Глава 5

Формы: проект FORMS

Проект FORMS знакомит с особенностями приложений, использующих несколько форм, и демонстрирует различные способы настройки внешнего вида форм и режимы их отображения на экране. Рассматриваются вопросы взаимодействия форм в рамках одного приложения и, в частности, проблемы, связанные с закрытием немодальных подчиненных форм. Описываются настройки для диалоговых окон и методы, обеспечивающие вывод на экран стандартных диалогов.

5.1. Настройка визуальных свойств форм. Открытие форм в обычном и модальном режиме

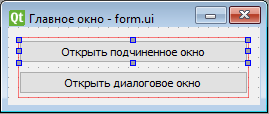
После создания проекта FORMS (форма типа **Widget**) добавьте к нему две новые формы типа **Dialog without Buttons** (дайте имена Form2 и Form3). Разместите в форме Form две кнопки pushButton и pushButton\_2, скомпонуйте их по вертикали. Настройте свойства всех форм и компонентов (листинг 5.1, рис. 5.1 – 5.3). Подключите в файле form.h файлы form2.h и form3.h. Добавьте в класс в раздел private описания двух полей:

Form2 \*form2 = new Form2(this);

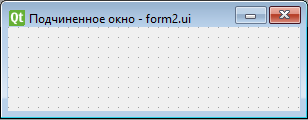
Form3 \*form3 = new Form3(this);

Дополните конструктор класса Form (листинг 5.2) и определите обработчики сигнала clicked для кнопок pushButton и pushButton\_2 (листинг 5.3). Добавьте в конструктор класса Form3 оператор

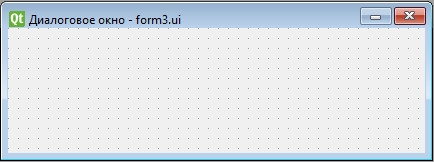
setFixedSize(size());



**Рис. 5.1.** Окончательный вид формы Form для проекта FORMS



**Рис. 5.2.** Вид формы Form2 для проекта FORMS на начальном этапе разработки



**Рис. 5.3.** Вид формы Form3 для проекта FORMS на начальном этапе разработки

**Листинг 5.1. Настройка свойств**

Form: windowTitle = **Главное окно**

pushButton: text = Открыть подчиненное окно

pushButton\_2: text = Открыть диалоговое окно

Form2: windowTitle = **Подчиненное окно**

Form3: windowTitle = **Диалоговое окно**, modal = **True**

**Листинг 5.2. Новый вариант конструктора формы Form**

Form::**Form**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::Form)

{

ui->setupUi(this);

**setFixedSize(size());**

**form2->setWindowFlags(Qt::Dialog | Qt::WindowMinMaxButtonsHint | Qt::WindowCloseButtonHint);**

**form3->setWindowFlags(Qt::Dialog | Qt::WindowCloseButtonHint);**

}

**Листинг 5.3. Обработчики on\_pushButton\_clicked и on\_pushButton\_2\_clicked**

void Form::on\_pushButton\_clicked()

{

**form2->move(geometry().right() - 10, geometry().bottom() - 10);**

**form2->show();**

}

void Form::on\_pushButton\_2\_clicked()

{

**form3->show();**

}

**Результат.** Программа содержит три формы, демонстрирующие основные типы окон в графических приложениях: окно фиксированного размера (класс Form), окно переменного размера (класс Form2), диалоговое окно (класс Form3). Форма Form содержит две кнопки (см. рис. 5.1); формы Form2 и Form3 пока не содержат компонентов. Форма Form является главной; она автоматически создается при запуске приложения и сразу отображается на экране. Кроме того, она создает две формы с именами form2 и form3 — экземпляры классов Form2 и Form3 соответственно.

Форма form2 (подчиненная форма) вызывается из главной формы нажатием кнопки **Открыть подчиненное окно**; при этом она отображается в обычном режиме (методом show). Форма form3 также является подчиненной; она вызывается нажатием кнопки **Открыть диалоговое окно** и отображается в *модальном (диалоговом)* режиме. Особенность модального режима состоит в том, что если некоторая форма приложения находится в этом режиме, то до ее закрытия нельзя переключаться на другие формы приложения (хотя возможно переключение на другие запущенные приложения). Для завершения программы надо закрыть ее главную форму.

Главная форма Form имеет фиксированные размеры. Размеры подчиненной формы form2 можно изменять; кроме того, форму form2 можно разворачивать на весь экран. Визуальные свойства формы form3 соответствуют стандартным свойствам диалогового окна: размеры формы form3 нельзя изменять и, кроме того, в ее заголовке отображается только текст и кнопка закрытия (см. рис. 5.3).

При открытии Form по умолчанию отображается в центре, форма form2 отображается около правого нижнего угла формы Form с небольшим наложением, форма form3 также отображается в центре. См. также комментарии 1-3.

Комментарии

1. Явное указание в конструкторе форм Form2 и Form3 формы Form позволяет закрывать вызванные окна при закрытии главного окна. В противном случае (вызов без параметров) сложилась бы ситуация, при которой, несмотря на закрытие главное окна, подчиненное или диалоговое окна оставались бы открытыми. Подчиненная форма всегда отображается поверх главной, даже если главная форма является активной. Кроме того, при минимизации главной формы ее подчиненные формы также минимизируются, а также кнопки для подчиненных форм не отображаются на панели задач в нижней части экрана.
2. По умолчанию для формы типа QDialog в заголовке отображаются иконка, название формы, кнопки вызова подсказки и закрытия окна. В данном проекте вызов подсказок не требуется, поэтому выполняется замена флагов окна: для подчиненного окна это комбинация флагов максимизации/минимизации и закрытия окна, для диалогового окна (имеет фиксированный размер) достаточно флага закрытия окна.
3. Наиболее простым способом создания фиксированного размера окна является вызов функции setFixedSize, параметром которой служит предполагаемый размер типа QSize. Постоянный размер окна будет регулярно использоваться нами в других проектах, в которых вызов этой функции будет называться *оператором фиксации размера*.

5.2. Контроль за состоянием подчиненной формы. Воздействие подчиненной формы на главную

Измените метод on\_pushButton\_clicked (листинг 5.4), добавьте в файл form.h описание слота **void setPushButtonText(bool visible);**, определите его в класса form.cpp (листинг 5.5). Добавьте в класс Form2 (form2.h) следующие операторы

**protected:**

**void *showEvent*(QShowEvent\*);**

**void *closeEvent*(QCloseEvent\*);**

**signals:**

**void visibleChanged(bool visible);**

Определите события showEvent и closeEvent (листинг 5.6). Соедините в конструкторе формы Form сигнал visibleChanged(bool) формы form2 со слотом setPushButtonText(bool).

**Листинг 5.4. Новый вариант метода on\_pushButton\_clicked**

void Form::on\_pushButton\_clicked()

{

form2->move(geometry().right() - 10, geometry().bottom() - 10);

**if (form2->isVisible())**

**form2->close();**

**else**

**form2->show();**

}

**Листинг 5.5. Слот setPushButtonText**

**void Form::setPushButtonText(bool visible)**

**{**

**ui->pushButton->setText(visible ? "Открыть подчиненное окно" :**

**"Закрыть подчиненное окно");**

**}**

**Листинг 5.6. Cобытия showEvent и** **closeEvent**

**void Form2::*showEvent*(QShowEvent\*)**

**{**

**emit visibleChanged(false);**

**}**

**void Form2::*closeEvent*(QCloseEvent\*)**

**{**

**emit visibleChanged(true);**

**}**

**Результат.** Теперь текст кнопки pushButton и действия при ее нажатии зависят от того, отображается на экране подчиненное окно form2 или нет: если подчиненное окно присутствует на экране, то оно исчезает, а если его на экране нет, то оно появляется. Подчиненное окно можно закрыть не только с помощью кнопки pushButton, но и любым стандартным способом (например, с помощью комбинации клавиш <Alt>+<F4>); при любом способе закрытия подчиненного окна текст кнопки pushButton будет изменен.

Комментарии

1. Для того, чтобы подчиненная форма могла обратиться к элементам главной формы и наоборот, можно было бы сделать объекты классов публичными полями. Однако лучше использовать механизм сигналов и слотов: при возникновании событий открытия/закрытия окна происходит вызов сигнала visibleChanged c соответствующим флагом. Так как сигнал всегда является доступным, то его можно соединить с необходимым слотом в главной форме.
2. В методе setPushButtonText (см. листинг 5.5) использована тернарная операция

*условие ? выражение1 : выражение2*

Если условие истинно, то вычисляется и возвращается *выражение1*, а если *условие* ложно, то вычисляется и возвращается *выражение2*. Подчеркнем, что при выполнении тернарной операции вычисляется только то выражение, значение которого будет возвращено. Мы использовали тернарную операцию, так как она приводит к более компактному коду, чем эквивалентный ей вариант с полным условным оператором if–then–else:

if (visible)

ui->pushButton->setText("Открыть подчиненное окно");

else

ui->pushButton->setText("Закрыть подчиненное окно");

1. Так как в обработчиках событий не требуется информация об их свойствах, достаточно указать в параметрах функции указатель на событие без имени.

5.3. Компоненты, подстраивающиеся под размер окна

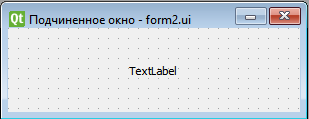
Разместите в форме Form2 компонент-метку типа QLabel (метка получит имя label) и настройте ее свойство alignment.Horizontal равным **AlignHCenter**. Щелкните правой кнопкой мыши на форме в свободной от метки месте и выберите пункт **Компоновка**, а далее **Скомпоновать по сетке**.

В секции private класса Form2 (form2.h) добавьте описание поля count:

**int count;**

Заметим, что при создании формы ее поле count будет автоматически инициализировано нулевым значением.

В метод showEvent добавьте новые операторы (листинг 5.7).



**Рис. 5.4.** Окончательный вид формы Form для проекта FORMS

**Листинг 5.7. Новый варианта обработчик showEvent**

void Form2::*showEvent*(QShowEvent\*)

{

**ui->label->setText(QString("Окно открыто в %1-й раз.").arg(++count));**

emit visibleChanged(false);

}

**Результат.** При изменении размеров подчиненного окна Form2 размеры находящейся на нем метки label изменяются так, чтобы она занимала всю внутреннюю *(клиентскую)* часть окна. Текст метки содержит информацию о том, сколько раз было открыто подчиненное окно.

Комментарий

При использовании *операции инкремента* вида ++i (префиксный вариант операции) вначале происходит увеличение значения переменной i на 1, а затем данная переменная используется в выражении. Для постфиксной операции i++ действия выполняются в обратном порядке: вначале прежнее значение i используется в выражении, а затем это значение увеличивается на 1. Аналогичным образом ведут себя префиксный и постфиксный варианты операции декремента --.