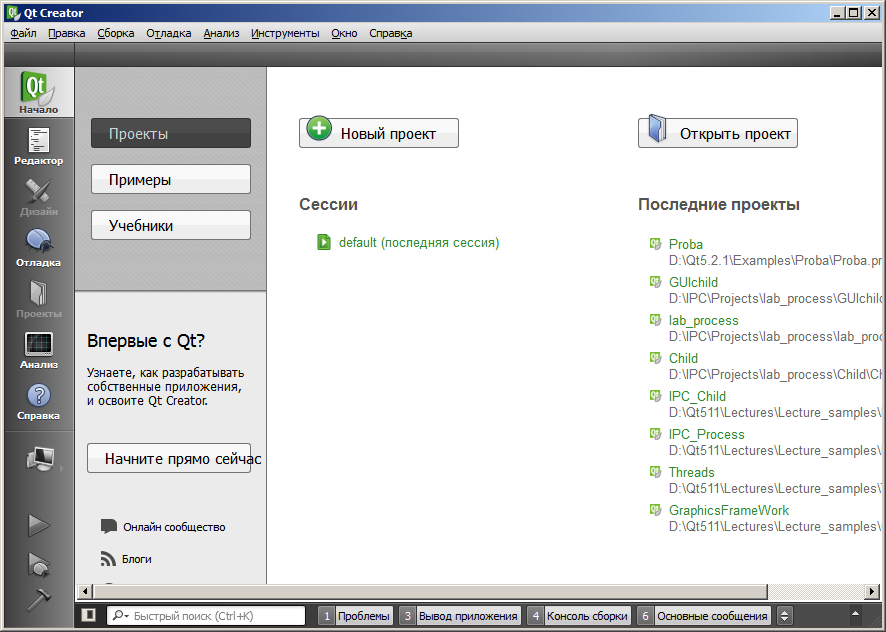
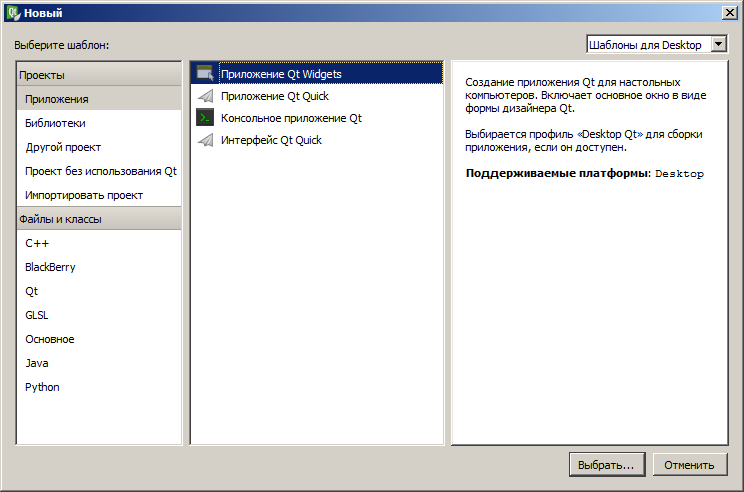
# Создание каркаса приложения

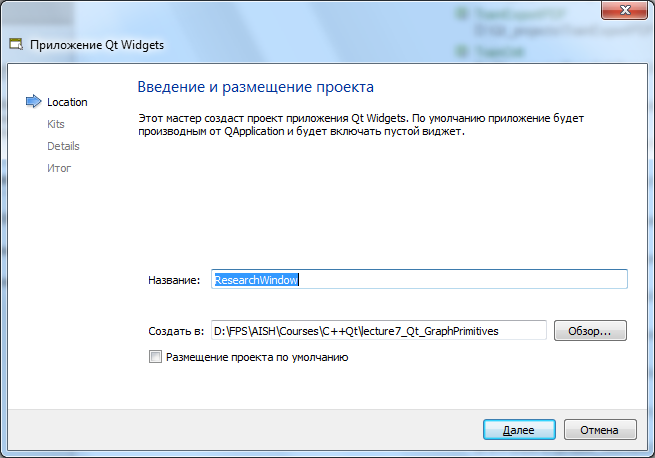
1. Запустить IDE Qt Creator



Нажать кнопку «Новый проект»:

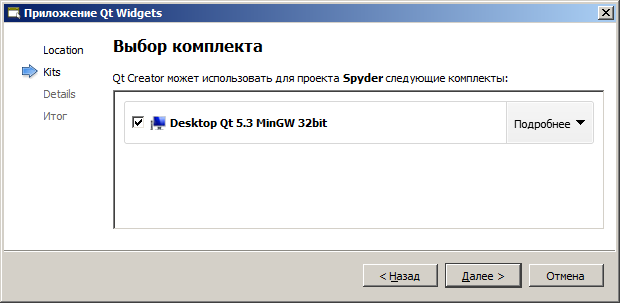


Выбрать отмеченные на рисунке тип проекта и “Приложение Qt Widgets”. Нажать кнопку «Выбрать»



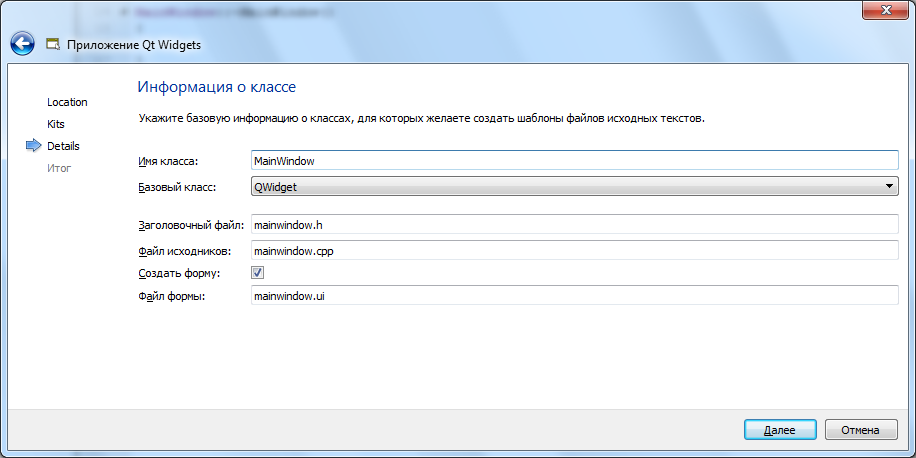
Ввести название проекта и путь. «Далее»

1. Подтвердите выбор инструментария



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

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



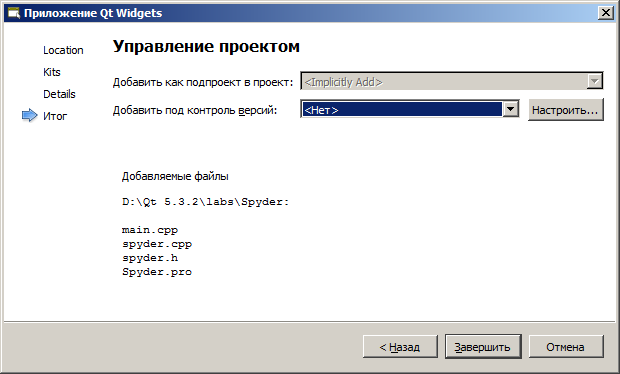
Класс MainWindow будет поддерживать главное окно приложения. Чтобы получить всю базовую функциональность виджета, унаследуем наш класс от класса Qt – **QWidget**.

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

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

«Далее»

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

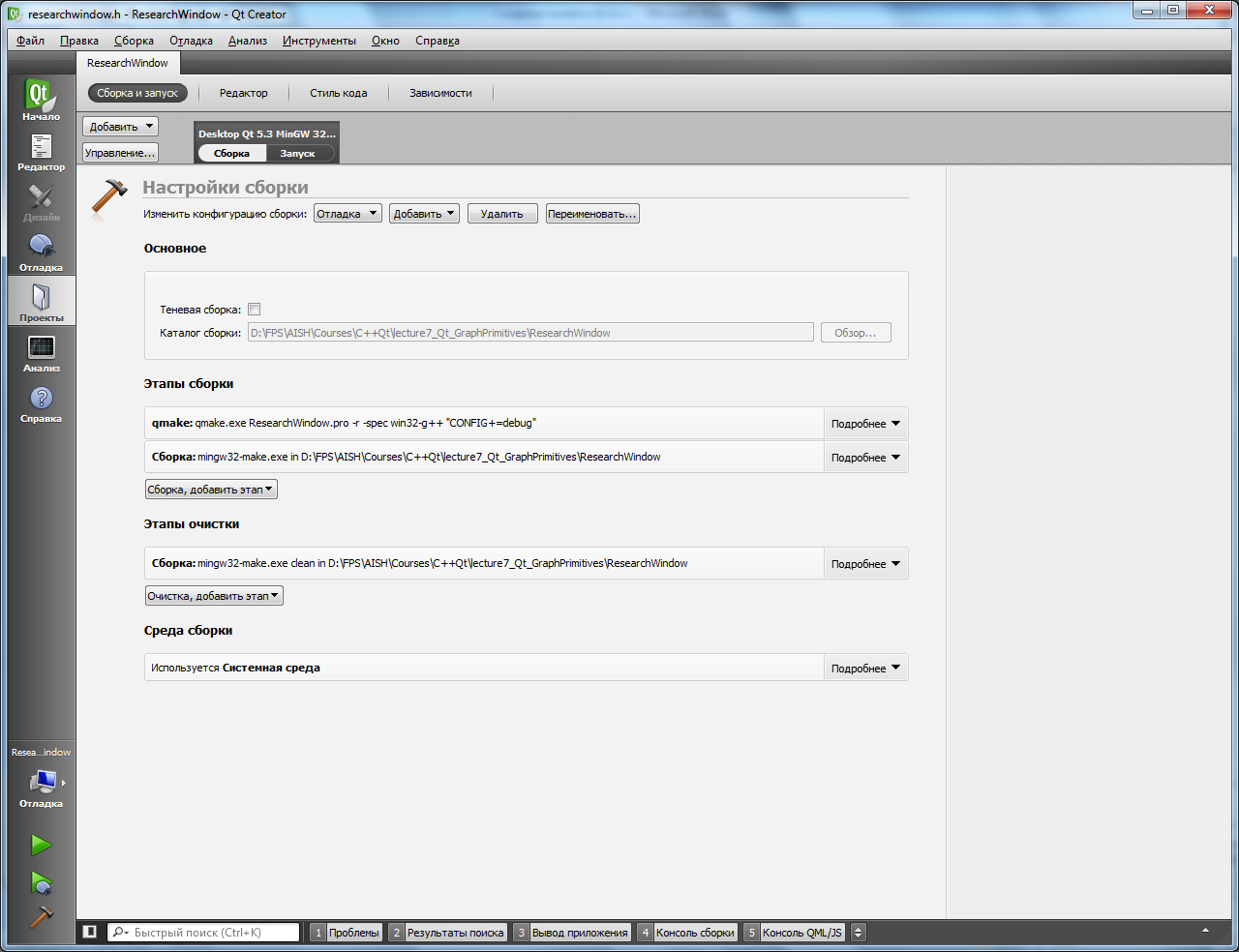


«Завершить»

# 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– в другой папке, что позволяет легко получать разные конфигурации сборки для одного набора исходников.

Для нашего первого учебного проекта теневую сборку можно отключить:



# Первое приложение

#include "researchwindow.h"

#include <QApplication>

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

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

{

QApplication a(argc, argv);//осуществляет контроль

//и управление приложением

ResearchWindow w;//создание объекта для поддержки главного

//окна приложения. В конструкторе класса

//можно осуществить все действия для

//инициализации главного окна

w.show();//команда отобразить главное окно,

//иначе по умолчанию оно после запуска

//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

}

Researchwindow.h

#include <QWidget>

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

{

Q\_OBJECT //макрос, необходимый для поддержки механизма сигналов/слотов

public:

ResearchWindow(QWidget \*parent = 0); //у главного окна приложения

//родителя нет

//(по умолчанию родительским окном //является desktop)

~*ResearchWindow*();

};

# Заголовок главного окна

void QWidget::setWindowTitle(const QString&);

# Состояние главного окна приложения

void QWidget::setWindowState ([Qt::WindowStates](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qt.html) *windowState* );

# Иконка приложения

* #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  
  void [QWidget::setWindowIcon](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qwidget.html)(const QIcon&)

# Цвет фона окна

#include<QPalette>

Создаем объект палитры QPalette c помощью конструктора QPalette**::**QPalette**()**

Задаем цвета  
voidQPalette::[setColor](file:///C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Word\\qpalette.html) ( ColorGroup *group*, ColorRole *role*, const QColor& *color* ) //только для указанной группы

voidQPalette::[setColor](file:///C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Word\\qpalette.html) ( ColorRole *role*, const QColor& *color* ) //для всех групп

void QWidget::**setPalette** ( const QPalette& )

# Классы геометрии.

## Точка

### Конструкторы

### QPoint::QPoint()

### QPoint::QPoint(int*xpos*, int*ypos*)

### **Получение** координат точки (X, Y):

### int QPoint::x() const

### int QPoint::y() const

### **Изменение** координат

### int & QPoint::rx()

### int & QPoint::ry()

## Прямоугольник

### Конструкторы

[**QRectF**](qthelp://org.qt-project.qtcore.532/qtcore/qrectf.html)()

[**QRectF**](qthelp://org.qt-project.qtcore.532/qtcore/qrectf.html)(const QPointF &*topLeft*, const QPointF &*bottomRight*);

[**QRectF**](qthelp://org.qt-project.qtcore.532/qtcore/qrectf.html)(qreal*x*, qreal*y*, qreal*width*, qreal*height*);

### Получение параметров прямоугольника:

### [qreal](qthelp://org.qt-project.qtcore.532/qtcore/qtglobal.html#qreal-typedef) QRectF::height() const

### [qreal](qthelp://org.qt-project.qtcore.532/qtcore/qtglobal.html#qreal-typedef) QRectF::width() const

### int QRect::x() const

### int QRect::y() const

## Линия

### Конструкторы

[**Q**](qthelp://org.qt-project.qtcore.532/qtcore/qrectf.html)[**Line**](qthelp://org.qt-project.qtcore.532/qtcore/qlinef.html)[**F**](qthelp://org.qt-project.qtcore.532/qtcore/qrectf.html)()

[**QLineF**](qthelp://org.qt-project.qtcore.532/qtcore/qlinef.html)(const QPointF &*p1*, const QPointF &*p2*);

[**QLineF**](qthelp://org.qt-project.qtcore.532/qtcore/qlinef.html)(qreal*x1*, qreal*y1*, qreal*x2*, qreal*y2*);

### Позиция начальной точки

### [qreal](qthelp://org.qt-project.qtcore.532/qtcore/qtglobal.html#qreal-typedef) QLineF::x1() const

### [qreal](qthelp://org.qt-project.qtcore.532/qtcore/qtglobal.html#qreal-typedef) QLineF::y1() const

### Позиция конечной точки

### [qreal](qthelp://org.qt-project.qtcore.532/qtcore/qtglobal.html#qreal-typedef) QLineF::x2() const

### [qreal](qthelp://org.qt-project.qtcore.532/qtcore/qtglobal.html#qreal-typedef) QLineF::y2() const

# Методы рисования.

## Линии

**void QPainter::**drawLine**(const [QLineF](qthelp://org.qt-project.qtgui.532/qtcore/qlinef.html) &*line*)**

**void QPainter::**drawLine**(const [QPointF](qthelp://org.qt-project.qtgui.532/qtcore/qpointf.html) &*p1*, const [QPointF](qthelp://org.qt-project.qtgui.532/qtcore/qpointf.html) &*p2*)**

**void QPainter::**drawLine**(int*x1*, int*y1*, int*x2*,  int*y2*)**

## Прямоугольник

**void QPainter::**drawRect**(const [QRectF](qthelp://org.qt-project.qtgui.532/qtcore/qrectf.html) &*rectangle*)**

**void QPainter::**drawRect**(int*x*, int*y*, int*width*, int*height*)**

# Размеры и расположение окна

* Функции задания размеров и положения^

Функция, учитывающие рамку:

move (xpos, ypos);

Функция, НЕ учитывающие рамку:

setGeometry(x, y, width, height);

# Размеры экрана

QDesktopWidget dw;

QRect r = dw.screenGeometry();

# Фиксированные размеры экрана

* void QWidget::setFixedHeight (int height)
* void QWidget::setFixedSize (const QSize& s)
* void QWidget::setFixedSize (int width, int height)
* void QWidget::setFixedWidth (int width)