Класс QLabel

Класс QLabel в библиотеке Qt представляет собой виджет, который может отображать текст или изображение. Вот краткое описание его основных методов, сигналов и слотов:

Методы:

- text() - возвращает текст метки.
- setText(const QString&) - устанавливает текст метки.
- setPixmap(const QPixmap&) - устанавливает изображение для метки.
- setAlignment(Qt::Alignment) - устанавливает выравнивание текста в метке.
- setWordWrap(bool) - включает или отключает перенос слов в тексте метки.

Сигналы:

ссылки, но мы ими не пользовались

Слоты:

- clear() - очищает содержимое метки.
- setNum(int num) - устанавливает числовое значение для отображения в метке.
- setNum(double num) - устанавливает числовое значение с плавающей точкой для отображения в метке.

Класс QPushButton

Методы:

- QPushButton() - конструктор кнопки.
- ~QPushButton() - деструктор.
- setText(const QString &) - устанавливает текст кнопки.
- setFlat(bool) - делает кнопку “плоской”, то есть без рамки.

Сигналы:

- clicked() - испускается, когда кнопка нажата и отпущена.
- pressed() - испускается, когда кнопка нажата.
- released() - испускается, когда кнопка отпущена.
- toggled(bool) - испускается, когда состояние кнопки переключается (если она checkable).

Слоты:

- Слоты для QPushButton обычно связаны с сигналами и могут выполнять любые действия, например, изменение текста кнопки или открытие нового окна.

Пример использования QPushButton в C++:

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    QPushButton button("Нажми меня");
    QObject::connect(&button, &QPushButton::clicked, {
        qDebug("Кнопка была нажата!");
    });
    button.show();

    return app.exec();
}
```

Класс QCheckBox

Класс QCheckBox в Qt представляет собой флажок с текстовой меткой, который может находиться в состоянии включено (checked) или выключено (unchecked). Вот краткое описание его методов, сигналов и слотов:

Методы:

- QCheckBox(const QString &text, QWidget *parent = nullptr) - конструктор, который создает флажок с текстом.
- ~QCheckBox() - деструктор.
- bool isChecked() const - возвращает true, если флажок установлен.
- void setChecked(bool) - устанавливает или снимает флажок.

Сигналы:

- void stateChanged(int state) - испускается, когда состояние флажка изменяется.

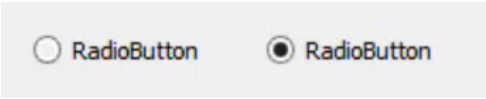
Слоты:

- void toggle() - переключает состояние флажка.

В этом примере создается окно с QCheckBox, который выводит сообщение в консоль при изменении его состояния.

```
QCheckBox checkBox("Выберите меня");
QObject::connect(&checkBox, &QCheckBox::stateChanged, {
    if (state == Qt::Checked) {
        qDebug("Флажок установлен!");
    } else {
        qDebug("Флажок снят!");
    }
});
```

Класс QRadioButton



при создании нескольких “радиокнопок” нажата быть может только одна

Методы:

- QRadioButton(QWidget *parent = nullptr) - конструктор, создающий радиокнопку.
- QRadioButton(const QString &text, QWidget *parent = nullptr) - конструктор, создающий радиокнопку с текстом.
- ~QRadioButton() - деструктор.
- bool isChecked() const - возвращает true, если кнопка выбрана.
- void setChecked(bool check) - устанавливает или снимает выбор с кнопки.
- void setText(const QString &text) - устанавливает текст кнопки.
- QString text() const - возвращает текст кнопки.

Сигналы:

- void toggled(bool checked) - испускается, когда состояние кнопки изменяется.

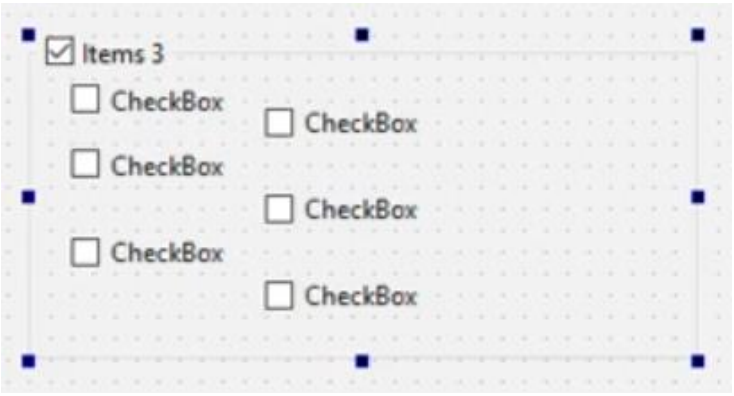
Слоты:

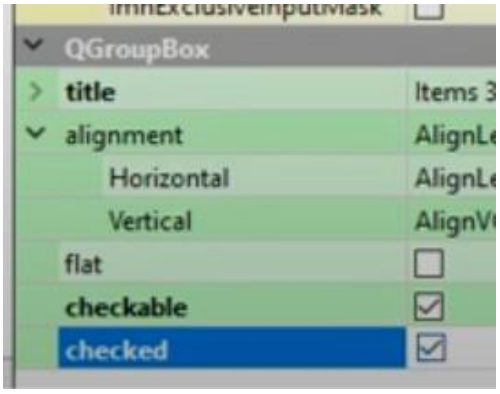
- void setChecked(bool) - устанавливает состояние кнопки в выбранное или не выбранное.

```
// Создание группы радиокнопок
QButtonGroup group;
QRadioButton radio1("Опция 1");
QRadioButton radio2("Опция 2");
QRadioButton radio3("Опция 3");

// Добавление кнопок в группу
group.addButton(&radio1);
group.addButton(&radio2);
group.addButton(&radio3);

// Установка одной из кнопок выбранной по умолчанию
radio1.setChecked(true);

// Добавление радиокнопок в вертикальное расположение
layout.addWidget(&radio1);
```

Только одна кнопка в группе может быть выбрана в данный момент времени.

Класс QRadioButton





Методы:

- QGroupBox(const QString &title, QWidget *parent = nullptr) - конструктор, создающий групповую рамку с заголовком.
- ~QGroupBox() - деструктор.
- QString title() const - возвращает заголовок групповой рамки.
- void setTitle(const QString &title) - устанавливает заголовок групповой рамки.
- bool isCheckable() const - возвращает true, если групповая рамка может быть выбрана.
- void setCheckable(bool checkable) - делает групповую рамку выбираемой.
- bool isChecked() const - возвращает true, если групповая рамка выбрана.
- void setChecked(bool checked) - устанавливает или снимает выбор с групповой рамки.
- void setFlat(bool flat) - делает групповую рамку плоской.

Сигналы:

- void clicked(bool checked = false) - испускается, когда на групповую рамку кликают.
- void toggled(bool on) - испускается, когда состояние выбора групповой рамки изменяется.

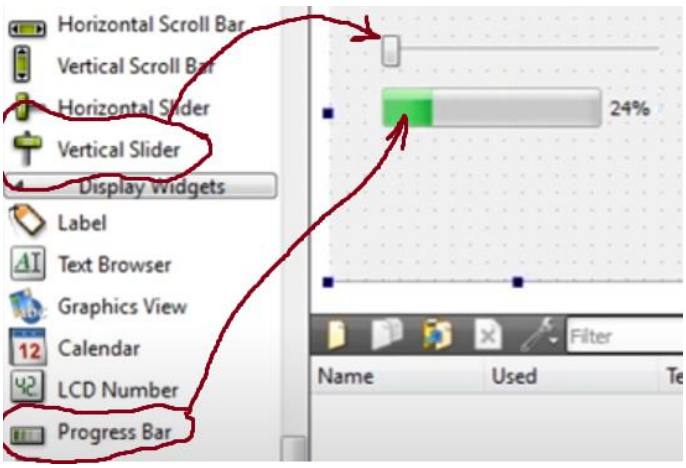
Слоты:

- void setChecked(bool checked) - устанавливает или снимает выбор с групповой рамки.

```
QGroupBox groupBox("Эксклюзивные радиокнопки");
QRadioButton radio1("&Радиокнопка 1");
QRadioButton radio2("Р&адиокнопка 2");
QRadioButton radio3("Ра&диокнопка 3");

radio1.setChecked(true);

QVBoxLayout vbox;
vbox.addWidget(&radio1);
vbox.addWidget(&radio2);
vbox.addWidget(&radio3);
vbox.addStretch(1);
groupBox.setLayout(&vbox);
```

В этом примере создается QGroupBox с заголовком “Эксклюзивные радиокнопки” и тремя радиокнопками внутри. Первая радиокнопка установлена как выбранная по умолчанию

Класс QSlider



Методы:

- QSlider(QWidget *parent = nullptr) - конструктор по умолчанию.
- QSlider(Qt::Orientation orientation, QWidget *parent = nullptr) - конструктор с указанием ориентации.
- void setTickInterval(int ti) - устанавливает интервал между делениями шкалы.
- void setTickPosition(QSlider::TickPosition position) - устанавливает позицию делений шкалы.
- int tickInterval() const - возвращает текущий интервал между делениями.

Сигналы:

- void valueChanged(int value) - испускается, когда значение ползунка изменяется.
- void sliderPressed() - испускается, когда пользователь начинает перемещение ползунка.
- void sliderMoved(int position) - испускается, когда ползунок перемещается.
- void sliderReleased() - испускается, когда пользователь отпускает ползунок.

Слоты:

- void setValue(int value) - устанавливает значение ползунка.
- void setMinimum(int min) - устанавливает минимальное значение диапазона.

- void setMaximum(int max) - устанавливает максимальное значение диапазона.

В этом примере создается горизонтальный ползунок (QSlider) с диапазоном значений от 0 до 100 и начальным значением 50. Также создается метка (QLabel), отображающая текущее значение ползунка. При изменении положения ползунка значение метки обновляется[1].

```
    QSlider slider(Qt::Horizontal);
    slider.setMinimum(0);
    slider.setMaximum(100);
    slider.setValue(50);

    QLabel label;
    label.setText("Значение: " + QString::number(slider.value()));

    // Соединение сигнала изменения значения ползунка со слотом обновления
текста метки
    QObject::connect(&slider, &QSlider::valueChanged, &{
        label.setText("Значение: " + QString::number(newValue));
    });

    layout.addWidget(&slider);
    layout.addWidget(&label);
    window.setLayout(&layout);
```

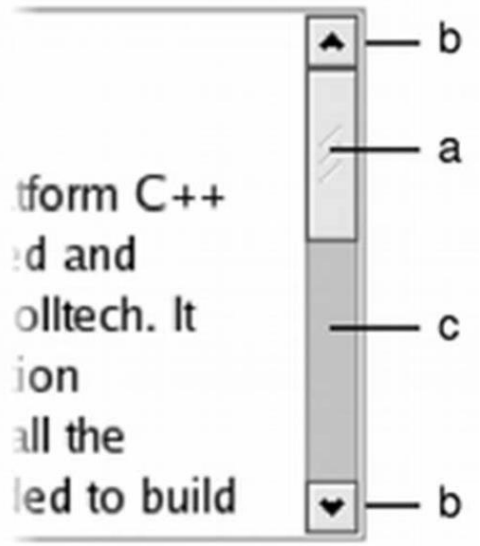
Класс QScrollBar

Методы:

- QScrollBar(QWidget *parent = nullptr) - конструктор по умолчанию.
- QScrollBar(Qt::Orientation orientation, QWidget *parent = nullptr) - конструктор с указанием ориентации полосы прокрутки.
- void setValue(int value) - устанавливает текущее значение полосы прокрутки.
- void setMinimum(int min) - устанавливает минимальное значение диапазона.
- void setMaximum(int max) - устанавливает максимальное значение диапазона.
- void setSingleStep(int step) - устанавливает шаг одиночного перемещения.
- void setPageStep(int step) - устанавливает шаг перемещения при прокрутке страницы.

Сигналы:

- void valueChanged(int value) - испускается, когда значение полосы прокрутки изменяется.
- void sliderPressed() - испускается, когда пользователь начинает перемещение слайдера.
- void sliderMoved(int position) - испускается, когда слайдер перемещается.
- void sliderReleased() - испускается, когда пользователь отпускает слайдер.



```
    QScrollBar scrollBar(Qt::Horizontal);
    scrollBar.setMinimum(0);
    scrollBar.setMaximum(100);

    QLabel label;
    label.setAlignment(Qt::AlignCenter);

    // Соединение сигнала изменения значения полосы прокрутки со слотом
обновления текста метки
    QObject::connect(&scrollBar, &QScrollBar::valueChanged, &{
        label.setText(QString("Значение: %1").arg(newValue));
    });

    layout.addWidget(&scrollBar);
    layout.addWidget(&label);
    window.setLayout(&layout);
    window.show();
```

В этом примере создается горизонтальная полоса прокрутки (QScrollBar) с диапазоном значений от 0 до 100. Также создается метка (QLabel), отображающая текущее значение полосы прокрутки. При изменении положения полосы прокрутки значение метки обновляется[1]

Классы QLineEdit и QTextEdit

QLineEdit:

- Предназначен для однострочного ввода текста.
- Обычно используется для ввода коротких текстов, таких как имя пользователя или пароль.
- Не поддерживает форматирование текста.

Методы QLineEdit:

- setText(const QString &text) - устанавливает текст в поле ввода.
- text() - возвращает текущий текст из поля ввода.
- setPlaceholderText(const QString &text) - устанавливает текст-подсказку.

Сигналы QLineEdit:

- textChanged(const QString &text) - испускается при изменении текста.
- textEdited(const QString &text) - испускается при редактировании текста пользователем.
- returnPressed() - испускается при нажатии клавиши Enter.

QTextEdit:

- Предназначен для многострочного ввода и отображения текста.
- Поддерживает как простой текст, так и форматированный (например, HTML).
- Имеет богатые возможности редактирования и форматирования текста.

Методы QTextEdit:

- setPlainText(const QString &text) - устанавливает простой текст в редактор.
- toPlainText() - возвращает текущий простой текст из редактора.
- setHtml(const QString &text) - устанавливает форматированный текст в редактор.
- toHtml() - возвращает текущий форматированный текст из редактора.

Сигналы QTextEdit:

- textChanged() - испускается при изменении текста в редакторе.

В этом примере создается окно с однострочным полем ввода (QLineEdit) и многострочным текстовым редактором (QTextEdit). Оба виджета имеют текст-подсказку, который отображается, когда они пусты

```
QLineEdit lineEdit;
lineEdit.setPlaceholderText("Введите текст...");

QTextEdit textEdit;
textEdit.setPlaceholderText("Введите многострочный текст...");

layout.addWidget(&lineEdit);
layout.addWidget(&textEdit);
window.setLayout(&layout);
window.show();
```

Класс QSpinBox

Класс QSpinBox в Qt предназначен для виджета, который позволяет пользователю выбирать целочисленные значения с помощью стрелок вверх и вниз или путем ввода числа в поле. Вот краткое описание его методов, сигналов и слотов:

Методы:

- QSpinBox(QWidget *parent = nullptr) - конструктор.
- int value() const - возвращает текущее значение.
- void setValue(int val) - устанавливает значение.
- void setMinimum(int min) - устанавливает минимальное значение.
- void setMaximum(int max) - устанавливает максимальное значение.
- void setRange(int min, int max) - устанавливает диапазон значений.
- void setSingleStep(int step) - устанавливает шаг изменения значения.

Сигналы:

- void valueChanged(int i) - испускается при изменении значения.
- void textChanged(const QString &text) - испускается при изменении текста в поле ввода.

Слоты:

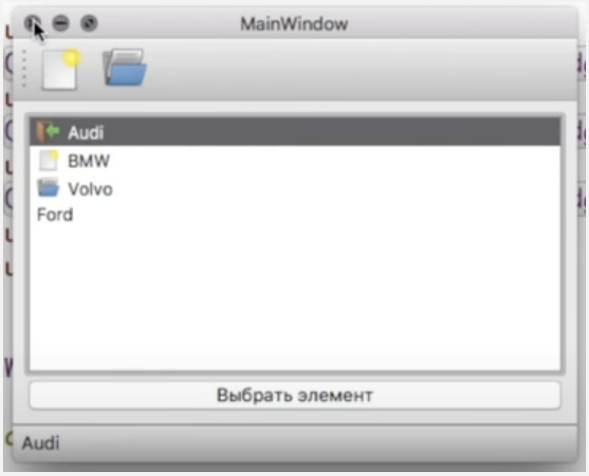
- void setValue(int val) - слот для установки значения.

В этом примере создается виджет QSpinBox, который позволяет выбирать значения от 0 до 100 с шагом 1. Текущее значение установлено на 50. При изменении значения испускается сигнал valueChanged, который может быть соединен со слотом для выполнения дополнительных действий[1].

```
QSpinBox spinBox;
spinBox.setRange(0, 100);
spinBox.setSingleStep(1);
spinBox.setValue(50);
```

Класс QListWidget

Класс QListWidget в Qt представляет собой удобный элемент управления, который позволяет отображать список элементов. Вот краткое описание его методов, сигналов и слотов:



Методы:

- addItem(const QString &label) - добавляет элемент с указанным текстом.
- addItems(const QStringList &labels) - добавляет список элементов.
- clear() - удаляет все элементы из списка.
- currentItem() - возвращает текущий выбранный элемент.
- item(int row) - возвращает элемент по указанному индексу.
- setCurrentItem(QListWidgetItem *item) - устанавливает текущий выбранный элемент.

Сигналы:

- itemClicked(QListWidgetItem *item) - испускается при клике на элемент.
- currentItemChanged(QListWidgetItem *current, QListWidgetItem *previous) - испускается при изменении текущего выбранного элемента.

Слоты:

- clear() - слот для удаления всех элементов из списка.
- scrollToItem(const QListWidgetItem *item) - прокручивает список до элемента.

В этом примере создается окно с QListWidget, содержащим три элемента. При клике на элемент в консоль выводится сообщение с текстом выбранного элемента[1].

```
QListWidget listWidget;
listWidget.addItem("Элемент 1");
listWidget.addItem("Элемент 2");
listWidget.addItem("Элемент 3");

QObject::connect(&listWidget, &QListWidget::itemClicked, {
    qDebug() << "Выбран элемент:" << item->text();
});

layout.addWidget(&listWidget);
window.setLayout(&layout);
window.show();
```

Класс QTableWidget

Класс QTableWidget в Qt является удобным элементом управления для отображения таблиц с элементами. Он наследуется от QTableView и предоставляет модель по умолчанию для работы с таблицами на основе элементов.

Методы:

- setItem(int row, int column, QTableWidgetItem *item) - устанавливает элемент в указанную ячейку.
- item(int row, int column) - возвращает элемент из указанной ячейки.
- setRowCount(int rows) и setColumnCount(int columns) - устанавливают количество строк и столбцов.
- insertRow(int row) и insertColumn(int column) - вставляют строку или столбец.
- removeRow(int row) и removeColumn(int column) - удаляют строку или столбец.

Сигналы:

- cellChanged(int row, int column) - испускается, когда содержимое ячейки изменяется.
- cellClicked(int row, int column) - испускается при клике на ячейку.
- cellDoubleClicked(int row, int column) - испускается при двойном клике на ячейку.

Слоты:

- clear() - очищает все ячейки таблицы.
- clearContents() - очищает содержимое ячеек, но сохраняет структуру таблицы.

```
    QVBoxLayout layout(&window);

    QTableWidget table(3, 3); // Таблица 3x3

    // Заполнение таблицы элементами
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            QTableWidgetItem *item = new QTableWidgetItem(QString("Ячейка
%1,%2").arg(i).arg(j));
            table.setItem(i, j, item);
        }
    }
```

В этом примере создается таблица размером 3x3, каждая ячейка которой заполняется текстом с указанием ее позиции. Приложение отображает таблицу, и пользователь может взаимодействовать с ней[1][2].

Класс QComboBox

Класс QComboBox в Qt представляет собой виджет, который сочетает в себе кнопку и выпадающий список. Вот краткое описание его методов, сигналов и слотов:

Методы:

- addItem(const QString &text) - добавляет элемент в комбобокс.
- addItems(const QStringList &texts) - добавляет список элементов.
- setCurrentIndex(int index) - устанавливает текущий выбранный элемент по индексу.
- currentText() - возвращает текст текущего выбранного элемента.
- removeItem(int index) - удаляет элемент по указанному индексу.

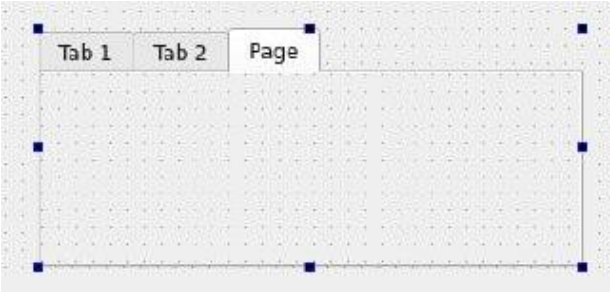
Сигналы:

- currentIndexChanged(int index) - испускается при изменении выбранного элемента.
- activated(int index) - испускается при выборе элемента пользователем.
- highlighted(int index) - испускается при наведении на элемент списка.

Слоты:

- clear() - очищает все элементы из комбобокса.
- setCurrentIndex(int index) - устанавливает текущий выбранный элемент по индексу.

Класс QTabWidget



Класс QTabWidget в Qt представляет собой контейнер для вкладок, который позволяет организовать содержимое в виде вкладок. Вот краткое описание его методов, сигналов и слотов:

Методы:

- addTab(QWidget *page, const QString &label) - добавляет вкладку с виджетом и меткой.
- removeTab(int index) - удаляет вкладку по индексу.
- setCurrentIndex(int index) - устанавливает текущую вкладку по индексу.
- setTabText(int index, const QString &label) - устанавливает текст метки вкладки.
- setTabIcon(int index, const QIcon &icon) - устанавливает иконку вкладки.

Сигналы:

- currentChanged(int index) - испускается при изменении текущей вкладки.
- tabCloseRequested(int index) - испускается при запросе закрытия вкладки.

Слоты:

- setCurrentIndex(int index) - устанавливает текущую вкладку по индексу.
- setCurrentWidget(QWidget *widget) - устанавливает текущую вкладку по виджету.

В этом примере создается QTabWidget с двумя вкладками, каждая из которых содержит метку (QLabel) с текстом. Пользователь может переключаться между вкладками, чтобы увидеть разное содержимое[1].

```
// Добавление первой вкладки
QLabel *label1 = new QLabel("Содержимое вкладки 1");
tabWidget.addTab(label1, "Вкладка 1");

// Добавление второй вкладки
QLabel *label2 = new QLabel("Содержимое вкладки 2");
tabWidget.addTab(label2, "Вкладка 2");
```

Класс QFile

Класс QFile в Qt используется для работы с файлами. Он предоставляет интерфейс для чтения и записи файлов на дискеQFile является подклассом QIODevice, поэтому он наследует все методы для ввода/вывода, такие как read(), write(), open(), close() и другие[1].

Основные методы:

- setFileName(const QString &name) - устанавливает имя файла для объекта QFile.
- open(QIODevice::OpenMode mode) - открывает файл с указанным режимом доступа.
- write(const QByteArray &byteArray) - записывает массив байтов в файл.
- readAll() - читает все данные из файла и возвращает их как QByteArray.
- close() - закрывает файл.

```cpp
int main() {
    QFile file("example.txt");

    // Открытие файла для записи
    if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QTextStream stream(&file);
        stream << "Пример текста в файле.\n";
        file.close();
    } else {
        qDebug() << "Не удалось открыть файл для записи";
    }

    // Открытие файла для чтения
    if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QTextStream stream(&file);
        QString line = stream.readLine();
        qDebug() << line;
        file.close();
    } else {
        qDebug() << "Не удалось открыть файл для чтения";
    }
```

Сигналы и слоты: QFile не имеет специфических сигналов и слотов, так как это не виджет и он не предназначен для взаимодействия с пользовательским интерфейсом. Однако, он использует сигналы и слоты QIODevice, такие как readyRead() и bytesWritten(qint64), для асинхронной работы с файлами.
В этом примере создается объект QFile для работы с файлом “example.txt”. Сначала файл открывается для записи, в него записывается строка текста, а затем файл закрывается. После этого файл открывается снова, но уже для чтения, и из него читается первая строка