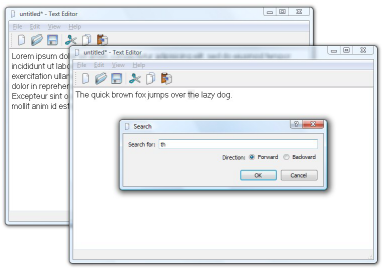
**Лабораторная работа №1. (лекции 1-4)**

Цель: научиться создавать полноценное Qt-приложение. Акцентировать внимание на понимании того как устроено Qt-приложение, научиться пользоваться справочной системой.

Qt Meta-object system, Qt resource system, Qt Signal-Slot mechanism, Qt Creator. QApplication, QMainWindow, QTextEdit, QAction, QEvent, QSettings, QFont, QString, QMessageBox, QFile, QFileDialog, QFileInfo, QTextStream

Продолжительность: 3 часа

Задача: разработать приложение - текстовый редактор с поддержкой работы с несколькими документами, загрузкой-сохранением в текстовый файл, отменой изменений, работой с буфером обмена и т.д.



Требования к приложению:

1. В приложении должно быть меню с командами::
   1. “Файл”
      1. “Создать” (Ctrl+N)
      2. “Открыть” (Ctrl+O)
      3. “Сохранить” (Ctrl+S)
      4. “Сохранить как” (Ctrl+Shift+S)
      5. “Закрыть” (Ctrl+W)
      6. “Выход”
   2. “Редактирование”
      1. “Копировать” (Ctrl+C)
      2. “Вырезать” (Ctrl+X)
      3. “Вставить” (Ctrl+V)
      4. “Отменить” (Ctrl+Z)
      5. “Повторить” (Ctrl+Y)
   3. “Вид”
      1. “Выбрать шрифт”
   4. Помощь
      1. “О программе”
      2. “О Qt”
2. Меню должно быть продублировано кнопками на панели инструментов (кроме “Сохранить как”).
3. Действие команд должны соответствовать общепринятой практике в текстовых редакторах.

**Оценка.**

Максимальная оценка за выполнение работы - 25 баллов. Оценка выставляется по результатам проверки работоспособности программы по пунктам (1 балл за каждый пункт):

1. По команде “Создать” (здесь и далее в меню, панели инструментов и горячими клавишами) открывается отдельное окно с новым документом
2. Все окна имеют иконку. Все кнопки на панели инструментов имеют иконку. Заголовок окна содержит имя файла (или “Новый документ”)
3. Для хранения иконок используется система ресурсов Qt. (Для доступа к иконкам используется файл ресурсов).
4. По команде “Закрыть” закрывает текущее окно (если документ не изменялся).
5. По команде “Выход” закрываются все окна (если документ не изменялся) и завершается работа приложения
6. Индикатор [\*] появляется в заголовке при изменении текста (документ изменен)
7. При закрытии измененного документа командами “Закрыть” ,”Выход” и через оконный менеджер (Х) должно появиться предупреждение “Документ изменен. Сохранить?” с вариантами ответа “Да”, “Нет”, “Отмена”. По умолчанию активная кнопка - “Нет”.
8. Разное поведение приложения при ответе пользователя :"да" - сохранить и закрыть, "нет" - не сохранять и закрыть, "отмена" - не закрывать.
9. Откройте несколько документов. Измените некоторые из них. По команде ”Выход” окна с не измененными документами закрылись, для остальных показывается предупреждение, и процедура закрытия прерывается если ответить “Нет” на предупреждение.
10. Можно выбрать новый шрифт в конкретном окне. Шрифт текущего документа должен измениться, в остальных остаться прежним.
11. Можно отменить выбор шрифта (работает кнопка “Отменить” в диалоге выбора шрифта)
12. Изменить шрифт и открыть новое окно. В нем должен использоваться новый шрифт
13. Выбранный шрифт сохраняется при перезапуске приложения.
14. По команде “О программе” показывается диалог с информацией о программе и ее авторе (ФИО, группа).
15. По команде “О Qt” показывается диалог с версией Qt, используемой при сборке (не разработке) приложения.
16. Команды “Отменить” и “Повторить” в текстовом редакторе работают как должны.
17. Команды “Отменить” и “Повторить” активны когда ожидается, в том числе при запуске приложения.
18. Команда “Вырезать”, “Копировать”, “Вставить” активны и работают как ожидается, в том числе при запуске приложения.
19. Документ можно открыть из файла и его содержимое загружается в редактор.
20. Документ сохраняется в файл из которого был открыт.
21. Документ сохраняется в новый файл
22. При открытии и обоих способах сохранения заголовок окна меняется правильно.
23. При открытии и обоих сохранениях признак измененного документа ведет себя правильно.
24. При попытке закрыть измененный новый документ должен быть диалог выбора имени файла при запросе на сохранение изменений.
25. При попытке закрыть измененный новый документ. Ответьте, что хотите сохранить документ, но выберете “Отменить” в диалоге выбора имени нового файла. Окно не должно закрыться, документ не должен сохраниться в файле..

При выполнении работы на 25 баллов самостоятельно, без использования методического материала, оценка увеличивается на **20%** (т.е. можно получить 30 баллов). Если выполнение работы с помощью методического материала займет более чем неделю, оценка уменьшается на **20%** (т.е. можно получить не более 20 баллов).

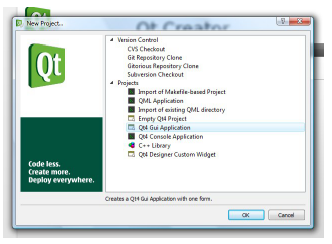
**Методические указания по выполнению лабораторной работы №1.**

Во время работы к приложению постепенно будут добавляться новые функции. Преследуются две цели: первая - понять как разрабатывается реальное приложение. Функциональность приложения наращивается постепенно. Вторая - получить рабочее приложение с самого начала, так чтобы можно было последовательно исследовать, тестировать и отлаживать функциональность приложения. Не откладывайте известную ошибку на потом, постоянно убеждайтесь, что приложение работает. Будет гораздо сложнее исправить ошибку в более сложном приложении.

В ЛР указания будут менее детальные, чем в примере. Это значит, что вам придется самим искать необходимые классы в документации и подключать их.

**1.Каркас приложения**

Будем строить приложение на основе шаблона **GUI приложение Qt .** Создайте приложение и подключите модули *QtCore* и *QtGui*. Главное окно сделайте на основе QMainWindow. Назовите проект *TextEditor*.



Полученный проект содержит файлы:

* TextEditor.pro – файл описания проекта.
* mainwindow.ui – интерфейс для класса MainWindow
* mainwindow.cpp/h – реализация и объявление класса MainWindow.
* main.cpp – основная(main) функция.

Просмотрите файлы и убедитесь, что проект собирается и запускается.

**2. Добавление элементов интерфейса**

Следующим шагом добавим виджет редактирования текста QTextEdit и основные пользовательские операции: создать файл, закрыть файл и выход к главному окну.

Но сначала добавим в приложение ресурс: иконку, они будут использоваться далее.

Добавьте в проект файл ресурсов, назовем его icons.qrc. Откройте файл ресурсов, добавьте префикс icons и добавьте файлы \*.png. из материалов к ЛР (или возьмите другие, какие нравятся).

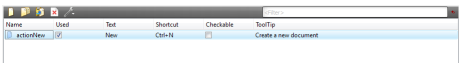
Теперь вы готовы работать над интерфейсом. Откройте mainwindow.ui в режиме дизайнера.

Добавьте виджет QTextEdit в главное окно и скомпонуйте его, чтобы оно занимало окно целиком (табличный компоновщик).



Посмотрите что получилось, запустив предварительный просмотр (Alt+Shift+R).

Пользовательские действия представлены объектом класса QAction. Они создаются и управляются в *редакторе действий* .



Создайте следующие действия:

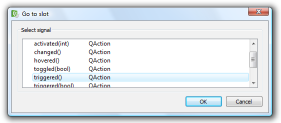
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Text** | **Name** | **Icon** | **Shortcut** | **ToolTip** |
| Новый | actionNew |  | Ctrl+N | Создать новый документ |
| Закрыть | actionClose |  | Ctrl+W | Закрыть текущее окно |
| Выход | actionExit |  |  | Завершить работу |

Добавьте пункт “Файл” в меню приложения. Нажмите на поле “Пишите здесь”и введите “Файл”. Теперь перетащите действия в меню. Добавьте разделитель между каждым пунктом. (если написать &Файл, вы получите горячую клавишу, чтобы & отображался в меню, напишите его дважды &&)

Соберите и запустите приложение. Проверьте его работу.

**3. Добавляем функциональность**

Каждое действие нужно реализовать. Для этого по правой кнопке выберете пункт “Перейти к слоту” и выберете из списка сигнал triggered()



Начните с реализации actionNew - on\_actionNew\_triggered(). В этом слоте создайте новый объект класса MainWindow и покажите его. Для остальных двух действий actionClose и actionExit уже созданы слоты. Соедините их в конструкторе MainWindow после вызова setupUi следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| **Источник сигнала** | **Сигнал** | **Обработчик** | **Слот** |
| actionClose | triggered() | this (текущее окно) | close() |
| actionExit | triggered() | qApp (приложение) | closeAllWindows() |

~~Также не забудьте вызвать setAttribute в конструкторе MainWindow и установить атрибут Qt::WA\_DeleteOnClose.~~

*Контрольные вопросы*

1. Новые окна открываются
2. Файл - Закрыть закрывает текущее окно
3. Файл - Выход закрывает все окна и завершает работу приложения
4. ~~Объясните почему надо установить атрибут Qt::WA\_DeleteOnClose. Что он делает и почему это важно.~~

**4. Редактирование и закрытие**

Сейчас пользователь по ошибке может закрыть окно и потерять не сохраненный текст. Это не то что он ожидает. Он ждет, что будет предупрежден о риске потерять изменения. Существует двухэтапное решение такой ситуации. Первый шаг - выставлять флаг при изменении документа, второй шаг - выдавать предупреждение, если флаг изменения установлен.

Отслеживая сигнал textChanged() объекта QTextEdit можно узнать об изменении документа. Создайте private слот documentModified() в классе MainWindow. В конструкторе соедините их. В слоте установите свойство windowModified в значение true.

Свойство windowModified взаимодействует со свойством windowTitle, поэтому в заголовке вы увидите звездочку. Звездочка добавляется в свойство windowTitle как “[\*]”, после этого Qt будет синхронизировать эти свойства автоматически.

Второй шаг решения проблемы - выдать предупреждение пользователю перед закрытием документа. Это делается переопределением метода closeEvent объекта класса MainWindow. Добавьте метод в объявление класса :

protected:

void closeEvent(QCloseEvent \*e);

и пустое тело в реализацию:

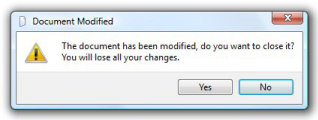
void MainWindow::closeEvent(QCloseEvent \*e)

{

}

Событие может быть принято и обработано или проигнорировано используя его методы e->accept() и e->ignore(). Реализуйте функцию таким образом, чтобы событие было принято для неизмененных докуметов. Если документ изменен, используйте функцию QMessageBox::warning для выдачи предупреждения и соответственно примите или игнорируйте событие.

Предупреждение должно содержать две кнопки “Да” и “Нет”. причем “Нет” - по умолчанию.



Диалог предупреждения возвращает какую кнопку нажал пользователь.

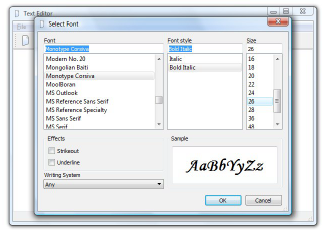
После реализации функция закрытия, пункты меню “Закрыть” и “Выход” будут работать как должны.

*Контрольные вопросы*

* Индикатор [\*] появляется в заголовке при изменении текста
* Откройте документ, измените его, попробуйте его закрыть следующими способами: через меню “Файл - Закрыть” ,”Файл - Выход” и через менеджер окон (Х). Убедитесь что появляется предупреждение.
* Убедитесь, что используя предупреждение вы можете предотвратить закрытие окна выбрав “Нет” или закрыв предупреждение другим способом.
* Откройте документ, не изменяя его, попробуйте закрыть способами из предыдущего пункта. Убедитесь что предупреждения нет.
* Откройте несколько документов. Измените некоторые. Выберете ”Файл - Выход”. Убедитесь, что только не измененные окна закрылись и что процедура закрытия прерывается если вы ответите “Нет” на предупреждение.

**5. Настройки приложения**

Почти все современные приложения в том или ином виде имею возможность пользовательской настройки. Это может что угодно, начиная от выбора единиц измерения и настройки порядка панелей инструментов и заканчивая тонкими специфическими настройками. Добавим к нашему приложению возможность выбора шрифта для отображения текста.



настройки будем хранить в зависимости от платформы, т.е. в случае Windows - в реестре, в Linux - в скрытых файлах и т.д. Используем для этой цели класс QSettings.

***5.1. Подготовка***

Для правильной реализации указанного поведения требуется указать название приложения, его версию, наименование и область деятельности организации. Просто добавьте следующий код в функцию main, где a экземпляр класса QApplication:

a.setApplicationName("Text Editor");

a.setApplicationVersion("0.1");

a.setOrganizationName("MephiSoft");

a.setOrganizationDomain("mephi.ru");

Также установите свойство windowIcon объекта QApplication для использования файла “:/icons/new.png" как иконки приложения.

***5.2. добавление действий***

Добавьте действие к главному окну приложения:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Text** | **Name** | **Icon** | **Shortcut** | **ToolTip** |
| Выбрать шрифт... | actionSelectFont |  |  | Выбор шрифта текста |

добавьте действие к меню “Вид” (его следует создать). Через контекстное меню действия добавьте слот для сигнала triggered(). Вы должны получить пустую функцию слота:

void MainWindow::on\_actionSelectFont\_triggered()

{

}

Используйте QFontDialog::getFont( \*ok, initial, parent ) для получения нового объекта QFont. Используйте ui->textEdit->font() для получения текущего шрифта.

Если пользователь подтвердил выбор нового шрифта, т.е. ok==true, установите новый шрифт использую свойство font объекта ui->textEdit. Используйте объект QSettings для сохранения выбранного шрифта в переменной “viewFont”.

Реализуйте установку шрифта из сохраненного значения “viewFont” (используйте QSettings) для отображения текста в виджете ui->textEdit. Сделайте это в конструкторе MainWindow и используйте QApplication::font() как шрифт по умолчанию при чтении из переменной “viewFont” .

*Контрольные вопросы*

* все окна имеют иконку
* можно выбрать новый шрифт в конкретном окне
* можно отменить выбор шрифта (работает кнопка “Отмена”)
* изменить шрифт и открыть новое окно. В нем должен использоваться новый шрифт
* выбранный шрифт сохраняется при перезапуске приложения

**6. Буфер обмена и история изменений**

Добавить обычные операции работы с буфером обмена: вырезать, копировать и вставить с поддержкой отмены и повтора. В Qt это настолько просто, что больше времени занимает реализация диалога “О программе”.

Большинство виджетов в Qt подготовлено для использования в реальных программах. QTextEdit тут не исключение. Он имеет готовые слоты для копирования, вырезки вставки отмены и повтора действий, так же как и сигналы. Поэтому все что нужно - добавить действия для всех этих операций и соединить их.

***6.1. добавление действий***

Добавьте следующие действия к главному окну приложения:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Text** | **Name** | **Icon** | **Shortcut** | **ToolTip** |
| О программе | actionAbout |  |  |  |
| О Qt | actionAboutQt |  |  |  |
| Вырезать | actionCut |  | Ctrl+X | Вырезать |
| Копировать | actionCopy |  | Ctrl+C | Копировать |
| Вставить | actionPaste |  | Ctrl+V | Вставить |
| Отменить | actionUndo |  | Ctrl+Z | Отмена последнего действия |
| Повторить | actionRedo |  | Ctrl+Y | повтор последнего действия |

Разместите действие actionAbout с меню “Помощь” (нужно создать). Остальные действия - в меню “Редактирование”(нужно создать). Разместите пункты меню в следующем порядке: Файл, Редактирование, Вид, Помощь.

Добавьте действия с буфером обмена также и на панель инструментов(создайте кнопки). Используйте разделители ,чтобы отделить actionNew от действия с буфером.

***6.2 Реализация функциональности***

Все действия, кроме actionAbout, для того чтобы они работали, нужно только соединить со слотами. Это значит, что их функциональность уже реализована в Qt.

В конструкторе MainWindow соедините следующие сигналы и слоты:

|  |  |  |  |
| --- | --- | --- | --- |
| **Источник сигнала** | **Сигнал** | **Обработчик** | **Слот** |
| actionAboutQt | triggered() | qApp | aboutQt() |
| actionCut | triggered() | textEdit | cut() |
| actionCopy | triggered() | textEdit | copy() |
| actionPaste | triggered() | textEdit | paste() |
| actionUndo | triggered() | textEdit | undo() |
| actionRedo | triggered() | textEdit | redo() |
| textEdit | copyAvailable(bool) | actionCopy | setEnabled(bool) |
| textEdit | copyAvailable(bool) | actionCut | setEnabled(bool) |
| textEdit | undoAvailable(bool) | actionUndo | setEnabled(bool) |
| textEdit | redoAvailable(bool) | actionRedo | setEnabled(bool) |

Свойство enabled действий actionCopy, actionCut, actionUndo и actionRedo обновляется при получении соответствующего сигнала от QTextEdit. Но они еще не проинициализированы. В конструкторе добавьте код их инициализации - установите по умолчанию значение enabled равным false.

Осталось только одно действие без обработки - actionAbout. Создайте в дизайнере слот для его сигнала triggered:

void MainWindow::on\_actionAbout\_triggered()

{

}

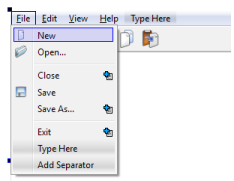
В слоте вызовете QMessageBox::about для показа информации о вашем приложении.

*Контрольные вопросы*

* в меню “Помощь - О программе” показывается диалог с информацией о программе
* в меню “Помощь - О Qt” показывается диалог с версией Qt
* меню “Отменить” и “Повторить” работают как ожидается
* меню “Отменить” и “Повторить” активны когда ожидается, в том числе при запуске приложения
* меню “Вырезать”, “Копировать”, “Вставить” работают как ожидается
* меню “Вырезать”, “Копировать”, “Вставить” активны когда ожидается, в том числе при запуске приложения

**7. Операции с файлами**

Самый большой недостаток текстового редактора - это отсутствие возможности работы с файлами. Нельзя сохранить или загрузить документ. Исправим это.



Сохранение и загрузка файлов оказывает влияние на другие части программы. Каждое окно должно теперь иметь в названии имя файла, а при закрытии (событие closeEvent) должно проверять возможность сохранения изменений в файл.

***7.1 Открыть файл***

Добавьте переменную QString m\_fileName в раздел private класса MainWindow.

Измените определение (или создайте новый) конструктора класса MainWindow. Добавьте в него имя файла:

MainWindow(const QString &fileName=QString(), QWidget \*parent=0);

Как вы понимаете, наша задача написать метод загрузки(открытия) файла. Добавим такой метод loadFile в раздел private класса MainWindow:

void MainWindow::loadFile(const QString &fileName)

{

}

В этом методе реализуйте следующий функционал:

1. если имя файла не задано (fileName пустое) вызовете setFileName(QString()) (метод setFileName пока еще нами не реализован) и верните управление.
2. Создайте объект QFile для файла с именем fileName.
3. Попробуйте открыть объект QFile на чтение текстового файла, т.е. с флагами QIODevice::ReadOnly и QIODevice::Text.
4. Если файл невозможно открыть, покажите сообщение с ошибкой используя QMessageBox::warning, вызовете setFileName(QString()) и верните управление.
5. Если файл открылся, создайте объект QTextStream для работы с ним.
6. Установите свойство text объекта textEdit равным значению, которое возвращает функция readAll объекта QTextStream.
7. Закройте объект QFile .
8. Вызовете setFileName(fileName).
9. Установите свойство windowModified равным false.

Теперь добавьте метод setFileName(const QString &) в раздел private класса MainWindow, в котором присвойте имя файла переменной m\_fileName и измените заголовок окна (свойство windowTitle) на строку вида:

QString("%1[\*] - %2")

.arg(m\_fileName.isNull()?"новый":QFileInfo(m\_fileName).fileName())

.arg(QApplication::applicationName())

Добавьте действие к главному окну MainWindow, вставьте его в меню “Файл” и панель инструментов:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Text** | **Name** | **Icon** | **Shortcut** | **ToolTip** |
| Открыть... | actionOpen |  | Ctrl+O | Открыть документ |

Создайте в дизайнере слот для сигнала triggered :

void MainWindow::on\_actionOpen\_triggered()

{

}

В слоте реализуйте следующий функционал:

1. Используйте следующий код для получения имени файла, который нужно открыть:  
    QString fileName = QFileDialog::getOpenFileName(this,  
    "Открыть документ", QDir::currentPath(), "Text documents (\*.txt)");
2. Если файл не выбран (fileName isNull) верните управление
3. Если текущее значение m\_fileName isNull и документ не изменен, вызовете loadFile(fileName), т.е. загрузите содержимое файла в текущее окно
4. Иначе создайте новое окно (new QMainWindow(fileName) ) и отобразите его, т.е. загрузите файл в новый редактор.

Поэкспериментируйте с приложением. Откройте текстовые файлы, создайте новые документы и откройте фалы в них. Закройте и убедитесь, что предупреждения показываются когда нужно.

***7.2 Сохранить документ.***

Сохранение документа немногим сложнее открытия. С одной стороны пользователь должен иметь возможность “Сохранить...” и “Сохранить как...” документ, с другой стороны важно сообщить пользователю о результате сохранения файла - успешно или нет.

Добавьте объявление слотов в private раздел класса MainWindow и их реализацию в .срр файл:

private slots:

bool saveFile();

bool saveFileAs();

Как видите оба слота возвращают bool. Смысл в том, чтобы показать успешно или нет прошло сохранение файла. Нам это понадобится позже при обработке закрытия измененного документа.

Слот saveFile сохраняет текущий документ в текущий файл, а saveFileAs сначала запрашивает имя нового файла, а потом сохраняет документ в него используя saveFile.

Реализуйте saveFileAs :

1. Используйте следующий код для получения имени нового файла:  
    QString fileName = QFileDialog::getSaveFileName(this, "Сохранить документ",  
    m\_fileName.isNull()?QDir::currentPath():m\_fileName, "Text documents (\*.txt)");
2. если имя файла не задано (fileName isNull) верните false
3. если имя файла задано, вызовете setFileName, а затем saveFile. Верните то что вернет saveFile.

Реализуйте saveFile:

1. если m\_fileName isNull, вызовете saveFileAs и верните, то что вернет она.
2. если m\_fileName не isNull, создайте объект QFile для этого имени
3. Попробуйте открыть файл QFile на запись, т.е.е с флагами QIODevice::WriteOnly и QIODevice::Text.
4. Если файл нельзя открыть, используйте QMessageBox::warning для вывода ошибки, вызовете setFileName(QString()) и верните false
5. Если файл открылся, создайте для него QTextStream
6. запишите содержимое документа textEdit->toPlainText() в QTextStream
7. Закройте файл
8. Установите windowModified равным false
9. верните true

Слоты saveFile и saveFileAs вызывают друг друга. Объясните, чем гарантируется, что они не зациклятся.

Добавьте действия в MainWindow. Создайте для них пункты в меню “Файл”, а для actionSave кнопку на панели инструментов:

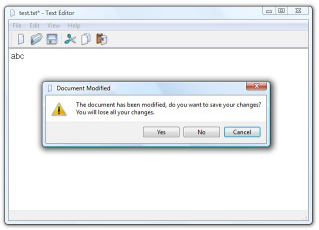
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Text** | **Name** | **Icon** | **Shortcut** | **ToolTip** |
| Сохранить... | actionSave |  | Ctrl+S | Сохранить документ |
| Сохранить как... | actionSaveAs |  | Ctrl+Shift+S | Сохранить документ как |

В конструкторе MainWindow соедините действия со слотами saveFile и saveFileAs.

Соберите и запустите приложение. Поэкспериментируйте с функциями. Документы должны открываться, сохраняться и сохраняться в новый файл. Также проверьте что признак измененного документа работает как ожидается и появляется предупреждения о потере данных когда нужно.

**8. Закрытие окна**

Осталось обработать ответ пользователя на вопрос закрывать измененный документа или нет. Наиболее распространенные варианты: “отменить изменения и закрыть” , “сохранить изменения и закрыть”, “не закрывать”. Чтобы обработать эти варианты ответов нужно добавить функции сохранения файла в обработчик события closeEvent.



Реализуйте обработчик closeEvent:

void MainWindow::closeEvent(QCloseEvent \*e){

if(m\_modified){

switch(QMessageBox::warning(this, "Document Modified",

"The document has been modified. "

"Do you want to save your changes?\n"

"You will lose and unsaved changes.",

QMessageBox::Yes | QMessageBox::No | QMessageBox::Cancel,

QMessageBox::Cancel)){

case QMessageBox::Yes:

// [1]

break;

case QMessageBox::No:

// [2]

break;

case QMessageBox::Cancel:

// [3]

break;

}

}

else{

// [4]

}

}

В строках, помеченных [1], [2], [3] и [4] добавьте код обработки(accept()) или игнорирования(ignore()) события где какой нужно. В одном из случаев нужно вызвать метод saveFile и вызвать accept() или ignore() в зависимости оттого что она вернет.

*Контрольные вопросы*

* документ можно открыть и он загружается в редактор
* документ сохраняется
* при открытии и обоих сохранениях заголовок окна меняется правильно
* при открытии и обоих сохранениях признак измененного документа ведет себя правильно
* попробуйте закрыть измененный новый документ. Должен быть диалог выбора имени файла при запросе на сохранение изменений.
* попробуйте закрыть измененный новый документ. Ответьте, что хотите сохранить документ, но выберете “отменить” в диалоге выбора имени нового файла. Окно не должно закрыться.