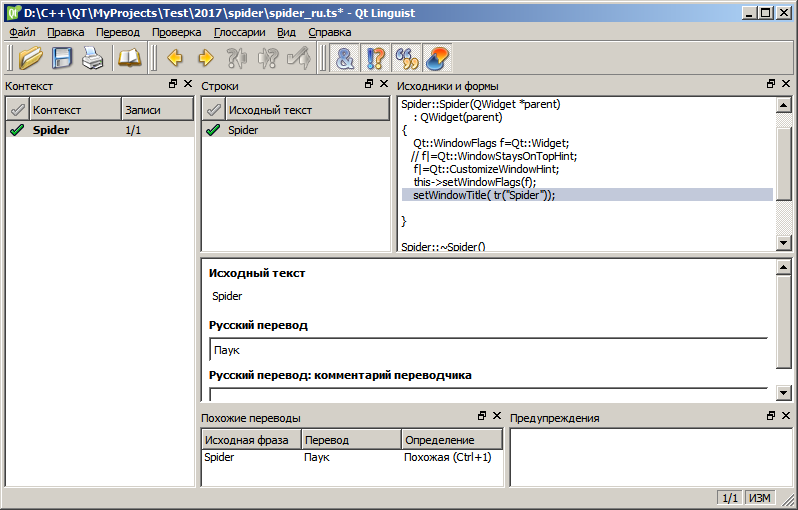
Конечно, зная синтаксис, можно модифицировать секцию перевода «руками», но гораздо удобнее сделать это с помощью утилиты **QtLinguist**

### Конвертация файла .ts в файл .qm. Утилита QtLinguist.

* Запустите QtLinguist
* Файл/Открыть - spider\_ru.ts



* В поле ввода «Русский перевод» введите – «Паук» и нажмите кнопку «Пометить перевод как завершенный и перейти к следующему». Чтобы увидеть результаты перевода в окне «Похожие переводы», достаточно кликнуть в поле «Русский перевод»



* Сохраните файл

В результате файл spider\_ru.ts будет модифицирован следующим образом:

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE TS>

<TS version="2.1" language="ru\_RU">

<context>

<name>Spyder</name>

<message>

<location filename="spider.cpp" line="25"/>

<source>spider</source>

<translation>Паук</translation>

</message>

</context>

</TS>

* Меню/Файл/**Скомпилировать как** – сохраните файл в формате .qm  
  В результате в папке проекта появится файл **spider\_ru.qm**

### Утилита lrelease

Конвертацию файла .ts в файл .qm мы выполнили с помощью QtLinguist. Но это также можно сделать с помощью утилиты lrelease (Меню/Инструменты/Внешние/Linguist/ lrelease)

**Загрузка файла .qm**

|  |
| --- |
| Синтаксис |
| bool QTranslator::[load](file:///D:\C++\QT\NEW%20COURSE\BASE_QT%20%20%20%20%20%20DEV-Qt10\LAB%201\Lab1_Qt10_задание\qtranslator.html#load)(const QString &filename,  const QString &directory = QString(),  const QString &search\_delimiters = QString(),  const QString &suffix = QString()); |
| bool QTranslator::[load](file:///D:\C++\QT\NEW%20COURSE\BASE_QT%20%20%20%20%20%20DEV-Qt10\LAB%201\Lab1_Qt10_задание\qtranslator.html#load-1)(const QLocale &locale, const QString &filename,  const QString &prefix = QString(),  const QString &directory = QString(),  const QString &suffix = QString()) |

### Применение переводчика ко всему приложению

|  |  |
| --- | --- |
| Синтаксис | Пример |
| bool QApplication::[installTranslator](file:///D:\C++\QT\NEW%20COURSE\BASE_QT%20%20%20%20%20%20DEV-Qt10\LAB%201\Lab1_Qt10_задание\qcoreapplication.html#installTranslator)(  QTranslator \*translationFile);  **[static]** | <загрузка файла перевода>  bool b=  QApplication ::[installTranslator](file:///D:\C++\QT\NEW%20COURSE\BASE_QT%20%20%20%20%20%20DEV-Qt10\LAB%201\Lab1_Qt10_задание\qcoreapplication.html#installTranslator)(&transl); |
|  |  |

### В тексте программы – загрузить транслятор

Так как нужно произвести глобальную инициализацию, то в функции main():

…

#include<QTranslator>

int main(int argc,char\*argv[])

{

QApplication a(argc,argv);

QTranslator translator; //создаем переводчик

translator.load("spider\_ru.qm", //имя файла перевода

"."); //директория - текущая

QApplication ::installTranslator(&translator); //применяет переводчик ко всему приложению

Важное замечание (!!!):

Чтобы применить переводчик ко всем окнам приложения, вызов **QApplication::installTranslator()** должен **предшествовать** созданию окна!

**Задание 9.1**

Проверьте работу переводчика

Замечание: пока наша программа еще слишком явно «привязана» исключительно к русскому переводу. Чтобы сделать ее более универсальной, нужно написать код похитрее:

Можно формирование спецификации файла возложить на QTranslator.

1 способ

//загружаем спецификацию qm файла

translator.load(QLocale::system(), // локаль

"spider", //имя файла перевода без информации о языке

"\_"); //префикс

2 способ

QString filespec=QString("Spider\_");

filespec += QLocale::system().name(); //Spider\_ru\_RU

QTranslator все сделает так: сначала будет искать файл точно c таким именем, какое указано (Spyder\_ru\_RU), если такой файл не найден, то "отрезается" расширение .qm и снова пытается найти файл. Если не найдет файла, то приделывает расширение .qm, но "откусывает" суффикс - RU (до символа \_).

//загружаем спецификацию qm файла

translator.load(filespec,".");//из текущей директории"Spyder\_ru.qm"

### Перевод текста в стандартных диалогах

Для того чтобы были переведены тексты, находящиеся на кнопках, в заголовках стандартных диалогов можно воспользоваться готовыми переводами, которые находятся в файле **qtbase\_ru.qm** в одной из папок Qt . Для этого требуется:

* Сформировать имя директории, содержащей файлы переводов. Для этого можно воспользоваться классом QLibrararyInfo.

QString directory=QLibraryInfo::location(QLibraryInfo::TranslationsPath);

* Создать еще один объект транслятора translator2 и загрузить ему файл перевода:

Translator2.load("qt\_" + QLocale::system().name(), //имя файла перевода

directory); //имя директории

* Применить переводчик к приложению

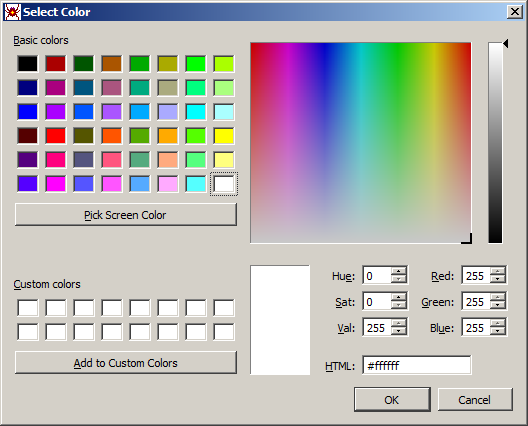
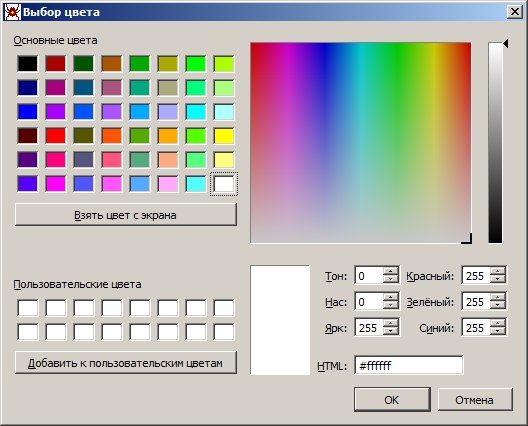
QApplication ::installTranslator(&translator2);

Замечание:

Готовые переводы для стандартных диалогов будут использоваться, если Вы не будете самостоятельно их переводить. А для этого можно либо вызывать стандартные диалоги без указания параметров, либо не обрабатывая строки методом tr(). Например, так:

QColor color= QColorDialog:: getColor();

QColor color= QColorDialog:: getColor (Qt::green, this, "");

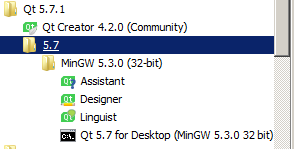
## Развертывание приложения Qt под Windows

Для того чтобы запускать под Windows (без QtCreator) приложение, разработанное с использованием библиотеки Qt, можно:

* Выполнить статическую сборку библиотеки Qt
* Предоставить вместе со своим исполняемым (exe) файлом все dll-ки, необходимые для работы (добавить их в каталог с exe-файлом ).
* Использовать утилиту windеployqt. (Но общий размер каталога получаем большой)

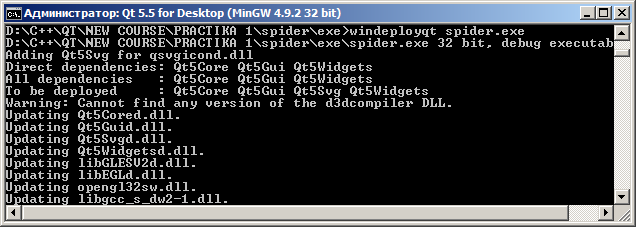
Мы будем пользоваться третьим вариантом. Прежде приходилось вручную отыскивать все dll в каталоге Qt и помещать их вместе со своим exe-файлом. Сейчас Qt предоставляет утилиту ***windеployqt*** , которая делает эту работу.

Для запуска утилиты ***windеployqt*** будем использовать командную строку.



Запуск ***windеployqt*** выглядит следующим образом:

windeployqt <путь\_к\_исполняемому\_файлу>



После завершения работы утилиты, все файлы, требуемые для работы программы, будут находиться в папке с исполняемым файлом.

Замечания:

1. Так как в самом начале работы мы отказались от теневой сборки, то рабочая папка проекта содержит папки Debug и Release. Причем, если используется конфигурация сборки «Отладка», то именно в папку Debug помещается исполняемый файл.

Имеет смысл отделить исполняемый файл от остального содержимого папки Debug . Для этого в файл проекта надо добавить: DESTDIR= < путь\_к\_папке\_с\_исполняемым\_файлом >

1. После того как изменили расположение exe- файла переводчик перестанет работать. Надо добавить файлы перевода в папку exe.

|  |  |  |
| --- | --- | --- |
|  | Рабочая папка проекта /Spider | Расположение exe- файла |
| После отмены теневой сборки |  | Папка /Spider/Debug |
| После добавления в файл проекта  DESTDIR= ./exe |  | Папка /Spider/ exe |

**Задание 10**

Воспользуйтесь утилитой windeployqt для развертывания приложения.

Проверьте, что не испортилась ли Ваша работа по интернационализации. Подумайте, как это исправить.