

Toynote
20191224

Создано системой Doxygen 1.8.13

Оглавление

1	Toynote	1
2	Часто задаваемые вопросы	3
2.1	Что такое архитектура модель/вид?	3
2.2	Что такое интерфейс класса?	3
2.3	Что такое константный метод?	4
2.4	Какова структура модели в Qt?	4
2.5	Что такое индекс элемента модели в Qt?	4
2.6	Что такое роль данных для элемента модели в Qt?	5
2.7	Что такое Qt Designer?	5
2.8	Как работает Qt Designer?	5
2.9	Что такое виджет?	6
2.10	Что такое ресурсы в приложении Qt?	6
3	Иерархический список классов	7
3.1	Иерархия классов	7
4	Алфавитный указатель классов	9
4.1	Классы	9
5	Список файлов	11
5.1	Файлы	11

6	Классы	13
6.1	Класс <code>EditNoteDialog</code>	13
6.1.1	Подробное описание	14
6.1.2	Конструктор(ы)	14
6.1.2.1	<code>EditNoteDialog()</code>	14
6.1.2.2	<code>~EditNoteDialog()</code>	15
6.1.3	Методы	15
6.1.3.1	<code>accept</code>	15
6.1.4	Данные класса	15
6.1.4.1	<code>mUi</code>	16
6.2	Класс <code>MainWindow</code>	16
6.2.1	Подробное описание	18
6.2.2	Конструктор(ы)	19
6.2.2.1	<code>MainWindow()</code>	19
6.2.2.2	<code>~MainWindow()</code>	19
6.2.3	Методы	19
6.2.3.1	<code>destroyNotebook()</code>	19
6.2.3.2	<code>notebookFileNameChanged</code>	19
6.2.3.3	<code>notebookOpened</code>	20
6.2.3.4	<code>saveNotebookToFile()</code>	20
6.2.4	Данные класса	20
6.2.4.1	<code>mNotebook</code>	20
6.2.4.2	<code>mUi</code>	21
6.3	Класс <code>Note</code>	21
6.3.1	Подробное описание	22
6.3.2	Конструктор(ы)	22
6.3.2.1	<code>Note()</code>	22
6.4	Класс <code>Notebook</code>	22
6.4.1	Подробное описание	24
6.4.2	Определения типов	24
6.4.2.1	<code>SizeType</code>	24
6.4.3	Методы	24
6.4.3.1	<code>columnCount()</code>	24
6.4.3.2	<code>data()</code>	25
6.4.3.3	<code>headerData()</code>	25
6.4.3.4	<code>operator[]()</code>	26
6.4.3.5	<code>rowCount()</code>	27

7	Файлы	29
7.1	Файл <code>config.hpp</code>	29
7.1.1	Подробное описание	30
7.1.2	Переменные	30
7.1.2.1	<code>applicationName</code>	30
7.1.2.2	<code>notebookFileNameFilter</code>	30
7.2	Файл <code>editnotedialog.cpp</code>	31
7.2.1	Подробное описание	31
7.3	Файл <code>editnotedialog.hpp</code>	31
7.3.1	Подробное описание	32
7.4	Файл <code>main.cpp</code>	33
7.4.1	Подробное описание	33
7.4.2	Функции	33
7.4.2.1	<code>main()</code>	33
7.5	Файл <code>mainwindow.cpp</code>	34
7.5.1	Подробное описание	34
7.6	Файл <code>mainwindow.hpp</code>	35
7.6.1	Подробное описание	35
7.7	Файл <code>note.cpp</code>	36
7.7.1	Подробное описание	36
7.8	Файл <code>note.hpp</code>	37
7.8.1	Подробное описание	38
7.8.2	Функции	38
7.8.2.1	<code>operator<<()</code>	38
7.8.2.2	<code>operator>>()</code>	38
7.9	Файл <code>notebook.cpp</code>	39
7.9.1	Подробное описание	39
7.10	Файл <code>notebook.hpp</code>	40
7.10.1	Подробное описание	41
7.10.2	Функции	41
7.10.2.1	<code>operator<<()</code>	41
7.10.2.2	<code>operator>>()</code>	41
	Алфавитный указатель	43

Глава 1

Toynote

Toynote — простое графическое приложение для заметок, написанное с использованием библиотеки Qt. Является кроссплатформенным на уровне исходного кода.

Рекомендуется просмотреть [Часто задаваемые вопросы](#).

Автор: Кирилл Владимирович Пушкарёв <kpushkarev@sfu-kras.ru>, 2019 г.

Глава 2

Часто задаваемые вопросы

2.1 Что такое архитектура модель/вид?

Архитектура модель/вид (model/view architecture) — это частный случай архитектуры модель/вид/контроллер (model/view/controller). Модель — это часть системы, которая отвечает за хранение и обработку данных. Вид отвечает за отображение данных. Контроллер отвечает за управление через пользовательский интерфейс. В Qt функции вида и контроллера объединены.

См. также [подробное описание](#) в документации Qt.

2.2 Что такое интерфейс класса?

Интерфейс в узком смысле представляет собой совокупность методов (с определёнными параметрами и типом возвращаемого значения). В C++ интерфейс оформляется в виде класса, содержащего виртуальные методы, которые производные классы могут определить (назначить тело функции) или переопределить (заменить тело функции).

Один класс может реализовать несколько интерфейсов. Например, класс Car (легковой автомобиль) может реализовать интерфейсы Vehicle (транспортное средство) и Asset (ценность). Интерфейс Vehicle может содержать методы для определения типа транспортного средства, типа двигателя, грузоподъёмности и т. д. Интерфейс Asset может содержать методы для определения типа ценности (материальная ценность, интеллектуальная собственность и т. д.), стоимости и т. д.

Чтобы можно было создать объект класса Car, все методы, в том числе унаследованные от интерфейсов, должны быть определены. Класс Car обязан определить все методы, не имеющие определения, а также может переопределить методы, уже имеющие определения в базовых (родительских) классах.

Польза интерфейсов в том, что алгоритмы могут работать не с конкретными классами, а с любыми классами, реализующими нужный им интерфейс. Например, у организации может быть много автомобилей (объектов класса Car). Алгоритм управления рассматривает автомобиль как объект Car, так как ему нужны методы рулевого управления и переключения передач. Алгоритм логистики тот же автомобиль рассматривает как объект Vehicle, так как его интересуют грузоподъёмность, скорость, расход топлива. С точки зрения бухгалтерского алгоритма, тот же автомобиль является объектом Asset, так как для него важны стоимость и износ.

В результате алгоритмы рассматривают объекты, с которыми работают, через призму интерфейсов, содержащих важные для них методы. Наличие или отсутствие каких-либо ещё методов не влияет на работу алгоритма. Благодаря этому, алгоритм может обрабатывать любой объект, имеющий нужный интерфейс, независимо от специфики объекта. Нет нужды писать отдельные бухгалтерские алгоритмы для работы с автомобилями, самолётами или мебелью, достаточно написать один алгоритм, работающий с объектами, реализующими интерфейс Asset.

2.3 Что такое константный метод?

Константным в C++ называется метод класса, который не имеет права изменять объект класса, для которого метод вызывается. Подразделение методов на константные и неконстантные нужно, чтобы компилятор мог предотвратить изменение константного объекта.

Константные методы объявляются с квалификатором `const` после списка параметров, например:

```
class A {
    int m_x;
public:
    A(): m_x(0) {} // Конструктор
    int get() const { return m_x; } // Константный метод
    void add(int x) { m_x += x; } // Неконстантный метод
};
```

Такие методы можно вызывать даже у объекта, который является константным:

```
const A obj;
int x;
x = obj.get(); // Нормально: вызов константного метода
obj.add(5); // Ошибка: вызов неконстантного метода у константного объекта!
```

2.4 Какова структура модели в Qt?

Модели, которые Qt использует в [архитектура model/view](#), являются древовидной структурой таблиц. Потомки каждого узла дерева организованы в виде таблицы. Каждый потомок, в свою очередь, тоже имеет таблицу потомков.

Корень этой структуры имеет индекс, который создаётся конструктором класса `QModelIndex` по умолчанию. Для такого индекса метод `isValid()` возвращает `false`. Корню модели не сопоставлены никакие данные, он лишь служит точкой отсчёта.

Чаще всего встречающиеся на практике список, таблица, дерево с линейным списком потомков являются частными случаями общей структуры иерархических таблиц.

В Toynote используется табличная модель, реализуемая классом [Notebook](#). Табличная модель является одноуровневой — состоит из корня и таблицы его потомков. Каждый потомок отвечает за одну ячейку таблицы и не может иметь собственных потомков.

См. [иллюстрации структуры моделей](#) в документации Qt.

2.5 Что такое индекс элемента модели в Qt?

Индекс — это объект, позволяющий идентифицировать определённый элемент, доступный в модели. Индексы являются объектами класса `QModelIndex`.

Индекс элемента состоит из трёх компонентов: индекса его родителя, а также номера строки и столбца в таблице потомков родителя. В классе `QModelIndex` предусмотрены методы, позволяющие узнать эти данные.

Индексы обычно передаются методам модели, чтобы указать на элемент, о котором идёт речь.

2.6 Что такое роль данных для элемента модели в Qt?

С одним элементом модели в Qt может быть связано несколько объектов данных, каждый из которых играет определённую роль. Стандартные роли, используемые Qt, собраны в перечисляемом типе `Qt::ItemDataRole`.

Из стандартных ролей абсолютно необходимой является `Qt::DisplayRole`. Данные с этой ролью — это текст, который отображается привязанным к модели видом для данного её элемента.

Остальные роли позволяют влиять на отображение или функционирование элемента модели. Например, `Qt::DecorationRole` — это цвет или иконка, отображаемые рядом с элементом, `Qt::ToolTipRole` — это текст для всплывающей подсказки.

2.7 Что такое Qt Designer?

Qt Designer — это визуальный конструктор [виджетов](#), входящий в IDE Qt Creator. Он автоматизирует создание программного кода, собирающего нужный графический интерфейс из классов библиотеки Qt.

2.8 Как работает Qt Designer?

Qt Designer позволяет пользователю сформировать вид интерфейса, параметры его элементов и связи между ними в режиме WYSIWYG. Результат сохраняется в файле с расширением `.ui` (UI-файле).

UI-файл написан на языке, который является разновидностью XML. В нём записана вся информация, необходимая для воссоздания разработанного пользователем графического интерфейса.

При сборке проекта из UI-файла автоматически генерируется код на языке C++, содержащий класс (UI-класс), состоящий из виджетов и кода для их настройки.

Далее обычно объект, который будет основой для вывода графического интерфейса (это может быть объект класса-потомка `QDialog` или `QMainWindow`), создаёт объект UI-класса и вызывает код настройки, передавая ему указатель на себя. Код настройки создаёт нужный интерфейс на объекте-основе.

См. [подробнее](#) в документации Qt.

Как добавить новое окно в Qt Designer?

Необходимо добавить в проект в Qt Creator элемент «Qt Designer Form Class» (File | New File or Project... | Files and Classes | Qt | Qt Designer Form Class). При этом добавится UI-файл, а также заголовочный файл и файл реализации для класса, который будет обеспечивать функционирование вашего окна.

2.9 Что такое виджет?

Виджет — это объект графического интерфейса. Окна, кнопки, поля для ввода и т. д. являются виджетами.

В Qt виджеты представляются классами-потомками `QWidget`. Таких классов много: `QDialog` (диалоговое окно), `MainWindow` (главное окно), `QPushButton` (кнопка), `QLineEdit` (поле для ввода строки текста) и т. д.

Виджеты можно добавлять в графический интерфейс приложения вручную, создавая объекты соответствующих классов, либо пользоваться конструктором [Qt Designer](#), который позволяет создать необходимый программный код автоматически.

2.10 Что такое ресурсы в приложении Qt?

Ресурсы в приложении Qt — это двоичные файлы, которые приложение использует в своей работе, упакованные в специальный файл или интегрированные в исполняемый файл приложения.

Обычно в виде ресурсов хранятся иконки приложений, но система ресурсов Qt позволяет хранить произвольные двоичные данные.

Ресурсы приложения доступны по виртуальному файловому пути, начинающемуся с двоеточия. Например, строка «`:/icons/open.png`» указывает на ресурс «`/icons/open.png`». Такие пути можно использовать вместо реальных файловых путей при создании объектов `QIcon` или `QFile` и др. Также ресурсы можно назначать в качестве иконок в Qt Designer.

См. [подробнее](#) в документации Qt.

Глава 3

Иерархический список классов

3.1 Иерархия классов

Иерархия классов.

Note	21
QAbstractTableModel	
Notebook	22
QDialog	
EditNoteDialog	13
QMainWindow	
MainWindow	16

Глава 4

Алфавитный указатель классов

4.1 Классы

Классы с их кратким описанием.

EditNoteDialog	Класс диалога (диалогового окна) редактирования заметки	13
MainWindow	Класс главного окна программы	16
Note	Класс заметки	21
Notebook	Класс записной книжки	22

Глава 5

Список файлов

5.1 Файлы

Полный список документированных файлов.

config.hpp	
Файл конфигурации	29
editnotedialog.cpp	
Файл реализации класса EditNoteDialog	31
editnotedialog.hpp	
Заголовочный файл класса EditNoteDialog	31
main.cpp	
Файл главной функции	33
mainwindow.cpp	
Файл реализации класса MainWindow	34
mainwindow.hpp	
Заголовочный файл класса MainWindow	35
note.cpp	
Файл реализации класса Note	36
note.hpp	
Заголовочный файл класса Note	37
notebook.cpp	
Файл реализации класса Notebook	39
notebook.hpp	
Заголовочный файл класса Notebook	40

Глава 6

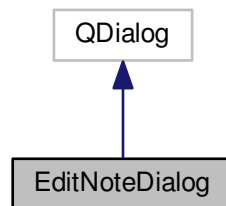
Классы

6.1 Класс EditNoteDialog

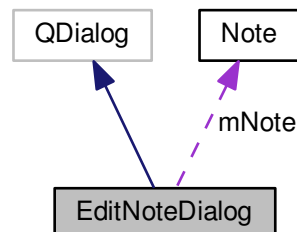
Класс диалога (диалогового окна) редактирования заметки.

```
#include <editnotedialog.hpp>
```

Граф наследования:EditNoteDialog:



Граф связей класса EditNoteDialog:



Открытые слоты

- `void accept () Q_DECL_OVERRIDE`
Обрабатывает подтверждение диалога.

Открытые члены

- `EditNoteDialog (QWidget *parent=0)`
Конструктор с необязательным указанием родительского объекта.
- `~EditNoteDialog ()`
Деструктор
- `Note * note () const`
Возвращает указатель на редактируемую заметку.
- `void setNote (Note *note)`
Устанавливает указатель на редактируемую заметку.

Закрытые данные

- `Ui::EditNoteDialog * mUi`
Указатель на сгенерированный интерфейс.
- `Note * mNote`
Указатель на редактируемую заметку

6.1.1 Подробное описание

Класс диалога (диалогового окна) редактирования заметки.

Является потомком класса `QDialog`, стандартного диалогового окна Qt. Интерфейс окна создаётся в редакторе Qt Designer и сохраняется в файле `editnotedialog.ui`. При сборке проекта, из UI-файла автоматически генерируется код на C++ — класс `Ui::EditNoteDialog`, который отвечает приведение нашего диалога в нужный вид.

Редактируемая заметка передаётся по указателю через метод `setNote()`. Прочитать текущий указатель можно через метод `note()`.

См. также

[Как работает Qt Designer?](#)

6.1.2 Конструктор(ы)

6.1.2.1 EditNoteDialog()

```
EditNoteDialog::EditNoteDialog (
    QWidget * parent = 0 ) [explicit]
```

Конструктор с необязательным указанием родительского объекта.

Аргументы

parent	Указатель на родительский объект.
--------	-----------------------------------

Конструирует объект класса с родительским объектом `parent`. Параметр `parent` имеет значение по умолчанию 0. Указывать родительский объект нужно, например чтобы дочерний объект был автоматически удалён при удалении родительского. Для [EditNoteDialog](#) родителем будет окно более высокого уровня, например главное.

6.1.2.2 ~EditNoteDialog()

```
EditNoteDialog::~EditNoteDialog ( )
```

Деструктор

Отвечает за уничтожение объектов [EditNoteDialog](#). Сюда можно поместить функции, которые надо выполнить перед уничтожением (например, закрыть какие-либо файлы или освободить память).

6.1.3 Методы

6.1.3.1 accept

```
void EditNoteDialog::accept ( ) [slot]
```

Обрабатывает подтверждение диалога.

Этот метод вызывается, когда пользователь подтверждает диалог, например нажатием кнопки «↔ ОК». Метод изначально определён в базовом классе `QDialog`, а здесь он переопределяется, то есть при вызове метода [accept\(\)](#) у объекта [EditNoteDialog](#) будет выполняться этот код, а не тот, что есть в классе `QDialog`.

Метод отвечает за обработку введенных пользователем в диалоге данных. Он считывает данные из полей диалога и записывает их в соответствующие атрибуты редактируемой заметки.

6.1.4 Данные класса

6.1.4.1 mUi

```
Ui::EditNoteDialog* EditNoteDialog::mUi [private]
```

Указатель на сгенерированный интерфейс.

Указатель на объект UI-класса, сгенерированного на основе UI-файла `mainwindow.ui`.

Через этот указатель можно обратиться к элементам главного окна, созданного в Qt Designer.

См. также

[Что такое Qt Designer?](#)

Заметки

`std::unique_ptr` обеспечивает автоматическое удаление объекта по указателю, когда уничтожается объект `EditNoteDialog`

Объявления и описания членов классов находятся в файлах:

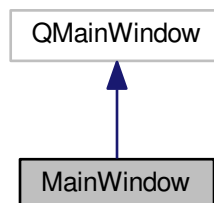
- [editnotedialog.hpp](#)
- [editnotedialog.cpp](#)

6.2 Класс MainWindow

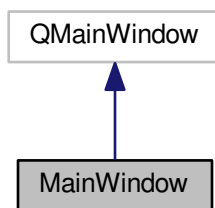
Класс главного окна программы.

```
#include <mainwindow.hpp>
```

Граф наследования:MainWindow:



Граф связей класса MainWindow:



Сигналы

- void [notebookFileNameChanged](#) (QString name)
Сигнал об изменении имени открытого файла.
- void [notebookReady](#) ()
Сигнализирует, что записная книжка готова к работе с пользователем (после создания или открытия).
- void [notebookCreated](#) ()
Сигнализирует, что записная книжка успешно создана.
- void [notebookOpened](#) (QString fileName)
Сигнализирует, что записная книжка успешно открыта.
- void [notebookSaved](#) ()
Сигнализирует, что записная книжка успешно сохранена.
- void [notebookClosed](#) ()
Сигнализирует, что записная книжка успешно закрыта.

Открытые члены

- [MainWindow](#) (QWidget *parent=0)
Конструктор с необязательным указанием родительского объекта `parent`.
- [~MainWindow](#) ()
Деструктор [MainWindow](#).

Закрытые слоты

- void [displayAbout](#) ()
Отображает окно с информацией о программе.
- void [newNotebook](#) ()
Создаёт новую записную книжку.
- bool [saveNotebook](#) ()
Сохраняет текущую записную книжку. Возвращает true в случае успеха.
- bool [saveNotebookAs](#) ()
Сохраняет текущую записную книжку в указанном пользователем файле. Возвращает true в случае успеха.

- `bool openNotebook ()`
Открывает записную книжку вместо текущей. Возвращает `true` в случае успеха.
- `bool closeNotebook ()`
Закрывает текущую записную книжку. Возвращает `true` в случае успеха.
- `bool newNote ()`
Создаёт новую заметку в текущей записной книжке. Возвращает `true` в случае успеха.
- `void deleteNotes ()`
Удаляет выбранные заметки из текущей записной книжки.
- `void refreshWindowTitle ()`
Обновляет заголовок окна.

Закрытые члены

- `void saveNotebookToFile (QString fileName)`
Сохраняет текущую записную книжку в файл.
- `bool isNotebookOpen () const`
Возвращает `true`, если в настоящий момент имеется открытая записная книжка.
- `void setNotebookFileName (QString name=QString())`
Устанавливает имя файла текущей записной книжки равным `name`.
- `QString notebookName () const`
Возвращает имя файла текущей записной книжки.
- `void createNotebook ()`
Создаёт новую записную книжку.
- `void setNotebook (Notebook *notebook)`
Устанавливает указатель на текущую записную книжку равным `notebook`.
- `void destroyNotebook ()`
Уничтожает объект текущей записной книжки.

Закрытые данные

- `Ui::MainWindow * mUi`
Указатель на сгенерированный интерфейс.
- `std::unique_ptr< Notebook > mNotebook`
Указатель на текущую записную книжку.
- `QString mNotebookFileName`
Имя файла текущей записной книжки.

6.2.1 Подробное описание

Класс главного окна программы.

Отвечает за реализацию функций, специфичных для данной конкретной программы. За реализацию функций, общих для всех программ с графическим интерфейсом, отвечает класс `QMainWindow` из библиотеки Qt.

`MainWindow` является производным классом от `QMainWindow` и может добавлять свои атрибуты и методы к тем, которые уже есть в `QMainWindow`.

6.2.2 Конструктор(ы)

6.2.2.1 MainWindow()

```
MainWindow::MainWindow (
    QWidget * parent = 0 ) [explicit]
```

Конструктор с необязательным указанием родительского объекта parent.

Конструирует объект класса с родительским объектом parent. Параметр parent имеет значение по умолчанию 0. Указывать родительский объект нужно, например чтобы дочерний объект был автоматически удалён при удалении родительского. В случае главного окна родителя можно не указывать.

6.2.2.2 ~MainWindow()

```
MainWindow::~MainWindow ( )
```

Деструктор [MainWindow](#).

Отвечает за уничтожение объектов [MainWindow](#). Сюда можно поместить функции, которые надо выполнить перед уничтожением (например, закрыть какие-либо файлы или освободить память).

6.2.3 Методы

6.2.3.1 destroyNotebook()

```
void MainWindow::destroyNotebook ( ) [private]
```

Уничтожает объект текущей записной книжки.

Прекращает отображение текущей записной книжки в графическом интерфейсе и удаляет объект [Notebook](#) по указателю mNotebook.

6.2.3.2 notebookFileNameChanged

```
void MainWindow::notebookFileNameChanged (
    QString name ) [signal]
```

Сигнал об изменении имени открытого файла.

Аргументы

name	Новое имя файла записной книжки.
------	----------------------------------

Сигнализирует, что имени открытого файла записной книжки присвоено значение name.

6.2.3.3 notebookOpened

```
void MainWindow::notebookOpened (  
    QString fileName ) [signal]
```

Сигнализирует, что записная книжка успешно открыта.

Аргументы

fileName	Имя файла открытой записной книжки.
----------	-------------------------------------

6.2.3.4 saveNotebookToFile()

```
void MainWindow::saveNotebookToFile (  
    QString fileName ) [private]
```

Сохраняет текущую записную книжку в файл.

Аргументы

fileName	Имя файла.
----------	------------

Сохраняет текущую записную книжку в файл fileName. Данный метод отвечает непосредственно за сохранение и не предусматривает диалога с пользователем.

6.2.4 Данные класса

6.2.4.1 mNotebook

```
std::unique_ptr<Notebook> MainWindow::mNotebook [private]
```

Указатель на текущую записную книжку.

Заметки

std::unique_ptr, в целом, ведёт себя как обычный указатель, но автоматически удаляет указываемый объект при присваивании другого значения или уничтожении самого unique_ptr.

6.2.4.2 mUi

```
Ui::MainWindow* MainWindow::mUi [private]
```

Указатель на сгенерированный интерфейс.

Указатель на объект UI-класса, сгенерированного на основе UI-файла `mainwindow.ui`.

Через этот указатель можно обратиться к элементам главного окна, созданного в Qt Designer.

См. также

[Что такое Qt Designer?](#)

Объявления и описания членов классов находятся в файлах:

- [mainwindow.hpp](#)
- [mainwindow.cpp](#)

6.3 Класс Note

Класс заметки.

```
#include <note.hpp>
```

Открытые члены

- [Note \(\)](#)
Конструктор по умолчанию
- [Note \(QString title, QString text\)](#)
Конструктор, устанавливающий заголовок и текст.
- [const QString & title \(\) const](#)
Возвращает заголовок заметки.
- [void setTitle \(const QString &title\)](#)
Устанавливает заголовок заметки равным title.
- [const QString & text \(\) const](#)
Возвращает текст заметки.
- [void setText \(const QString &text\)](#)
Устанавливает заголовок заметки равным text.
- [void save \(QDataStream &ost\) const](#)
Сохраняет заметку в поток ost.
- [void load \(QDataStream &ist\)](#)
Загружает заметку из потока ist.

Закрытые данные

- [QString mTitle](#)
Заголовок заметки.
- [QString mText](#)
Текст заметки.

6.3.1 Подробное описание

Класс заметки.

6.3.2 Конструктор(ы)

6.3.2.1 Note()

```
Note::Note (
    QString title,
    QString text )
```

Конструктор, устанавливающий заголовок и текст.

Аргументы

title	Заголовок заметки.
text	Текст заметки.

Создаёт объект [Note](#) с заголовком title и текстом text.

Объявления и описания членов классов находятся в файлах:

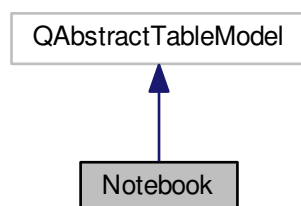
- [note.hpp](#)
- [note.cpp](#)

6.4 Класс Notebook

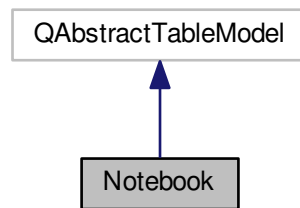
Класс записной книжки.

```
#include <notebook.hpp>
```

Граф наследования: Notebook:



Граф связей класса Notebook:



Открытые типы

- using `SizeType` = int
Тип индексов и размера контейнера.

Открытые члены

- `Notebook` ()
Конструктор по умолчанию.
- const `Note` & `operator[]` (`SizeType` idx) const
Оператор [].
- `SizeType` `size` () const
Определяет размер коллекции (количество заметок).
- void `save` (QDataStream &ost) const
Сохраняет записную книжку в поток ost.
- `SizeType` `load` (QDataStream &ist)
Очищает записную книжку и загружает новую из потока ist. Возвращает количество загруженных заметок.
- void `insert` (const `Note` ¬e)
Вставляет заметку note в записную книжку.
- void `erase` (`SizeType` idx)
Удаляет заметку с индексом idx из записной книжки.

Реализация интерфейса модели.

Ниже идут методы, являющиеся определениями методов интерфейса модели. Подробное описание их предназначения и требований к ним можно найти в документации классов Qt `QAbstractTableModel`, `QAbstractItemModel`.

- int `rowCount` (const `QModelIndex` &parent=`QModelIndex`()) const `Q_DECL_OVERRIDE`
Определяет количество строк в модели (заметок).
- int `columnCount` (const `QModelIndex` &parent=`QModelIndex`()) const `Q_DECL_OVERRIDE`
Определяет количество столбцов в модели.
- QVariant `data` (const `QModelIndex` &index, int role=`Qt::DisplayRole`) const `Q_DECL_OVERRIDE`
Возвращает данные указанной роли для указанного элемента модели.
- QVariant `headerData` (int section, `Qt::Orientation` orientation, int role=`Qt::DisplayRole`) const `Q_DECL_OVERRIDE`
Возвращает данные указанной роли для указанного заголовка модели.

Закрытые данные

- `std::vector< Note > mNotes`

Внутренний контейнер для хранения заметок записной книжки.

6.4.1 Подробное описание

Класс записной книжки.

Контейнер заметок (типа [Note](#)), реализующий интерфейс `QAbstractTableModel`, а через него `QAbstractItemModel`. Реализация интерфейса модели позволяет использовать объекты [Notebook](#) в качестве моделей при работе с видами Qt, в частности с видом `QTableView`, который отображает данные модели в виде таблицы.

На класс модели возлагаются обязательства уведомлять присоединённые виды о том, что данные или их структура изменились. Делается это с помощью вызова специальных методов, таких как `QAbstractItemModel::beginInsertRows()`, или отправки сигналов, таких как `QAbstractItemModel::dataChanged()`. Если поменять содержимое модели, никого не уведомив, то виды будут отображать его неправильно.

См. также

[Что такое интерфейс класса? Что такое архитектура модель/вид? Какова структура модели в Qt?](#)

6.4.2 Определения типов

6.4.2.1 SizeType

```
using Notebook::SizeType = int
```

Тип индексов и размера контейнера.

Используется `int`, так как это тип индексов строк в `QAbstractTableModel`. В то же время, в стандартных контейнерах C++ (`std::vector` и др.) для этого применяются беззнаковые типы.

6.4.3 Методы

6.4.3.1 columnCount()

```
int Notebook::columnCount (
    const QModelIndex & parent = QModelIndex\(\) ) const
```

Определяет количество столбцов в модели.

Аргументы

parent	Ссылка на индекс родительского объекта.
--------	---

Для каждой заметки отображается только её заголовок, поэтому метод возвращает 1 для корневого элемента. Для всех остальных элементов возвращает 0 (см. `QAbstractItemModel::columnCount()`).

См. также

[Какова структура модели в Qt?](#)

6.4.3.2 data()

```
QVariant Notebook::data (
    const QModelIndex & index,
    int role = Qt::DisplayRole ) const
```

Возвращает данные указанной роли для указанного элемента модели.

Аргументы

index	Индекс элемента модели.
role	Роль, для которой надо вернуть данные.

Возвращает для элемента модели с индексом `index` данные с ролью `role`.

Заметки

`QVariant` — это универсальный тип данных Qt, позволяющий хранить данные различных типов.

См. также

[Что такое индекс элемента модели в Qt? Что такое роль данных для элемента модели в Qt?](#)

6.4.3.3 headerData()

```
QVariant Notebook::headerData (
    int section,
    Qt::Orientation orientation,
    int role = Qt::DisplayRole ) const
```

Возвращает данные указанной роли для указанного заголовка модели.

Аргументы

section	Раздел заголовка (номер строки для вертикального заголовка или номер столбца для горизонтального).
orientation	Ориентация заголовка (вертикальный — заголовок строк, горизонтальный — заголовок столбцов).
role	Роль, для которой надо вернуть данные.

Возвращает для элемента section заголовка модели с ориентацией orientation данные с ролью role.

Горизонтальная ориентация означает заголовки столбцов, вертикальная — строк.

См. также

[Что такое индекс элемента модели в Qt? Что такое роль данных для элемента модели в Qt?](#)

6.4.3.4 operator[]()

```
const Note & Notebook::operator[] (
    Notebook::SizeType idx ) const
```

Оператор [].

Аргументы

idx	Индекс читаемого элемента.
-----	----------------------------

Возвращает

Константная ссылка на заметку.

Возвращает ссылку на заметку с индексом idx. Слово const после списка параметров означает, что это константная версия метода, она не может изменять данные класса. Результат также имеет квалификатор const, т. е. по этой ссылке заметку нельзя изменить.

Таким образом, данный метод позволяет прочесть заметку с индексом idx из коллекции, но не изменить её.

Можно было бы определить метод, позволяющий изменить заметку. Его объявление выглядело бы так:

```
Note &operator[] (SizeType idx);
```

В данном случае делать этого нельзя, поскольку возвращённая ссылка на заметку неподконтрольна данному классу и он не имеет возможности узнать, была ли заметка реально изменена и когда это произошло, а значит не может уведомить присоединённые виды о том, что данные изменились.

См. также

[Что такое константный метод?](#)

6.4.3.5 rowCount()

```
int Notebook::rowCount (
    const QModelIndex & parent = QModelIndex() ) const
```

Определяет количество строк в модели (заметок).

Аргументы

parent	Ссылка на индекс родительского объекта.
--------	---

Заметки

`Q_DECL_OVERRIDE` — это макроконстанта (`#define`), которая заменяется спецификатором `override`, если компилятор его поддерживает. Этот спецификатор означает, что данный метод является переопределением соответствующего метода базового класса.

Данная модель является табличной, каждая заметка занимает одну строку, поэтому метод возвращает количество заметок для корневого элемента. Для всех остальных элементов возвращает 0 (см. `QAbstractItemModel::rowCount()`).

См. также

[Какова структура модели в Qt?](#)

Объявления и описания членов классов находятся в файлах:

- [notebook.hpp](#)
- [notebook.cpp](#)

Глава 7

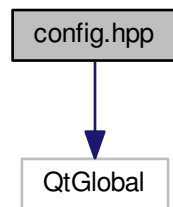
Файлы

7.1 Файл config.hpp

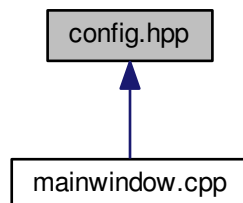
Файл конфигурации.

```
#include <QtGlobal>
```

Граф включаемых заголовочных файлов для config.hpp:



Граф файлов, в которые включается этот файл:



Переменные

- `const char Config::applicationName [] = QT_TRANSLATE_NOOP("Config", "Toynote")`
Название приложения.
- `const char Config::applicationVersion [] = "20191224"`
Версия приложения
- `const char Config::notebookFileNameFilter [] = QT_TRANSLATE_NOOP("Config", "Notebooks (*.tnb)")`
Фильтр для имён файлов записных книжек.

7.1.1 Подробное описание

Файл конфигурации.

Автор

Кирилл Пушкарёв

Дата

2017

7.1.2 Переменные

7.1.2.1 applicationName

```
const char Config::applicationName[] = QT_TRANSLATE_NOOP("Config", "Toynote")
```

Название приложения.

Заметки

`QT_TRANSLATE_NOOP()` используется, чтобы строки отображались в файле строк для перевода на другие языки в контексте `Config`.

7.1.2.2 notebookFileNameFilter

```
const char Config::notebookFileNameFilter[] = QT_TRANSLATE_NOOP("Config", "Notebooks (*.tnb)")
```

Фильтр для имён файлов записных книжек.

Фильтры используются диалогом `QFileDialog`, в котором пользователь выбирает файл при сохранении или открытии записной книжки. Обычно они отображаются в выпадающем списке внизу окна. При выборе фильтра в окне показываются только файлы, имена которых соответствуют одной из масок фильтра.

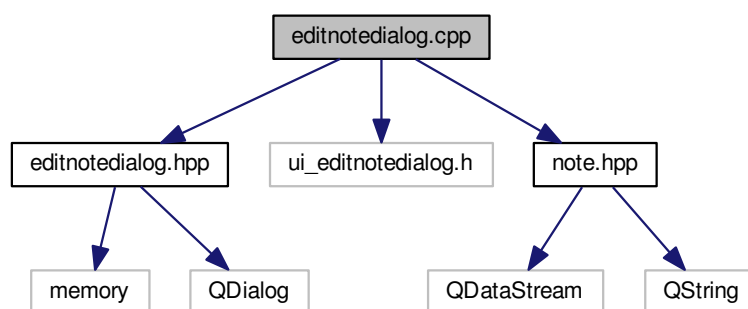
Маски фильтра указываются в скобках через пробел. В одной строке можно указать несколько фильтров через парную точку с запятой (;). Подробнее см. в документации `QFileDialog`.

7.2 Файл editnotedialog.cpp

Файл реализации класса [EditNoteDialog](#).

```
#include "editnotedialog.hpp"  
#include "ui_editnotedialog.h"  
#include "note.hpp"
```

Граф включаемых заголовочных файлов для editnotedialog.cpp:



7.2.1 Подробное описание

Файл реализации класса [EditNoteDialog](#).

Автор

Кирилл Пушкарёв

Дата

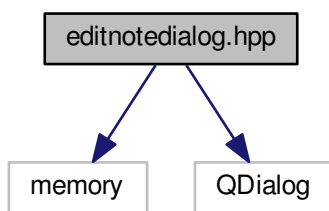
2017

7.3 Файл editnotedialog.hpp

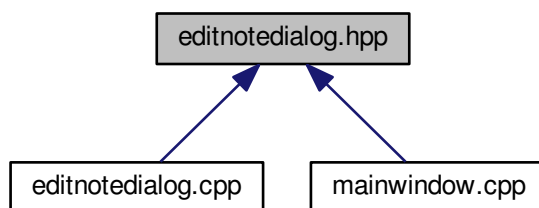
Заголовочный файл класса [EditNoteDialog](#).

```
#include <memory>  
#include <QDialog>
```

Граф включаемых заголовочных файлов для editnotedialog.hpp:



Граф файлов, в которые включается этот файл:



Классы

- class [EditNoteDialog](#)

Класс диалога (диалогового окна) редактирования заметки.

7.3.1 Подробное описание

Заголовочный файл класса [EditNoteDialog](#).

Автор

Кирилл Пушкарёв

Дата

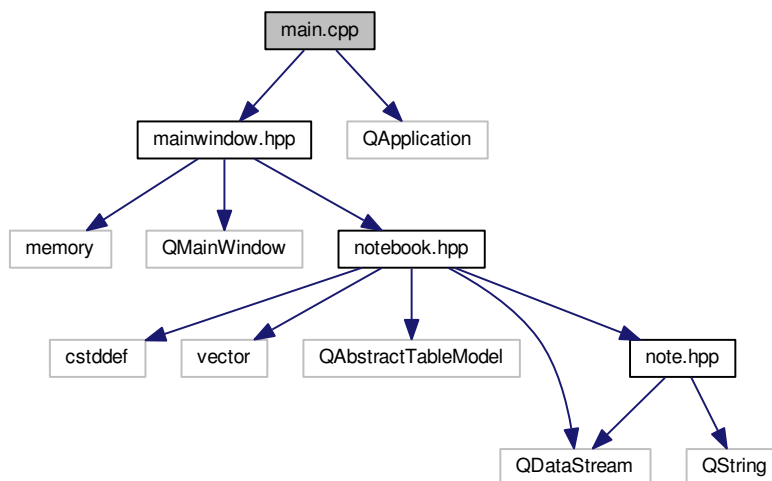
2017

7.4 Файл main.cpp

Файл главной функции.

```
#include "mainwindow.hpp"
#include <QApplication>
```

Граф включаемых заголовочных файлов для main.cpp:



Функции

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

7.4.1 Подробное описание

Файл главной функции.

Автор

Кирилл Пушкарёв

Дата

2017

7.4.2 Функции

7.4.2.1 main()

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

main

Аргументы

argc	количество параметров командной строки
argv	параметры командной строки

Возвращает

код результата

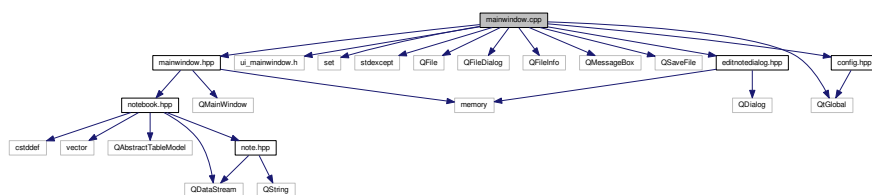
Главная функция программы

7.5 Файл mainwindow.cpp

Файл реализации класса [MainWindow](#).

```
#include "mainwindow.hpp"
#include "ui_mainwindow.h"
#include <set>
#include <stdexcept>
#include <QFile>
#include <QFileDialog>
#include <QFileInfo>
#include <QMessageBox>
#include <QSaveFile>
#include <QtGlobal>
#include "config.hpp"
#include "editnotedialog.hpp"
```

Граф включаемых заголовочных файлов для mainwindow.cpp:



7.5.1 Подробное описание

Файл реализации класса [MainWindow](#).

Автор

Кирилл Пушкарёв

Дата

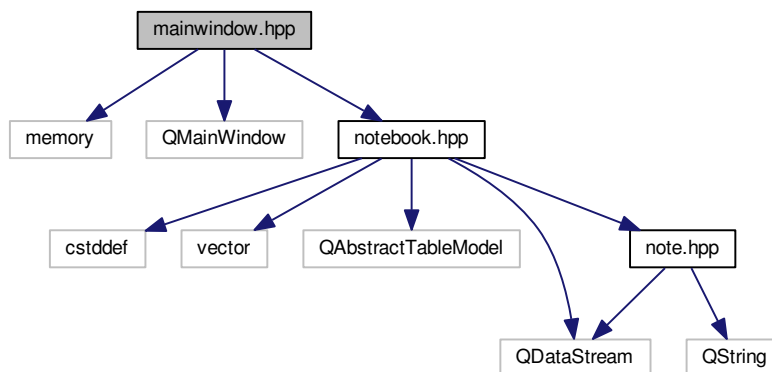
2017

7.6 Файл/mainwindow.hpp

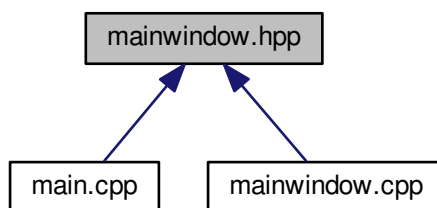
Заголовочный файл класса `MainWindow`.

```
#include <memory>
#include <QMainWindow>
#include "notebook.hpp"
```

Граф включаемых заголовочных файлов для/mainwindow.hpp:



Граф файлов, в которые включается этот файл:



Классы

- class `MainWindow`

Класс главного окна программы.

7.6.1 Подробное описание

Заголовочный файл класса `MainWindow`.

Автор

Кирилл Пушкарёв

Дата

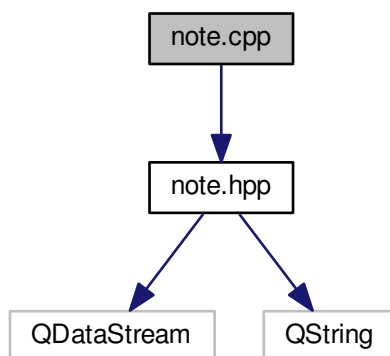
2017

7.7 Файл note.cpp

Файл реализации класса [Note](#).

```
#include "note.hpp"
```

Граф включаемых заголовочных файлов для note.cpp:



7.7.1 Подробное описание

Файл реализации класса [Note](#).

Автор

Кирилл Пушкарёв

Дата

2017

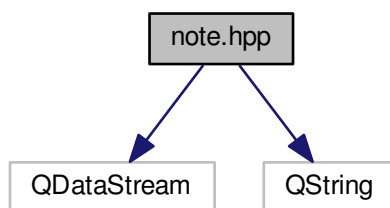
7.8 Файл note.hpp

Заголовочный файл класса [Note](#).

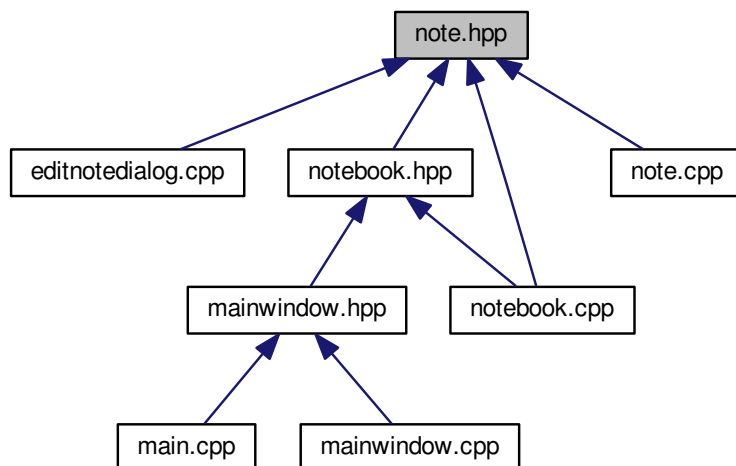
```
#include <QDataStream>
```

```
#include <QString>
```

Граф включаемых заголовочных файлов для note.hpp:



Граф файлов, в которые включается этот файл:



Классы

- class [Note](#)

Класс заметки.

Функции

- `QDataStream & operator<< (QDataStream &ost, const Note ¬e)`
Реализация оператора `<<` для вывода [Note](#) в `QDataStream`.
- `QDataStream & operator>> (QDataStream &ist, Note ¬e)`
Реализация оператора `>>` для ввода [Note](#) из `QDataStream`.

7.8.1 Подробное описание

Заголовочный файл класса [Note](#).

Автор

Кирилл Пушкарёв

Дата

2017

7.8.2 Функции

7.8.2.1 `operator<<()`

```
QDataStream& operator<< (  
    QDataStream & ost,  
    const Note & note ) [inline]
```

Реализация оператора `<<` для вывода [Note](#) в `QDataStream`.

Аргументы

ost	Поток для вывода.
note	Заметка.

Возвращает

Поток `ost` после вывода.

Данная функция объявлена `inline`, чтобы компилятор мог подставить её код вместо полноценного вызова функции. В данном случае это оправданно, так как функция всего лишь перенаправляет вызов методу `save()`.

7.8.2.2 `operator>>()`

```
QDataStream& operator>> (  
    QDataStream & ist,  
    Note & note ) [inline]
```

Реализация оператора >> для ввода [Note](#) из QDataStream.

Аргументы

ist	Поток для ввода.
note	Заметка.

Возвращает

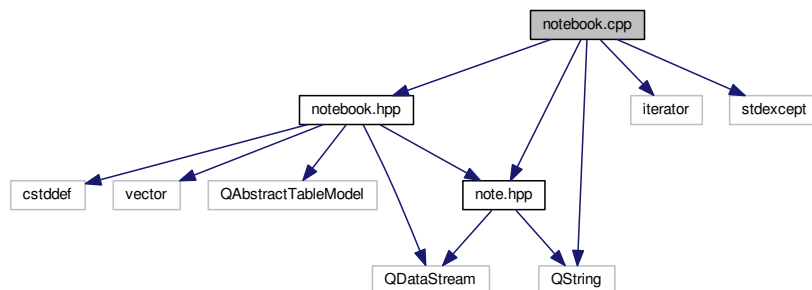
Поток ist после ввода.

7.9 Файл notebook.cpp

Файл реализации класса [Notebook](#).

```
#include "notebook.hpp"
#include <iterator>
#include <stdexcept>
#include <QString>
#include "note.hpp"
```

Граф включаемых заголовочных файлов для notebook.cpp:



7.9.1 Подробное описание

Файл реализации класса [Notebook](#).

Автор

Кирилл Пушкарёв

Дата

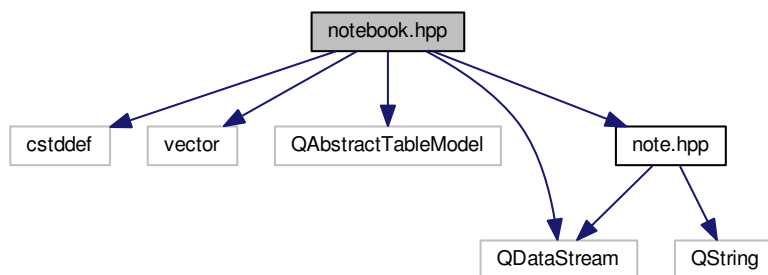
2017

7.10 Файл notebook.hpp

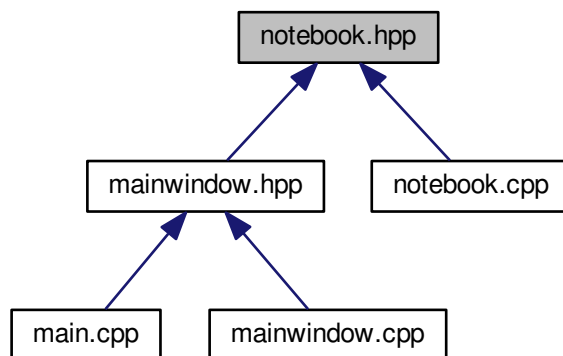
Заголовочный файл класса [Notebook](#).

```
#include <cstdint>
#include <vector>
#include <QAbstractTableModel>
#include <QDataStream>
#include "note.hpp"
```

Граф включаемых заголовочных файлов для notebook.hpp:



Граф файлов, в которые включается этот файл:



Классы

- class [Notebook](#)

Класс записной книжки.

Функции

- `QDataStream & operator<< (QDataStream &ost, const Notebook ¬ebook)`
Реализация оператора << для вывода `Notebook` в `QDataStream`.
- `QDataStream & operator>> (QDataStream &ist, Notebook ¬ebook)`
Реализация оператора >> для ввода `Notebook` из `QDataStream`.

7.10.1 Подробное описание

Заголовочный файл класса `Notebook`.

Автор

Кирилл Пушкарёв

Дата

2017

7.10.2 Функции

7.10.2.1 `operator<<()`

```
QDataStream& operator<< (  
    QDataStream & ost,  
    const Notebook & notebook ) [inline]
```

Реализация оператора << для вывода `Notebook` в `QDataStream`.

Аргументы

ost	Поток для вывода.
notebook	Записная книжка.

Возвращает

Поток ost после вывода.

Данная функция объявлена `inline`, чтобы компилятор мог подставить её код вместо полноценного вызова функции. В данном случае это оправданно, так как функция всего лишь перенаправляет вызов методу `save()`.

7.10.2.2 `operator>>()`

```
QDataStream& operator>> (  
    QDataStream & ist,  
    Notebook & notebook ) [inline]
```

Реализация оператора >> для ввода [Notebook](#) из QDataStream.

Аргументы

ist	Поток для ввода.
notebook	Записная книжка.

Возвращает

Поток ist после ввода.

Предметный указатель

- ~EditNoteDialog
 - EditNoteDialog, [15](#)
- ~MainWindow
 - MainWindow, [19](#)
- accept
 - EditNoteDialog, [15](#)
- applicationName
 - config.hpp, [30](#)
- columnCount
 - Notebook, [24](#)
- config.hpp, [29](#)
 - applicationName, [30](#)
 - notebookFileNameFilter, [30](#)
- data
 - Notebook, [25](#)
- destroyNotebook
 - MainWindow, [19](#)
- EditNoteDialog, [13](#)
 - ~EditNoteDialog, [15](#)
 - accept, [15](#)
 - EditNoteDialog, [14](#)
 - mUi, [15](#)
- editnotedialog.cpp, [31](#)
- editnotedialog.hpp, [31](#)
- headerData
 - Notebook, [25](#)
- mNotebook
 - MainWindow, [20](#)
- mUi
 - EditNoteDialog, [15](#)
 - MainWindow, [20](#)
- main
 - main.cpp, [33](#)
- main.cpp, [33](#)
 - main, [33](#)
- MainWindow, [16](#)
 - ~MainWindow, [19](#)
 - destroyNotebook, [19](#)
 - mNotebook, [20](#)
 - mUi, [20](#)
 - MainWindow, [19](#)
 - notebookFileNameChanged, [19](#)
 - notebookOpened, [20](#)
 - saveNotebookToFile, [20](#)
- mainwindow.cpp, [34](#)
- mainwindow.hpp, [35](#)
- Note, [21](#)
 - Note, [22](#)
- note.cpp, [36](#)
- note.hpp, [37](#)
 - operator<<, [38](#)
 - operator>>, [38](#)
- Notebook, [22](#)
 - columnCount, [24](#)
 - data, [25](#)
 - headerData, [25](#)
 - operator[], [26](#)
 - rowCount, [26](#)
 - SizeType, [24](#)
- notebook.cpp, [39](#)
- notebook.hpp, [40](#)
 - operator<<, [41](#)
 - operator>>, [41](#)
- notebookFileNameChanged
 - MainWindow, [19](#)
- notebookFileNameFilter
 - config.hpp, [30](#)
- notebookOpened
 - MainWindow, [20](#)
- operator<<
 - note.hpp, [38](#)
 - notebook.hpp, [41](#)
- operator>>
 - note.hpp, [38](#)
 - notebook.hpp, [41](#)
- operator[]
 - Notebook, [26](#)
- rowCount
 - Notebook, [26](#)
- saveNotebookToFile
 - MainWindow, [20](#)
- SizeType
 - Notebook, [24](#)