



ГЛАВА 21

Основные компоненты

Практически все компоненты графического интерфейса определены в модуле `QtWidgets` (за исключением `WebEngineWidgets`) и наследуют класс `QWidget`. Следовательно, методы этих классов, которые мы рассматривали в предыдущих главах, доступны всем компонентам. Если компонент не имеет родителя, он обладает собственным окном, и его положение отсчитывается, например, относительно экрана. Если же компонент имеет родителя, его положение отсчитывается относительно родительского компонента. Это обстоятельство важно учитывать при работе с компонентами. Обращайте внимание на иерархию наследования, которую мы будем показывать для каждого компонента.

21.1. Надпись

Надпись применяется для вывода подсказки пользователю, информирования пользователя о ходе выполнения операции, назначении клавиш быстрого доступа и т. п. Кроме того, надписи позволяют отображать отформатированный с помощью CSS текст в формате HTML, что позволяет реализовать простейший веб-браузер. В библиотеке PyQt 5 надпись реализуется с помощью класса `QLabel`. Иерархия наследования выглядит так:

```
(QObject, QPaintDevice) QWidget QFrame QLabel
```

Конструктор класса `QLabel` имеет два формата:

```
<Объект> QLabel([parent <Родитель>] [, flags <Тип окна>])
<Объект> QLabel(<Текст> [, parent <Родитель>] [, flags <Тип окна>])
```

В параметре `parent` указывается ссылка на родительский компонент. Если он не указан или имеет значение `None`, компонент будет обладать своим собственным окном, тип которого можно задать с помощью параметра `flags`. Параметр `<Текст>` позволяет задать текст, который будет отображен на надписи:

```
label = QtWidgets.QLabel("Текст надписи", flags QtCore.Qt.Window)
label.resize(300, 50)
label.show()
```

Класс `QLabel` поддерживает следующие основные методы (полный их список смотрите на странице <https://doc.qt.io/qt-5/qlabel.html>):

- ◆ `setText(<Текст>)` — задает текст, который будет отображен на надписи. Можно указать как обычный текст, так и содержащий CSS-форматирование текст в формате HTML:

```
label.setText("Текст <b>полужирный</b>")
```

Перевод строки в простом тексте осуществляется с помощью символа \n, а в тексте в формате HTML — с помощью тега
:

```
label.setText ("Текст\nна двух строках")
```

Внутри текста символ &, указанный перед буквой или цифрой, задает комбинацию клавиш быстрого доступа. В этом случае буква, перед которой указан символ &, будет — в качестве подсказки пользователю — подчеркнута. При одновременном нажатии клавиши <Alt> и подчеркнутой буквы компонент, ссылка на который передана в метод setBuddy(), окажется в фокусе ввода. Чтобы вывести сам символ &, необходимо его удвоить. Если надпись не связана с другим компонентом, символ & выводится в составе текста:

```
label = QtWidgets.QLabel ("&Пароль")
lineEdit = QtWidgets.QLineEdit ()
label.setBuddy (lineEdit)
```

Метод является слотом;

- ◆ setNum(<Число>) — преобразует целое или вещественное число в строку и отображает ее на надписи. Метод является слотом;
- ◆ setWordWrap(<Флаг>) — если в параметре указано значение True, текст может переноситься на другую строку. По умолчанию перенос строк не осуществляется;
- ◆ text() — возвращает текст надписи;
- ◆ setTextFormat(<Режим>) — задает режим отображения текста. Могут быть указаны следующие атрибуты класса QtCore.Qt:
 - PlainText — 0 — простой текст;
 - RichText — 1 — текст, отформатированный тегами HTML;
 - AutoText — 2 — автоматическое определение (режим по умолчанию). Если текст содержит HTML-теги, то используется режим RichText, в противном случае — режим PlainText;

- ◆ setAlignment(<Режим>) — задает режим выравнивания текста внутри надписи (допустимые значения мы рассматривали в разд. 20.2):

```
label.setAlignment (QtCore.Qt.AlignRight | QtCore.Qt.AlignBottom)
```

- ◆ setOpenExternalLinks(<Флаг>) — если в качестве параметра указано значение True, теги <a>, присутствующие в тексте, будут преобразованы в гиперссылки:

```
label.setText ('<a href="https://www.google.ru/">Это гиперссылка</a>')
label.setOpenExternalLinks (True)
```

- ◆ setBuddy(<Компонент>) — позволяет связать надпись с другим компонентом. В этом случае в тексте надписи можно задавать клавиши быстрого доступа, указав символ & перед буквой или цифрой. После нажатия комбинации клавиш в фокусе ввода окажется компонент, ссылка на который передана в качестве параметра;

- ◆ setPixmap(<QPixmap>) — позволяет вывести изображение на надпись. В качестве параметра указывается экземпляр класса QPixmap:

```
label.setPixmap (QtGui.QPixmap ("picture.jpg"))
```

Метод является слотом;

- ◆ setPicture(<QPicture>) — позволяет вывести рисунок. В качестве параметра указывается экземпляр класса QPicture. Метод является слотом;

- ◆ `setMovie(<QMovie>)` — позволяет вывести анимацию. В качестве параметра указывается экземпляр класса `QMovie`. Метод является слотом;
- ◆ `setScaledContents(<Флаг>)` — если в параметре указано значение `True`, то при изменении размеров надписи размер содержимого также будет изменяться. По умолчанию изменение размеров содержимого не осуществляется;
- ◆ `setMargin(<Отступ>)` — задает отступы от границ компонента до его содержимого;
- ◆ `setIndent(<Отступ>)` — задает отступ от рамки до текста надписи в зависимости от значения выравнивания. Если выравнивание производится по левой стороне, то задает отступ слева, если по правой стороне, то справа;
- ◆ `clear()` — удаляет содержимое надписи. Метод является слотом;
- ◆ `setTextInteractionFlags(<Режим>)` — задает режим взаимодействия пользователя с текстом надписи. Можно указать следующие атрибуты (или их комбинацию через оператор `|`) класса `QtCore.Qt`:
 - `NoTextInteraction` — `0` — пользователь не может взаимодействовать с текстом надписи;
 - `TextSelectableByMouse` — `1` — текст можно выделить мышью, чтобы, например, скопировать его в буфер обмена;
 - `TextSelectableByKeyboard` — `2` — текст можно выделить с помощью клавиш на клавиатуре. Внутри надписи будет отображен текстовый курсор;
 - `LinksAccessibleByMouse` — `4` — на гиперссылках, присутствующих в тексте надписи, можно щелкать мышью;
 - `LinksAccessibleByKeyboard` — `8` — с гиперссылками, присутствующими в тексте надписи, допускается взаимодействовать с помощью клавиатуры: перемещаться между гиперссылками можно с помощью клавиши `<Tab>`, а переходить по гиперссылке — по нажатию клавиши `<Enter>`;
 - `TextEditable` — `16` — текст надписи можно редактировать;
 - `TextEditorInteraction` — комбинация `TextSelectableByMouse | TextSelectableByKeyboard | TextEditable`;
 - `TextBrowserInteraction` — комбинация `TextSelectableByMouse | LinksAccessibleByMouse | LinksAccessibleByKeyboard`;
- ◆ `setSelection(<Индекс>, <Длина>)` — выделяет фрагмент длиной `<Длина>`, начиная с позиции `<Индекс>`;
- ◆ `selectionStart()` — возвращает начальный индекс выделенного фрагмента или значение `-1`, если ничего не выделено;
- ◆ `selectedText()` — возвращает выделенный текст или пустую строку, если ничего не выделено;
- ◆ `hasSelectedText()` — возвращает значение `True`, если фрагмент текста надписи выделен, и `False` — в противном случае.

Класс `QLabel` поддерживает следующие сигналы:

- ◆ `linkActivated(<URL>)` — генерируется при переходе по гиперссылке. Через параметр внутри обработчика доступен URL-адрес, заданный в виде строки;

- ◆ linkHovered(<URL>) — генерируется при наведении указателя мыши на гиперссылку. Через параметр внутри обработчика доступен URL-адрес в виде строки или пустая строка.

21.2. Командная кнопка

Командная кнопка используется для запуска какой-либо операции. Кнопка реализуется с помощью класса QPushButton. Иерархия наследования:

(QObject, QPaintDevice) — QWidget — QAbstractButton — QPushButton

Конструктор класса QPushButton имеет три формата:

```
<Объект> = QPushButton([parent=<Родитель>])
<Объект> = QPushButton(<Текст>, parent=<Родитель>)
<Объект> = QPushButton(<QIcon>, <Текст>, parent=<Родитель>)
```

В параметре parent указывается ссылка на родительский компонент. Если таковой не задан или имеет значение None, компонент будет обладать своим собственным окном. Параметр <Текст> позволяет задать текст, который отобразится на кнопке, а параметр <QIcon> — добавить перед текстом значок.

Класс QPushButton наследует следующие методы из класса QAbstractButton (здесь приведены только основные — полный их список смотрите на странице <https://doc.qt.io/qt-5/qabstractbutton.html>):

- ◆ setText(<Текст>) — задает текст, который будет отображен на кнопке. Внутри текста символ &, указанный перед буквой или цифрой, задает комбинацию клавиш быстрого доступа. В этом случае буква, перед которой указан символ &, будет — в качестве подсказки пользователю — подчеркнута. Одновременное нажатие клавиши <Alt> и подчеркнутой буквы приведет к нажатию этой кнопки. Чтобы вывести сам символ &, необходимо его удвоить;
- ◆ text() — возвращает текст, отображаемый на кнопке;
- ◆ setShortcut(<QKeySequence>) — задает комбинацию клавиш быстрого доступа. Вот примеры указания значения:

```
button.setShortcut("Alt+B")
button.setShortcut(QtGui.QKeySequence.mnemonic("&B"))
button.setShortcut(QtGui.QKeySequence("Alt+B"))
button.setShortcut(
    QtGui.QKeySequence(Qt.Core.Qt.ALT + Qt.Core.Qt.Key_E))
```

- ◆ setIcon(<QIcon>) — вставляет значок перед текстом кнопки;
- ◆ setIconSize(<QSize>) — задает размеры значка в виде экземпляра класса QSize. Метод является слотом;
- ◆ setAutoRepeat(<Флаг>) — если в качестве параметра указано значение True, сигнал clicked будет периодически генерироваться, пока кнопка находится в нажатом состоянии. Примером являются кнопки, изменяющие значение полосы прокрутки;
- ◆ animateClick([<Интервал>]) — имитирует нажатие кнопки пользователем. После нажатия кнопка находится в этом состоянии указанный промежуток времени, по истечении которого отпускается. Значение указывается в миллисекундах. Если параметр не указан, то интервал равен 100 миллисекундам. Метод является слотом;

- ◆ `click()` — имитирует нажатие кнопки без анимации. Метод является слотом;
- ◆ `setCheckable(<Флаг>)` — если в качестве параметра указано значение `True`, то кнопка является переключателем, который может находиться в двух состояниях: установленном и неустановленном;
- ◆ `setChecked(<Флаг>)` — если в качестве параметра указано значение `True`, кнопка переключатель будет находиться в установленном состоянии. Метод является слотом;
- ◆ `isChecked()` — возвращает значение `True`, если кнопка находится в установленном состоянии, и `False` — в противном случае;
- ◆ `toggle()` — переключает кнопку. Метод является слотом;
- ◆ `setAutoExclusive(<Флаг>)` — если в качестве параметра указано значение `True`, внутри контейнера может быть установлена только одна кнопка-переключатель;
- ◆ `setDown(<Флаг>)` — если в качестве параметра указано значение `True`, кнопка будет находиться в нажатом состоянии;
- ◆ `isDown()` — возвращает значение `True`, если кнопка находится в нажатом состоянии, и `False` — в противном случае.

Кроме указанных состояний, кнопка может находиться в неактивном состоянии. Для этого необходимо передать значение `False` в метод `setEnabled()`, унаследованный от класса `QWidget`. Проверить, активна ли кнопка, позволяет метод `isEnabled()`, возвращающий значение `True`, если кнопка находится в активном состоянии, и `False` — в противном случае. Это же касается и всех прочих компонентов, порожденных от класса `QWidget`.

Класс `QAbstractButton` поддерживает следующие сигналы:

- ◆ `pressed` — генерируется при нажатии кнопки;
- ◆ `released` — генерируется при отпускании ранее нажатой кнопки;
- ◆ `clicked(<Состояние>)` — генерируется при нажатии, а затем отпускании кнопки мыши над кнопкой. Именно для этого сигнала обычно назначают обработчики. Передаваемый обработчику параметр имеет значение `True`, если кнопка-переключатель установлена, и `False`, если она сброшена или это обычная кнопка, а не переключатель;
- ◆ `toggled(<Состояние>)` — генерируется при изменении состояния кнопки-переключателя. Через параметр доступно новое состояние кнопки.

Класс `QPushButton` определяет свои собственные методы (здесь приведены только основные — полный их список смотрите на странице <https://doc.qt.io/qt-5/qpushbutton.html>):

- ◆ `setFlat(<Флаг>)` — если в качестве параметра указано значение `True`, кнопка будет отображаться без рамки;
- ◆ `setAutoDefault(<Флаг>)` — если в качестве параметра указано значение `True`, кнопка может быть нажата с помощью клавиши `<Enter>`, при условии, что она находится в фокусе. По умолчанию нажать кнопку позволяет только клавиша `<Пробел>`. В диалоговых окнах для всех кнопок по умолчанию указано значение `True`, а для остальных окон — значение `False`;
- ◆ `setDefault(<Флаг>)` — задает кнопку по умолчанию. Метод работает только в диалоговых окнах. Эта кнопка может быть нажата с помощью клавиши `<Enter>`, когда фокус ввода установлен на другой компонент, — например, на текстовое поле;
- ◆ `setMenu(<QMenu>)` — устанавливает всплывающее меню, которое будет отображаться при нажатии кнопки. В качестве параметра указывается экземпляр класса `QMenu`;

- ◆ menu() — возвращает ссылку на всплывающее меню или значение None;
- ◆ showMenu() — отображает всплывающее меню. Метод является слотом.

21.3. Переключатель

Переключатели (иногда их называют *радиокнопками*) всегда используются группами. В такой группе может быть установлен только один переключатель — при попытке установить другой переключатель ранее установленный сбрасывается. Для объединения переключателей в группу можно воспользоваться классом QGroupBox, который мы уже рассматривали в разд. 20.7, а также классом QButtonGroup.

Переключатель реализуется классом QRadioButton. Иерархия наследования:

(QObject, QPaintDevice) — QWidget — QAbstractButton — QRadioButton

Конструктор класса QRadioButton имеет два формата:

<Объект> = QRadioButton([parent=<Родитель>])

<Объект> = QRadioButton(<Текст>[, parent=<Родитель>])

Класс QRadioButton наследует все методы класса QAbstractButton (см. разд. 21.2). Установить или сбросить переключатель позволяет метод setChecked(), а проверить его текущее состояние можно с помощью метода isChecked(). Отследить изменение состояния можно в обработчике сигнала toggled(<Состояние>), в параметре которого передается логическая величина, указывающая новое состояние переключателя.

21.4. Флажок

Флажок предназначен для включения или выключения какой-либо опции и может находиться в нескольких состояниях: установленном, сброшенном и промежуточном (неопределенном) — последнее состояние может быть запрещено программно. Флажок реализуется с помощью класса QCheckBox. Иерархия наследования:

(QObject, QPaintDevice) — QWidget — QAbstractButton — QCheckBox

Конструктор класса QCheckBox имеет два формата:

<Объект> = QCheckBox([parent=<Родитель>])

<Объект> = QCheckBox(<Текст>[, parent=<Родитель>])

Класс QCheckBox наследует все методы класса QAbstractButton (см. разд. 21.2), а также добавляет несколько новых:

- ◆ setCheckState(<Статус>) — задает состояние флажка. Могут быть указаны следующие атрибуты класса QtCore.Qt:
 - Unchecked — 0 — флажок сброшен;
 - PartiallyChecked — 1 — флажок находится в промежуточном состоянии;
 - Checked — 2 — флажок установлен;
- ◆ checkState() — возвращает текущее состояние флажка;
- ◆ setTristate([<Флаг>=True]) — если в качестве параметра указано значение True (значение по умолчанию), флажок может находиться во всех трех состояниях. По умолчанию поддерживаются только установленное и сброшенное состояния;

- ◆ `isTristate()` — возвращает значение `True`, если флагок поддерживает три состояния, и `False` — если только два.

Чтобы перехватить изменение состояния флагка, следует назначить обработчик сигнала `stateChanged(<Состояние>)`. Через параметр внутри обработчика доступно новое состояние флагка, заданное в виде целого числа.

Если используется флагок, поддерживающий только два состояния, установить или сбросить его позволяет метод `setChecked()`, а проверить текущее состояние — метод `isChecked()`. Обработать изменение состояния можно в обработчике сигнала `toggled(<Состояние>)`, параметр которого имеет логический тип.

21.5. Однострочное текстовое поле

Однострочное текстовое поле предназначено для ввода и редактирования текста небольшого объема. С его помощью можно также отобразить вводимые символы в виде звездочек (чтобы скрыть пароль) или вообще не отображать их (что позволит скрыть длину пароля). Поле поддерживает технологию `drag & drop`, стандартные комбинации клавиш быстрого доступа, работу с буфером обмена и многое другое.

Однострочное текстовое поле реализуется классом `QLineEdit`. Иерархия наследования:

`(QObject, QPaintDevice) — QWidget — QLineEdit`

Конструктор класса `QLineEdit` имеет два формата:

```
<Объект> = QLineEdit([parent=<Родитель>])
<Объект> = QLineEdit(<Текст>[, parent=<Родитель>])
```

В параметре `parent` указывается ссылка на родительский компонент. Если родитель не указан или имеет значение `None`, компонент будет обладать своим собственным окном. Параметр `<Текст>` позволяет задать текст, который будет отображен в текстовом поле.

21.5.1. Основные методы и сигналы

Класс `QLineEdit` поддерживает следующие методы (полный их список смотрите на странице <https://doc.qt.io/qt-5/qlineedit.html>):

- ◆ `setText(<Текст>)` — помещает указанный текст в поле. Метод является слотом;
- ◆ `insert(<Текст>)` — вставляет текст в текущую позицию текстового курсора. Если в поле был выделен фрагмент, он будет удален;
- ◆ `text()` — возвращает текст, содержащийся в текстовом поле;
- ◆ `displayText()` — возвращает текст, который видит пользователь. Результат зависит от режима отображения, заданного с помощью метода `setEchoMode()`, — например, в режиме `Password` строка будет состоять из звездочек;
- ◆ `clear()` — удаляет весь текст из поля. Метод является слотом;
- ◆ `backspace()` — удаляет выделенный фрагмент. Если выделенного фрагмента нет, удаляет символ, стоящий слева от текстового курсора;
- ◆ `del()` — удаляет выделенный фрагмент. Если выделенного фрагмента нет, удаляет символ, стоящий справа от текстового курсора;

- ◆ `setSelection(<Индекс>, <Длина>)` — выделяет фрагмент длиной `<Длина>`, начиная с позиции `<Индекс>`. Во втором параметре можно указать отрицательное значение;
- ◆ `selectedText()` — возвращает выделенный фрагмент или пустую строку, если ничего не выделено;
- ◆ `selectAll()` — выделяет весь текст в поле. Метод является слотом;
- ◆ `selectionStart()` — возвращает начальный индекс выделенного фрагмента или значение `-1`, если ничего не выделено;
- ◆ `hasSelectedText()` — возвращает значение `True`, если поле содержит выделенный фрагмент, и `False` — в противном случае;
- ◆ `deselect()` — снимает выделение;
- ◆ `isModified()` — возвращает `True`, если текст в поле был изменен пользователем, и `False` — в противном случае. Отметьте, что вызов метода `setText()` помечает поле как **неизмененное**;
- ◆ `setModified(<Флаг>)` — если передано значение `True`, поле ввода помечается как **измененное**, если `False` — как **неизмененное**;
- ◆ `setEchoMode(<Режим>)` — задает режим отображения текста. Могут быть указаны следующие атрибуты класса `QLineEdit`:
 - `Normal` — `0` — показывать символы как они были введены;
 - `NoEcho` — `1` — не показывать вводимые символы;
 - `Password` — `2` — вместо символов выводить звездочки (`*`);
 - `PasswordEchoOnEdit` — `3` — показывать символы при вводе, а после потери фокуса вместо них отображать звездочки (`*`);
- ◆ `setCompleter(<QCompleter>)` — позволяет предлагать возможные варианты значений, начинающиеся с введенных пользователем символов. В качестве параметра указывается экземпляр класса `QCompleter`:

```
lineEdit = QtWidgets.QLineEdit()
arr = ["кадр", "каменный", "камень", "камера"]
completer = QtWidgets.QCompleter(arr, parent=window)
lineEdit.setCompleter(completer)
```
- ◆ `setReadOnly(<Флаг>)` — если в качестве параметра указано значение `True`, поле будет доступно только для чтения;
- ◆ `isReadOnly()` — возвращает значение `True`, если поле доступно только для чтения, и `False` — в противном случае;
- ◆ `setAlignment(<Выравнивание>)` — задает выравнивание текста внутри поля;
- ◆ `setMaxLength(<Количество>)` — задает максимальное количество символов;
- ◆ `setFrame(<Флаг>)` — если в качестве параметра указано значение `False`, поле будет отображаться без рамки;
- ◆ `setDragEnabled(<Флаг>)` — если в качестве параметра указано значение `True`, режим перетаскивания текста из текстового поля с помощью мыши будет включен. По умолчанию однострочное текстовое поле только принимает перетаскиваемый текст;
- ◆ `setPlaceholderText(<Текст>)` — задает текст подсказки, который будет выводиться в поле, когда оно не содержит значения и не имеет фокуса ввода;

- ◆ `setTextMargins()` — задает величины отступов от границ компонента до находящегося в нем текста. Форматы метода:
`setTextMargins(<Слева>, <Сверху>, <Справа>, <Снизу>)`
`setTextMargins(<QMargins>)`
- ◆ `setCursorPosition(<Индекс>)` — задает положение текстового курсора;
- ◆ `cursorPosition()` — возвращает текущее положение текстового курсора;
- ◆ `cursorForward(<Флаг>[, steps=1])` — перемещает текстовый курсор вперед на указанное во втором параметре количество символов. Если в первом параметре указано значение `True`, выполняется выделение фрагмента;
- ◆ `cursorBackward(<Флаг>[, steps=1])` — перемещает текстовый курсор назад на указанное во втором параметре количество символов. Если в первом параметре указано значение `True`, выполняется выделение фрагмента;
- ◆ `cursorWordForward(<Флаг>)` — перемещает текстовый курсор вперед на одно слово. Если в параметре указано значение `True`, выполняется выделение фрагмента;
- ◆ `cursorWordBackward(<Флаг>)` — перемещает текстовый курсор назад на одно слово. Если в параметре указано значение `True`, выполняется выделение фрагмента;
- ◆ `home(<Флаг>)` — перемещает текстовый курсор в начало поля. Если в параметре указано значение `True`, выполняется выделение фрагмента;
- ◆ `end(<Флаг>)` — перемещает текстовый курсор в конец поля. Если в параметре указано значение `True`, выполняется выделение фрагмента;
- ◆ `cut()` — копирует выделенный текст в буфер обмена и удаляет его из поля при условии, что есть выделенный фрагмент и используется режим `Normal`. Метод является слотом;
- ◆ `copy()` — копирует выделенный текст в буфер обмена при условии, что есть выделенный фрагмент и используется режим `Normal`. Метод является слотом;
- ◆ `paste()` — вставляет текст из буфера обмена в текущую позицию текстового курсора при условии, что поле доступно для редактирования. Метод является слотом;
- ◆ `undo()` — отменяет последнюю операцию ввода пользователем при условии, что отмена возможна. Метод является слотом;
- ◆ `redo()` — повторяет последнюю отмененную операцию ввода пользователем, если это возможно. Метод является слотом;
- ◆ `isUndoAvailable()` — возвращает значение `True`, если можно отменить последнюю операцию ввода, и `False` — в противном случае;
- ◆ `isRedoAvailable()` — возвращает значение `True`, если можно повторить последнюю отмененную операцию ввода, и `False` — в противном случае;
- ◆ `createStandardContextMenu()` — создает стандартное меню, которое отображается при щелчке правой кнопкой мыши в текстовом поле. Чтобы изменить стандартное меню, следует создать класс, наследующий класс `QLineEdit`, и переопределить в нем метод `contextMenuEvent(self, <event>)`. Внутри этого метода можно создать свое собственное меню или добавить новый пункт в стандартное меню;
- ◆ `setClearButtonEnabled(<Флаг>)` — если передано `True`, в левой части непустого поля будет выводиться кнопка, нажатием которой можно очистить это поле, если `False`, кнопка очистки выводится не будет.

Класс QLineEdit поддерживает следующие сигналы:

- ◆ cursorPositionChanged(<Старая позиция>, <Новая позиция>) — генерируется при перемещении текстового курсора. Внутри обработчика через первый параметр доступна старая позиция курсора, а через второй параметр — новая позиция. Оба параметра являются целочисленными;
- ◆ editingFinished — генерируется при нажатии клавиши <Enter> или потере полем фокуса ввода;
- ◆ returnPressed — генерируется при нажатии клавиши <Enter>;
- ◆ selectionChanged — генерируется при изменении выделения;
- ◆ textChanged(<Новый текст>) — генерируется при изменении текста внутри поля пользователем или программно. Внутри обработчика через параметр доступен новый текст в виде строки;
- ◆ textEdited(<Новый текст>) — генерируется при изменении текста внутри поля пользователем. При задании текста вызовом метода setText() не генерируется. Внутри обработчика через параметр доступен новый текст в виде строки.

21.5.2. Ввод данных по маске

С помощью метода setInputMask(<Маска>) можно ограничить ввод символов допустимым диапазоном значений. В качестве параметра указывается строка, имеющая следующий формат:

"<Последовательность символов>[;<Символ-заполнитель>] "

В первом параметре указывается комбинация из следующих специальных символов:

- ◆ 9 — обязательна цифра от 0 до 9;
- ◆ 0 — разрешена, но не обязательна цифра от 0 до 9;
- ◆ D — обязательна цифра от 1 до 9;
- ◆ d — разрешена, но не обязательна цифра от 1 до 9;
- ◆ B — обязательна цифра 0 или 1;
- ◆ b — разрешена, но не обязательна цифра 0 или 1;
- ◆ H — обязательен шестнадцатеричный символ (0-9, A-F, a-f);
- ◆ h — разрешен, но не обязательен шестнадцатеричный символ (0-9, A-F, a-f);
- ◆ # — разрешена, но не обязательна цифра, знак плюс или минус;
- ◆ A — обязательна буква в любом регистре;
- ◆ a — разрешена, но не обязательна буква;
- ◆ N — обязательна буква в любом регистре или цифра от 0 до 9;
- ◆ n — разрешена, но не обязательна буква или цифра от 0 до 9;
- ◆ X — обязательен любой символ;
- ◆ x — разрешен, но не обязательен любой символ;
- ◆ > — все последующие буквы переводятся в верхний регистр;
- ◆ < — все последующие буквы переводятся в нижний регистр;

- ◆ ! — отключает изменение регистра;
- ◆ \ — используется для отмены действия спецсимволов.

Все остальные символы трактуются как есть. В необязательном параметре <Символ-заполнитель> можно указать символ, который будет отображаться в поле, обозначая место ввода. Если параметр не указан, заполнителем будет служить пробел:

```
lineEdit.setInputMask("Дата: 99.В9.9999;_") # Дата: __.__._____
lineEdit.setInputMask("Дата: 99.В9.9999;#") # Дата: ##.##.####
lineEdit.setInputMask("Дата: 99.В9.9999 г.") # Дата: . . . . г.
```

Проверить соответствие введенных данных маске позволяет метод `hasAcceptableInput()`. Если данные соответствуют маске, метод возвращает значение `True`, а в противном случае — `False`.

21.5.3. Контроль ввода

Контролировать ввод данных позволяет метод `setValidator(<QValidator>)`. В качестве параметра указывается экземпляр класса, наследующего класс `QValidator` из модуля `QtGui`. Существуют следующие стандартные классы, позволяющие контролировать ввод данных:

- ◆ `QIntValidator` — допускает ввод только целых чисел. Функциональность класса зависит от настройки локали. Форматы конструктора:

```
QIntValidator([parent=None])
QIntValidator(<Минимальное значение>, <Максимальное значение>
              [, parent=None])
```

Пример ограничения ввода диапазоном целых чисел от 0 до 100:

```
lineEdit.setValidator(QtGui.QIntValidator(0, 100, parent=window))
```

- ◆ `QDoubleValidator` — допускает ввод только вещественных чисел. Функциональность класса зависит от настройки локали. Форматы конструктора:

```
QDoubleValidator([parent=None])
QDoubleValidator(<Минимальное значение>, <Максимальное значение>,
                  <Количество цифр после точки>[, parent=None])
```

Пример ограничения ввода диапазоном вещественных чисел от 0.0 до 100.0 и двумя цифрами после десятичной точки:

```
lineEdit.setValidator(
    QtGui.QDoubleValidator(0.0, 100.0, 2, parent=window))
```

Чтобы позволить вводить числа в экспоненциальной форме, необходимо передать значение атрибута `ScientificNotation` в метод `setNotation()`. Если передать значение атрибута `StandardNotation`, будет разрешено вводить числа только в десятичной форме:

```
validator = QtGui.QDoubleValidator(0.0, 100.0, 2, parent=window)
validator.setNotation(QtGui.QDoubleValidator.StandardNotation)
lineEdit.setValidator(validator)
```

- ◆ `QRegExpValidator` — позволяет проверить данные на соответствие регулярному выражению. Форматы конструктора:

```
QRegExpValidator([parent=None])
QRegExpValidator(<QRegExp>[, parent=None])
```

Пример ввода только цифр от 0 до 9:

```
validator = QtGui.QRegExpValidator(  
    QtCore.QRegExp("[0-9]+"), parent=window)  
lineEdit.setValidator(validator)
```

Обратите внимание, что здесь производится проверка полного соответствия шаблону, поэтому символы ^ и \$ явным образом указывать не нужно.

Проверить соответствие введенных данных условию позволяет метод `hasAcceptableInput()`. Если данные соответствуют условию, метод возвращает значение `True`, а в противном случае — `False`.

21.6. Многострочное текстовое поле

Многострочное текстовое поле предназначено для ввода и редактирования как простого текста, так и текста в формате HTML. Поле поддерживает технологии drag & drop, стандартные комбинации клавиш быстрого доступа, работу с буфером обмена и многое другое. Многострочное текстовое поле реализуется с помощью класса `QTextEdit`. Иерархия наследования:

```
(QObject, QPaintDevice) — QWidget — QFrame —  
QAbstractScrollArea — QTextEdit
```

Конструктор класса `QTextEdit` имеет два формата вызова:

```
<Объект> = QTextEdit([parent=<Родитель>])  
<Объект> = QTextEdit(<Текст>, parent=<Родитель>)
```

В параметре `parent` указывается ссылка на родительский компонент. Если параметр не указан или имеет значение `None`, компонент будет обладать своим собственным окном. Параметр `<Текст>` позволяет задать текст в формате HTML, который будет отображен в текстовом поле.

ПРИМЕЧАНИЕ

Класс `QTextEdit` предназначен для отображения как простого текста, так и текста в формате HTML. Если поддержка HTML не нужна, то следует воспользоваться классом `QPlainTextEdit`, который оптимизирован для работы с простым текстом большого объема.

21.6.1. Основные методы и сигналы

Класс `QTextEdit` поддерживает следующие основные методы (полный их списоксмотрите на странице <https://doc.qt.io/qt-5/qtextedit.html>):

- ◆ `setText(<Текст>)` — помещает указанный текст в поле. Текст может быть простым или в формате HTML. Метод является слотом;
- ◆ `setPlainText(<Текст>)` — помещает в поле простой текст. Метод является слотом;
- ◆ `setHtml(<Текст>)` — помещает в поле текст в формате HTML. Метод является слотом;
- ◆ `insertPlainText(<Текст>)` — вставляет простой текст в текущую позицию текстового курсора. Если в поле был выделен фрагмент, он будет удален. Метод является слотом;
- ◆ `insertHtml(<Текст>)` — вставляет текст в формате HTML в текущую позицию текстового курсора. Если в поле был выделен фрагмент, он будет удален. Метод является слотом;

- ◆ `append(<Текст>)` — добавляет новый абзац с указанным текстом в формате HTML в конец поля. Метод является слотом;
- ◆ `setDocumentTitle(<Текст>)` — задает текст заголовка документа (для тега `<title>`);
- ◆ `documentTitle()` — возвращает текст заголовка (из тега `<title>`);
- ◆ `toPlainText()` — возвращает простой текст, содержащийся в текстовом поле;
- ◆ `toHtml()` — возвращает текст в формате HTML;
- ◆ `clear()` — удаляет весь текст из поля. Метод является слотом;
- ◆ `selectAll()` — выделяет весь текст в поле. Метод является слотом;
- ◆ `zoomIn([range=1])` — увеличивает размер шрифта. Метод является слотом;
- ◆ `zoomOut([range=1])` — уменьшает размер шрифта. Метод является слотом;
- ◆ `cut()` — копирует выделенный текст в буфер обмена и удаляет его из поля при условии, что есть выделенный фрагмент. Метод является слотом;
- ◆ `copy()` — копирует выделенный текст в буфер обмена при условии, что есть выделенный фрагмент. Метод является слотом;
- ◆ `paste()` — вставляет текст из буфера обмена в текущую позицию текстового курсора при условии, что поле доступно для редактирования. Метод является слотом;
- ◆ `canPaste()` — возвращает `True`, если из буфера обмена можно вставить текст, и `False` — в противном случае;
- ◆ `setAcceptRichText(<Флаг>)` — если в качестве параметра указано значение `True`, в поле можно будет ввести, вставить из буфера обмена или при помощи перетаскивания текст в формате HTML. Значение `False` дает возможность заносить в поле лишь обычный текст;
- ◆ `acceptRichText()` — возвращает значение `True`, если в поле можно занести текст в формате HTML, и `False` — если доступно занесение лишь обычного текста;
- ◆ `undo()` — отменяет последнюю операцию ввода пользователем при условии, что отмена возможна. Метод является слотом;
- ◆ `redo()` — повторяет последнюю отмененную операцию ввода пользователем, если это возможно. Метод является слотом;
- ◆ `setUndoRedoEnabled(<Флаг>)` — если в качестве значения указано значение `True`, операции отмены и повтора действий разрешены, а если `False` — то запрещены;
- ◆ `isUndoRedoEnabled()` — возвращает значение `True`, если операции отмены и повтора действий разрешены, и `False` — если запрещены;
- ◆ `createStandardContextMenu([<QPoint>])` — создает стандартное меню, которое отображается при щелчке правой кнопкой мыши в текстовом поле. Чтобы изменить стандартное меню, следует создать класс, наследующий класс `QTextEdit`, и переопределить в нем метод `contextMenuEvent(self, <event>)`. Внутри этого метода можно создать свое собственное меню или добавить новый пункт в стандартное меню;
- ◆ `ensureCursorVisible()` — прокручивает область таким образом, чтобы текстовый курсор оказался в зоне видимости;
- ◆ `find()` — производит поиск фрагмента (по умолчанию в прямом направлении без учета регистра символов) в текстовом поле. Если фрагмент найден, он выделяется, и метод возвращает значение `True`, в противном случае — значение `False`. Форматы метода:

```
find(<Искомый текст>[, <Режим>])
find(<QRegExp>[, <Режим>])
```

Искомый текст можно указать либо строкой, либо регулярным выражением, представленным экземпляром класса `QRegExp`. В необязательном параметре `<Режим>` можно указать комбинацию (через оператор `|`) следующих атрибутов класса `QTextDocument` из модуля `QtGui`:

- `FindBackward` — 1 — поиск в обратном, а не в прямом направлении;
 - `FindCaseSensitively` — 2 — поиск с учетом регистра символов;
 - `FindWholeWords` — 4 — поиск целых слов, а не фрагментов;
- ◆ `print(<QPagePaintDevice>)` — отправляет содержимое текстового поля на печать. В качестве параметра указывается экземпляр одного из классов, порожденных от `QPagePaintDevice`: `QPrinter` или `QPdfWriter`. Вот пример вывода документа в файл в формате PDF:

```
pdf = QtGui.QPdfWriter("document.pdf")
textEdit.print(pdf)
```

Класс `QTextEdit` поддерживает следующие сигналы:

- ◆ `currentCharFormatChanged(<QTextCharFormat>)` — генерируется при изменении формата текста. Внутри обработчика через параметр доступен новый формат;
- ◆ `cursorPositionChanged` — генерируется при изменении положения текстового курсора;
- ◆ `selectionChanged` — генерируется при изменении выделения текста;
- ◆ `textChanged` — генерируется при изменении текста в поле;
- ◆ `copyAvailable(<Флаг>)` — генерируется при выделении текста или, наоборот, снятии выделения. Значение параметра `True` указывает, что фрагмент выделен, и его можно скопировать, значение `False` — обратное;
- ◆ `undoAvailable(<Флаг>)` — генерируется при изменении возможности отменить операцию ввода. Значение параметра `True` указывает, что операция ввода может быть отменена, значение `False` говорит об обратном;
- ◆ `redoAvailable(<Флаг>)` — генерируется при изменении возможности повторить отмененную операцию ввода. Значение параметра `True` обозначает возможность повтора отмененной операции, а значение `False` — невозможность сделать это.

21.6.2. Изменение параметров поля

Задать другие параметры поля можно вызовами следующих методов класса `QTextEdit` (полный их списоксмотрите на странице <https://doc.qt.io/qt-5/qtextedit.html>):

- ◆ `setTextInteractionFlags(<Режим>)` — задает режим взаимодействия пользователя с текстом. Можно указать следующие атрибуты (или их комбинацию через оператор `|`) класса `QtCore.Qt`:
 - `NoTextInteraction` — 0 — пользователь не может взаимодействовать с текстом;
 - `TextSelectableByMouse` — 1 — текст можно выделить мышью;
 - `TextSelectableByKeyboard` — 2 — текст можно выделить с помощью клавиатуры.
- Внутри поля будет отображен текстовый курсор;

- `LinksAccessibleByMouse` — 4 — на гиперссылках, присутствующих в тексте, можно щелкать мышью;
 - `LinksAccessibleByKeyboard` — 8 — с гиперссылками, присутствующими в тексте, можно взаимодействовать с клавиатурой: перемещаться между гиперссылками — с помощью клавиши `<Tab>`, а переходить по гиперссылке — нажав клавишу `<Enter>`;
 - `TextEditable` — 16 — текст можно редактировать;
 - `TextEditorInteraction` — комбинация `TextSelectableByMouse` | `TextSelectableByKeyboard` | `TextEditable`;
 - `TextBrowserInteraction` — комбинация `TextSelectableByMouse` | `LinksAccessibleByMouse` | `LinksAccessibleByKeyboard`;
- ◆ `setReadOnly(<Флаг>)` — если в качестве параметра указано значение `True`, поле будет доступно только для чтения;
- ◆ `isReadOnly()` — возвращает значение `True`, если поле доступно только для чтения, и `False` — в противном случае;
- ◆ `setLineWrapMode(<Режим>)` — задает режим переноса строк. В качестве значения могут быть указаны следующие атрибуты класса `QTextEdit`:
- `NoWrap` — 0 — перенос строк не производится;
 - `WidgetWidth` — 1 — перенос строк при достижении ими ширины поля;
 - `FixedPixelWidth` — 2 — перенос строк при достижении ими фиксированной ширины в пикселях, которую можно задать с помощью метода `setLineWrapColumnOrWidth()`;
 - `FixedColumnWidth` — 3 — перенос строк при достижении ими фиксированной ширины в символах, которую можно задать с помощью метода `setLineWrapColumnOrWidth()`;
- ◆ `setLineWrapColumnOrWidth(<Значение>)` — задает фиксированную ширину строк, при достижении которой будет выполняться перенос;
- ◆ `setWordWrapMode(<Режим>)` — задает режим переноса по словам. В качестве значения могут быть указаны следующие атрибуты класса `QTextOption` из модуля `QtGui`:
- `NoWrap` — 0 — перенос по словам не производится;
 - `WordWrap` — 1 — перенос строк только по словам;
 - `ManualWrap` — 2 — аналогичен режиму `NoWrap`;
 - `WrapAnywhere` — 3 — перенос строки может быть внутри слова;
 - `WrapAtWordBoundaryOrAnywhere` — 4 — по возможности перенос по словам, но может быть выполнен и перенос внутри слова;
- ◆ `setOverwriteMode(<Флаг>)` — если в качестве параметра указано значение `True`, вводимый текст будет замещать ранее введенный. Значение `False` отключает замещение;
- ◆ `overwriteMode()` — возвращает значение `True`, если вводимый текст замещает ранее введенный, и `False` — в противном случае;
- ◆ `setAutoFormatting(<Режим>)` — задает режим автоматического форматирования. В качестве значения могут быть указаны следующие атрибуты класса `QTextEdit`:
- `AutoNone` — автоматическое форматирование не используется;
 - `AutoBulletList` — автоматическое создание маркированного списка при вводе пользователем в начале строки символа *;

- AutoAll — включить все режимы. На данный момент эквивалентно режиму AutoBulletList;
- ◆ setCursorWidth (<Ширина>) — задает ширину текстового курсора;
- ◆ setTabChangesFocus (<Флаг>) — если параметром передать значение False, то с помощью нажатия клавиши <Tab> можно вставить в поле символ табуляции. Если указано значение True, клавиша <Tab> используется для передачи фокуса следующему компоненту;
- ◆ setTabStopWidth (<Ширина>) — задает ширину табуляции в пикселях;
- ◆ tabStopWidth() — возвращает ширину табуляции в пикселях.

21.6.3. Указание параметров текста и фона

Для изменения параметров текста и фона предназначены следующие методы класса QTextEdit (полный их список смотрите на странице <https://doc.qt.io/qt-5/qtextedit.html>):

- ◆ setCurrentFont (<QFont>) — задает текущий шрифт. Метод является слотом. В качестве параметра указывается экземпляр класса QFont из модуля QtGui. Конструктор этого класса имеет следующий формат:

```
<Шрифт> = QFont(<Название шрифта>[, pointSize=-1][, weight=-1]
                  [, italic=False])
```

В первом параметре задается название шрифта в виде строки. Необязательный параметр pointSize устанавливает размер шрифта. В параметре weight можно указать степень жирности шрифта: число от 0 до 99 или значение атрибутов Light, Normal, DemiBold, Bold или Black класса QFont. Если в параметре italic указано значение True, шрифт будет курсивным;

- ◆ currentFont() — возвращает экземпляр класса QFont с текущими характеристиками шрифта;
- ◆ setFontFamily(<Название шрифта>) — задает название текущего шрифта. Метод является слотом;
- ◆ fontFamily() — возвращает название текущего шрифта;
- ◆ setFontPointSize(<Размер>) — задает размер текущего шрифта. Метод является слотом;
- ◆ fontPointSize() — возвращает размер текущего шрифта;
- ◆ setFontWeight(<Жирность>) — задает жирность текущего шрифта. Метод является слотом;
- ◆ fontWeight() — возвращает жирность текущего шрифта;
- ◆ setFontItalic(<Флаг>) — если в качестве параметра указано значение True, шрифт будет курсивным. Метод является слотом;
- ◆ fontItalic() — возвращает True, если шрифт курсивный, и False — в противном случае;
- ◆ setFontUnderline(<Флаг>) — если в качестве параметра указано значение True, текст будет подчеркнутым. Метод является слотом;
- ◆ fontUnderline() — возвращает True, если текст подчеркнутый, и False — в противном случае;
- ◆ setTextColor(<QColor>) — задает цвет текущего текста. В качестве значения можно указать атрибут класса QtCore.Qt (например, black, white и т. д.) или экземпляр класса

`QColor` из модуля `QtGui` (например, `QColor("red")`, `QColor("#ff0000")`, `QColor(255, 0, 0)` и др.). Метод является слотом;

- ◆ `textColor()` — возвращает экземпляр класса `QColor` с цветом текущего текста;
- ◆ `setTextBackgroundColor(<QColor>)` — задает цвет фона. В качестве значения можно указать атрибут из класса `QtCore.Qt` (например, `black`, `white` и т. д.) или экземпляр класса `QColor` (например, `QColor("red")`, `QColor("#ff0000")`, `QColor(255, 0, 0)` и др.). Метод является слотом;
- ◆ `textBackgroundColor()` — возвращает экземпляр класса `QColor` с цветом фона;
- ◆ `setAlignment(<Выравнивание>)` — задает горизонтальное выравнивание текста внутри абзаца (допустимые значения мы рассматривали в разд. 20.2). Метод является слотом;
- ◆ `alignment()` — возвращает значение выравнивания текста внутри абзаца.

Задать формат символов можно также с помощью класса `QTextCharFormat`, который определен в модуле `QtGui` и поддерживает дополнительные настройки. После создания экземпляра класса его следует передать в метод `setCurrentCharFormat(<QTextCharFormat>)`. Получить экземпляр класса с текущими настройками позволяет метод `currentCharFormat()`. За подробной информацией по классу `QTextCharFormat` обращайтесь к странице <https://doc.qt.io/qt-5/qtextcharformat.html>.

21.6.4. Класс `QTextDocument`

Класс `QTextDocument` из модуля `QtGui` представляет документ, который отображается в многострочном текстовом поле. Получить ссылку на текущий документ позволяет метод `document()` класса `QTextEdit`. Установить новый документ можно с помощью метода `setDocument(<QTextDocument>)`. Иерархия наследования:

`QObject` — `QTextDocument`

Конструктор класса `QTextDocument` имеет два формата:

```
<Объект> = QTextDocument([parent=<Родитель>])
<Объект> = QTextDocument(<Текст>[, parent=<Родитель>])
```

В параметре `parent` указывается ссылка на родительский компонент. Параметр `<Текст>` позволяет задать простой текст (не в HTML-формате), который будет отображен в текстовом поле.

Класс `QTextDocument` поддерживает следующий набор методов (полный их список смотрите на странице <https://doc.qt.io/qt-5/qtextdocument.html>):

- ◆ `setPlainText(<Текст>)` — помещает в документ простой текст;
- ◆ `setHtml(<Текст>)` — помещает в документ текст в формате HTML;
- ◆ `toPlainText()` — возвращает простой текст, содержащийся в документе;
- ◆ `toHtml([<QByteArray>])` — возвращает текст в формате HTML. В качестве параметра можно указать кодировку документа, которая будет выведена в теге `<meta>`;
- ◆ `clear()` — удаляет весь текст из документа;
- ◆ `isEmpty()` — возвращает значение `True`, если документ пустой, и `False` — в противном случае;
- ◆ `setModified(<Флаг>)` — если передано значение `True`, документ помечается как измененный, если `False` — как неизмененный. Метод является слотом;

- ◆ `isModified()` — возвращает значение `True`, если документ был изменен, и `False` — в противном случае;
- ◆ `undo()` — отменяет последнюю операцию ввода пользователем при условии, что отмена возможна. Метод является слотом;
- ◆ `redo()` — повторяет последнюю отмененную операцию ввода пользователем, если это возможно. Метод является слотом;
- ◆ `isUndoAvailable()` — возвращает значение `True`, если можно отменить последнюю операцию ввода, и `False` — в противном случае;
- ◆ `isRedoAvailable()` — возвращает значение `True`, если можно повторить последнюю отмененную операцию ввода, и `False` — в противном случае;
- ◆ `setUndoRedoEnabled(<Флаг>)` — если в качестве параметра указано значение `True`, то операции отмены и повтора действий разрешены, а если `False` — то запрещены;
- ◆ `isUndoRedoEnabled()` — возвращает значение `True`, если операции отмены и повтора действий разрешены, и `False` — если запрещены;
- ◆ `availableUndoSteps()` — возвращает количество возможных операций отмены;
- ◆ `availableRedoSteps()` — возвращает количество возможных повторов отмененных операций;
- ◆ `clearUndoRedoStacks([stacks=UndoAndredoStacks])` — очищает список возможных отмен и/или повторов. В качестве параметра можно указать следующие атрибуты класса `QTextDocument`:
 - `UndoStack` — только список возможных отмен;
 - `RedoStack` — только список возможных повторов;
 - `UndoAndredoStacks` — очищаются оба списка;
- ◆ `print(<QPagedPaintDevice>)` — отправляет содержимое документа на печать. В качестве параметра указывается экземпляр одного из классов, порожденных от `QPagedPaintDevice`: `QPrinter` или `QPdfWriter`;
- ◆ `find()` — производит поиск фрагмента в документе. Метод возвращает экземпляр класса `QTextCursor` из модуля `QtGui`. Если фрагмент не найден, то возвращенный экземпляр объекта будет нулевым. Проверить успешность операции можно с помощью метода `isNull()` класса `QTextCursor`. Форматы метода:

```
find(<Текст>[, position=0] [, options=0])
find(<QRegExp>[, position=0] [, options=0])
find(<Текст>, <QTextCursor>[, options=0])
find(<QRegExp>, <QTextCursor>[, options=0])
```

Параметр `<Текст>` задает искомый фрагмент, а параметр `<QRegExp>` позволяет указать регулярное выражение. По умолчанию обычный поиск производится без учета регистра символов в прямом направлении, начиная с позиции `position` или от текстового курсора, указанного в параметре `<QTextCursor>`. Поиск по регулярному выражению по умолчанию производится с учетом регистра символов. Чтобы поиск производился без учета регистра, необходимо передать атрибут `QtCore.Qt.CaseInsensitive` в метод `setCaseSensitivity()` регулярного выражения. В необязательном параметре `options` можно указать комбинацию (через оператор `|`) следующих атрибутов класса `QTextDocument`:

- `FindBackward` — 1 — поиск в обратном, а не в прямом направлении;
- `FindCaseSensitively` — 2 — поиск с учетом регистра символов. При использовании регулярного выражения значение игнорируется;
- `FindWholeWords` — 4 — поиск целых слов, а не фрагментов;

◆ `setFont (<QFont>)` — задает шрифт по умолчанию для документа. В качестве параметра указывается экземпляр класса `QFont` из модуля `QtGui`. Конструктор класса `QFont` имеет следующий формат:

```
<Шрифт> = QFont (<Название шрифта>[, pointSize=-1][, weight=-1]
[, italic=False])
```

В первом параметре указывается название шрифта в виде строки. Необязательный параметр `pointSize` задает размер шрифта. В параметре `weight` можно выставить степень жирности шрифта: число от 0 до 99 или значение атрибутов `Light`, `Normal`, `DemiBold`, `Bold` или `Black` класса `QFont`. Если в параметре `italic` указано значение `True`, шрифт будет курсивным;

- ◆ `setDefaultStyleSheet (<CSS>)` — устанавливает для документа таблицу стилей CSS по умолчанию;
- ◆ `setDocumentMargin (<Отступ>)` — задает отступ от краев поля до текста;
- ◆ `documentMargin ()` — возвращает величину отступа от краев поля до текста;
- ◆ `setMaximumBlockCount (<Количество>)` — задает максимальное количество текстовых блоков в документе. Если количество блоков становится больше указанного значения, первый блок будет удален;
- ◆ `maximumBlockCount ()` — возвращает максимальное количество текстовых блоков;
- ◆ `characterCount ()` — возвращает количество символов в документе;
- ◆ `lineCount ()` — возвращает количество абзацев в документе;
- ◆ `blockCount ()` — возвращает количество текстовых блоков в документе;
- ◆ `firstBlock ()` — возвращает экземпляр класса `QTextBlock`, объявленного в модуле `QtGui`, который содержит первый текстовый блок документа;
- ◆ `lastBlock ()` — возвращает экземпляр класса `QTextBlock`, который содержит последний текстовый блок документа;
- ◆ `findBlock (<Индекс символа>)` — возвращает экземпляр класса `QTextBlock`, который содержит текстовый блок документа, включающий символ с указанным индексом;
- ◆ `findBlockByLineNumber (<Индекс абзаца>)` — возвращает экземпляр класса `QTextBlock`, который содержит текстовый блок документа, включающий абзац с указанным индексом;
- ◆ `findBlockByNumber (<Индекс блока>)` — возвращает экземпляр класса `QTextBlock`, который содержит текстовый блок документа с указанным индексом.

Класс `QTextDocument` поддерживает сигналы:

- ◆ `undoAvailable (<Флаг>)` — генерируется при изменении возможности отменить операцию ввода. Значение параметра `True` обозначает наличие возможности отменить операцию ввода, а `False` — отсутствие такой возможности;
- ◆ `redoAvailable (<Флаг>)` — генерируется при изменении возможности повторить отмененную операцию ввода. Значение параметра `True` обозначает наличие возможности повторить отмененную операцию ввода, а `False` — отсутствие такой возможности;

- ◆ `undoCommandAdded` — генерируется при добавлении операции ввода в список возможных отмен;
- ◆ `blockCountChanged(<Новое количество блоков>)` — генерируется при изменении количества текстовых блоков. Внутри обработчика через параметр доступно новое количество текстовых блоков, заданное целым числом;
- ◆ `cursorPositionChanged(<QTextCursor>)` — генерируется при изменении позиции текстового курсора из-за операции редактирования. При простом перемещении текстового курсора сигнал не генерируется;
- ◆ `contentsChange(<Позиция курсора>, <Количество добавленных символов>, <Количество удаленных символов>)` — генерируется при изменении текста. Все три параметра целочисленные;
- ◆ `contentsChanged` — генерируется при любом изменении документа;
- ◆ `modificationChanged(<Флаг>)` — генерируется при изменении состояния документа: из неизмененного в измененное или наоборот. Значение параметра `True` обозначает, что документ помечен как измененный, значение `False` — что он теперь неизмененный.

21.6.5. Класс `QTextCursor`

Класс `QTextCursor` из модуля `QtGui` предоставляет инструмент для доступа к документу, представленному экземпляром класса `QTextDocument`, и для его правки, — иными словами, текстовый курсор. Конструктор класса `QTextCursor` поддерживает следующие форматы:

```
<Объект> = QTextCursor()
<Объект> = QTextCursor(<QTextDocument>)
<Объект> = QTextCursor(<QTextFrame>)
<Объект> = QTextCursor(<QTextBlock>)
<Объект> = QTextCursor(<QTextCursor>)
```

Создать текстовый курсор, установить его в документе и управлять им позволяют следующие методы класса `QTextEdit`:

- ◆ `textCursor()` — возвращает видимый в данный момент текстовый курсор (экземпляр класса `QTextCursor`). Чтобы изменения затронули текущий документ, необходимо передать этот объект в метод `setTextCursor()`;
- ◆ `setTextCursor(<QTextCursor>)` — устанавливает текстовый курсор, ссылка на который указана в качестве параметра;
- ◆ `cursorForPosition(<QPoint>)` — возвращает текстовый курсор, который соответствует позиции, указанной в качестве параметра. Позиция задается с помощью экземпляра класса `QPoint` в координатах области;
- ◆ `moveCursor(<Позиция>[, mode=MoveAnchor])` — перемещает текстовый курсор внутри документа. В первом параметре можно указать следующие атрибуты класса `QTextCursor`:
 - `NoMove` — 0 — не перемещать курсор;
 - `Start` — 1 — в начало документа;
 - `Up` — 2 — на одну строку вверх;
 - `StartOfLine` — 3 — в начало текущей строки;
 - `StartOfBlock` — 4 — в начало текущего текстового блока;

- StartOfWord — 5 — в начало текущего слова;
- PreviousBlock — 6 — в начало предыдущего текстового блока;
- PreviousCharacter — 7 — на предыдущий символ;
- PreviousWord — 8 — в начало предыдущего слова;
- Left — 9 — сдвинуть на один символ влево;
- WordLeft — 10 — влево на одно слово;
- End — 11 — в конец документа;
- Down — 12 — на одну строку вниз;
- EndOfLine — 13 — в конец текущей строки;
- EndOfWord — 14 — в конец текущего слова;
- EndOfBlock — 15 — в конец текущего текстового блока;
- NextBlock — 16 — в начало следующего текстового блока;
- NextCharacter — 17 — на следующий символ;
- NextWord — 18 — в начало следующего слова;
- Right — 19 — сдвинуть на один символ вправо;
- WordRight — 20 — в начало следующего слова.

Помимо указанных, существуют также атрибуты `NextCell`, `PreviousCell`, `NextRow` и `PreviousRow`, позволяющие перемещать текстовый курсор внутри таблицы. В необязательном параметре `mode` можно указать следующие атрибуты из класса `QTextCursor`:

- `MoveAnchor` — 0 — если существует выделенный фрагмент, выделение будет снято, и текстовый курсор переместится в новое место (значение по умолчанию);
- `KeepAnchor` — 1 — фрагмент текста от старой позиции курсора до новой будет выделен.

Класс `QTextCursor` поддерживает следующие методы (здесь приведены только основные — полный их список смотрите на странице <https://doc.qt.io/qt-5/qtextcursor.html>):

- ◆ `isNull()` — возвращает значение `True`, если объект курсора является нулевым (создан с помощью конструктора без параметра), и `False` — в противном случае;
- ◆ `setPosition(<Позиция> [, mode=MoveAnchor])` — перемещает текстовый курсор внутри документа. В первом параметре указывается позиция внутри документа. Необязательный параметр `mode` аналогичен одноименному параметру в методе `moveCursor()` класса `QTextEdit`;
- ◆ `movePosition(<Позиция> [, mode=MoveAnchor] [, n=1])` — перемещает текстовый курсор внутри документа. Параметры `<Позиция>` и `mode` аналогичны одноименным параметрам в методе `moveCursor()` класса `QTextEdit`. Необязательный параметр `n` позволяет указать количество перемещений — например, переместить курсор на 10 символов вперед можно так:

```
cur = textEdit.textCursor()
cur.movePosition(QtGui.QTextCursor.NextCharacter,
                 mode=QtGui.QTextCursor.MoveAnchor, n=10)
textEdit.setTextCursor(cur)
```

Метод movePosition() возвращает значение True, если операция успешно выполнена указанное количество раз. Если было выполнено меньшее количество перемещений (например, из-за достижения конца документа), метод возвращает значение False;

- ◆ position() — возвращает позицию текстового курсора внутри документа;
- ◆ positionInBlock() — возвращает позицию текстового курсора внутри блока;
- ◆ block() — возвращает экземпляр класса QTextBlock, который описывает текстовый блок, содержащий курсор;
- ◆ blockNumber() — возвращает номер текстового блока, содержащего курсор;
- ◆ atStart() — возвращает значение True, если текстовый курсор находится в начале документа, и False — в противном случае;
- ◆ atEnd() — возвращает значение True, если текстовый курсор находится в конце документа, и False — в противном случае;
- ◆ atBlockStart() — возвращает значение True, если текстовый курсор находится в начале блока, и False — в противном случае;
- ◆ atBlockEnd() — возвращает значение True, если текстовый курсор находится в конце блока, и False — в противном случае;
- ◆ select(<Режим>) — выделяет фрагмент в документе в соответствии с указанным режимом. В качестве параметра можно указать следующие атрибуты класса QTextCursor:
 - WordUnderCursor — 0 — выделяет слово, в котором расположен курсор;
 - LineUnderCursor — 1 — выделяет строку, в которой расположен курсор;
 - BlockUnderCursor — 2 — выделяет текстовый блок, в котором находится курсор;
 - Document — 3 — выделяет весь документ;
- ◆ hasSelection() — возвращает значение True, если существует выделенный фрагмент, и False — в противном случае;
- ◆ hasComplexSelection() — возвращает значение True, если выделенный фрагмент содержит сложное форматирование, а не просто текст, и False — в противном случае;
- ◆ clearSelection() — снимает выделение;
- ◆ selectionStart() — возвращает начальную позицию выделенного фрагмента;
- ◆ selectionEnd() — возвращает конечную позицию выделенного фрагмента;
- ◆ selectedText() — возвращает текст выделенного фрагмента;

ВНИМАНИЕ!

Если выделенный фрагмент занимает несколько строк, то вместо символа перевода строки вставляется символ с кодом \u2029. Попытка вывести этот символ в окно консоли приведет к исключению, поэтому следует произвести замену символа с помощью метода replace():

```
print(cur.selectedText().replace("\u2029", "\n"))
```

- ◆ selection() — возвращает экземпляр класса QTextDocumentFragment, который описывает выделенный фрагмент. Получить текст позволяют методы toPlainText() (возвращает простой текст) и toHtml() (возвращает текст в формате HTML) этого класса;
- ◆ removeSelectedText() — удаляет выделенный фрагмент;

- ◆ `deleteChar()` — если нет выделенного фрагмента, удаляет символ справа от курсора, в противном случае удаляет выделенный фрагмент;
- ◆ `deletePreviousChar()` — если нет выделенного фрагмента, удаляет символ слева от курсора, в противном случае удаляет выделенный фрагмент;
- ◆ `beginEditBlock()` и `endEditBlock()` — задают начало и конец блока инструкций. Эти инструкции могут быть отменены или повторены как единое целое с помощью методов `undo()` и `redo()`;
- ◆ `joinPreviousEditBlock()` — делает последующие инструкции частью предыдущего блока инструкций;
- ◆ `setKeepPositionOnInsert(<Флаг>)` — если в качестве параметра указано значение `True`, то после операции вставки курсор сохранит свою предыдущую позицию. По умолчанию позиция курсора при вставке изменяется;
- ◆ `insertText(<Текст>[, <QTextCharFormat>])` — вставляет простой текст;
- ◆ `insertHtml(<Текст>)` — вставляет текст в формате HTML.

С помощью методов `insertBlock()`, `insertFragment()`, `insertFrame()`, `insertImage()`, `insertList()` и `insertTable()` можно вставить различные элементы: изображения, списки и др. Изменить формат выделенного фрагмента позволяют методы `mergeBlockCharFormat()`, `mergeBlockFormat()` и `mergeCharFormat()`. За подробной информацией по этим методам обращайтесь к странице документации <https://doc.qt.io/qt-5/qtextcursor.html>.

21.7. Текстовый браузер

Класс `QTextBrowser` расширяет возможности класса `QTextEdit` и реализует текстовый браузер с возможностью перехода по гиперссылкам. Иерархия наследования выглядит так:

```
(QObject, QPaintDevice) – QWidget – QFrame –
QAbstractScrollArea – QTextEdit – QTextBrowser
```

Формат конструктора класса `QTextBrowser`:

```
<Объект> = QTextBrowser([parent=<Родитель>])
```

Класс `QTextBrowser` поддерживает следующие основные методы (полный их список смотрите на странице <https://doc.qt.io/qt-5/qtextbrowser.html>):

- ◆ `setSource(<QUrl>)` — загружает ресурс. В качестве параметра указывается экземпляр класса `QUrl` из модуля `QtCore`:

```
# Загружаем и выводим содержимое текстового файла
url = QtCore.QUrl("text.txt")
browser.setSource(url)
```

Метод является слотом;

- ◆ `source()` — возвращает экземпляр класса `QUrl` с адресом текущего ресурса;
- ◆ `reload()` — перезагружает текущий ресурс. Метод является слотом;
- ◆ `home()` — загружает первый ресурс из списка истории. Метод является слотом;
- ◆ `backward()` — загружает предыдущий ресурс из списка истории. Метод является слотом;
- ◆ `forward()` — загружает следующий ресурс из списка истории. Метод является слотом;

- ◆ backwardHistoryCount() — возвращает количество предыдущих ресурсов из списка истории;
- ◆ forwardHistoryCount() — возвращает количество следующих ресурсов из списка истории;
- ◆ isBackwardAvailable() — возвращает значение True, если существует предыдущий ресурс в списке истории, и False — в противном случае;
- ◆ isForwardAvailable() — возвращает значение True, если существует следующий ресурс в списке истории, и False — в противном случае;
- ◆ clearHistory() — очищает список истории;
- ◆ historyTitle(<Количество позиций>) — если в качестве параметра указано отрицательное число, возвращает заголовок предыдущего ресурса, отстоящего от текущего на заданное число позиций, если 0 — заголовок текущего ресурса, а если положительное число — заголовок следующего ресурса, также отстоящего от текущего на заданное число позиций;
- ◆ historyUrl(<Количество позиций>) — то же самое, что historyTitle(), но возвращает адрес ресурса в виде экземпляра класса QUrl;
- ◆ setOpenLinks(<Флаг>) — если в качестве параметра указано значение True, то автоматический переход по гиперссылкам будет разрешен (значение по умолчанию). Значение False запрещает переход.

Класс QTextBrowser поддерживает сигналы:

- ◆ anchorClicked(<QUrl>) — генерируется при переходе по гиперссылке. Внутри обработчика через параметр доступен адрес (URL) гиперссылки;
- ◆ backwardAvailable(<Признак>) — генерируется при изменении статуса списка предыдущих ресурсов. Внутри обработчика через параметр доступно значение True, если в списке истории имеются предыдущие ресурсы, и False — в противном случае;
- ◆ forwardAvailable(<Признак>) — генерируется при изменении статуса списка следующих ресурсов. В обработчике через параметр доступно значение True, если в списке истории имеются следующие ресурсы, и False — в противном случае;
- ◆ highlighted(<QUrl>) — генерируется при наведении указателя мыши на гиперссылку и выведении его. Внутри обработчика через параметр доступен адрес (URL) ссылки или пустой объект;
- ◆ highlighted(<Адрес>) — генерируется при наведении указателя мыши на гиперссылку и выведении его. Внутри обработчика через параметр доступен адрес (URL) ссылки в виде строки или пустая строка;
- ◆ historyChanged — генерируется при изменении списка истории;
- ◆ sourceChanged(<Адрес>) — генерируется при загрузке нового ресурса. Внутри обработчика через параметр доступен адрес (URL) загруженного ресурса.

21.8. Поля для ввода целых и вещественных чисел

Для ввода целых чисел предназначен класс QSpinBox, для ввода вещественных чисел — класс QDoubleSpinBox. Эти поля могут содержать две кнопки, которые позволяют щелчками

мыши увеличивать и уменьшать значение внутри поля. Пример такого поля ввода можно увидеть на рис. 21.1. Иерархия наследования:

```
(QObject, QPaintDevice) - QWidget - QAbstractSpinBox - QSpinBox
(QObject, QPaintDevice) - QWidget - QAbstractSpinBox - QDoubleSpinBox
```

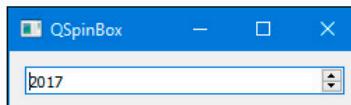


Рис. 21.1. Компонент QSpinBox

Форматы конструкторов классов QSpinBox и QDoubleSpinBox:

```
<Объект> - QSpinBox([parent-<Родитель>])
<Объект> - QDoubleSpinBox([parent-<Родитель>])
```

Классы QSpinBox и QDoubleSpinBox наследуют следующие методы из класса QAbstractSpinBox (здесь приведены только основные — полный их список смотрите на странице <https://doc.qt.io/qt-5/qabstractspinbox.html>):

- ◆ `setButtonSymbols(<Режим>)` — задает режим отображения кнопок, предназначенных для изменения значения поля с помощью мыши. Можно указать следующие атрибуты класса QAbstractSpinBox:
 - `UpDownArrows` — 0 — отображаются кнопки со стрелками;
 - `PlusMinus` — 1 — отображаются кнопки с символами + и -. Обратите внимание, что при использовании некоторых стилей это значение может быть проигнорировано;
 - `NoButtons` — 2 — кнопки не отображаются;
- ◆ `setAlignment(<Режим>)` — задает режим выравнивания значения внутри поля;
- ◆ `setWrapping(<Флаг>)` — если в качестве параметра указано значение `True`, то значение внутри поля будет при нажатии кнопок изменяться по кругу: максимальное значение сменится минимальным и наоборот;
- ◆ `setSpecialValueText(<Строка>)` — позволяет задать строку, которая будет отображаться внутри поля вместо минимального значения;
- ◆ `setReadOnly(<Флаг>)` — если в качестве параметра указано значение `True`, поле будет доступно только для чтения;
- ◆ `setFrame(<Флаг>)` — если в качестве параметра указано значение `False`, поле будет отображаться без рамки;
- ◆ `stepDown()` — уменьшает значение на одно приращение. Метод является слотом;
- ◆ `stepUp()` — увеличивает значение на одно приращение. Метод является слотом;
- ◆ `stepBy(<Количество>)` — увеличивает (при положительном значении) или уменьшает (при отрицательном значении) значение поля на указанное количество приращений;
- ◆ `text()` — возвращает текст, содержащийся внутри поля;
- ◆ `clear()` — очищает поле. Метод является слотом;
- ◆ `selectAll()` — выделяет все содержимое поля. Метод является слотом.

Класс QAbstractSpinBox поддерживает сигнал `editingFinished`, который генерируется при потере полем фокуса ввода или при нажатии клавиши <Enter>.

Классы QSpinBox и QDoubleSpinBox поддерживают следующие методы (здесь приведены только основные – полные их списки доступны на страницах <https://doc.qt.io/qt-5/qspinbox.html> и <https://doc.qt.io/qt-5/qdoublespinbox.html> соответственно):

- ◆ setValue(<Число>) задает значение поля. Метод является слотом, принимающим в зависимости от компонента, целое или вещественное значение;
- ◆ value() возвращает целое или вещественное число, содержащееся в поле;
- ◆ cleanText () возвращает целое или вещественное число в виде строки;
- ◆ setRange(<Минимум>, <Максимум>), setMinimum(<Минимум>) и setMaximum(<Максимум>) задают минимальное и максимальное допустимые значения;
- ◆ setPrefix(<Текст>) задает текст, который будет отображаться внутри поля перед значением;
- ◆ setSuffix(<Текст>) задает текст, который будет отображаться внутри поля после значения;
- ◆ setSingleStep(<Число>) задает число, которое будет прибавляться или вычитаться из текущего значения поля на каждом шаге.

Класс QDoubleSpinBox также поддерживает метод setDecimals(<Количество>), который задает количество цифр после десятичной точки.

Классы QSpinBox и QDoubleSpinBox поддерживают сигналы valueChanged(<Целое число>) (только в классе QSpinBox), valueChanged(<Вещественное число>) (только в классе QDoubleSpinBox) и valueChanged(<Строка>), которые генерируются при изменении значения внутри поля. Внутри обработчика через параметр доступно новое значение в виде числа или строки в зависимости от типа параметра.

21.9. Поля для ввода даты и времени

Для ввода даты и времени предназначены классы QDateTimeEdit (ввод даты и времени), QDateEdit (ввод даты) и QTimeEdit (ввод времени). Поля могут содержать кнопки, которые позволяют щелчками мыши увеличивать и уменьшать значение внутри поля. Пример такого поля показан на рис. 21.2.

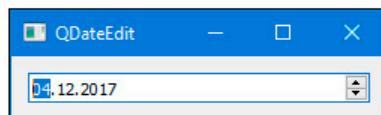


Рис. 21.2. Компонент QDateTimeEdit с кнопками-стрелками

Иерархия наследования:

```
(QObject, QPaintDevice) - QWidget - QAbstractSpinBox - QDateTimeEdit
(QObject, QPaintDevice) - QWidget - QAbstractSpinBox - QDateTimeEdit - QDateEdit
(QObject, QPaintDevice) - QWidget - QAbstractSpinBox - QDateTimeEdit - QTimeEdit
```

Форматы конструкторов классов:

```
<Объект> = QDateTimeEdit([parent=<Родитель>])
<Объект> = QDateTimeEdit(<QDateTime>[, parent=<Родитель>])
<Объект> = QDateTimeEdit(<QDate>[, parent=<Родитель>])
```

```
<Объект> = QDateTimeEdit(<QTime>[, parent=<Родитель>])
<Объект> = QDateEdit([parent=<Родитель>])
<Объект> = QDateEdit(<QDate>[, parent=<Родитель>])
<Объект> = QTimeEdit([parent=<Родитель>])
<Объект> = QTimeEdit(<QTime>[, parent=<Родитель>])
```

В параметре `<QDateTime>` можно указать экземпляр класса `QDateTime` или экземпляр класса `datetime` из языка Python. Преобразовать экземпляр класса `QDateTime` в экземпляр класса `datetime` позволяет метод `toPyDateTime()` класса `QDateTime`:

```
>>> from PyQt5 import QtCore
>>> d = QtCore.QDateTime()
>>> d
PyQt5.QtCore.QDateTime()
>>> d.toPyDateTime()
datetime.datetime(0, 0, 0, 255, 255, 255, 16776216)
```

В качестве параметра `<QDate>` можно указать экземпляр класса `QDate` или экземпляр класса `date` из языка Python. Преобразовать экземпляр класса `QDate` в экземпляр класса `date` позволяет метод `toPyDate()` класса `QDate`.

В параметре `<QTime>` можно указать экземпляр класса `QTime` или экземпляр класса `time` из языка Python. Преобразовать экземпляр класса `QTime` в экземпляр класса `time` позволяет метод `toPyTime()` класса `QTime`.

Классы `QDateTime`, `QDate` и `QTime` определены в модуле `QtCore`.

Класс `QDateTimeEdit` наследует все методы из класса `QAbstractSpinBox` (см. разд. 21.8) и дополнительно реализует следующие методы (здесь приведены только самые полезные — полный их список смотрите на странице <https://doc.qt.io/qt-5/qdatetimeedit.html>):

- ◆ `setDateTime(<QDateTime>)` — устанавливает дату и время. В качестве параметра указывается экземпляр класса `QDateTime` или экземпляр класса `datetime` из языка Python. Метод является слотом;
- ◆ `setDate(<QDate>)` — устанавливает дату. В качестве параметра указывается экземпляр класса `QDate` или экземпляр класса `date` из языка Python. Метод является слотом;
- ◆ `setTime(<QTime>)` — устанавливает время. В качестве параметра указывается экземпляр класса `QTime` или экземпляр класса `time` из языка Python. Метод является слотом;
- ◆ `dateTime()` — возвращает экземпляр класса `QDateTime` с датой и временем;
- ◆ `date()` — возвращает экземпляр класса `QDate` с датой;
- ◆ `time()` — возвращает экземпляр класса `QTime` со временем;
- ◆ `setDateTimeRange(<Минимум>, <Максимум>), setMinimumDateTime(<Минимум>)` и `setMaximumDateTime(<Максимум>)` — задают минимальное и максимальное допустимые значения для даты и времени. В параметрах указывается экземпляр класса `QDateTime` или экземпляр класса `datetime` из языка Python;
- ◆ `setDateRange(<Минимум>, <Максимум>), setMinimumDate(<Минимум>)` и `setMaximumDate(<Максимум>)` — задают минимальное и максимальное допустимые значения для даты. В параметрах указывается экземпляр класса `QDate` или экземпляр класса `date` из языка Python;
- ◆ `setTimeRange(<Минимум>, <Максимум>), setMinimumTime(<Минимум>)` и `setMaximumTime(<Максимум>)` — задают минимальное и максимальное допустимые значения для времени.

ни. В параметрах указывается экземпляр класса `QTime` или экземпляр класса `time` из языка Python;

- ◆ `setDisplayFormat(<Формат>)` задает формат отображения даты и времени. В качестве параметра указывается строка, содержащая специальные символы. Пример задания строки формата:

```
dateTimeEdit.setDisplayFormat("dd.MM.yyyy HH:mm:ss")
```

- ◆ `setTimeSpec(<Зона>)` задает зону времени. В качестве параметра можно указать атрибуты `LocalTime`, `UTC` или `OffsetFromUTC` класса `QtCore.Qt`;

- ◆ `setCalendarPopup(<Флаг>)` если в качестве параметра указано значение `True`, то дату можно будет выбрать с помощью календаря, который появится на экране при щелчке на кнопке с направленной вниз стрелкой, выведенной вместо стандартных кнопок-стрелок (рис. 21.3);

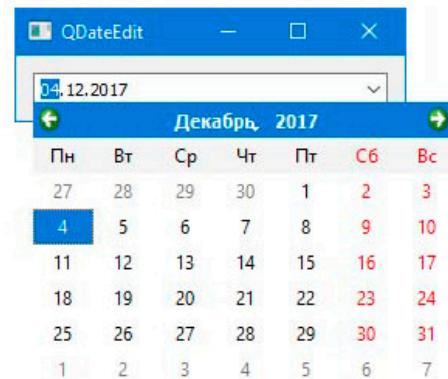


Рис. 21.3. Компонент `QDateEdit` с открытым календарем

- ◆ `setSelectedSection(<Секция>)` выделяет указанную секцию. В качестве параметра можно задать атрибуты `NoSection`, `DaySection`, `MonthSection`, `YearSection`, `HourSection`, `MinuteSection`, `SecondSection`, `MSecSection` или `AmPmSection` класса `QDateTimeEdit`;
- ◆ `setCurrentSection(<Секция>)` делает указанную секцию текущей;
- ◆ `setCurrentSectionIndex(<Индекс>)` делает секцию с указанным индексом текущей;
- ◆ `currentSection()` возвращает тип текущей секции;
- ◆ `currentSectionIndex()` возвращает индекс текущей секции;
- ◆ `sectionCount()` возвращает количество секций внутри поля;
- ◆ `sectionAt(<Индекс>)` возвращает тип секции по указанному индексу;
- ◆ `sectionText(<Секция>)` возвращает текст указанной секции.

При изменении значений даты или времени генерируются сигналы `timeChanged(<QTime>)`, `dateChanged(<QDate>)` и `dateTimeChanged(<QDateTime>)`. Внутри обработчиков через параметр доступно новое значение.

Классы `QDateEdit` (поле для ввода даты) и `QTimeEdit` (поле для ввода времени) созданы для удобства и отличаются от класса `QDateTimeEdit` только форматом отображаемых данных. Эти классы наследуют методы базовых классов и не добавляют никаких своих методов.

21.10. Календарь

Класс `QCalendarWidget` реализует календарь с возможностью выбора даты и перемещения по месяцам с помощью мыши и клавиатуры (рис. 21.4). Иерархия наследования:

(`QObject`, `QPaintDevice`) — `QWidget` — `QCalendarWidget`

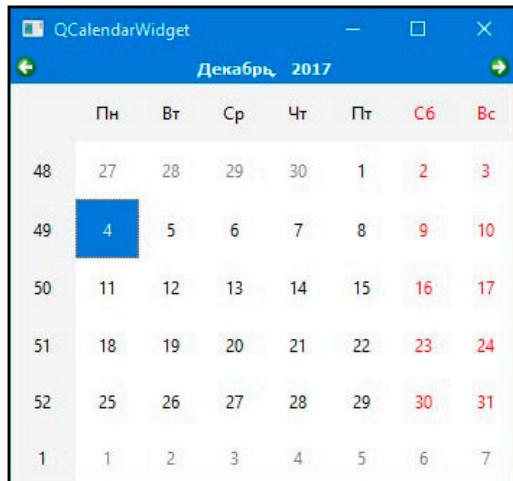


Рис. 21.4. Компонент `QCalendarWidget`

Формат конструктора класса `QCalendarWidget`:

<Объект> — `QCalendarWidget` ([`parent`=<Родитель>])

Класс `QCalendarWidget` поддерживает следующие методы (здесь представлены только основные — полный их список смотрите на странице <https://doc.qt.io/qt-5/qcalendarwidget.html>):

- ◆ `setSelectedDate(<QDate>)` — устанавливает дату, заданную в качестве параметра экземпляром класса `QDate` или экземпляром класса `date` языка Python. Метод является слотом;
- ◆ `selectedDate()` — возвращает экземпляр класса `QDate` с выбранной датой;
- ◆ `setDateRange(<Минимум>, <Максимум>), setMinimumDate(<Минимум>) и setMaximumDate(<Максимум>)` — задают минимальное и максимальное допустимые значения для даты. В параметрах указывается экземпляр класса `QDate` или экземпляр класса `date` из языка Python. Метод `setDateRange()` является слотом;
- ◆ `setCurrentPage(<Год>, <Месяц>)` — делает текущей страницу календаря с указанными годом и месяцем, которые задаются целыми числами. Выбранная дата при этом не изменяется. Метод является слотом;
- ◆ `monthShown()` — возвращает месяц (число от 1 до 12), отображаемый на текущей странице;
- ◆ `yearShown()` — возвращает год, отображаемый на текущей странице;
- ◆ `showSelectedDate()` — отображает страницу с выбранной датой. Выбранная дата при этом не изменяется. Метод является слотом;
- ◆ `showToday()` — отображает страницу с сегодняшней датой. Выбранная дата при этом не изменяется. Метод является слотом;

- ◆ `showPreviousMonth()` — отображает страницу с предыдущим месяцем. Выбранная дата при этом не изменяется. Метод является слотом;
- ◆ `showNextMonth()` — отображает страницу со следующим месяцем. Выбранная дата при этом не изменяется. Метод является слотом;
- ◆ `showPreviousYear()` — отображает страницу с текущим месяцем в предыдущем году. Выбранная дата не изменяется. Метод является слотом;
- ◆ `showNextYear()` — отображает страницу с текущим месяцем в следующем году. Выбранная дата при этом не изменяется. Метод является слотом;
- ◆ `setFirstDayOfWeek(<День>)` — задает первый день недели. По умолчанию используется воскресенье. Чтобы первым днем недели сделать понедельник, следует в качестве параметра указать атрибут `Monday` класса `QtCore.Qt`;
- ◆ `setNavigationBarVisible(<Флаг>)` — если в качестве параметра указано значение `False`, то панель навигации выводиться не будет. Метод является слотом;
- ◆ `setHorizontalHeaderFormat(<Формат>)` — задает формат горизонтального заголовка. В качестве параметра можно указать следующие атрибуты класса `QCalendarWidget`:
 - `NoHorizontalHeader` — 0 — заголовок не отображается;
 - `SingleLetterDayNames` — 1 — отображается только первая буква из названия дня недели;
 - `ShortDayNames` — 2 — отображается сокращенное название дня недели;
 - `LongDayNames` — 3 — отображается полное название дня недели;
- ◆ `setVerticalHeaderFormat(<Формат>)` — задает формат вертикального заголовка. В качестве параметра можно указать следующие атрибуты класса `QCalendarWidget`:
 - `NoVerticalHeader` — 0 — заголовок не отображается;
 - `ISOWeekNumbers` — 1 — отображается номер недели в году;
- ◆ `setGridVisible(<Флаг>)` — если в качестве параметра указано значение `True`, линии сетки будут отображены. Метод является слотом;
- ◆ `setSelectionMode(<Режим>)` — задает режим выделения даты. В качестве параметра можно указать следующие атрибуты класса `QCalendarWidget`:
 - `NoSelection` — 0 — дата не может быть выбрана пользователем;
 - `SingleSelection` — 1 — может быть выбрана одна дата;
- ◆ `setHeaderTextFormat(<QTextCharFormat>)` — задает формат ячеек заголовка. В параметре указывается экземпляр класса `QTextCharFormat` из модуля `QtGui`;
- ◆ `setWeekdayTextFormat(<День недели>, <QTextCharFormat>)` — определяет формат ячеек для указанного дня недели. В первом параметре задаются атрибуты `Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday`, `Saturday` или `Sunday` класса `QtCore.Qt`, а во втором параметре — экземпляр класса `QTextCharFormat`;
- ◆ `setDateTextFormat(<QDate>, <QTextCharFormat>)` — задает формат ячейки с указанной датой. В первом параметре указывается экземпляр класса `QDate` или экземпляр класса `date` из языка Python, а во втором параметре — экземпляр класса `QTextCharFormat`.

Класс `QCalendarWidget` поддерживает такие сигналы:

- ◆ `activated(<QDate>)` — генерируется при двойном щелчке мышью или нажатии клавиши `<Enter>`. Внутри обработчика через параметр доступна текущая дата;

- ◆ `clicked(<QDate>)` генерируется при щелчке мышью на доступной дате. Внутри обработчика через параметр доступна выбранная дата;
- ◆ `currentPageChanged(<Год>, <Месяц>)` генерируется при изменении страницы. Внутри обработчика через первый параметр доступен год, а через второй — месяц. Обе величины задаются целыми числами;
- ◆ `selectionChanged` генерируется при изменении выбранной даты пользователем или из программного кода.

21.11. Электронный индикатор

Класс `QLCDNumber` реализует электронный индикатор, в котором цифры и буквы отображаются отдельными сегментами как на электронных часах или дисплее калькулятора (рис. 21.5). Индикатор позволяет отображать числа в двоичной, восьмеричной, десятичной и шестнадцатеричной системах счисления. Иерархия наследования выглядит так:

`(QObject, QPaintDevice) – QWidget – QFrame – QLCDNumber`



Рис. 21.5. Компонент `QLCDNumber`

Форматы конструктора класса `QLCDNumber`:

```
<Объект> = QLCDNumber ([parent=<Родитель>])
<Объект> = QLCDNumber (<Количество цифр>, parent=<Родитель>)
```

В параметре `<Количество цифр>` указывается количество отображаемых цифр — если оно не указано, используется значение 5.

Класс `QLCDNumber` поддерживает следующие методы (здесь приведены только основные полный их список смотрите на странице <https://doc.qt.io/qt-5/qlcdnumber.html>):

- ◆ `display(<Значение>)` задает новое значение. В качестве параметра можно указать целое, вещественное число или строку:

```
lcd.display(1048576)
```

Метод является слотом;

- ◆ `checkOverflow(<Число>)` возвращает значение `True`, если целое или вещественное число, указанное в параметре, не может быть отображено индикатором. В противном случае возвращает значение `False`;
- ◆ `intValue()` возвращает значение индикатора в виде целого числа;
- ◆ `value()` возвращает значение индикатора в виде вещественного числа;
- ◆ `setSegmentStyle(<Стиль>)` задает стиль индикатора. В качестве параметра можно указать атрибуты `Outline`, `Filled` или `Flat` класса `QLCDNumber`;
- ◆ `setMode(<Режим>)` задает режим отображения чисел. В качестве параметра можно указать следующие атрибуты класса `QLCDNumber`:

- Hex — 0 — шестнадцатеричное значение;
- Dec — 1 — десятичное значение;
- Oct — 2 — восьмеричное значение;
- Bin — 3 — двоичное значение.

Вместо метода `setMode()` удобнее воспользоваться методами-слотами `setHexMode()`, `setDecMode()`, `setOctMode()` и `setBinMode()`:

- ◆ `setSmallDecimalPoint(<Флаг>)` — если в качестве параметра указано значение `True`, десятичная точка будет отображаться как отдельный элемент (при этом значение выводится более компактно без пробелов до и после точки), а если значение `False` — то десятичная точка будет занимать позицию цифры (значение используется по умолчанию). Метод является слотом;
- ◆ `setDigitCount(<Число>)` — задает количество отображаемых цифр. Если в методе `setSmallDecimalPoint()` указано значение `False`, десятичная точка считается отдельной цифрой.

Класс `QLCDNumber` поддерживает сигнал `overflow`, генерируемый при попытке задать значение, которое не может быть отображено индикатором.

21.12. Индикатор хода процесса

Класс `QProgressBar` реализует индикатор хода процесса, с помощью которого можно информировать пользователя о текущем состоянии выполнения длительной операции. Иерархия наследования выглядит так:

(`QObject`, `QPaintDevice`) — `QWidget` — `QProgressBar`

Формат конструктора класса `QProgressBar`:

`<Объект> = QProgressBar([parent=<Родитель>])`

Класс `QProgressBar` поддерживает следующий набор методов, которые могут быть нам полезны (полный их список смотрите на странице <https://doc.qt.io/qt-5/qprogressbar.html>):

- ◆ `setValue(<Значение>)` — задает новое целочисленное значение. Метод является слотом;
- ◆ `value()` — возвращает текущее значение индикатора в виде числа;
- ◆ `text()` — возвращает текст, отображаемый на индикаторе или рядом с ним;
- ◆ `setRange(<Минимум>, <Максимум>)`, `setMinimum(<Минимум>)` и `setMaximum(<Максимум>)` — задают минимальное и максимальное значения в виде целых чисел. Если оба значения равны нулю, то внутри индикатора будут постоянно по кругу перемещаться сегменты, показывая ход выполнения процесса с неопределенным количеством шагов. Методы являются слотами;
- ◆ `reset()` — сбрасывает значение индикатора. Метод является слотом;
- ◆ `setOrientation(<Ориентация>)` — задает ориентацию индикатора. В качестве значения указываются атрибуты `Horizontal` или `Vertical` класса `QtCore.Qt`. Метод является слотом;
- ◆ `setTextVisible(<Флаг>)` — если в качестве параметра указано значение `False`, текст с текущим значением индикатора отображаться не будет;
- ◆ `setTextDirection(<Направление>)` — задает направление вывода текста при вертикальной ориентации индикатора. Обратите внимание, что при использовании стилей

"windows", "windowsxp" и "macintosh" при вертикальной ориентации текст вообще не отображается. В качестве значения указываются следующие атрибуты класса QProgressBar:

- TopToBottom — 0 — текст поворачивается на 90 градусов по часовой стрелке;
 - BottomToTop — 1 — текст поворачивается на 90 градусов против часовой стрелки;
- ◆ setInvertedAppearance(<Флаг>) — если в качестве параметра указано значение True, направление увеличения значения будет изменено на противоположное (например, не слева направо, а справа налево — при горизонтальной ориентации);
- ◆ setFormat(<Формат>) — задает формат вывода текстового представления значения. Параметром передается строка формата, в которой могут использоваться следующие специальные символы: %v — само текущее значение, %m — заданный методами setMaximum() или setRange() максимум, %p — текущее значение в процентах:

```
lcd.setFormat('Выполнено %v шагов из %m')
```

При изменении значения индикатора генерируется сигнал valueChanged(<Новое значение>). Внутри обработчика через параметр доступно новое значение, заданное целым числом.

21.13. Шкала с ползунком

Класс QSlider реализует шкалу с ползунком, который можно перемещать с помощью мыши или клавиатуры. Иерархия наследования выглядит так:

```
(QObject, QPaintDevice) – QWidget – QAbstractSlider – QSlider
```

Форматы конструктора класса QSlider:

```
<Объект> = QSlider( [parent=<Родитель>])  
<Объект> = QSlider(<Ориентация>, [ parent=<Родитель>])
```

Параметр <Ориентация> позволяет задать ориентацию шкалы. В качестве значения указываются атрибуты Horizontal или Vertical (значение по умолчанию) класса QtCore.Qt.

Класс QSlider наследует следующие методы из класса QAbstractSlider (здесь приведены только основные — полный их список смотрите на странице <https://doc.qt.io/qt-5/qabstractslider.html>):

- ◆ setValue(<Значение>) — задает новое целочисленное значение. Метод является слотом;
- ◆ value() — возвращает текущее значение в виде числа;
- ◆ setSliderPosition(<Значение>) — задает текущее положение ползунка;
- ◆ sliderPosition() — возвращает текущее положение ползунка в виде числа. Если отслеживание перемещения ползунка включено (принято по умолчанию), то возвращаемое значение будет совпадать со значением, возвращаемым методом value(). Если отслеживание выключено, то при перемещении метод sliderPosition() вернет текущее положение, а метод value() — положение, которое имел ползунок до перемещения;
- ◆ setRange(<Минимум>, <Максимум>), setMinimum(<Минимум>) и setMaximum(<Максимум>) — задают минимальное и максимальное значения, представленные целыми числами. Метод setRange() является слотом;
- ◆ setOrientation(<Ориентация>) — задает ориентацию шкалы. В качестве значения указываются атрибуты Horizontal или Vertical класса QtCore.Qt. Метод является слотом;

- ◆ `setSingleStep(<Значение>)` — задает значение, на которое сдвинется ползунок при нажатии клавиш со стрелками;
- ◆ `setPageStep(<Значение>)` — задает значение, на которое сдвинется ползунок при нажатии клавиш `<Page Up>` и `<Page Down>`, повороте колесика мыши или щелчке мышью на шкале;
- ◆ `setInvertedAppearance(<Флаг>)` — если в качестве параметра указано значение `True`, направление увеличения значения будет изменено на противоположное (например, не слева направо, а справа налево — при горизонтальной ориентации);
- ◆ `setInvertedControls(<Флаг>)` — если в качестве параметра указано значение `False`, то при изменении направления увеличения значения будет изменено и направление перемещения ползунка при нажатии клавиш `<Page Up>` и `<Page Down>`, повороте колесика мыши и нажатии клавиш со стрелками вверх и вниз;
- ◆ `setTracking(<Флаг>)` — если в качестве параметра указано значение `True`, отслеживание перемещения ползунка будет включено (принято по умолчанию). При этом сигнал `valueChanged` при перемещении ползунка станет генерироваться постоянно. Если в качестве параметра указано значение `False`, то сигнал `valueChanged` будет генерирован только при отпускании ползунка;
- ◆ `hasTracking()` — возвращает значение `True`, если отслеживание перемещения ползунка включено, и `False` — в противном случае.

Класс `QAbstractSlider` поддерживает сигналы:

- ◆ `actionTriggered(<Действие>)` — генерируется, когда производится взаимодействие с ползунком (например, при нажатии клавиши `<Page Up>`). Внутри обработчика через параметр доступно произведенное действие, которое описывается целым числом. Также можно использовать атрибуты `SliderNoAction` (0), `SliderSingleStepAdd` (1), `SliderSingleStepSub` (2), `SliderPageStepAdd` (3), `SliderPageStepSub` (4), `SliderToMinimum` (5), `SliderToMaximum` (6) и `SliderMove` (7) класса `QAbstractSlider`;
- ◆ `rangeChanged(<Минимум>, <Максимум>)` — генерируется при изменении диапазона значений. Внутри обработчика через параметры доступны новые минимальное и максимальное значения, заданные целыми числами;
- ◆ `sliderPressed` — генерируется при нажатии ползунка;
- ◆ `sliderMoved(<Положение>)` — генерируется постоянно при перемещении ползунка. Внутри обработчика через параметр доступно новое положение ползунка, выраженное целым числом;
- ◆ `sliderReleased` — генерируется при отпусканье ранее нажатого ползунка;
- ◆ `valueChanged(<Значение>)` — генерируется при изменении значения. Внутри обработчика через параметр доступно новое значение в виде целого числа.

Класс `QSlider` дополнительно определяет следующие методы (здесь приведены только основные — полный их список смотрите на странице <https://doc.qt.io/qt-5/qslider.html>):

- ◆ `setTickPosition(<Позиция>)` — задает позицию рисок. В качестве параметра указываются следующие атрибуты класса `QSlider`:
 - `NoTicks` — без рисок;
 - `TicksBothSides` — риски по обе стороны;
 - `TicksAbove` — риски выводятся сверху;

- TicksBelow — риски выводятся снизу;
 - TicksLeft — риски выводятся слева;
 - TicksRight — риски выводятся справа;
- ◆ setTickInterval (<Расстояние>) — задает расстояние между рисками.

21.14. Круговая шкала с ползунком

Класс `QDial` реализует круглую шкалу с ползунком, который можно перемещать по кругу с помощью мыши или клавиатуры. Компонент, показанный на рис. 21.6, напоминает регулятор, используемый в различных устройствах для изменения или отображения каких-либо настроек. Иерархия наследования:

(`QObject`, `QPaintDevice`) — `QWidget` — `QAbstractSlider` — `QDial`

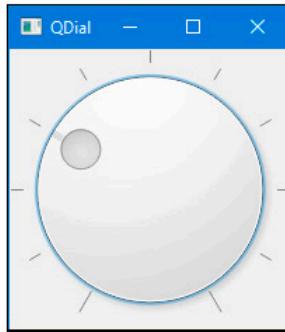


Рис. 21.6. Компонент `QDial`

Формат конструктора класса `QDial`:

<Объект> — `QDial` ([`parent`=<Родитель>])

Класс `QDial` наследует все методы и сигналы класса `QAbstractSlider` (см. разд. 21.13) и определяет несколько дополнительных методов (здесь приведена только часть методов — полный их список смотрите на странице <https://doc.qt.io/qt-5/qdial.html>):

- ◆ `setNotchesVisible (<Флаг>)` — если в качестве параметра указано значение `True`, будут отображены риски. По умолчанию риски не выводятся. Метод является слотом;
- ◆ `setNotchTarget (<Значение>)` — задает рекомендуемое количество пикселов между рисками. В качестве параметра указывается вещественное число;
- ◆ `setWrapping (<Флаг>)` — если в качестве параметра указано значение `True`, то начало шкалы будет совпадать с ее концом. По умолчанию между началом шкалы и концом расположено пустое пространство. Метод является слотом.

21.15. Полоса прокрутки

Класс `QScrollBar` представляет горизонтальную или вертикальную полосу прокрутки. Изменить положение ползунка на полосе можно нажатием на кнопки, расположенные по краям полосы, щелчками мышью на полосе, собственно перемещением ползунка мышью,

нажатием клавиш на клавиатуре, а также выбрав соответствующий пункт из контекстного меню. Иерархия наследования:

(QObject, QPaintDevice) – QWidget – QAbstractSlider – QScrollBar

Форматы конструктора класса QScrollBar:

<Объект> = QScrollBar([parent=<Родитель>])

<Объект> = QScrollBar(<Ориентация>[, parent=<Родитель>])

Параметр <Ориентация> позволяет задать ориентацию полосы прокрутки. В качестве значения указываются атрибуты `Horizontal` или `Vertical` (значение по умолчанию) класса `QtCore.Qt`.

Класс `QScrollBar` наследует все методы и сигналы класса `QAbstractSlider` (см. разд. 21.13) и не определяет дополнительных методов.

ПРИМЕЧАНИЕ

Полоса прокрутки редко используется отдельно. Гораздо удобнее воспользоваться областью с полосами прокрутки, которую реализует класс `QScrollArea` (см. разд. 20.12).

21.16. Веб-браузер

Класс `QWebEngineView`, определенный в модуле `QtWebEngineWidgets`, реализует полнофункциональный веб-браузер, поддерживающий HTML, CSS, JavaScript, вывод изображений и пр. (на рис. 21.7 в этом компоненте выведена главная страница Google). Иерархия наследования:

(QObject, QPaintDevice) – QWidget – QWebEngineView

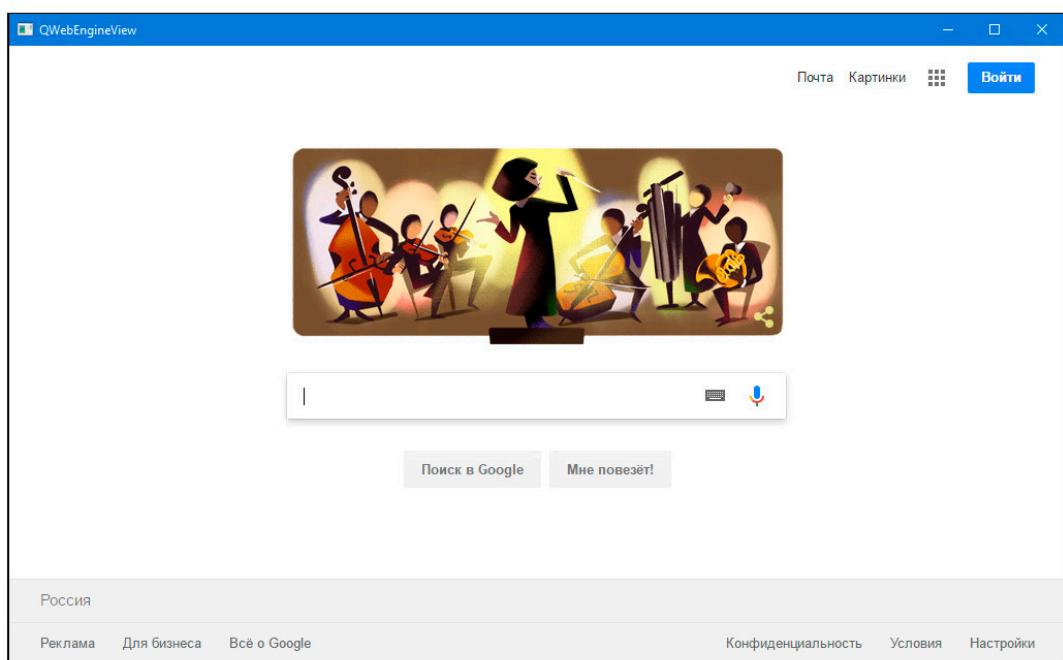


Рис. 21.7. Компонент QWebEngineView

Формат конструктора класса `QWebEngineView`:

```
<Объект> = QWebEngineView( [parent=<Родитель>] )
```

ПРИМЕЧАНИЕ

До версии 5.5 библиотеки PyQt вывод веб-страниц осуществлялся средствами класса `QWebView` из модуля `QtWebKitWidgets`. Однако в версии 5.5 этот модуль был объявлен не рекомендованным к использованию, а в версии 5.6 — удален.

Класс `QWebEngineView` поддерживает следующие полезные для нас методы (полный их список смотрите на странице <https://doc.qt.io/qt-5/qwebengineview.html>):

- ◆ `load(<QUrl>)` и `setUrl(<QUrl>)` — загружают и выводят страницу с указанным в параметре адресом, который задается в виде экземпляра класса `QUrl` из модуля `QtCore`:
`webView.load(QtCore.QUrl('https://www.google.ru/'))`
- ◆ `url()` — возвращает адрес текущей страницы в виде экземпляра класса `QUrl`;
- ◆ `title()` — возвращает заголовок (содержимое тега `<title>`) текущей страницы;
- ◆ `setHtml(<HTML-код>[, baseUrl=QUrl()])` — задает HTML-код страницы, которая будет отображена в компоненте.

Необязательный параметр `baseUrl` указывает базовый адрес, относительно которого будут отсчитываться относительные адреса в гиперссылках, ссылках на файлы изображений, таблицы стилей, файлы сценариев и пр. Если этот параметр не указан, в качестве значения по умолчанию используется «пустой» экземпляр класса `QUrl`, и относительные адреса будут отсчитываться от каталога, где находится сам файл страницы:

```
webView.setHtml("<h1>Заголовок</h1>")  
# файл page2.html будет загружен с сайта http://www.somesite.ru/  
webView.setHtml("<a href='page2.html'>Вторая страница</a>", QtCore.QUrl('http://www.somesite.ru/'))
```

- ◆ `selectedText()` — возвращает выделенный текст или пустую строку, если ничего не было выделено;
- ◆ `hasSelection()` — возвращает `True`, если фрагмент страницы был выделен, и `False` — в противном случае;
- ◆ `setZoomFactor(<Множитель>)` — задает масштаб самой страницы. Значение 1 указывает, что страница будет выведена в оригинальном масштабе, значение меньше единицы — в уменьшенном, значение больше единицы — увеличенном масштабе;
- ◆ `zoomFactor()` — возвращает масштаб страницы;
- ◆ `back()` — загружает предыдущий ресурс из списка истории. Метод является слотом;
- ◆ `forward()` — загружает следующий ресурс из списка истории. Метод является слотом;
- ◆ `reload()` — перезагружает страницу. Метод является слотом;
- ◆ `stop()` — останавливает загрузку страницы. Метод является слотом;
- ◆ `icon()` — возвращает в виде экземпляра класса `QIcon` значок, заданный для страницы;
- ◆ `findText(<Искомый текст>[, options=QWebEnginePage.FindFlags()] [, resultCallback=0])` — позволяет найти на странице заданный фрагмент текста. Все найденные фрагменты будут выделены желтым фоном непосредственно в компоненте `QWebEngineView`.

Необязательный параметр `option` позволяет указать дополнительные параметры в виде экземпляра класса `QWebEnginePage.FindFlags` из того же модуля `QtWebEngineWidgets`. В конструкторе этого класса можно указать один из атрибутов класса `QWebEnginePage` из модуля `QtWebEngineWidgets` или их комбинации через оператор `|`:

- `FindBackward` — 1 — выполнять поиск в обратном, а не в прямом направлении;
- `FindCaseSensitively` — 2 — поиск с учетом регистра символов (по умолчанию выполняется поиск без учета регистра).

В необязательном параметре `resultCallback` можно указать функцию, которая будет вызвана по окончании поиска. Эта функция должна принимать единственный параметр, которым станет логическая величина `True`, если поиск увенчался успехом, или `False` — в противном случае.

Вот пример поиска с учетом регистра и выделением всех найденных совпадений:

```
web.findText('Python', options=QtWebEngineWidgets.QWebEnginePage.FindFlags(  
    QtWebEngineWidgets.QWebEnginePage.FindBackward |  
    QtWebEngineWidgets.QWebEnginePage.FindCaseSensitively))
```

◆ `triggerPageAction(<Действие> [, checked=False])` — выполняет над страницей указанное действие. В качестве действия задается один из атрибутов класса `QWebEnginePage` — этих атрибутов очень много, и все они приведены на странице <https://doc.qt.io/qt-5/qwebenginepage.html#WebAction-enum>. Необязательный параметр `checked` имеет смысл указывать лишь для действий, принимающих логический флаг:

```
# Выделение всей страницы  
web.triggerPageAction(QtWebEngineWidgets.QWebEnginePage.SelectAll)  
# Копирование выделенного фрагмента страницы  
web.triggerPageAction(QtWebEngineWidgets.QWebEnginePage.Copy)  
# Перезагрузка страницы, минуя кэш  
web.triggerPageAction(QtWebEngineWidgets.QWebEnginePage.ReloadAndBypassCache)
```

◆ `page()` — возвращает экземпляр класса `QWebEnginePage` из модуля `QtWebEngineWidgets`, представляющий открытую веб-страницу.

Класс `QWebView` поддерживает следующий набор полезных для нас сигналов (полный их список смотрите на странице <https://doc.qt.io/qt-5/qwebengineview.html>):

- ◆ `iconChanged` — генерируется после загрузки или изменения значка, заданного для страницы;
- ◆ `loadFinished(<Признак>)` — генерируется по окончании загрузки страницы. Значение `True` параметра указывает, что загрузка выполнена без проблем, `False` — что при загрузке произошли ошибки;
- ◆ `loadProgress(<Процент выполнения>)` — периодически генерируется в процессе загрузки страницы. В качестве параметра передается целое число от 0 до 100, показывающее процент загрузки;
- ◆ `loadStarted` — генерируется после начала загрузки страницы;
- ◆ `selectionChanged` — генерируется при выделении нового фрагмента содержимого страницы;
- ◆ `titleChanged(<Текст>)` — генерируется при изменении текста заголовка страницы (содержимого тега `<title>`). В параметре, передаваемом обработчику, доступен этот текст в виде строки;

- ♦ `urlChanged(<QUrl>)` — генерируется при изменении адреса текущей страницы, что может быть вызвано, например, загрузкой новой страницы. Параметр — новый адрес.

Класс `QWebEnginePage`, представляющий открытую в веб-браузере страницу и получаемый вызовом метода `page()` класса `QWebEngineView`, поддерживает следующие полезные для нас методы (полный их список смотрите на странице <https://doc.qt.io/qt-5/qwebenginepage.html>):

- ♦ `print(<QPrinter>, <Функция>)` — печатает содержимое страницы на заданном принтере. Вторым параметром указывается функция, которая будет вызвана после окончания печати и должна принимать единственный параметр, которым будет значение `True`, если печать завершилась успешно, и `False` — в противном случае. Метод поддерживается PyQt, начиная с версии 5.8;

- ♦ `printToPdf()` — преобразует страницу в формат PDF. Форматы метода:

```
printToPdf(<Путь файла>,
            pageLayout=QPageLayout(QPageSize(QPageSize::A4),
            QPageLayout::Portrait, QMarginsF()))
printToPdf(<Функция>,
            pageLayout=QPageLayout(QPageSize(QPageSize::A4),
            QPageLayout::Portrait, QMarginsF()))
```

Первый формат сразу сохраняет преобразованную страницу в файле, чей путь указан первым параметром. Второй формат после преобразования вызывает указанную в первом параметре функцию, передавая ей в качестве единственного параметра массив типа `bytes`, хранящий преобразованную страницу.

Необязательный параметр `pageLayout` задает настройки страницы в виде экземпляра класса `QPageLayout` (он будет описан в главе 29). Если параметр не указан, будет выполнена печать на бумаге типоразмера A4, в портретной ориентации, без отступов.

Метод поддерживается PyQt, начиная с версии 5.7;

- ♦ `save(<Путь файла>, format=QWebEngineDownloadItem::MimeHtmlSaveFormat)` — сохраняет страницу в файле, чей путь указан в первом параметре. Необязательный параметр `format` задает формат файла — для него можно задать один из следующих атрибутов класса `QWebEngineDownloadItem`, определенного в модуле `QtWebEngineWidgets`:

- `SingleHtmlSaveFormat` — 0 — обычный HTML-файл. Связанные со страницей файлы (изображения, аудио- и видеоролики, таблицы стилей и пр.) не сохраняются;
- `CompleteHtmlSaveFormat` — 1 — то же самое, только связанные файлы будут сохранены в каталоге, находящемся там же, где и файл со страницей, и имеющим то же имя;
- `MimeHtmlSaveFormat` — 2 — страница и все связанные файлы сохраняются в одном файле.

Метод поддерживается PyQt, начиная с версии 5.8;

- ♦ `contentsSize()` — возвращает экземпляр класса `QSizeF`, хранящий размеры содержимого страницы. Метод поддерживается PyQt, начиная с версии 5.7;
- ♦ `scrollPosition()` — возвращает экземпляр класса `QPointF`, хранящий позицию прокрутки содержимого страницы. Метод поддерживается PyQt, начиная с версии 5.7;
- ♦ `setAudioMuted(<Флаг>)` — если с параметром передать значение `True`, все звуки, воспроизводящиеся на странице, будут приглушены. Чтобы снова сделать их слышимыми, нужно передать значение `False`. Метод поддерживается PyQt, начиная с версии 5.7;

- ◆ `isAudioMuted()` — возвращает `True`, если все звуки, воспроизводящиеся на странице, приглушены, и `False` — в противном случае. Метод поддерживается PyQt, начиная с версии 5.7;
- ◆ `setBackgroundColor(<Цвет>)` — задает для страницы фоновый цвет, указанный в виде экземпляра класса `QColor`. Метод поддерживается PyQt, начиная с версии 5.6;
- ◆ `backgroundColor()` — возвращает фоновый цвет страницы в виде экземпляра класса `QColor`. Метод поддерживается PyQt, начиная с версии 5.6.