Глава 14

Работа с графическими файлами,  
рисование тонким пером:  
проект PNGEDIT1

Проект PNGEDIT1 является первым в серии проектов, посвященных работе с графической библиоте­кой, используемой в Qt-приложениях. В этом проекте рассматриваются основные действия с графиче­скими файлами (создание, сохранение и загрузка) и описываются особенности основных графических классов: QPixmap и QPainter. В проекте также реализуется отслеживание текущих координат изобра­жения, рисование тонким пером (с использованием класса QPen) и очистка изображения.

14.1. Создание, сохранение и загрузка графических файлов

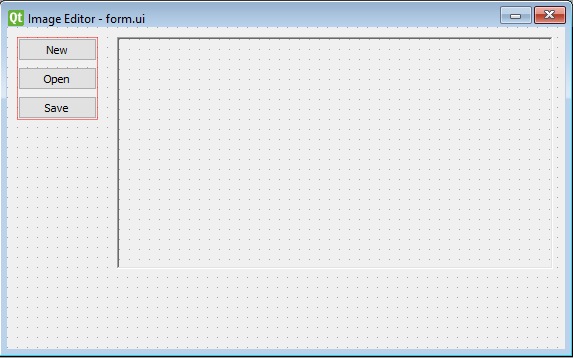
После создания формы типа **Widget** добавьте в нее три кнопки (pushButton-pushButton\_3) и прокру­чиваемую панель QScrollArea (scrollArea), скомпонуйте кнопки по вертикали. Настройте свойства формы Form и всех добавленных компонентов (листинг 14.1) и расположите компоненты в соответ­ствии с рис. 14.1.

Добавьте новую форму типа **Dialog with Button Bottom** (дайте ей имя Form2) и разместите в ней две метки (label и label\_2) и два компонента типа QSpinBox (spinBox и spinBox\_2). Скомпонуйте новые компоненты в два столбца (добавится лейаут formLayout). Автоматически будет создан компонент buttonBox типа QDialogButtonBox, содержащий две кнопки: OK и Cancel. Настройте свойства формы Form2 и ее компонентов (листинг 14.2) и расположите компоненты в соответствии с рис. 14.2. С помо­щью вкладки **Редактор сигналов и слотов** соедините сигналы accepted и rejected виджета buttonBox со слотами accept и reject формы соответственно.

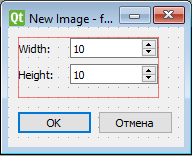
Создайте файлы для обеих форм и выполните их конвертацию в py-файлы с помощью утилиты pyuic5. В описание класса Form добавьте подключение класса Form2 и поле

**self.form2 = Form2(self)**

В конструктор класса Form добавьте новые операторы (листинг 14.3) и определите слоты для сигнала clicked кнопок pushButton, pushButton\_2 и pushButton\_3, размещенных в форме Form (листинг 14.4).



**Рис. 14.1.** Вид формы Form для проекта PNGEDIT1 на начальном этапе разработки



**Рис. 14.2.** Вид формы Form2 для проекта PNGEDIT1

Добавьте в конструктор класса Form2 операторы:

**self.setWindowFlags(Qt.Dialog | Qt.WindowCloseButtonHint)**

**self.buttonBox.accepted.connect(self.new)**

**Листинг 14.1. Настройка свойств формы Form и ее компонентов**

Form: windowTitle = **Image Editor**

pushButton: text = **&New**

pushButton\_2: text = **&Open**

pushButton\_3: text = **&Save**

scrollArea: frameShape = **WinPanel**

**Листинг 14.2. Настройка свойств формы Form2 и ее компонентов**

Form2: windowTitle = **New** **Image**

formLayout: layoutHorizontalSpacing = **16**

label: text = **Width:**

label\_2: text = **Height:**

spinBox: minimum = **10**, maximum = **800**, singleStep = **10**

spinBox\_2: minimum = **10**, maximum = **600**, singleStep = **10**

**Листинг 14.3. Новый вариант конструктора формы Form**

def \_\_init\_\_(self):

super(self.\_\_class\_\_, self).\_\_init\_\_()

self.setupUi(self)

**self.setFixedSize(self.size())**

**self.label = QLabel()**

**self.label.setAlignment(Qt.AlignTop)**

**self.scrollArea.setWidget(self.label)**

self.form2 = Form2(self)

**self.form2.spinBox.setValue(self.scrollArea.width())**

**self.form2.spinBox\_2.setValue(self.scrollArea.height())**

**self.pushButton.clicked.connect(self.new)**

**self.pushButton\_2.clicked.connect(self.open)**

**self.pushButton\_3.clicked.connect(self.save)**

**Листинг 14.4. Слоты new, open и save**

**def new(self):**

**self.form2.spinBox.setFocus(True)**

**if self.form2.exec\_() == 1:**

**self.setWindowTitle('Image Editor')**

**w = self.form2.spinBox.value() - 4**

**h = self.form2.spinBox\_2.value() - 4**

**pix = QPixmap(w, h)**

**pix.fill(Qt.white)**

**self.label.setPixmap(pix)**

**def open(self):**

**s = QFileDialog.getOpenFileName(self, 'Открытие', '',**

**'Image files (\*.bmp \*.jpg \*.png)')[0]**

**if s:**

**self.label.setPixmap(QPixmap(s, '1'))**

**self.setWindowTitle('Image Editor - ' + s)**

**def save(self):**

**s = QFileDialog.getSaveFileName(self, 'Сохранение', '',**

**'PNG-files (\*.png)')[0]**

**if s:**

**self.label.pixmap().save(s)**

**self.setWindowTitle('Image Editor - ' + s)**

**Результат.** В редактор можно загружать графические файлы формата BMP, JPG, PNG, GIF любого размера (кнопка **Open**). Если размер файла превосходит размер выделенной для его отображения па­нели scrollArea, то панель снабжается полосами прокрутки, позволяющими перемещаться к любому фрагменту изображения. При загрузке файла его полное имя отображается в заголовке окна. Загру­женный файл можно сохранить под новым именем в формате PNG (кнопка **Save**). В качестве каталога для загруженных и сохраняемых файлов по умолчанию предлагается рабочий каталог приложения. Кроме того, редактор позволяет создавать новые изображения (кнопка **New**). Размер нового изображе­ния (в пикселах) запрашивается в диалоговом окне **New Image** и может изменяться от 10 × 10 до 800 × 600. При первом отображении диалогового окна предлагаются размеры, обеспечивающие заполне­ние всей отображаемой части панели scrollArea; в дальнейшем в окне сохраняются размеры, указан­ные пользователем. Созданное изображение автоматически закрашивается белым цветом.

См. также комментарии 1—4.

**Недочет.** При запуске программы изображение в редакторе отсутствует.

**Исправление.** Чтобы не дублировать код, отвечающий за создание нового изображения (см. слот new в листинге 14.4), "делегируем" задачу создания нового изображения форме form2. Для этого опреде­лим обработчик слот new для компонента buttonBox, размещенной в форме Form2 (листинг 14.5), и изменим слот new формы Form (листинг 14.6; в тексте метода следует удалить несколько операторов, ничего не добавляя). Кроме того, в конструктор класса Form добавим новый оператор:

**self.form2.new()**

В конструктор класса Form2 же нужно добавить оператор соединения:

**self.buttonBox.accepted.connect(self.new)**

**Листинг 14.5. Слот new (для формы Form2)**

**def new(self):**

**w = self.spinBox.value() - 4**

**h = self.spinBox\_2.value() - 4**

**pix = QPixmap(w, h)**

**pix.fill(Qt.white)**

**self.parent().label.setPixmap(pix)**

**Листинг 14.6. Новый вариант слота new формы Form**

def new(self):

self.form2.spinBox.setFocus(True)

if self.form2.exec\_() == 1:

self.setWindowTitle('Image Editor')

**Результат.** Теперь при запуске программы в ней автоматически создается новое изображение, раз­меры которого совпадают с размерами клиентской части панели, содержащей данное изображение.

Комментарии

1. При вызове статической функции getOpenFileName (аналогично и для getSaveFileName) класса QFileDialog для определения рабочего каталога приложения достаточно указать пустую строку. Для указанной конкретной начальной папки необходимо указывать каталог явно.

Перечислим другие параметры данного метода: первый – родительский компонент для QFileDialog (в данном случае – сама форма), второй – заголовок вызываемого диалогового окна, четвертый – список расширений файлов, которые доступны диалогу.

1. Так как в компоненте scrollArea используется граница типа WinPanel, необходимо дважды вы­честь ее ширину (равную 2) из получаемых от счетчиков значений. Компонент-метка label созда­ется программно, так как при добавлении через Qt Designer метка не будет основным виджетом в scrollArea и прокрутка выполняться не будет.
2. Компонент QSpinBox, называемый *счетчиком с прокруткой* (spin control), позволяет задавать числа из определенного диапазона, используя как явный ввод с клавиатуры, так и вспомогательные кнопки со стрелками для увеличения/уменьшения значения счетчика на фиксированную величину. Диапазон допустимых значений определяют свойства minimum и maximum (значения по умолчанию **0** и **99** соответственно), шаг изменения счетчика — свойство singleStep (значение по умолчанию **1**), а текущее значение счетчика — свойство value (значение по умолчанию **0**). Кроме того, суще­ствует аналогичный компонент для вещественных чисел – QDoubleSpinBox.
3. Следует четко понимать роль объектов, участвующих в выводе и обработке изображения. Для ви­зуализации изображений с возможностью прокрутки удобнее всего использовать комбинацию компонентов QScrollArea и QLabel. Рисование будет происходить в поле метки pixmap типа QPixmap с помощью графического класса QPainter. Для вывода графической информации в Qt су­ществует специальный виджет QGraphicsView, однако реализация прокрутки и рисования некото­рых графических объектов в данном компоненте слишком сложна.

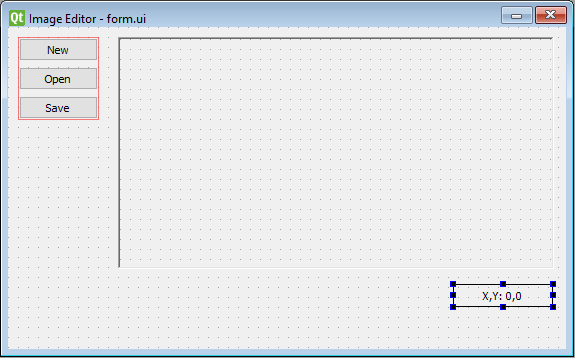
Графический класс QPixmap имеет несколько конструкторов. В данной главе рассмотрены два: в первом класс инициализируется длиной и шириной изображения (см. слот new), во втором происходит загрузка содержимого из файла (см. слот open): в этом случае, помимо имени, необходимо указать параметр '1' – формат изображения.

1. В аналогичных проектах (например, для библиотеки GDI+) была возможность работать с GIF-анимацией. В Qt GIF-файлы доступны только для чтения, поэтому данное расширение не исполь­зуется.

14.2. Отслеживание текущих координат изображения

Разместите в форме Form метку label\_2 и настройте её свойства (листинг 14.7) и положение (рис. 14.3; ширина метки должна быть достаточной для того, чтобы отобразить трехзначные координаты вместе с комментарием **X,Y:**).

Создайте класс ImageLabel (наследник класса QLabel) в файле imagelabel.py (листинг 14.8). Замените в конструкторе формы Form для метки label тип на ImageLabel, а также соедините сигнал mouseMoved компонента label со слотом mouseMove формы Form (листинг 14.9).



**Рис. 14.3.** Вид формы Form для проекта PNGEDIT1 на промежуточном этапе разработки

**Листинг 14.7. Настройка свойств компонента label\_2 формы Form**

label\_2: text = **X,Y: 0,0**, frameShape = **Box**, alignment.Horizontal = **AlignHCenter**

**Листинг 14.8. Класс ImageLabel**

**from PyQt5.QtCore import \***

**from PyQt5.QtGui import \***

**from PyQt5.QtWidgets import \***

**class ImageLabel(QLabel):**

**mouseMoved = pyqtSignal(QMouseEvent)**

**def mouseMoveEvent(self, event):**

**self.mouseMoved.emit(event)**

**Листинг 14.9. Слот mouseMove**

**def mouseMove(self, event):**

**self.label\_2.setText('X,Y: {},{}'.format(event.x(), event.y()))**

**Результат.** При перемещении мыши над изображением (т. е. над компонентом label) в метке label\_2 отображаются координаты текущего пиксела изображения.

***ПРИМЕЧАНИЕ***

Следует подчеркнуть, что выводятся именно координаты текущей точки изображения, а не позиции, отсчитываемой от левого верхнего угла компонента scrollArea. Различие между этими величинами проявляется при прокрутке изображения, имеющего размеры, которые превышают размеры панели: в подобной ситуации левый верхний угол панели будет содержать точку изображения с координатами, отличными от (0, 0).

14.3. Рисование тонким пером

Добавьте в класс Form описания новых полей:

**pen = QPen(Qt.black)**

**startPt = QPoint()**

Определите сигнал mousePressed для класса ImageLabel, добавьте обработчик для события mousePressEvent (листинг 14.10), определите слот mousePress и дополните слот mouseMove формы Form (листинг 14.11).

**Листинг 14.10. Обработчик события *mousePressEvent* класса ImageLabel**

**def mousePressEvent(self, event):**

**self.mousePressed.emit(event)**

**Листинг 14.11. Слоты mouseMove и mousePress**

def mouseMove(self, event):

self.label\_2.setText('X,Y: {},{}'.format(event.x(), event.y()))

**if event.buttons() == Qt.LeftButton:**

**painter = QPainter(self.label.pixmap())**

**painter.setPen(self.pen)**

**painter.drawLine(self.startPt, event.pos())**

**self.startPt = event.pos()**

**self.label.repaint()**

**def mousePress(self, event):**

**self.startPt = event.pos()**

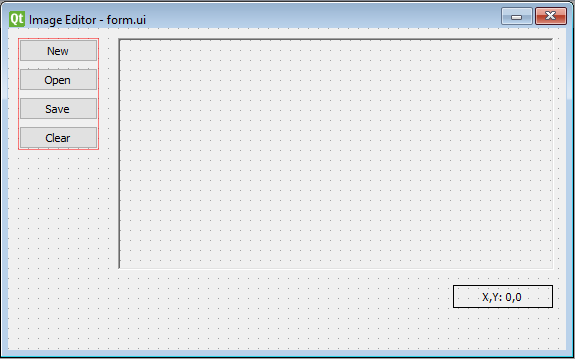
**Результат.** На изображении можно рисовать линии черного цвета толщиной 1 пиксел. Для этого надо переместить курсор на начальную позицию, нажать левую кнопку мыши и, не отпуская ее, нарисовать нужную линию. Рисование возможно как на созданных, так и на загруженных изображениях. См. также комментарии 1—3.

Комментарии

1. В новом варианте слота mouseMove (см. листинг 14.11) использован метод drawLine графической канвы (т. е. объекта типа QPainter). При перемещении мыши с зажатой левой клавишей над меткой label происходит захват объектом класса QPainter поля pixmap метки. Для рисования линии следует указать *перо*, которым она будет рисоваться, а также ее начальную и конечную точку. Поле pen инициализируется стандартным черным пером толщины 1, которое можно получить с помощью класса QPen, содержащего уже созданные перья различных цветов. В дальнейшем в редакторе будут предусмотрены средства для изменения свойств текущего пера.
2. Поле startPt типа QPoint используется для хранения информации о предыдущей позиции мыши; значение этого поля необходимо обновлять после выполнения каждой операции рисования (см. последний оператор в листинге 14.11).
3. Для того чтобы результат применения операций, связанных с рисованием, отобразился на экране, необходимо вызвать метод repaint компонента label (см. последний оператор в методе mouseMove в листинге 14.11). Название метода связано с тем, что он вынуждает операционную систему перерисовать объект.

14.4. Очистка изображения

Разместите в менеджере компоновки формы Form еще одну кнопку (pushButton\_4), положите ее свойство text равным **&Clear** (рис 14.4) и соедините ее сигнал clicked со слотом clear (листинг 14.12).



**Рис. 14.4.** Окончательный вид формы Form для проекта PNGEDIT1

**Листинг 14.12. Слоты clear**

**def clear(self):**

**pix = self.label.pixmap()**

**pix.fill(Qt.white)**

**self.label.setPixmap(pix)**

**Результат.** При нажатии кнопки **Clear** изображение очищается, т. е. закрашивается белым цветом.