Министерство образования и науки Российской Федерации
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

РАЗРАБОТКА ПРИЛОЖЕНИЙ НА C# С ИСПОЛЬЗОВАНИЕМ СУБД PostgreSQL

Утверждено Редакционно-издательским советом университета
в качестве учебного пособия

НОВОСИБИРСК
2015

УДК 004.65:004.43(075.8)
Р 177

Коллектив авторов:

И.А. Васюткина, *Г.В. Трошина*,
М.И. Бычков, *С.А. Менжулин*

Рецензенты:

д-р техн. наук, профессор *В.И. Гужов*
канд. техн. наук, доцент *С.П. Ильиных*

Работа подготовлена на кафедре вычислительной техники
для студентов, обучающихся по направлениям 09.03.01 и 09.03.04

Р 177 **Разработка приложений на C# с использованием СУБД PostgreSQL**: учебное пособие / И.А. Васюткина, Г.В. Трошина, М.И. Бычков, С.А. Менжулин. – Новосибирск: Изд-во НГТУ, 2015. – 143 с.

ISBN 978-5-7782-2699-9

Представлены основные приемы по использованию языка C# и баз данных PostgreSQL при создании информационных систем различного назначения. Учебное пособие предназначено для студентов, обучающихся по направлениям 09.03.01 – «Информатика и вычислительная техника», 09.03.04 – «Программная инженерия».

УДК 004.65:004.43(075.8)

ISBN 978-5-7782-2699-9

ПРЕДИСЛОВИЕ

Настоящее пособие содержит изложение основных вопросов по использованию языка C# и СУБД PostgreSQL при создании информационных систем (приложений) различного назначения.

Пособие рассчитано на студентов всех форм обучения направлений «Информатика и вычислительная техника» и «Программная инженерия», знакомых с языком запросов SQL и основами языка C#.

Цель написания данного учебного пособия – дать единую теоретическую базу, необходимую для выполнения курсовых, контрольных и лабораторных работ по курсам «Информационные системы», «Базы данных», «Безопасность баз данных», «Технология программирования».

Предлагаемое учебное пособие интегрирует и логически увязывает теоретический материал нескольких курсов и позволяет получить необходимый опыт практической разработки приложений.

Каждый из четырех разделов учебного пособия включает в себя примеры, упражнения, контрольные вопросы, необходимые для более глубокого понимания излагаемого материала и получения практических навыков его использования.

Основное внимание уделено вопросам построения баз данных, разработки графического интерфейса пользователя (GUI) в Microsoft Visual Studio.Net Windows Forms на языке C# и языковым средствам доступа к данным с использованием СУБД PostgerSQL, защите баз данных.

Учебное пособие может быть использовано при выполнении курсовых, контрольных и лабораторных работ по всем указанным дисциплинам.

1. ЭЛЕМЕНТЫ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

1.1. Особенности формирования пользовательского интерфейса

Пользовательский интерфейс – это средство взаимодействия пользователя с приложением. От того, как он разработан, будет зависеть время, которое требуется для его освоения, и эффективность работы с приложением. Правильно разработанный интерфейс должен учитывать ряд основных моментов:

1) учет потребностей пользователей приложения;

2) учет логики работы приложения и последовательность выполняемых действий;

3) оптимизация размещения элементов управления;

4) простота и привлекательность внешнего вида.

Требования, которые предъявляются к интерфейсам программных продуктов, разнообразны и зависят от конкретных задач. Однако в целом можно выявить много общих требований, которые необходимо определить еще до начала разработки интерфейса пользователя. Очень часто это игнорируют, и в результате интерфейсы строятся без учета закономерностей мышления и поведения человека.

Создание хороших интерфейсов требует от разработчика объемной и напряженной работы. При этом нужно учитывать и уметь применять как уже устоявшиеся подходы к разработке интерфейсов, так и новые, которые могут сделать интерфейс более удобным и практичным.

Интерфейс должен быть построен так, чтобы минимизировать как выполнение рутинных часто повторяющихся операций, так и время, необходимое для его освоения. Основное внимание следует уделять продуктивности интерфейса.

Интерфейсом желательно заниматься на начальных стадиях разработки приложения. На более поздних стадиях возможности улучшения качества взаимодействия между пользователем и приложением могут

быть потеряны. Начинать разработку интерфейса можно уже после того, как определены задачи, для решения которых предназначено создаваемое приложение.

На первом этапе следует определить, какие действия должен сделать пользователь, для того чтобы получить тот или иной результат. Необходимо также представить реакцию приложения на эти действия. В ходе разработки приложения возможны корректировки, связанные с изменением как ставящихся задач, так и интерфейса. По этой причине разработка интерфейса представляет собой повторяющийся процесс.

Интерфейс должен быть ориентирован на пользователя, отвечать его нуждам и учитывать его особенности. Интерфейсы не должны быть неоправданно сложны, запутаны, неэкономны и побуждающими к ошибкам. Следует также обращать внимание на необходимость быстрой реакции приложения на действия, осуществляемые пользователем.

Часто действия пользователя при работе с интерфейсом могут иметь несколько интерпретаций. Примером может служить нажатие на клавишу «Return», которая в одном случае означает вставку символа возврата каретки, а в другом – обеспечивает выполнение команды. Состояние интерфейса, при котором интерпретация действий пользователя остается неизменной, называют режимом.

Использование различных режимов (modes) может быть источником ошибок, путаницы и сложности в интерфейсе. Режимы создают проблемы, связанные с тем, когда привычные действия приводят к неожиданным результатам. Для устранения этих проблем целесообразно:

1) обеспечить четкое различие между режимами, например, использовать индикаторы текущего состояния системы;

2) не использовать режимы (не допускать неоднозначной интерпретации действий);

3) не использовать одинаковые команды в разных режимах.

Пользовательские настройки являются одним из примеров использования режимов. Обычно они ориентированы на пользователей с разными навыками и предпочтениями. Они, конечно же, расширяют возможности приложений, но вместе с тем создают и затруднения, связанные с необходимостью их изучения и использования.

Режимы, которые исчезают после однократного применения, называют временными режимами. Эти режимы создают меньше ошибок, чем постоянные режимы, так как они имеют меньше времени на то,

чтобы эти ошибки вызвать. Например, нажатие на любую кнопку приводит к выходу компьютера из режима ожидания, но после этого эти кнопки выполняют действия, для которых они предназначены.

Включение и удерживание элемента управления во время выполнения другого действия называют квазирежимом, или режимом, удерживаемым пользователем. Примером такого режима может быть выбор варианта в выпадающем меню. Использование таких режимов устраняет недостатки постоянных режимов. Однако квазирежимы зачастую требуют запоминания команд. Для сохранения эффективности работы с приложением число квазирежимов должно быть небольшим – примерно от 4 до 7.

Источником ошибок пользователя при работе с интерфейсом могут быть и неточные формулировки подписей элементов управления. Например, вместо формулировки Lock (Блокировка) лучше применить более точную формулировку Locked (Заблокировано).

Для нормальной работы интерфейса должны быть видимы только необходимые его элементы, те, что обеспечивают работу приложения в данный момент времени и определяют способ взаимодействия с приложением. Функция элемента управления и способ ее использования должны выявляться уже по одному его виду. Иногда эту роль могут выполнять пиктограммы. В разработке интерфейса следует оптимизировать качество восприятия, что важно с точки зрения эргономики.

Многие программные продукты предлагают сразу несколько методов достижения того или иного результата. Например, панель инструментов зачастую дублирует действия пунктов главного меню. Кроме того, некоторые команды можно выполнять как с помощью меню, так и с помощью сочетания клавиш.

Такая тактика предъявления действий на выбор в некоторых случаях может быть оправдана, но нужно учитывать, что это приводит к увеличению стоимости и сложности программного продукта и увеличивает время обучения работе с ним.

В тех случаях, когда этого не требуется в силу особых условий, следует придерживаться того, чтобы действие пользователя имело один и только один результат. Такой подход обусловливает только одно соответствие между причиной (командами) и следствием (действиями).

1.2. Формы. Основные сведения

Интерфейс проектов Windows Forms основан на визуальном представлении элементов управления и строится на основе форм Windows. Формы служат для создания окон программы. Windows Forms – это одна из технологий реализации графического интерфейса в платформе .Net. Она включает в себя кмножество типов, которые представлены двумя пространствами имен – System.Windows.Forms и System.Drawing. Первое из них служит для реализации элементов интерфейса, а второе – для рисования в клиентской области окна формы.

Формы являются основным компонентом интерфейса пользователя и представляют собой самые крупные визуальные объекты – контейнеры для размещения в них разнообразных элементов управления приложением.

Для работы с простыми приложениями нужна как минимум одна форма. Для работы с более сложными приложениями их может быть несколько. Одна из форм с именем Form1 создается автоматически при создании проекта Windows Forms. Другие формы могут быть включены в проект приложения с помощью меню или соответствующей кнопки на панели инструментов.

Как и все визуальные компоненты интерфейса, формы имеют свои свойства, методы и события. Управление ими может происходить как на этапе разработки интерфейса, так и в процессе работы приложения.

Для редактирования форм на этапе разработки интерфейса в Visual Studio существует специальный конструктор, в среде которого можно добавлять элементы управления, настраивать их свойства и свойства самих форм.

Добавить форму в проект на этапе разработки интерфейса можно путем выбора пункта меню «Проект –> Добавить форму Windows» (Project -> Add Windows Form…). В открывшемся окне Add New Item нужно выбрать категорию Windows Forms, и в разделе Шаблоны (Templates) выбрать шаблон с именем Windows Form, после чего можно заполнить поле Name, указав новое имя формы. Доступ к окну «Add New Item» можно получить и с помощью кнопки «Add Windows Form», расположенной на панели инструментов.

В результате добавления формы создается класс, производный от класса Form и имеющий по умолчанию имя Form2. Имя этого класса можно увидеть в поле Name панели свойств (Properties) формы. Имя формы в C# используется в пространстве имен для уникальной иден-

тификации класса Form2, и по этой причине не является средством доступа к экземпляру формы по умолчанию.

Получить доступ к свойствам текущей формы можно не только из панели свойств, но и из кода самой формы. Для этого необходимо использовать ключевое слово this. Например, для изменения заголовка формы можно использовать оператор this.Text=”Вторая форма”.

Доступ к свойствам первой (стартовой) формы можно получить из второй формы, создав переменную типа Form1 и присвоив ей значение new Form1().

Пример 1.1:

```
Form1 f1=new Form1();
f1.Text="Первая форма";
f1.Show();
```

Стартовая форма и метод Main

Метод Main() определяет точку входа в программу, и описание этого метода содержится в файле Program.cs. Файл виден в обозревателе решений. Если его содержимое не отображается в редакторе кода приложения, то нужно выделить этот файл в обозревателе решений, и в контекстном меню файла выбрать пункт View Code. После выполнения указанных действий в редакторе кода приложения появится описание метода Main():

```
static void Main()
    {
      Application.EnableVisualStyles();
      Application.SetCompatibleTextRenderingDefault(false);
      Application.Run(new Form1());
    }
```

С выполнения этого метода начинается работа с приложением. Если приложение содержит несколько форм, то одну из них нужно назначить стартовой. При запуске приложения она загружается первой.

Для того чтобы другую форму назначить стартовой, нужно в методе Main() поменять имя класса, которому принадлежит форма. Например, вместо Application.Run(new Form1()); записать Application.Run(new Form2());.

Класс Program, который создается в процессе создания проекта, уже реализует метод Main(), и по умолчанию будет запускать при старте форму по умолчанию (Forml). Стартовым объектом в C# считается любой класс, который реализует метод Main ().

Структура окна формы

Наиболее часто используются формы, окна которых содержат следующие составные части.

1. Строка заголовка окна, в которой находится системное меню и системные кнопки для управления окном (свернуть, раскрыть, закрыть). Системное меню располагается в левой части строки, а системные кнопки – в правой.

2. Строка меню приложения – для выполнения различных операций, например, с файлами, базами данных и т. д. Строка меню чаще всего располагается вверху окна формы, но может быть расположена и в любой другой части окна.

3. Инструментальная панель, содержащая элементы управления для ускоренного доступа к часто выполняемым операциям. Окно формы может содержать несколько инструментальных панелей, расположенных в различных частях формы. Чаще всего инструментальная панель располагается сразу за строкой меню.

4. Рабочая (клиентская) область, используемая пользователем, как правило, для отображения и редактирования информации.

5. Статусная панель (строка состояния), отображающая состояние системы, например, установленный режим. В эту панель можно выводить информационные сообщения о ходе работы приложения.

Кроме перечисленных элементов формы могут содержать боковые стационарные и выдвижные панели, окна контекстных меню, окна подсказок и т. д.

Виды окон

Все разнообразие окон можно представить тремя группами:

1) основные окна, представляющие все приложение и инициирующие создание других окон;

2) диалоговые окна, используемые для получения информации и запуска на выполнение вспомогательных задач приложения;

3) элементы управления (controls), представляющие собой дочерние окна, которые используются для выполнения команд пользователя или для работы с информацией.

Некоторые свойства и события форм

Базовым классом для всех визуальных элементов управления, включая формы, является класс Control. Свойства, методы и события этого класса наследуются всеми его потомками. Некоторые свойства и события форм приведены в табл. 1.1 и 1.2 соответственно.

Таблица 1.1

Свойства класса Form

Свойство	Описание
AcceptButton	Определяет кнопку, запуск действия которой будет продублирован нажатием на клавишу «Enter»
CancelButton	Определяет кнопку, запуск действия которой будет продублирован нажатием на клавишу «Esc»
FormBorderStyle	Свойство, определяющее тип границы окна формы (находится в категории Font). По умолчанию это свойство имеет значение Sizable. Для диалоговых окон применяется значение FixedDialog
Text	Строка, определяющая заголовок формы
MainMenuStrip	Ссылка на компонент, формирующий основное меню формы
DialogResult	Свойство, определяющее способ закрытия «модального» диалогового окна. Доступно только программным способом. Значения, которые может принимать свойство, ограничены определенным набором. Наиболее часто используемые значения: None, OK, Cancel
MinimizeBox	Логическое свойство, определяющее необходимость размещения кнопки минимизации у формы. Как правило, у диалоговых окон эта кнопка отсутствует
MaximizeBox	Логическое свойство, определяющее необходимость размещения кнопки максимизации у формы. Как правило, у диалоговых окон эта кнопка отсутствует
ShowInTaskbar	Логическое свойство, определяющее необходимость отображения формы на панели задач. Как правило, диалоговые окна на панели задач не отображаются

Методы отображения форм

Формы могут отображаться в двух режимах – модальном, и не модальном. В первом из них вызываемые формы до завершения работы с

ними, блокируют работу в других формах. Второй режим обеспечивает параллельную работу с другими окнами. Методы отображения форм представлены в табл. 1.3.

Таблица 1.2

События, возникающие при создании и уничтожении форм

Событие	Описание
Activated	Возникает при каждом получении формой фокуса ввода
Deactivated	Возникает, когда форма перестает быть активной (при переходе пользователя к работе с другим приложением)
Closing	Возникает в процессе закрытия формы и предоставляет возможность управлять процессом закрытия приложения
Closed	Возникает после закрытия формы, позволяя освободить все выделенные ресурсы (закрыть файлы, закрыть соединение с базой данных)
Load	Возникает после инициализации формы, но до её отображения на экране. Полезно для выполнения действий, подготавливающих форму к работе

Таблица 1.3

Методы отображения класса Form

Метод	Описание
Show	Запускает отображение обычного окна (простое отображение формы). При этом может осуществляться одновременно работа как в основной форме, так и в вызванной
ShowDialog	Запускает отображение модального окна (отображение формы в виде диалогового окна – в «модальном» режиме). При таком отображении происходит блокирование всех остальных форм приложения (в том числе и основной) до тех пор, пока работа с диалоговым окном не будет завершена. Результат завершения работы диалогового окна класса DialogResult возвращается как значение функции ShowDialog

Диалоговые окна отображаются в модальном режиме и, как правило, имеют две кнопки «OK» и «Cancel». Для первой кнопки свойство DialogResult должно быть установлено в значение «OK», а для второй это же свойство должно принимать значение «Cancel». Первая кнопка используется для подтверждения ввода данных в диалоговом окне, а вторая (Cancel) – для отказа от ввода данных в этом окне.

Модальные и немодальные формы
Отличия и пример создания

Модальные окна – это окна, которые в открытом состоянии блокируют работу с другими окнами. Переключиться на работу с другим окном можно только после закрытия модального окна. Форма со свойством перечисления FormBorderStyle, для которого установлено значение FixedDialog, относится к числу диалоговых окон.

Основное назначение модальных окон – поддержание диалога с пользователем. До завершения диалога модальную форму покинуть нельзя. Часто использование этого типа окон определяется необходимостью подтверждения каких-либо действий путем нажатия определенной кнопки. Широко используются кнопки с надписями «OK», «Cancel», «Yes», «No». Для открытия модальных окон используется метод ShowDialog.

Немодальные (обычные) окна равноправны со всеми другими окнами приложения. Они дают возможность параллельной работы в разных окнах. Для открытия немодальных окон используется метод Show(). Если переменная f ссылается на объект класса, производного от класса Form, то вызов метода f.Show() позволяет переключиться на этот объект. Метод же ShowDialog позволяет сделать это переключение только после закрытия модальной формы.

Другим отличием методов Show и ShowDialog является то, что метод Show при завершении не возвращает никакого значения, а метод ShowDialog возвращает одно из значений перечисления DialogResult, являющееся идентификатором, поясняющим причину закрытия диалога.

Следует заметить, что идентификаторы создаются автоматически дизайнером форм при добавлении в форму элементов управления и компонентов.

Перечисления могут содержать следующие причины закрытия диалога:

1) OK – работа с формой завершилась успешно (пользователь выполнил требуемую задачу);

2) Cancel – работа с формой завершилась не успешно (пользователь не выполнил требуемую задачу);

3) Abort – прервать;

4) Retry – повторить;

5) Ignore – пропустить;

6) Yes – пользователь ответил утвердительно на заданный вопрос;

7) No – пользователь ответил отрицательно на заданный вопрос.

При закрытии обычного окна объект окна уничтожается, и по этой причине повторно его вызвать с помощью метода Show() нельзя. При закрытии модального окна, созданного с использованием метода ShowDialog, экземпляр класса окна не уничтожается и поэтому не нужно перед каждым новым вызовом создавать экземпляр класса. Он может быть создан всего лишь раз, например, в методе Main() модуля Program.cs.

Еще одно отличие модального окна от обычного состоит в том, что свойства модального окна можно менять только в самом классе этого окна, в то время как свойства обычных окон можно менять из классов других окон.

Нужно иметь в виду, что любую обычную форму можно запустить как модальное окно, если вместо метода Show() использовать метод ShowDialog().

Пример создания и отображения обычного окна на основе класса Form2, производного от класса Form:

```
Form2 f2 = new Form2();
f2.Show();
```

Пример создания и отображения модального окна на основе класса Form2, производного от класса Form:

```
Form2 f2 = new Form2();
f2. ShowDialog();
```

Настройка модального диалогового окна

Пример создания и отображения модального окна на основе класса Form2, производного от класса Form

Используя кнопку Add New Item, добавим в проект приложения дополнительную форму класса Form2, производного от класса Form. В обозревателе решений увидим новый компонент Form2.cs. Выбирая в контекстном меню этого компонента пункт View Designer, отобразим его содержимое.

В конструкторе формы установим следующие значения свойств:

1) FormBorderStyle = FormBorderStyle.FixedDialog; //не изменять размер
2) ControlBox = false; //убрать из заголовка меню управления и кнопки
3) ShowInTaskbar = false; //не показывать в панели задач

Добавим на форму элемент управления Label и установим его значение Text в виде строки «Выберите одну из альтернатив: ».

Добавим на форму три управляющих элемента Button с надписями «Да», «Нет», «Отменить». В обработчиках событий Click этих элементов присвоим свойству DialogResult значение, соответствующее назначению кнопки, например, DialogResult = DialogResult.OK;.

На форму класса Form1 добавим элемент Label, и изменим его значение на пустую строку. Установим заголовок формы вида «Стартовое окно». Добавим на форму управляющий элемент – Button, в обработчике события Click которого пропишем код:

```
Form2 f2 = new Form2();
f2.Text = "Диалоговое окно"; // Заголовок окна
f2.ShowDialog(); // Открыть окно в режиме диалога
string str = "Вы нажали кнопку " + f2.DialogResult;
if (Convert.ToString(f2.DialogResult) == "OK") this.label1.Text = str;
if (Convert.ToString(f2.DialogResult) == "Cancel") this.label1.Text = str;
if (Convert.ToString(f2.DialogResult) == "Abort") this.label1.Text = str;
```

После запуска данного приложения, на экране появится стартовая форма вида, изображенного на рис. 1.1.

Нажатие на кнопку стартовой формы позволяет отобразить диалоговое окно вида, показанного на рис. 1.2.

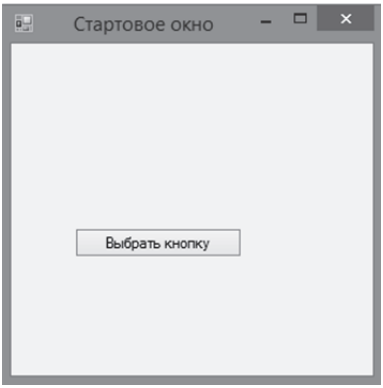

Рис. 1.1. Внешний вид окна стартовой формы

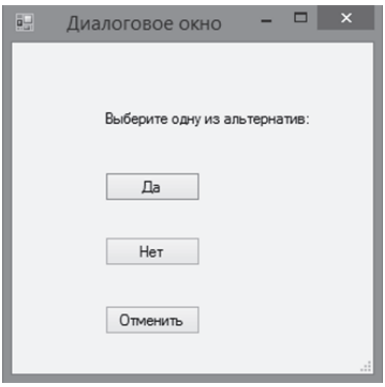

Рис. 1.2. Внешний вид диалогового окна