

СЕМИНАР 1

QIODevice

Сериализация данных

QFile

Отработка ПО подводной системы

Этапы отработки ПО:

1. Отработка алгоритмов на моделях и имитаторах подсистем НПА;
2. Полунатурная отработка ПО;
3. Натурная отработка подводной системы.



Отработка ПО подводной системы в натуральных условиях



Полунатурная отработка ПО

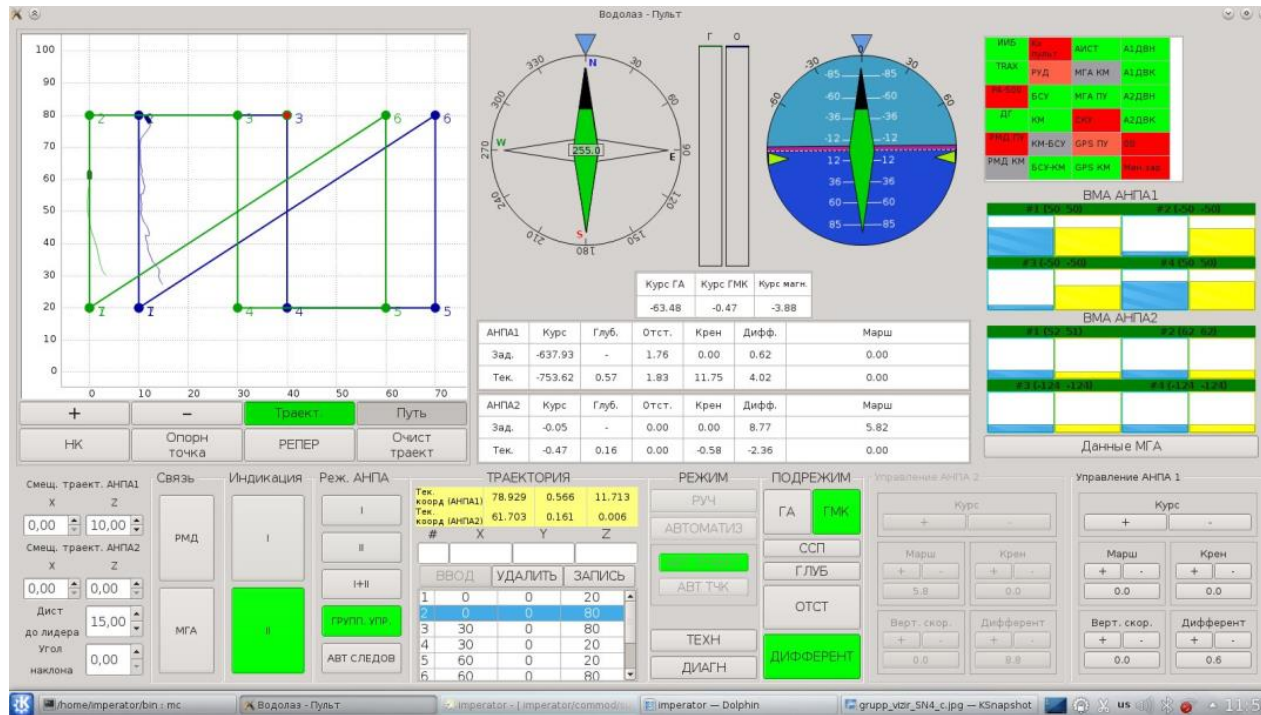


Полунатурная отработка ПО



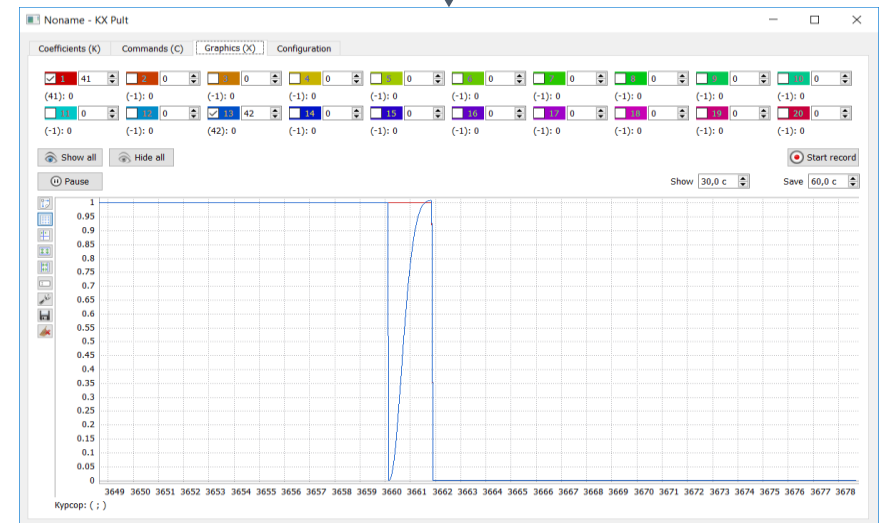
Отработка алгоритмов на моделях и имитаторах

Пульт управления:



ПО ТНПА
(СУ и математическая
модель ТНПА)

kx_pult:



Классы Qt для работы с каталогами/файлами

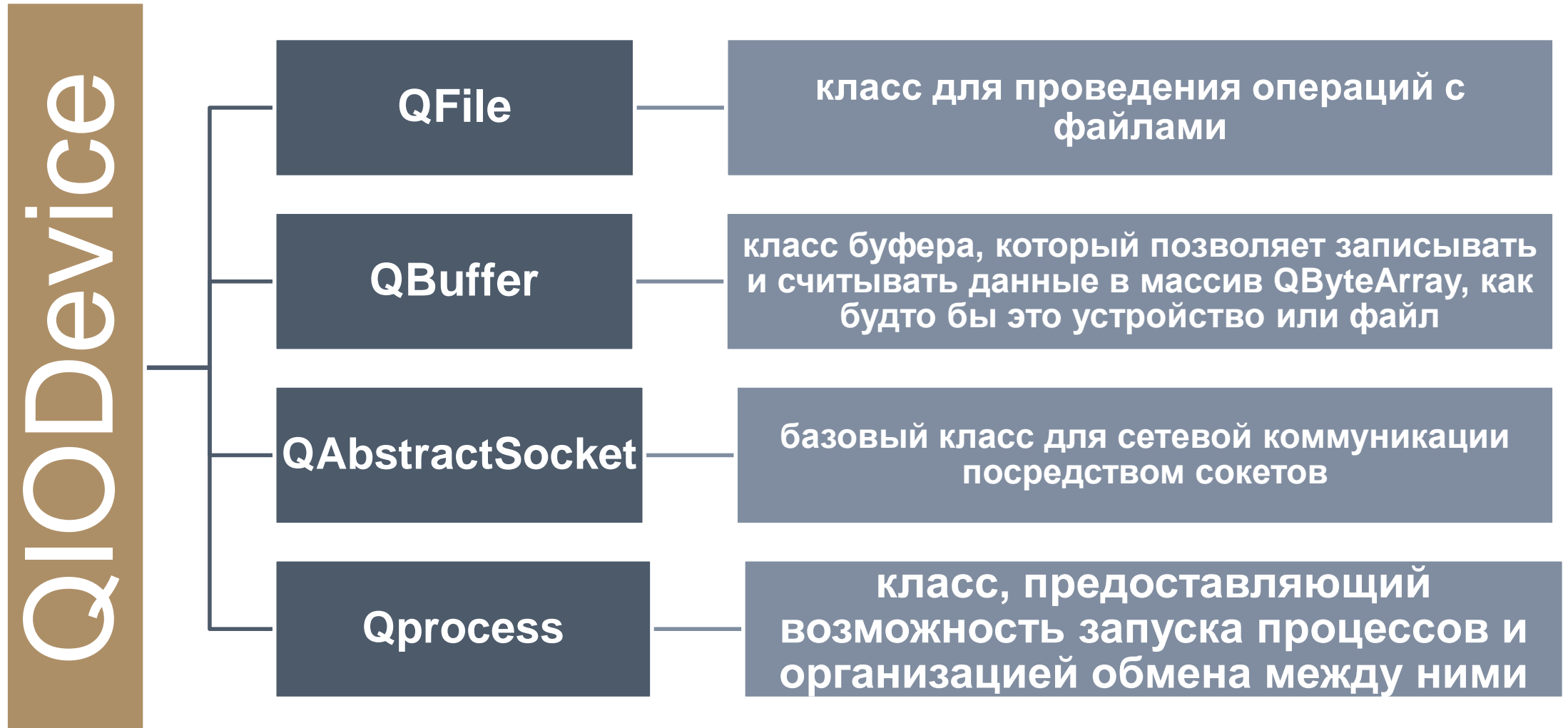
- QIODevice – абстрактный класс ввода вывода
- QDir – для работы с каталогами
- QFile – для работы с файлами
- QFileInfo – для получения файловой информации
- QBuffer – для эмуляции файлов в памяти компьютера

QIODevice

QIODevice — абстрактный класс, обобщающий устройство ввода/вывода, который содержит виртуальные методы для открытия и закрытия устройства ввода/вывода, а также для чтения и записи блоков данных или отдельных символов.

.

Наследники QIODevice



QFile. Этапы работы.

Создание объекта QFile для работы с файлом



Проверка существования файла



Открытие файла



Чтение/запись в файл



Закрытие файла

QFile. Этапы работы.

1. Создание объекта для работы с файлом:

```
QFile file ("имя_файла");
```

или

```
QFile file;  
File.setName ("имя_файла");
```

*при указании пути используется «/» вне зависимости от ОС

**если путь к файлу не указан, а указано только имя файла, то путь к файлу относительный и берет отсчет от папки, в которой располагается исполняемый файл проекта.

QFile. Этапы работы.

2. Проверить существует ли нужный вам файл можно статическим методом `QFile::exists()` :

`QFile::exists("имя_файла") ; //вернет true, если файл существует`
или нестатическим методом `exists()`:
`file.exists() ;`

QFile. Этапы работы.

3. Открытие файла:

```
file.open(QIODevice::”режим_открытия_файла”);
```

Режимы открытия файла:

- **ReadOnly** – открытие только для чтения;
- **WriteOnly** – открытие только для записи данных;
- **ReadWrite** – открытие для чтения и записи данных;
- **Append** – открытие для добавления данных;
- **Unbuffered** – открытие для непосредственного доступа к данным в обход промежуточных буферов чтения и записи;
- **Text** – преобразование символов переноса строки в зависимости от платформы. (В Windows “\r\n”, а в Mac OS и UNIX – “\r”);
- **Truncate** - все данные устройства, по возможности должны быть удалены при открытии.
- **NotOpen** – устройство не открыто.

QFile. Этапы работы.

4. Чтение и запись.

`QIODevice::read()` – считывание файлов блоками

`QIODevice::write()` – запись файлов блоками

Для считывания всех данных за 1 раз `QIODevice::readAll()`;

Для считывания строки – `QIODevice::readLine()`;

Для считывания символа – `QIODevice::getChar()`;

Рекомендуемый способ чтения/записи – с использованием потоков ввода/вывода.

QFile. Этапы работы.

5. В конце работы файл необходимо закрыть.

```
file.close();
```

*С закрытием осуществляется запись всех данных буфера.

Если требуется выполнить запись данных буфера в файл без его закрытия. То вызывается метод

```
QFile::flush().
```

Кроме того файл можно удалить используя статический метод
`QFile::remove("имя файла");`

Потоки ввода/вывода.

Для записи и чтения данных в файл помимо стандартных функций можно воспользоваться потоками ввода/вывода (QTextStream и QDataStream).

Преимущества потоков ввода/вывода перед стандартными функциями чтения/записи QFile:

- Просто передавать данные разных типов (от строк до изображений и т.п.);
- Можно использовать для передачи данных своих типов, унаследованных от QIODevice;
- Для записи данных в поток используется оператор “<<”;
- Для чтения данных из потока можно использовать оператор “>>”.

Потоки ввода/вывода. QTextStream

QTextStream предназначен для чтения текстовых данных.

В качестве текстовых данных могут выступать не только объекты, созданные классами, унаследованными от QIODevice, но и переменные ТИПОВ:

`char, Qchar, char*, QString, QByteArray, short, int, long, float и double.`

(числовые данные, передаваемые в поток автоматически преобразуются в текст)

Потоки ввода/вывода. QDataStream

QDataStream – гарант того, что формат, в котором будут записаны данные, останется платформонезависимым и его можно будет считать и обработать и на других платформах.

*по этой причине этот класс незаменим при передаче данных по сети с использованием сокетных соединений.

!Формат данных, используемый QDataStream, в процессе разработки версий Qt претерпел множество изменений и продолжает изменяться. По этой причине этот класс содержит информацию о версии, поэтому необходимо вызвать метод `setVersion(идентификатор_версии);`

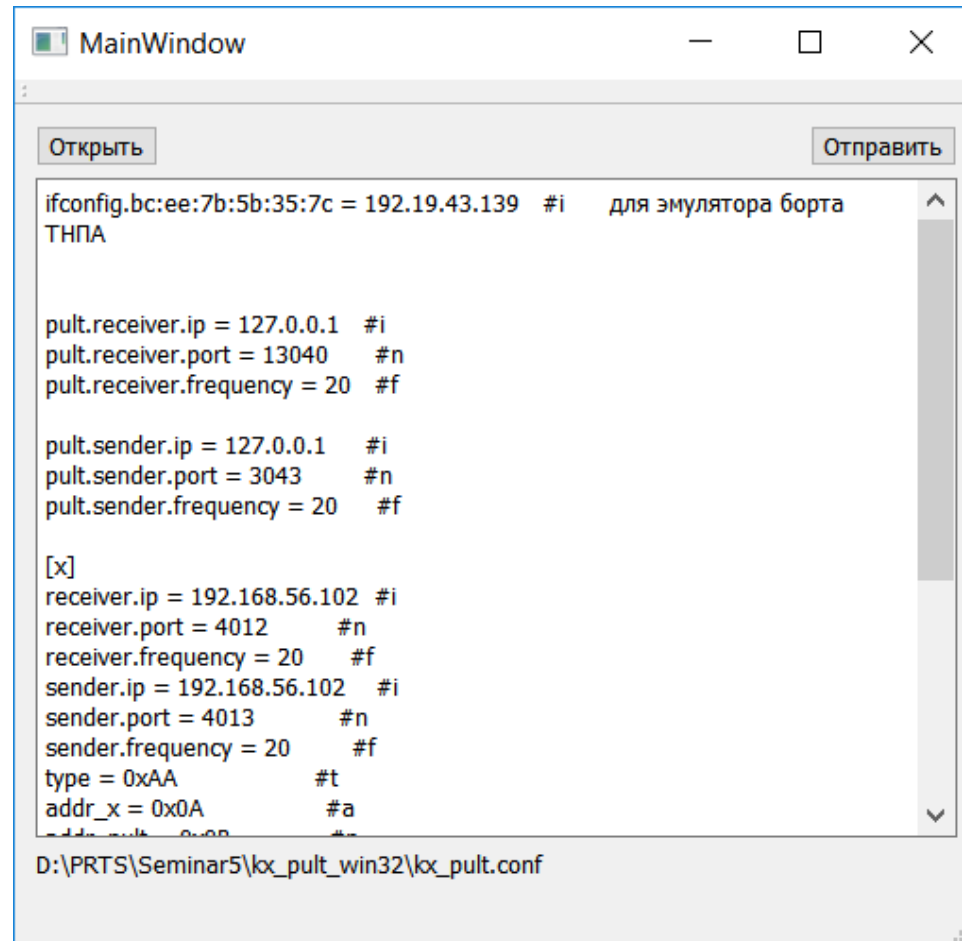
Сериализация данных

- Сериализация – перевод структуры или объекта в бинарный вид. Обратная операция – десериализация.
- Для чего используется:
 - Передача объектов по сети
 - Сохранение объектов в файлы
 - Сохранение информации о текущем состоянии программы
 - Вызов удаленных процедур

Результат сериализации может быть представлен в формате JSON, XML, HTTP и т.д.

QFile. Практическая часть 1.

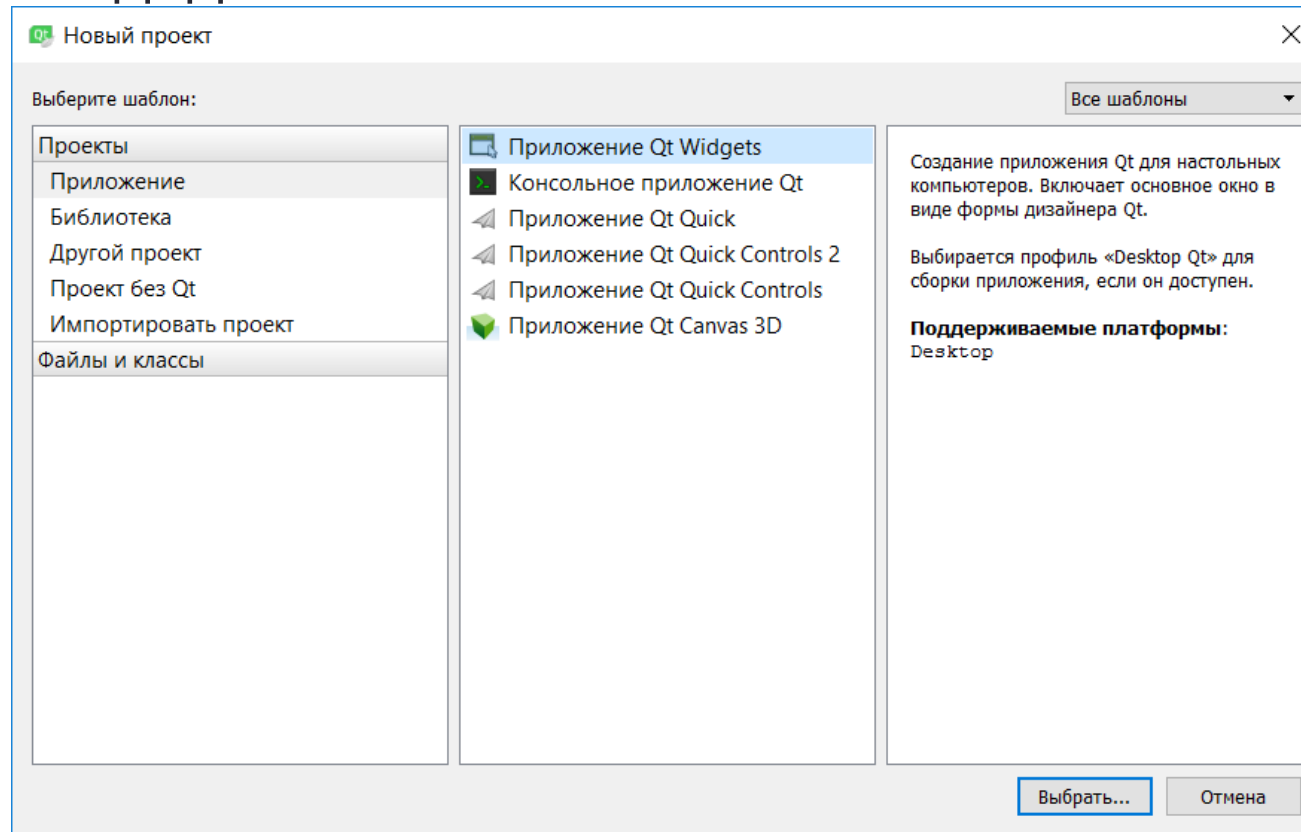
Создадим приложение, которое открывает файл и выводит его содержимое.



QFile. Практическая часть 1.

Создадим новый проект.

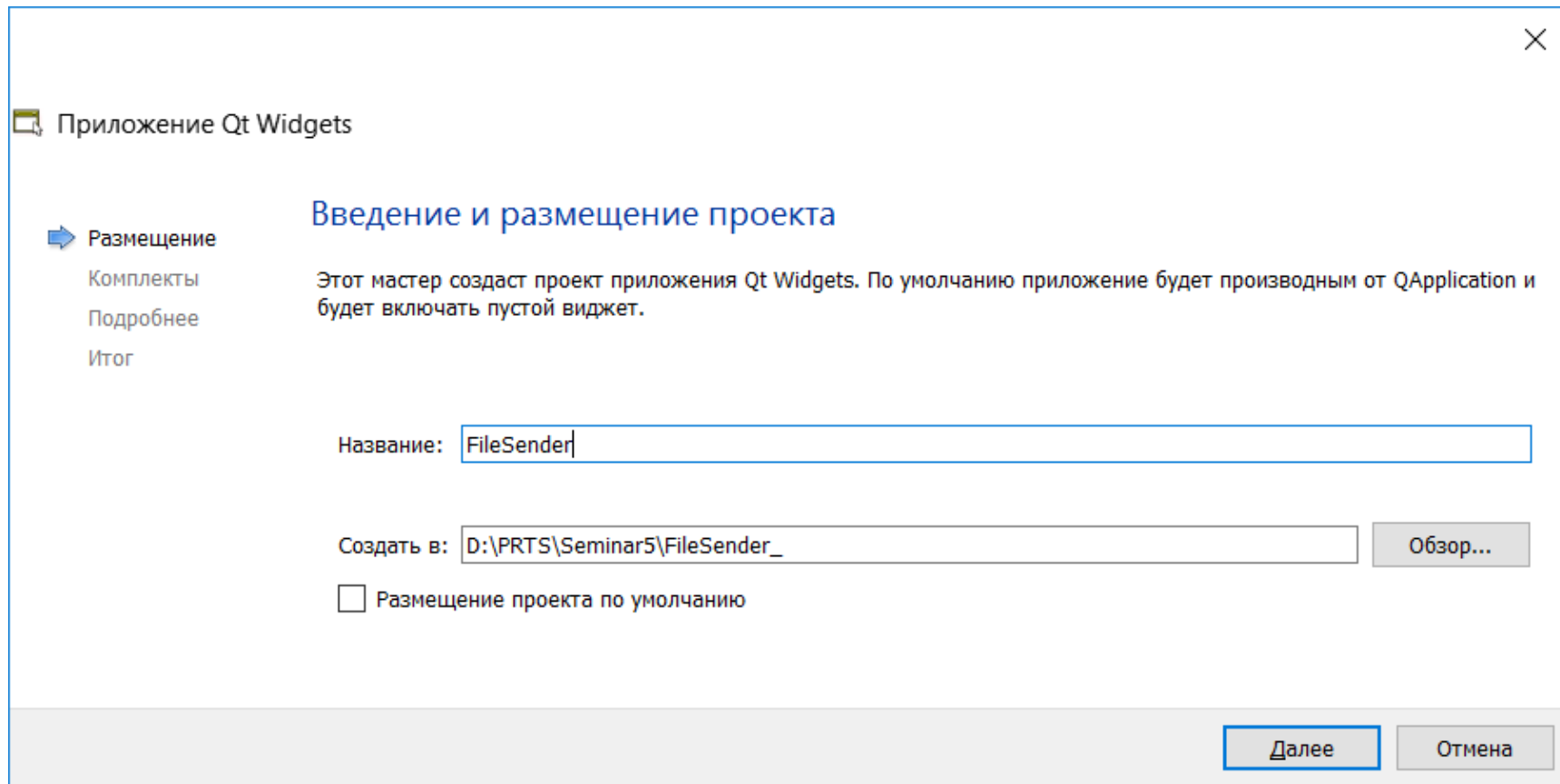
В этом проекте создадим класс MainWindow..



QFile. Практическая часть 1.

Создадим новый проект.

В этом проекте создадим класс MainWindow..



Приложение Qt Widgets

Введение и размещение проекта

Этот мастер создаст проект приложения Qt Widgets. По умолчанию приложение будет производным от QApplication и будет включать пустой виджет.

Название:

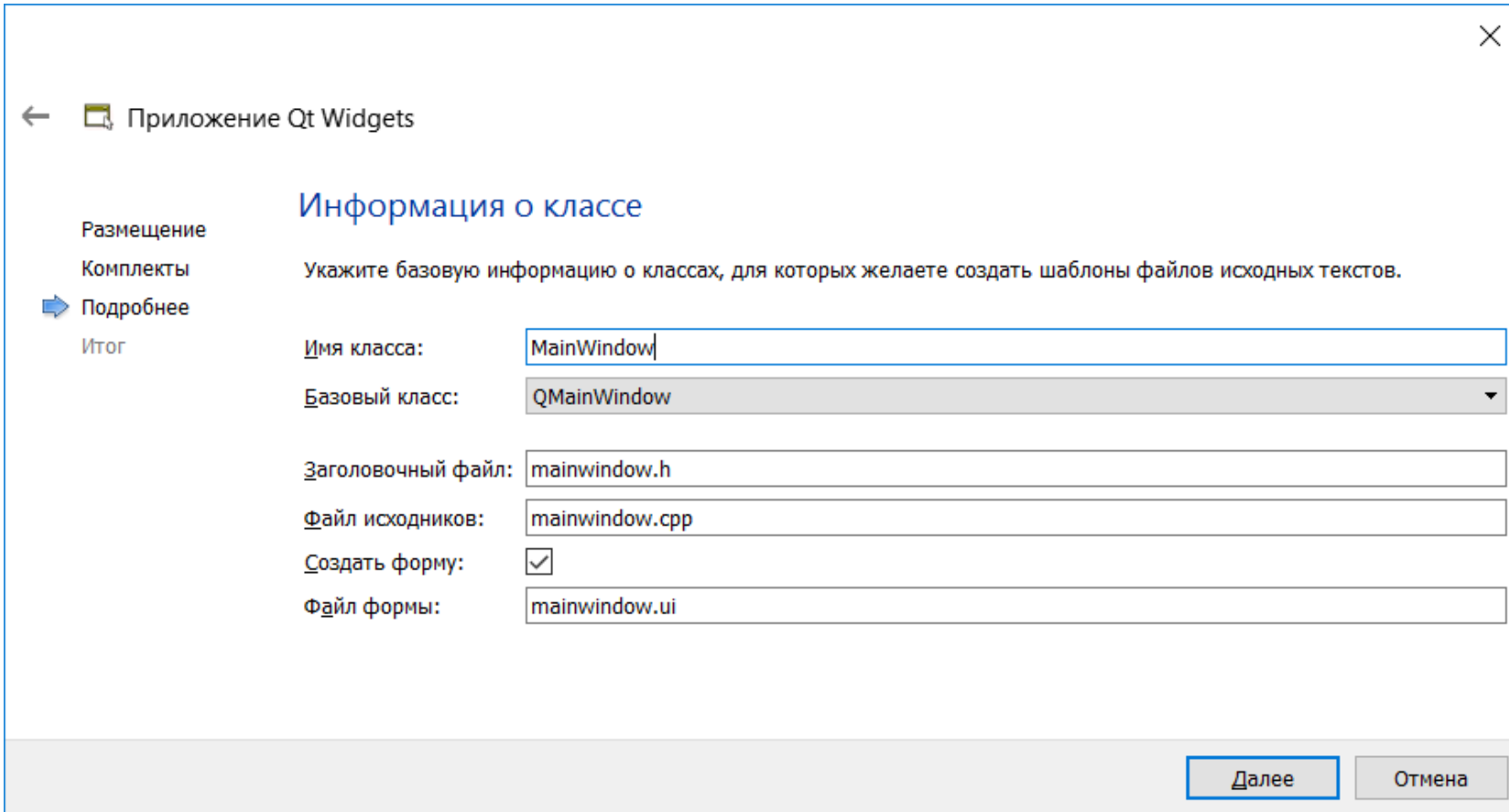
Создать в: Обзор...

☐ Размещение проекта по умолчанию

Далее Отмена

QFile. Практическая часть 1.

В этом проекте создадим класс MainWindow.. и даже подключим к нему форму



← Приложение Qt Widgets

Размещение
Комплекты
➔ Подробнее
Итог

Информация о классе

Укажите базовую информацию о классах, для которых желаете создать шаблоны файлов исходных текстов.

Имя класса:

Базовый класс:

Заголовочный файл:

Файл исходников:

Создать форму: ☒

Файл формы:

Далее Отмена

QFile. Практическая часть 1.

Mainwindow.h

Qt Creator

Меню Окно Справка

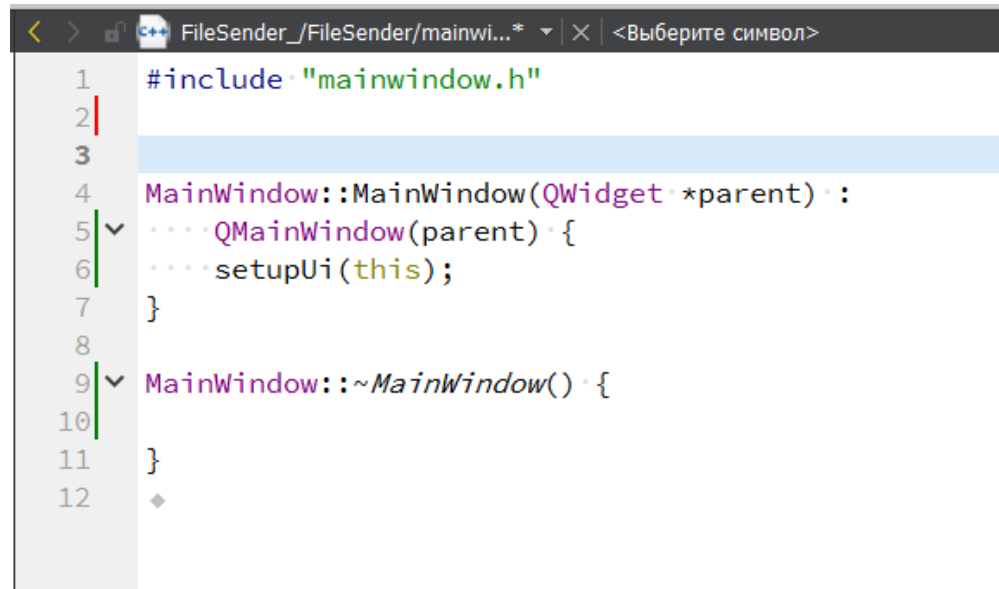
< > FileSender_/FileSender/mainwin... ~MainWindow()

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5
6  #include "ui_mainwindow.h"
7
8  class MainWindow : public QMainWindow, public Ui::MainWindow {
9      Q_OBJECT
10
11  public:
12      explicit MainWindow(QWidget *parent = 0);
13      ~MainWindow();
14
15  private:
16
17  };
18
19  #endif // MAINWINDOW_H
20
```

Вид класса MainWindow для ленивых фанатов
множественного наследования формы)))

QFile. Практическая часть 1.

Mainwindow.cpp

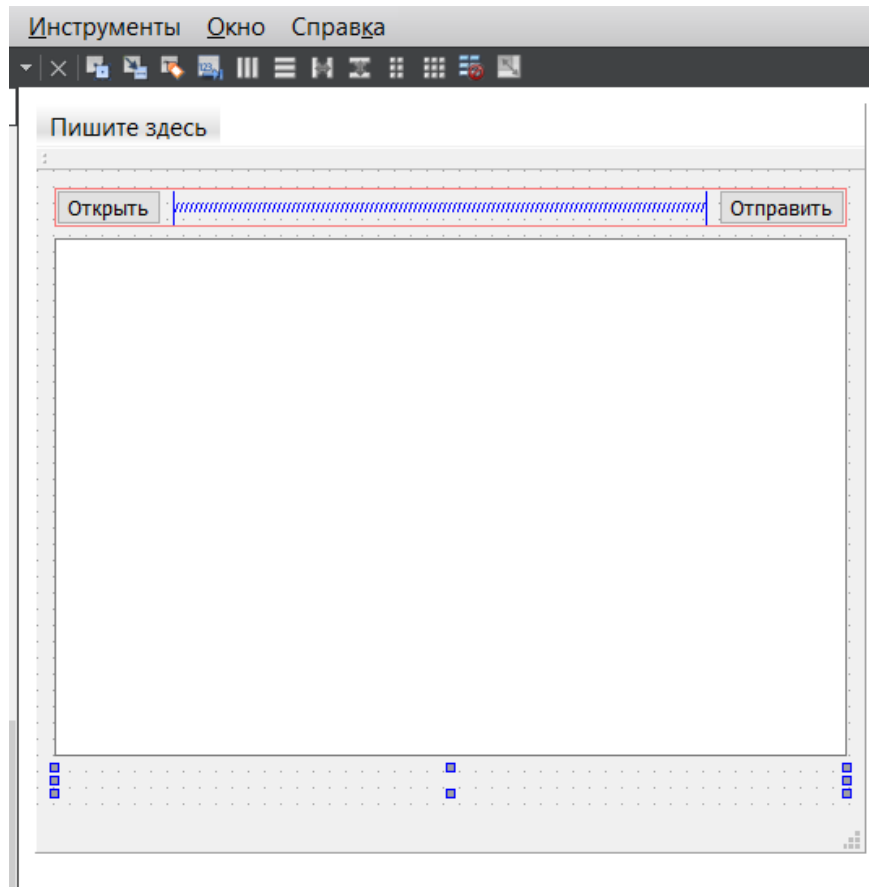


```
< > FileSender_/FileSender/mainwi... * | X | <Выберите символ>
1  #include "mainwindow.h"
2
3
4  MainWindow::MainWindow(QWidget *parent) :
5      QMainWindow(parent) {
6      setupUi(this);
7  }
8
9  MainWindow::~MainWindow() {
10
11 }
12 ◆
```

Вид класса MainWindow для ленивых фанатов
множественного наследования формы)))

QFile. Практическая часть 1.

Mainwindow.ui

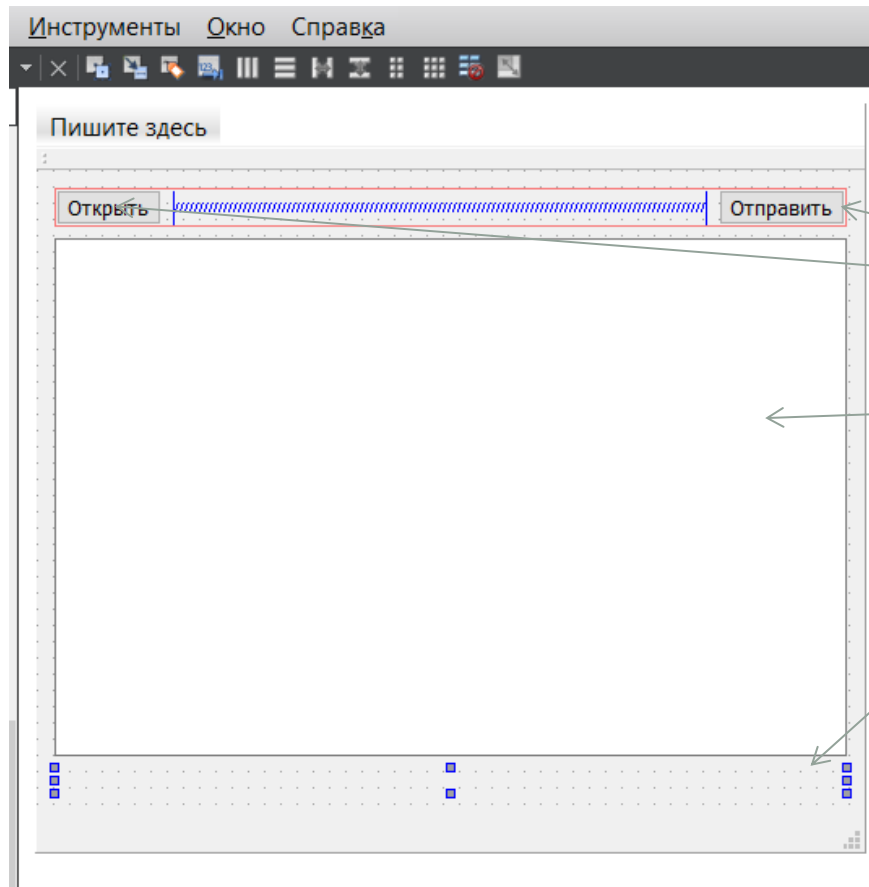


Для неравнодушных..
Названия виджетов формы

Объект	Класс
▼ MainWindow	QMainWindow
▼ centralWidget	QWidget
▼ hlayBtns	QHBoxLayout
spacerBtns	Spacer
tbtnOpen	QToolButton
tbtnSend	QToolButton
lblPath	QLabel
txtBrFile	QTextBrowser
menuBar	QMenuBar
mainToolBar	QToolBar
statusBar	QStatusBar

QFile. Практическая часть 1.

Mainwindow.ui

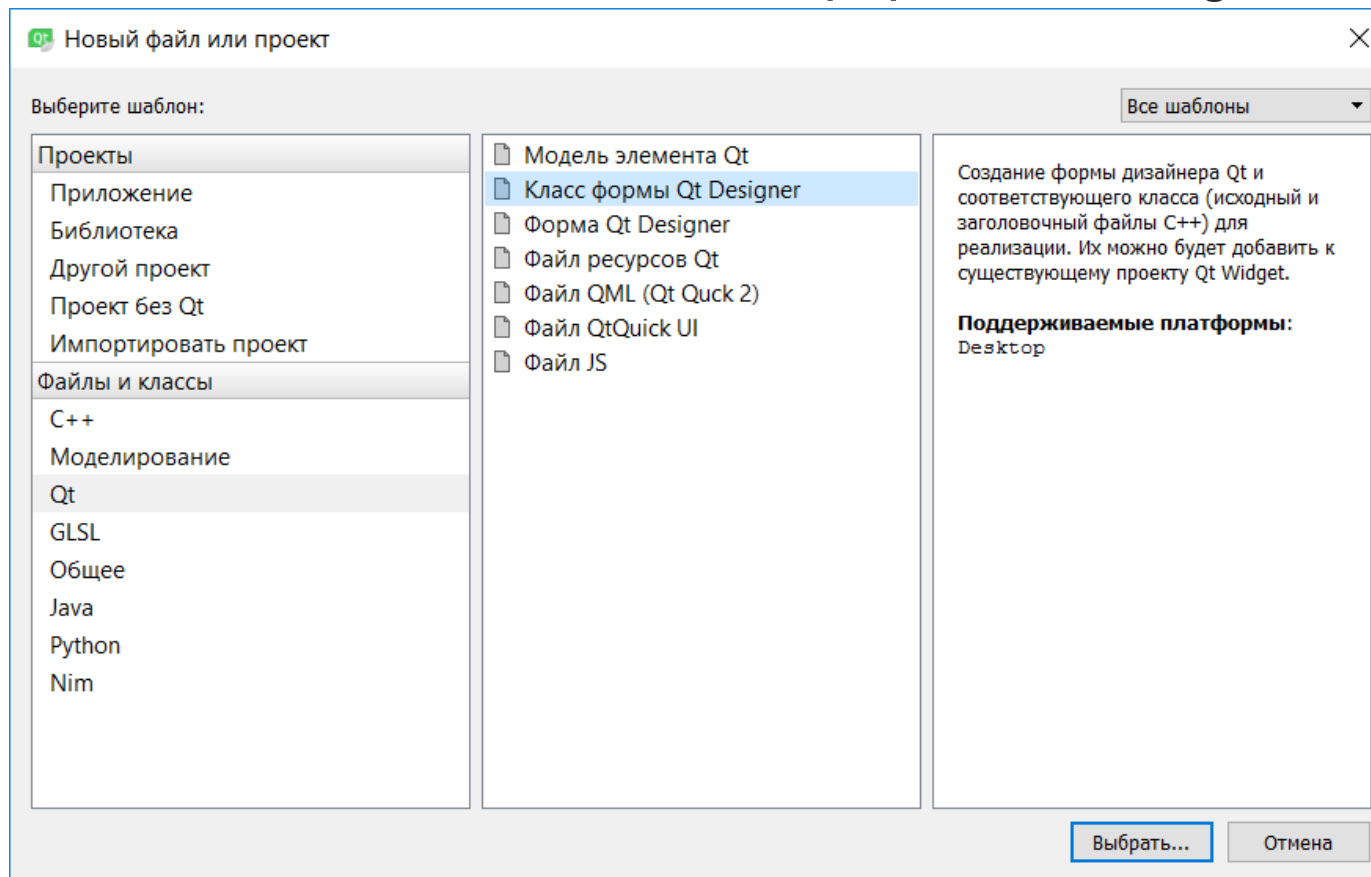


Для неравнодушных..
Названия виджетов формы

Объект	Класс
▼ MainWindow	QMainWindow
▼ centralWidget	QWidget
▼ hlayBtns	QHBoxLayout
spacerBtns	Spacer
tbtnOpen	QToolButton
tbtnSend	QToolButton
lblPath	QLabel
txtBrFile	QTextBrowser
menuBar	QMenuBar
mainToolBar	QToolBar
statusBar	QStatusBar

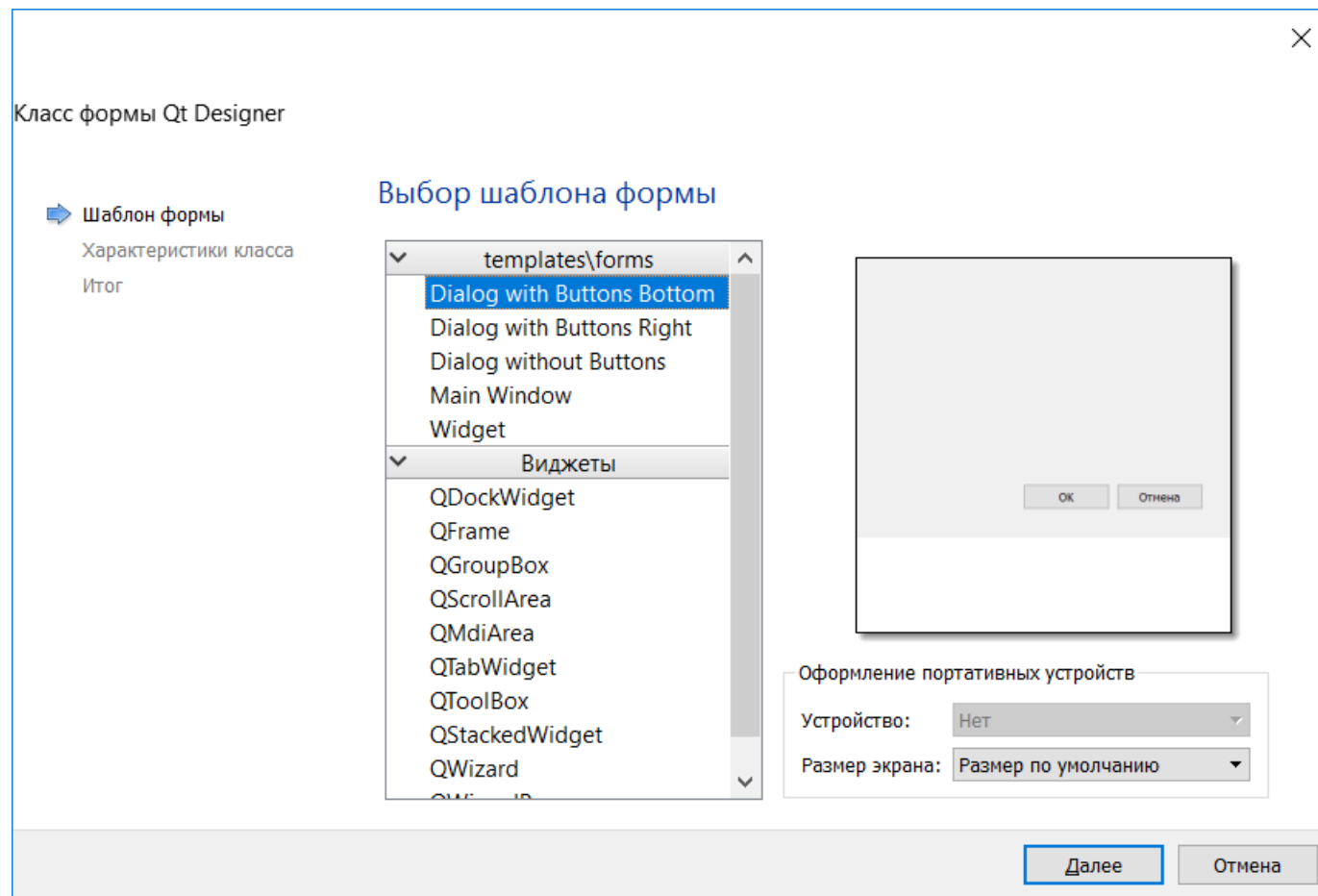
QFile. Практическая часть 1.

При нажатии на кнопку открыть, пользователь может ввести путь к файлу, который необходимо открыть. Для этого необходимо создать диалоговое окно. Для этого создадим класс формы QtDesigner.



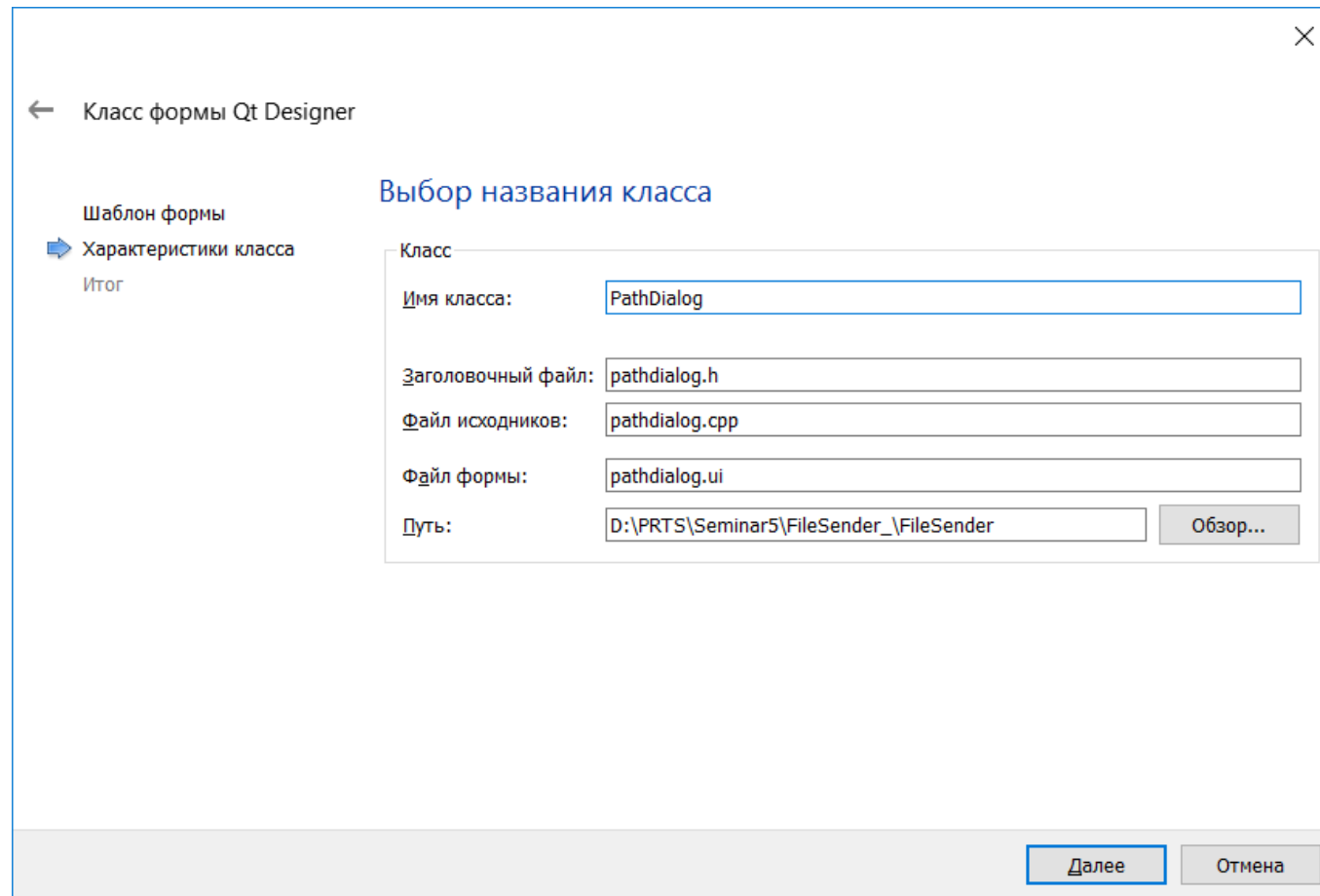
QFile. Практическая часть 1.

При выборе шаблона формы остановимся на каком-либо виде формы Dialog (я выбрала Dialog with Buttons Bottom)



QFile. Практическая часть 1.

При выборе шаблона формы остановимся на каком-либо виде формы Dialog (я выбрала Dialog with Buttons Bottom)



The screenshot shows the 'Class Form Name' dialog box in Qt Designer. The left sidebar contains a tree view with the following items: '← Класс формы Qt Designer', 'Шаблон формы', '➔ Характеристики класса' (which is selected), and 'Итог'. The main area is titled 'Выбор названия класса' and contains a 'Класс' section with the following fields:

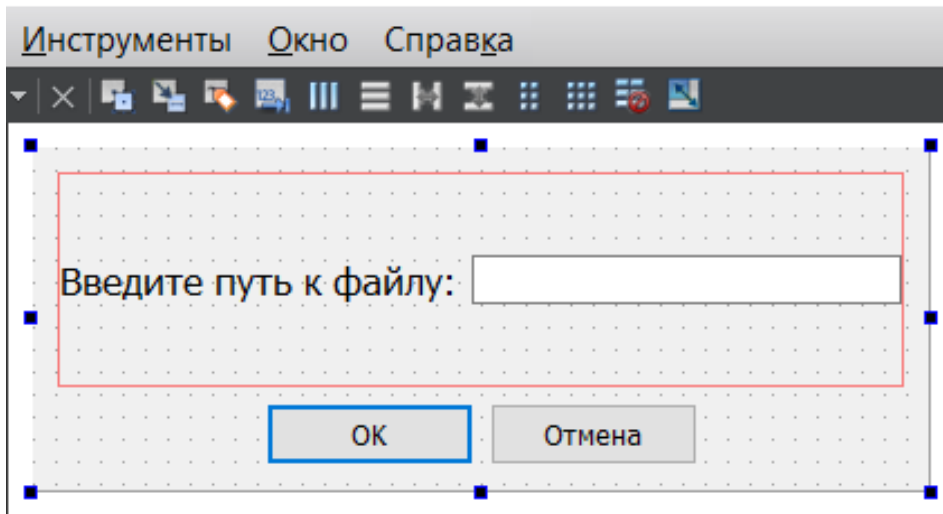
- Имя класса: PathDialog
- Заголовочный файл: pathdialog.h
- Файл исходников: pathdialog.cpp
- Файл формы: pathdialog.ui
- Путь: D:\PRTS\Seminar5\FileSender_\FileSender (with an 'Обзор...' button next to it)

At the bottom right of the dialog, there are two buttons: 'Далее' and 'Отмена'.

QFile. Практическая часть 1.

При выборе шаблона формы остановимся на каком-либо виде формы Dialog (я выбрала Dialog with Buttons Bottom)

pathdialog.ui



Для неравнодушных..
Названия виджетов диалога

Объект	Класс
PathDialog	QDialog
horizontalLayout	QHBoxLayout
edtPath	QLineEdit
lblInputPath	QLabel
btnBox	QDialogButtonBox

QFile. Практическая часть 1.

Pathdialog.h

```
менты Окно Справка
< > pathdialog.h* ~PathDialog()
1 #ifndef PATHDIALOG_H
2 #define PATHDIALOG_H
3
4 #include <QDialog>
5
6 #include "ui_pathdialog.h"
7
8 class PathDialog : public QDialog, public Ui::PathDialog {
9     Q_OBJECT
10
11 public:
12     explicit PathDialog(QWidget *parent = 0);
13     ~PathDialog();
14
15 private:
16     ....
17 };
18
19 #endif // PATHDIALOG_H
20
```

Pathdialog.cpp

```
менты Окно Справка
< > pathdialog.cpp PathDialog::~PathDialog()
1 #include "pathdialog.h"
2
3 PathDialog::PathDialog(QWidget *parent) :
4     QDialog(parent) {
5     setupUi(this);
6 }
7
8 PathDialog::~PathDialog() {
9
10 }
11
```

QFile. Практическая часть 1.

При нажатии на кнопку «Открыть» перед пользователем должно высветиться диалоговое окно для ввода пути к файлу:

Mainwindow.h

```
< > FileSender_/FileSender/mainwi...* on_tbtnOpen_clicked(): void
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5
6  #include "ui_mainwindow.h"
7
8  class MainWindow : public QMainWindow, public Ui::MainWindow {
9      Q_OBJECT
10
11  public:
12      explicit MainWindow(QWidget *parent = 0);
13      ~MainWindow();
14
15  private:
16
17  private slots:
18      //добавим слот, который будет вызываться при нажатии
19      //на кнопку "Открыть"
20      void on_tbtnOpen_clicked();
21
22      ....
23  };
24
25  #endif // MAINWINDOW_H
```

QFile. Практическая часть 1.

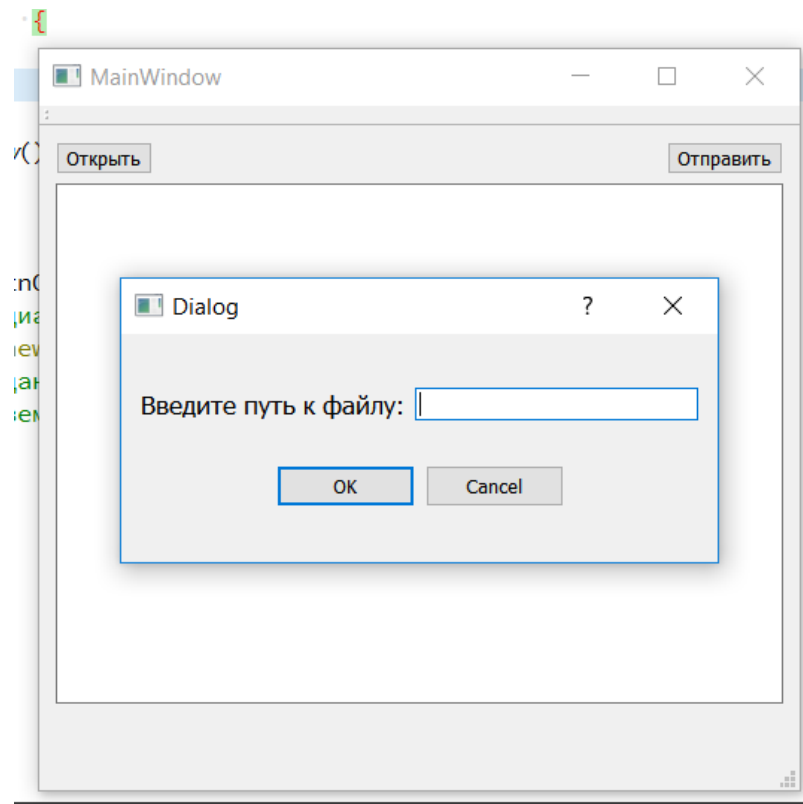
Добавим реализацию слота вызова диалогового окна в классе MainWindow

Mainwindow.cpp

```
> FileSender_/FileSender/mainwi... * | X | MainWindow::~~MainWindow()
1  #include "mainwindow.h"
2
3  //подключим заголовочный файл класса диалогового окна
4  #include "pathdialog.h"
5
6  MainWindow::MainWindow(QWidget *parent) :
7  ... QMainWindow(parent) {
8  ... setupUi(this);
9  }
10
11  MainWindow::~~MainWindow() {
12
13  }
14
15  void MainWindow::on_tbtnOpen_clicked() {
16  ... //создадим объект диалогового окна
17  ... PathDialog *dial= new PathDialog(this);
18  ... //по умолчанию созданный объект не будет виден, для
19  ... //отображения вызовем метод show()
20  ... dial->show();
21  }
22  ♦
```

QFile. Практическая часть 1.

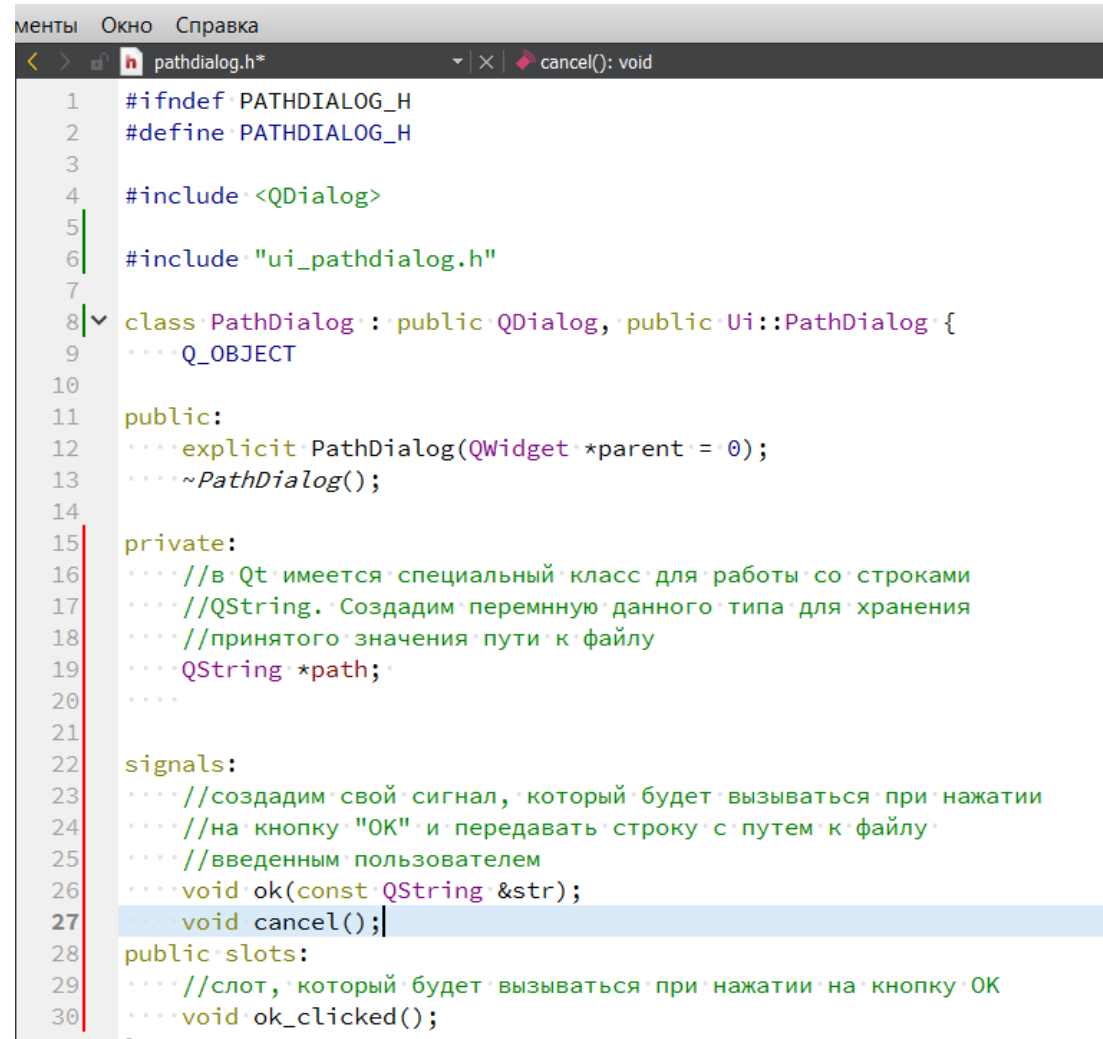
Насладимся промежуточным результатом и убедимся, что при нажатии на кнопку «открыть» у нас появляется диалоговое окно.



QFile. Практическая часть 1.

Введенный пользователем путь к файлу как-то должен попасть в код нашей программы...

Добавим в класс диалогового окна новый сигнал, в параметрах которого передадим строку с введенным путем к файлу. И соединим этот сигнал со слотом в классе главного окна.



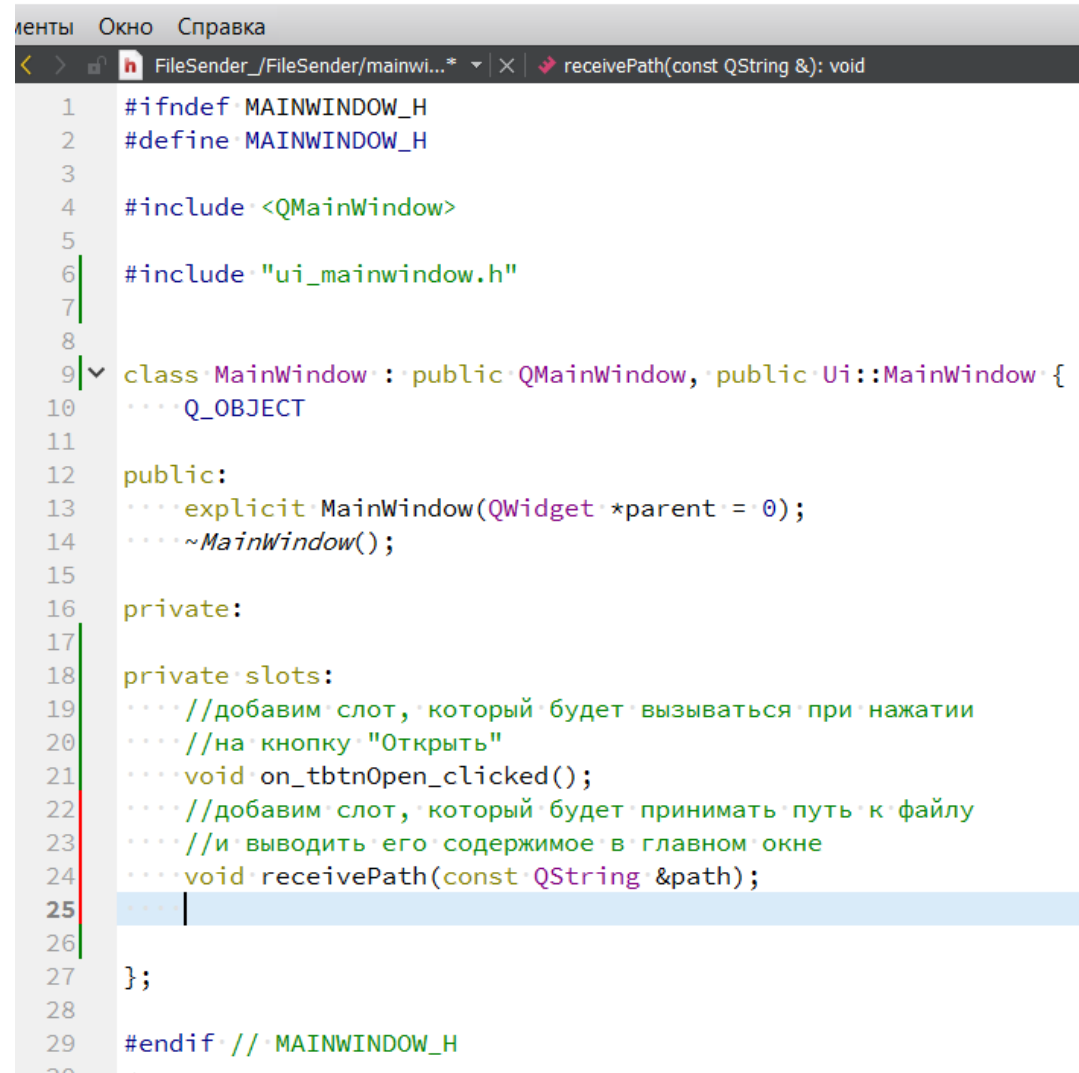
```
1  #ifndef PATHDIALOG_H
2  #define PATHDIALOG_H
3
4  #include <QDialog>
5
6  #include "ui_pathdialog.h"
7
8  class PathDialog : public QDialog, public Ui::PathDialog {
9      Q_OBJECT
10
11 public:
12     explicit PathDialog(QWidget *parent = 0);
13     ~PathDialog();
14
15 private:
16     //в Qt имеется специальный класс для работы со строками
17     //QString. Создадим переменную данного типа для хранения
18     //принятого значения пути к файлу
19     QString *path;
20
21
22 signals:
23     //создадим свой сигнал, который будет вызываться при нажатии
24     //на кнопку "ОК" и передавать строку с путем к файлу
25     //введенным пользователем
26     void ok(const QString &str);
27     void cancel();
28
29 public slots:
30     //слот, который будет вызываться при нажатии на кнопку ОК
31     void ok_clicked();
```


QFile. Практическая часть 1.

```
pathdialog.cpp* PathDialog::ok_clicked(): void
1  #include "pathdialog.h"
2  #include <QPushButton>
3
4
5
6  PathDialog::PathDialog(QWidget *parent) :
7  {
8      QDialog(parent) {
9          setupUi(this);
10         path = new QString("");
11         //соединим сигнал clicked(), который вызывается при нажатии на кнопку OK
12         //со слотом ok_clicked(), который реализуем далее в классе нашего диалогового
13         //окна
14         connect(btnBox->button(QDialogButtonBox::Ok), SIGNAL(clicked()),
15                 this, SLOT(ok_clicked()));
16         connect(btnBox->button(QDialogButtonBox::Cancel), SIGNAL(clicked()),
17                 this, SLOT(close()));
18     }
19
20 PathDialog::~PathDialog() {
21 }
22 //реализация метода слота ok_clicked()
23 void PathDialog::ok_clicked() {
24     //передадим сигнал с путем к файлу
25     emit ok(edtPath->text());
26     //закроем окно
27     close();
28 }
```

QFile. Практическая часть 1.

В классе MainWindow добавим слот receivePath()



```
менты Окно Справка
FileSender_/FileSender/mainwi... * × | receivePath(const QString &): void
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5
6  #include "ui_mainwindow.h"
7
8
9  class MainWindow : public QMainWindow, public Ui::MainWindow {
10     ... Q_OBJECT
11
12     public:
13         ... explicit MainWindow(QWidget *parent = 0);
14         ... ~MainWindow();
15
16     private:
17
18     private slots:
19         ... //добавим слот, который будет вызываться при нажатии
20         ... //на кнопку "Открыть"
21         ... void on_tbtnOpen_clicked();
22         ... //добавим слот, который будет принимать путь к файлу
23         ... //и выводить его содержимое в главном окне
24         ... void receivePath(const QString &path);
25         ...
26
27     };
28
29 #endif // MAINWINDOW_H
```

QFile. Практическая часть 1.

В классе MainWindow добавим реализацию слота receivePath(), а также соединим его с сигналом диалогового окна:

```
void MainWindow::on_tbtnOpen_clicked() {  
    //создадим объект диалогового окна  
    PathDialog *dial= new PathDialog(this);  
    //по умолчанию созданный объект не будет виден, для  
    //отображения вызовем метод show()  
    dial->show();  
    //соединим сигнал диалогового окна с путем к файлу  
    //и слот receivePath() нашего класса  
    connect(dial, SIGNAL(ok(QString)),  
            this, SLOT(receivePath(QString)));  
}
```

Qt Creator

Файлы Окно Справка

FileSender_/FileSender/mainwi... * | X | MainWindow::receivePath(const QString &): void

```
19     //по умолчанию созданный объект не будет виден, для  
20     //отображения вызовем метод show()  
21     dial->show();  
22 }  
23  
24 void MainWindow::receivePath(const QString &path){  
25     //создадим объект файла  
26     QFile file(path);  
27     QByteArray buffer; //класс Qt, который используется для  
28     //промежуточного хранения данных  
29  
30     //проверим существует ли файл, если файл не существует выведем  
31     //информацию об ошибке  
32     if (!file.exists()) lblPath->setText(tr("The file doesn't exists"));  
33     //если файл существует, то  
34     else {  
35         //откроем файл для чтения (QFile::ReadOnly)  
36         file.open(QFile::ReadOnly);  
37         //считаем все содержимое файла методом readAll() в buffer  
38         buffer=file.readAll();  
39         //выведем содержимое файла и путь к нему  
40         txtBrFile->setText(buffer);  
41         lblPath->setText(path);  
42     }  
43 }
```

QFile. Практическая часть 1.

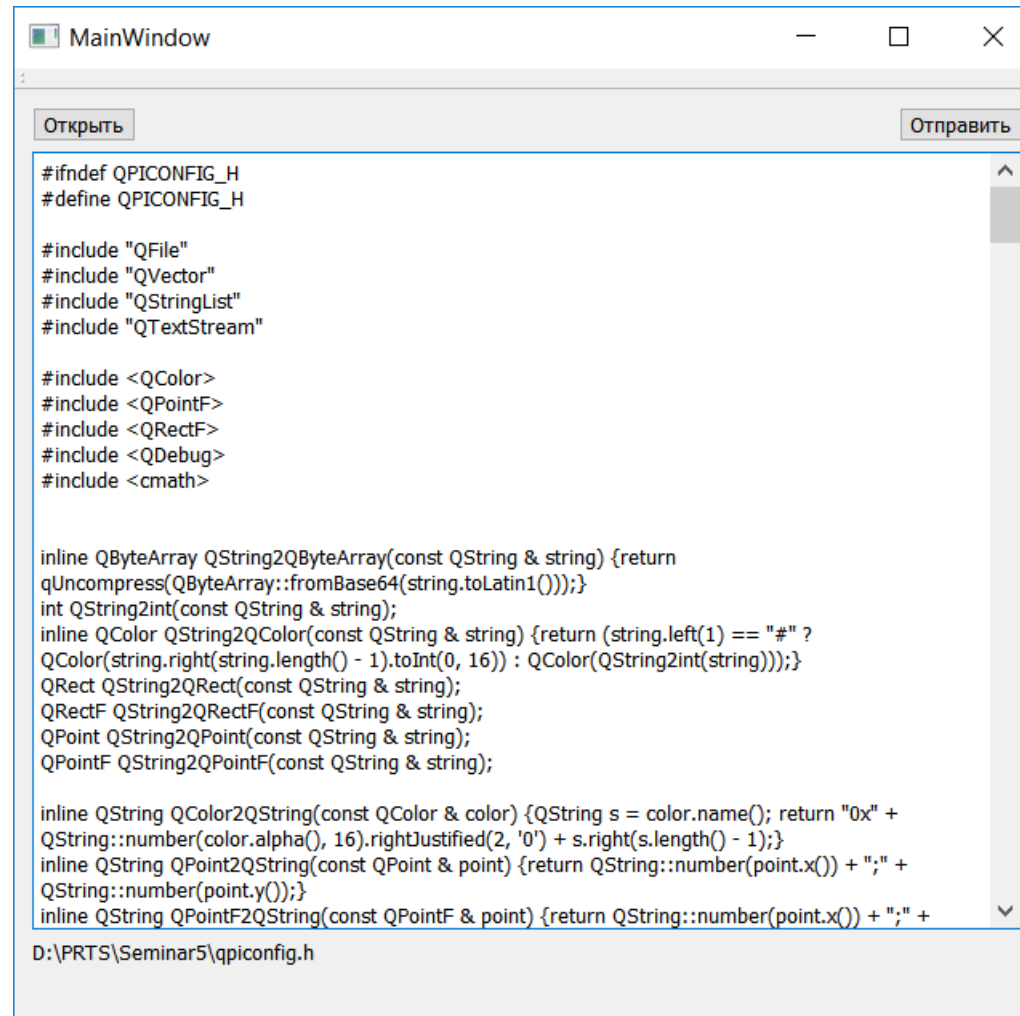
Чтение данных из файла и вывод его содержимого в главное окно можно осуществить с помощью класса QTextStream.

*Данный метод предпочтителен.

```
FileSender_/FileSender/mainwi... * X | ◆ MainWindow::receivePath(const QString &): void | # Line: 54, Col: 2
29
30
31 void MainWindow::receivePath(const QString &path){
32     //создадим объект файла
33     QFile file(path);
34     //проверим существует ли файл, если файл не существует выведем
35     //информацию об ошибке
36     if (!file.exists()) lblPath->setText(tr("The file doesn't exists"));
37     //если файл существует, то
38     else {
39         //откроем файл для чтения (QFile::ReadOnly)
40         file.open(QFile::ReadOnly);
41         //создадим поток ввода/вывода QTextStream, предназначенный
42         //для чтения текстовых данных
43         //при создании объекта передадим ему указатель на файл,
44         //из которого производим чтение
45         QTextStream stream(&file);
46         //выведем содержимое файла и путь к нему
47         txtBrFile->setText(stream.readAll());
48         lblPath->setText(path);
49     }
50     //по хорошему файл надо было бы закрыть, но т.к. в Qt
51     //многие нюансы уже учтены, файл автоматически закрывается с
52     //уничтожением объекта QFile (а он уничтожается при выходе из
53     //этой функции)
54 }
55
```

QFile. Практическая часть 1.

Насладимся промежуточным результатом:



```
#ifndef QPICONFIG_H
#define QPICONFIG_H

#include "QFile"
#include "QVector"
#include "QStringList"
#include "QTextStream"

#include <QColor>
#include <QPointF>
#include <QRectF>
#include <QDebug>
#include <cmath>

inline QByteArray QString2QByteArray(const QString & string) {return
qUncompress(QByteArray::fromBase64(string.toLatin1()));}
int QString2int(const QString & string);
inline QColor QString2QColor(const QString & string) {return (string.left(1) == "#" ?
QColor(string.right(string.length() - 1).toInt(0, 16)) : QColor(QString2int(string)));}
QRect QString2QRect(const QString & string);
QRectF QString2QRectF(const QString & string);
QPoint QString2QPoint(const QString & string);
QPointF QString2QPointF(const QString & string);

inline QString QColor2QString(const QColor & color) {QString s = color.name(); return "0x" +
QString::number(color.alpha(), 16).rightJustified(2, '0') + s.right(s.length() - 1);}
inline QString QPoint2QString(const QPoint & point) {return QString::number(point.x()) + ";" +
QString::number(point.y());}
inline QString QPointF2QString(const QPointF & point) {return QString::number(point.x()) + ";" +
```

D:\PRTS\Seminar5\qpiconfig.h