

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Брянский государственный технический университет

**Утверждаю**

**Ректор университета**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О.Н.Федонин**

**«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2017 г.**

**Программирование в среде Microsoft .NET**

**Создание приложений Windows Forms для взаимодействия с базами данных**

**Методические указания**

**к выполнению лабораторной работы №12**

**для студентов очной формы обучения по направлениям подготовки 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.01 – «Информатика и вычислительная техника» и 09.03.04 – «Программная инженерия»**

**Брянск 2017**

УКД 004.43

Программирование в среде Microsoft .Net. Создание приложений Windows Forms для взаимодействия с базами данных [Текст] + [Электронный ресурс]: Методические указания к выполнению лабораторной работы №12 для студентов очной формы обучения по направлениям подготовки 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.01 – «Информатика и вычислительная техника» и 09.03.04 – «Программная инженерия». – Брянск: БГТУ, 2017. – 20 с.

Разработал:

Д.А.Коростелёв

канд. техн. наук, доц.

Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол №8 от 29.06.2017)

**Методические издания публикуются в авторской редакции**

# Цель работы

Целью работы является изучение основ построения приложений Windows Form для работы с базами данных в автономном режиме.

Продолжительность работы – 2 часа.

# Работа с базами данных

Приложения, которые работают с базами данных, имеют одну общую схему функционирования. Обычно данные необходимо выбрать из базы данных в приложение, а пользовательский интерфейс приложения должен быть обновлен для соответствующего отображения данных. Для больших баз размером, например, в 100 Мбайт, необходимо использовать механизм разбивки на страницы (для того чтобы пользователь мог перемещаться взад и вперед по «потоку» данных). После того как данные благополучно доберутся до пользовательского интерфейса приложения, необходимо заняться движением данных от приложения к базе данных. Любые изменения в данных должны быть согласованы и зафиксированы в базе данных.

***Привязка данных*** – это термин для реализации проектной схемы, которая обрабатывает все аспекты цикла обработки данных, начиная от структуры данных до элементов управления приложения и обратно. Несмотря на то что ***структура данных*** – это чаще всего база данных, она может быть также контейнерным объектом любого типа (таким как массив или коллекция). Среда Microsoft .NET в дальнейшем разделяет концепцию привязки данных на простую и сложную. Оба этих термина относятся к внутренним возможностям элемента управления в более широком контексте процесса привязки данных.

***Простая привязка данных*** – это способность элемента управления привязаться к единственному элементу данных из большого набора данных и отобразить его. Элемент управления TextBox является отличным примером часто используемого при простой привязке данных элемента. Так, можно использовать TextBox для отображения фамилии служащего в том виде, в каком она хранится в таблице служащих внутри базы данных. Поддержка простой привязки данных имеется в элементах управления как windows-форм, так и web-форм.

Термин «сложная привязка данных» относится к способности элемента управления отображать одновременно множество элементов данных. Можно считать это способностью к «многострочности»: если элемент управления можно использовать для просмотра множества строк данных одновременно, то он поддерживает сложную привязку данных.

Элементы управления DataGridView (для форм Windows) и DataGrid (для форм Web) – это основные примеры элементов управления, которые были созданы специально для обработки табличных (многострочных и многостолбцовых) данных. Внутренние механизмы реализации привязки данных запутанны, сложны и трудны для понимания, однако по большей части инструментам Visual Studio удалось абстрагировать реализацию привязки данных до уровня приятного и простого процесса перетаскивания.

Рассмотрим, как быстро создать поддержку привязки данных.

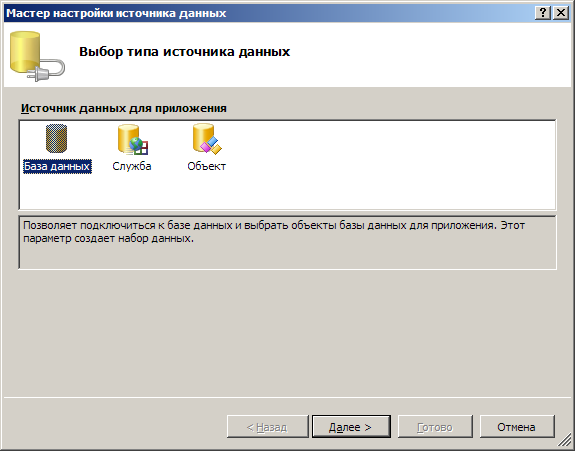
Несмотря на то что в среде Visual Studio существуют различные способы реализации привязанных к данным элементов управления, все они содержат одни и те же два шага:

1. Определить источник данных.
2. Установить соответствие между членами источника данных и элементами управления (или их свойствами).

После этого визуальные конструкторы Form Designers могут генерировать правильные элементы управления и размещать их на форме. Создается исходный код привязки данных; все, о чем необходимо позаботиться, – это компоновка, позиционирование и настройка аспектов пользовательского интерфейса элементов управления. Форма Windows может иметь элементы управления, которые используют простую или сложную привязку данных (либо и ту, и другую одновременно).

# Выбор источника данных

Убедимся в среде Visual Studio, что работа осуществляется в проекте приложения Windows Application, после чего используем окно Data Sources для выбора источника данных. Если это окно не видно, то в меню Data среды Visual Studio выберем пункт Show Data Sources. Если в проекте приложения нет ни одного определения источника данных, то необходимо будет его создать. Нажмем кнопку Add New Data Source на панели инструментов окна для запуска мастера Data Source Configuration Wizard. На первой странице этого мастера (рис. 1) выберем тип источника данных.



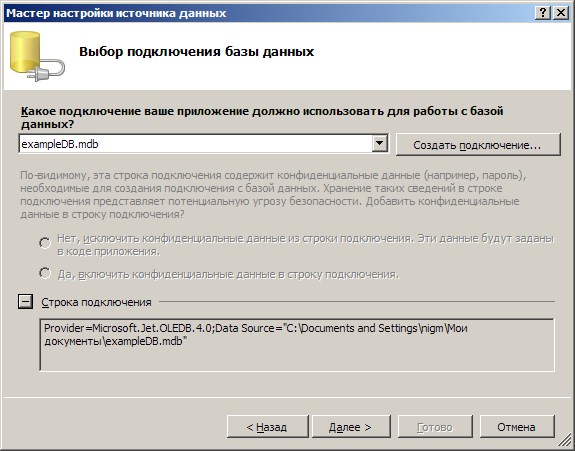
***Рис. 1. Выбор типа источника данных***

Здесь есть три варианта:

* Database – источник данных является таблицей в реляционной базе данных;
* Web Service – источник данных является web-сервисом, который будет выдавать данные, привязываемые к элементам управления формы;
* Object – источник данных является объектом, который будет обеспечивать данные (это полезно в тех случаях, когда за поставку данных в форму отвечает бизнес-объект из другого слоя приложения).

Так как концепции привязки данных легче всего понять в контексте базы данных, то в качестве типа источника данных будем использовать именно базу данных.

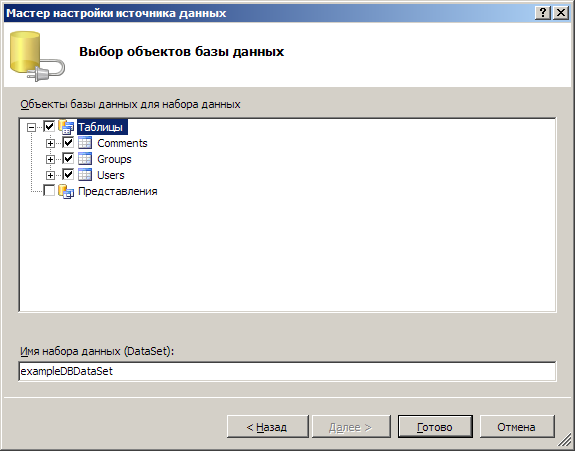
Если выбран тип Database (База данных) источника данных, то на второй странице мастера выбирается подключение для базы данных. Здесь в раскрывающемся списке будут по умолчанию показаны все ранее установленные подключения как для других источников данных, так и для использования в Server Explorer (рис. 2). Также есть возможность указать новое подключение. Если строка подключения к базе данных имеет такую конфиденциальную информацию, как пароль пользователя, то можно исключить его из строки.



***Рис. 2. Выбор подключения***

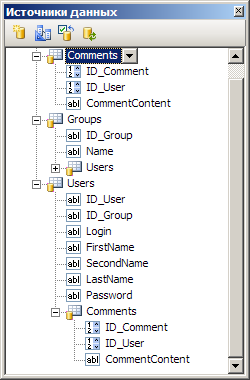
Следующий шаг мастера позволяет сохранить информацию строки подключения в локальный файл конфигурации проекта приложения. Сохранение этой информации обычно является хорошей идеей, так как это позволяет при необходимости модифицировать строку (например, при изменении базы данных). Однако необходимо соблюдать осторожность, если решено сохранить в строке конфиденциальную информацию.

На последней странице мастера (рис. 3) указывается, какой из объектов базы данных должен использоваться как источник данных. Можно выбрать любой элемент данных, имеющийся в любой таблице, представлении, хранимой процедуре или пользовательской функции базы данных.



***Рис. 3. Выбор объектов источника данных***

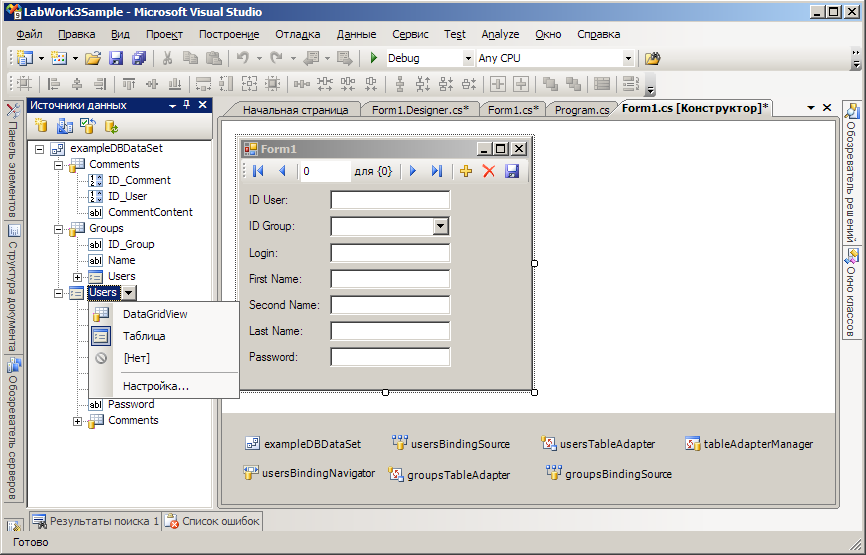
После завершения работы мастера источник данных будет виден в окне Data Sources (рис. 4).



***Рис. 4. Окно Data Source***

# Установление соответствия между источниками данных и элементами управления

Чтобы легко и быстро создать привязанные к данным элементы управления, необходимо позволить сделать это среде Visual Studio. В окне Data Sources нажмите кнопку раскрывающегося списка возле имени источника данных. В результате этого отобразится меню, которое позволяет настроить параметры генерирования элементов управления, а также отвечает на вопрос: какие элементы управления необходимо сгенерировать на основе таблицы источника данных? Если указать здесь DataGridView, то можно сгенерировать элемент управления DataGridView для просмотра и редактирования источника данных. Опция Details (Таблица) позволяет генерировать последовательности простых (привязанных к данным) элементов управления для просмотра или редактирования данных в источнике данных (рис. 5). По существу без всякого кодирования можно создать с нуля целое приложение для работы с данными.



***Рис. 5. Автоматически сгенерированные элементы управления,***

***привязанные к таблице***

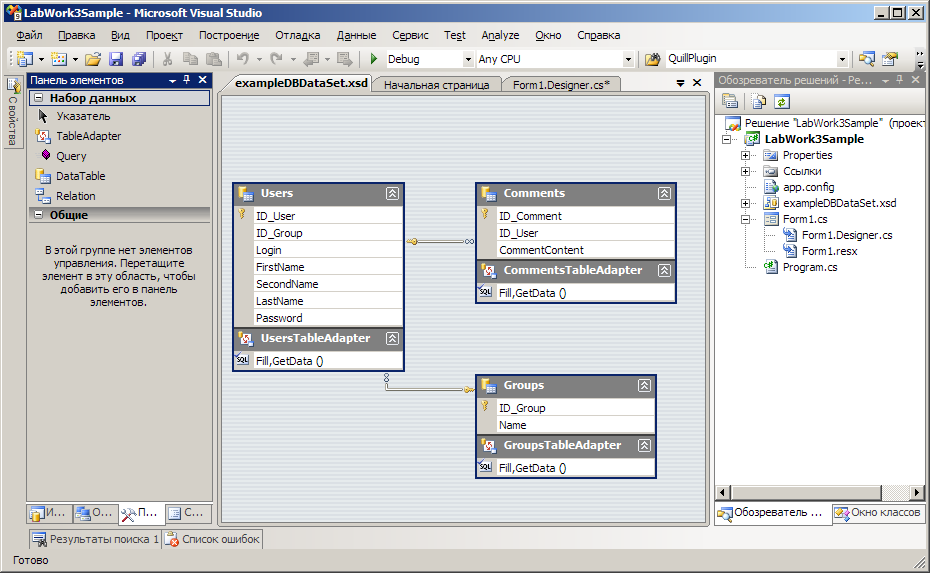
Использование простой привязки данных может плохо сочетаться с требованиями к дизайну пользовательского интерфейса, поэтому всегда можно начать работать со сложной привязкой данных (с помощью DataGridView).

В среде Visual Studio имеется визуальный конструктор, предназначенный исключительно для редактирования (и создания) типизированных наборов данных – это DataSet Designer. Этот визуальный конструктор запускается автоматически тогда, когда открывается элемент DataSet (такой как файл exampleDBDataSet.xsd, который был автоматически создан при исследовании привязки данных).

Класс DataSet, представляющий набор данных, был разработан как автономное хранилище данных. Он не содержит понятия соединений баз данных. По сути, данные, содержащиеся в классе DataSet, не обязательно должны поступать из базы – они просто могут быть записями из файла CSV (файл значений, разделяемых запятой) либо показаниями измерительного прибора.

Класс DataSet состоит из набора таблиц данных (классов DataTable), каждая из которых подразумевает наличие ряда столбцов данных (классов DataColumn) и ряда строк данных (классов DataRow). В классе DataSet могут определяться не только эти таблицы, но и *связи* между ними. Одним из наиболее типичных сценариев является определение связи или отношения типа «родитель­ский-дочерний» (которое также часто называется отношением «главный-подчиненный»). Так, одна запись в таблице (скажем, таблице Order) может связываться с множеством записей в другой таблице (например, Order Details). Это отношение может быть определено в классе DataSet.

Конструктор DataSet Designer можно легко использовать для настройки наборов данных (изменением любых их компонентов, в том числе и запросов для заполнения наборов данных). Ранее созданный DataSet, открытый в конструкторе DataSet Designer, показан на рис. 6.



***Рис. 6. Конструктор DataSet Designer в среде Visual Studio***

Обратите внимание, что здесь визуально представлен каждый элемент набора данных, и для выполнения изменений можно взаимодействовать с этими элементами. Добавление элемента DataSet в проект приложения позволит начать с чистого листа, добавляя в набор данных таблицы, запросы и т. п. для удовлетворения потребностей приложения в данных. Это особенно полезно для тех приложений, которые читают и пишут данные, но не обязательно взаимодействуют с базой данных. Такие наборы данных могут использоваться как простой способ хранения. Затем они могут быть легко привязаны к реляционной базе данных.

# Сохранение и получение значений объектов BLOB в базе данных

Объекты BLOB в базе данных – это не просто строки с названиями и адресами, цифровые или денежные значения. Объекты BLOB – это графики и фотографии; документы, сохраненные в двоичных форматах; и даже полные сборки или выполняемые файлы, которые необходимо сохранить в базе данных. В отличие от выполнения запросов или хранимых процедур, которые возвра­щают «простые» типы данных, работать с двоичными объектами немного сложнее.

При сохранении и получении двоичных данных могут возникнуть проблемы, которые при запросе стандартных записей данных никогда не появлялись. Проблемы, скорее всего, возникнут, если не перемещать весь объект BLOB целиком (например, большой двоичный файл размером несколько мегабайт), а разбить его на несколько частей, так как загрузка всего объекта BLOB в переменную расхо­дует много памяти и может значительно повлиять на производительность приложения. Однако если придется работать с таблицей, которая состоит из объектов BLOB, то сразу же возникают проблемы.

Среда Microsoft .NET предоставляет классы, спе­циально предназначенные для перемещения больших объемов двоичных данных. В частности, доступ к этим классам (например, классам BinaryReader и BinaryWriter, FileStream и MemoryStream и т.д.) активируется в пространстве имен System.IO.

Если в проекте для работы с базой данных используется DataSet, то считывание данных из поля типа BLOB, хранящего изображение, и загрузки их в элемент управления PictureBox можно осуществлять следующим образом:

DataRowView drw = (DataRowView)usersBindingSource.Current;

exampleDBDataSet.UsersRow ur = (exampleDBDataSet.UsersRow)(drw.Row);

MemoryStream ms = new MemoryStream(ur.Photo);

photoPictureBox.Image = Image.FromStream(ms,true, true);

ms.Close();

Сохранение графических данных в соответствующую запись таблицы базы данных осуществляется следующим образом:

DataRowView drw = (DataRowView)usersBindingSource.Current;

exampleDBDataSet.UsersRow ur = (exampleDBDataSet.UsersRow)(drw.Row);

MemoryStream ms = new MemoryStream();

photoPictureBox.Image.Save(ms, System.Drawing.Imaging.ImageFormat.Jpeg);

ur.Photo = ms.ToArray();

ms.Close();

usersTableAdapter.Update(ur);

# Печать

Обычно, реализуя возможность печати данных в приложении, необходимо добавить три команды в главное меню приложения File (Файл):

* Page Setup (Параметры страницы) – для выбора опций печати, например, ка­кие страницы печатать, какой принтер использовать и тому подобное;
* Print Preview (Предварительный просмотр) – для открытия новой формы, де­монстрирующей внешний вид макета для печати;
* Print (Печать) – для отправки документа на устройство печати.

Для простоты не будем реализовывать команду меню Page Setup. Печать будет выполняться только с установками по умолчанию. Однако отметим, что если необходимо реализовать Page Setup, то среда Microsoft .NET подготовила класс диалогово­го окна настройки печати – System.Windows.Forms.PrintDialog. Можно напи­сать обработчик события меню, который отобразит эту диалоговую форму Windows и сохранит выбранные пользователем настройки.

Во многих отношениях печать – это то же самое, что и отображение на экране. Используется контекст устройства (экземпляр Graphics) и вызываются обычные команды отображения с этим экземпляром. В Microsoft разработали множество классов, при­званных помочь в этом. Вот два главных класса, которые для этого понадобятся:

* System.Drawing.Printing.PrintDocument;
* System.Drawing.Printing.PrintPreviewDialog.

Эти два класса берут на себя обеспечение доставки команд рисования необходимому контексту устройств для выполнения печати и тем самым избавляют разработчика от рутинной работы и позволяют сосредоточиться на логике того, что необходимо напечатать.

Несмотря на эти сложности сам процесс печати достаточно прост. Необходимые шаги программного характера, которые для этого выполняют, сводятся к описанным ниже.

**Печать.** Создается экземпляр объекта PrintDocument и вызывается его метод Print(). Этот метод посылает событие PrintPage для печати первой страницы. Событие PrintPage принимает параметр PrintPageEventArgs, который содержит ин­формацию относительно размера бумаги и настроек, а также объект Graphics, используемый для команд рисования. Таким образом, необходимо написать обра­ботчик для этого события и реализовать его для печати страницы. Этот обработчик также должен устанавливать свойство PrintPageEventArgs булевского типа под названием HasMorePages в true или false для указания того, есть ли еще страницы для печати. Метод PrintDocument.Print() выполняет повто­ряющуюся инициализацию события PrintPage до тех пор, пока не увидит, что HasMorePages установлено в false.

**Предварительный просмотр.** Вэтом случае создаются экземпляры объектов PrintDocument и PrintPreviewDialog. Экземпляр PrintDocument присоединяется к PrintPreviewDialog (через свойство PrintPreviewDialog.Document), после чего вызывается метод диалога ShowDialog(). Этот метод модально отображает диалоговое окно, которое представляет собой стандартную форму Windows для предварительного просмотра печати с загруженными в нее стра­ницами документа. Внутренне страницы отображаются последовательной генерацией события PrintPage до тех пор, пока свойство HasMorePages не станет равно false. При этом нет необходимости писать отдельный обработчик событий для этого; используется тот же обработчик, что служит для печати каждой страницы, поскольку код рисования в обоих случаях идентичен (в конце концов, предварительный просмотр печати должен показывать в точности то, что будет напечатано).

# Реализация команд меню Print и Print Preview

Реализуем в исходном коде выводе текстового документа на печать. Исходный код примера доступен в виде проекта Printing по адресу: <ftp://iipo.tu-bryansk.ru/pub/DotNet/Bachelor/06/Printing.zip>.

Начнем с использования представления конструктора среды Visual Studio, добавив две новых команды в меню File (Файл);

Print (Печать) и Print Preview (Предварительный просмотр). С помощью окна свойств переименуем их в menuFilePrint и menuFilePrintPreview, а также сделаем их неактивными при запуске приложения (нельзя ничего печатать, пока документ не открыт).

private void LoadFile(string FileName)

{

StreamReader sr = new StreamReader(FileName);

string nextLine;

documentLines.Clear();

nLines = 0;

TextLineInformation nextLineInfo;

while ((nextLine = sr.ReadLine()) != null)

{

nextLineInfo = new TextLineInformation();

nextLineInfo.Text = nextLine;

documentLines.Add(nextLineInfo);

++nLines;

}

sr.Close();

if (nLines > 0)

{

documentHasData = true;

menuFilePrint.Enabled = true;

menuFilePrintPreview.Enabled = true;

}

else

{

documentHasData = false;

menuFilePrint.Enabled = false;

menuFilePrintPreview.Enabled = false;

}

CalculateLineWidths();

CalculateDocumentSize();

this.Text = standardTitle + " - " + FileName;

this.Invalidate();

}

Здесь выделен новый исходный код, добавленный к методу. Далее добавим поле-член в класс Form1:

private int printingPageNo = 0;

Это поле будет использоваться для указания номера текущей печатаемой страни­цы. Объявляем его полем-членом, потому что информацию о текущей странице необходимо держать в памяти между вызовами обработчика событий PrintPage.

Далее следуют обработчики событий команд меню Print и PrintPreview

private void menuFilePrintPreview\_Click(object sender, System.EventArgs e)

{

PrintPreviewDialog ppd = new PrintPreviewDialog();

PrintDocument pd = new PrintDocument();

pd.PrintPage += new PrintPageEventHandler

(this.pd\_PrintPage);

ppd.Document = pd;

ppd.ShowDialog();

}

private void menuFilePrint\_Click(object sender, System.EventArgs e)

{

PrintDocument pd = new PrintDocument();

pd.PrintPage += new PrintPageEventHandler

(this.pd\_PrintPage);

pd.Print();

MessageBox.Show(pd.PrinterSettings.PrinterName);

}

Эти обработчики событий реализуют описанные шаги, которые необходимо выполнить для осуществления печати. В обоих случаях создается экземпляр объекта PrintDocument и к его событию PrintPage присоединяется обработчик. При печати вызывается метод PrintDocument.Print(), в то время как для предварительного просмотра объект PrintDocument присоединяется к событию PrintPreviewDialog, и вызывается метод ShowDialog() диалогового окна предварительного просмотра. Реальная работа собы­тия PrintPage выполняется в обработчике события, который выглядит следующим образом

private void pd\_PrintPage(object sender, PrintPageEventArgs e)

{

float yPos = 0;

float leftMargin = e.MarginBounds.Left;

float topMargin = e.MarginBounds.Top;

string line = null;

// Calculate the number of lines per page.

int linesPerPage = (int)(e.MarginBounds.Height /

mainFont.GetHeight(e.Graphics));

// linesPerPage = 10;

int lineNo = this.printingPageNo \* linesPerPage;

// Print each line of the file.

int count = 0;

while (count < linesPerPage && lineNo < this.nLines)

{

line = ((TextLineInformation)this.documentLines[lineNo]).Text;

yPos = topMargin + (count \* mainFont.GetHeight(e.Graphics));

e.Graphics.DrawString(line, mainFont, Brushes.Blue,

leftMargin, yPos, new StringFormat());

lineNo++;

count++;

}

// If more lines exist, print another page.

if (this.nLines > lineNo)

e.HasMorePages = true;

else

e.HasMorePages = false;

printingPageNo++;

}

После объявления нескольких локальных переменных первое, что здесь делаем – это вычисляем количество строк, которое может быть отображено на одной странице, как высоту страницы, деленную на высоту строки с округлением в меньшую сторону. Высоту страницы можно получить из свойства MarginBounds объекта класса PrintPageEventArgs. Это свойство представляет собой структуру RectangleF, которая инициализирована размерами страницы. Высота строки получается из поля Form1.mainFont, представляющего шрифт, используемый для отображения текста. Нет причин использовать для печати какой-то другой шрифт.

Обратите внимание, что в примере PrintingCapsEditor количество строк на страницу всегда одно и то же, поэтому имело бы смысл его запомнить после первого вычисления. Однако эти вычисления не особенно сложны, к тому же в более разви­том приложении это количество может меняться, поэтому ничего плохого не случится, если станем повторно вычислять его при печати каждой страницы.

Также инициализируем здесь переменную с именем lineNo. Она представляет начинающийся с нуля индекс строки документа, которая будет первой на странице. Эта информация значима, так как в принципе метод pdPrintPage() может быть вызван для печати любой страницы, а не только первой. Значение lineNo вычисля­ется как количество строк на странице, умноженное на количество уже напечатанных страниц.

Далее запускается цикл, печатающий каждую строку. Этот цикл прерывается либо когда обнаруживается, что уже напечатаны все строки текста документа, либо когда будут напечатаны все строки, помещающиеся на текущей страницы – в зависимости от того, что произойдет раньше. И, наконец, проверяем, остались ли еще стро­ки документа для печати, и соответствующим образом устанавливаем свойство HasMorePages аргумента PrintPageEventArgs, а также увеличиваем на единицу значение поля pages Printed, чтобы знать, какую страницу напечатать, когда в следующий раз будет вызван обработчик события Print Page.

Отметим один момент относительно этого обработчика событий, а именно: нам не требуется беспокоиться о том, как выполняются команды рисования. Только используем объект Graphics, который получаем из PrintPageEventArgs. Класс Print Document, разработанный Microsoft, сам позаботится о том, что если мы собираемся печатать, то объект Graphics должен быть «прикреплен» к принтеру. Если же речь идет о предварительном просмотре, он «прикрепляется» к форме предварительного просмотра на экране.

И, наконец, необходимо убедиться, что пространство имен System.Drawing.Printing включено для поиска определений типов:

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Printing;

using System.Text;

using System.Windows.Forms;

using System.IO;

Все, что остается – скомпилировать проект приложения и проверить, как оно работает.

# Задание на лабораторную работу

1. Разработать клиентское приложение Windows Form для работы с базой данных, разработанной в предыдущей лабораторной работе, через DataSet. При этом необходимо реализовать следующий функционал:

* для второй таблицы базы данных реализовать возможность просмотра, добавления, редактирования, удаления записей в табличном виде (использовать элемент управления DataGridView);
* задействовать для второй таблицы компонент ErrorProvider;
* для первой таблицы реализовать возможность просмотра и удаления записей в табличном виде, а редактирование и добавление в детализированном виде (для каждого поля использовать соответствующий элемент управления: для текстовых полей – TextBox; для числовых полей – NumericUpDown; для булевых полей – CheckBox; для полей, использующих данные из другой таблицы – ComboBox; для изображений – PictureBox);
* на главной форме приложения необходимо максимально задействовать следующие элементы управления: главное меню (MenuStrip) и статусную строку (StatusStrip).

1. Для элемента управления PictureBox, связанного с полем изображения из первой таблицы, разработать контекстное меню (ContextMenu) и реализовать его функционал: загрузка изображения из файла, сохранение изображения в файл, свойства. О завершении действий (загрузка, сохранение) информировать в статусной строке. При выборе пункта «Свойства» должен отображаться обозреватель свойств PropertyGrid для соответствующего элемента управления PictureBox.
2. Реализовать поддержку печати содержимого текущей записи первой таблицы в форме отчета.

# Контрольные вопросы

1. Какие стандартные элементы управления используются для работы с базами данных с помощью технологии Windows Forms?
2. Что такое привязка данных? Какие бывают её виды?
3. Какова последовательность при выборе источника данных?
4. Каким образом можно установить соответствие между источниками данных и элементами управления?
5. Перечислите особенности сохранения и получения значений BLOB в базе данных.
6. Какие стандартные средства имеются в среде Microsoft .NET для организации вывода на печать?

# Список рекомендуемой литературы

1. Нейгел, К. C# 5.0 и платформа .NET 4.5 для профессионалов / К.Нейгел, Б.Ивьен, Дж.Глинн, М.Скиннер, К.Уотсон. – М.: Диалектика, 2013. – 1440 с.
2. Неш, Т. C# 2010: ускоренный курс для профессионалов / Т.Неш – М.: Вильямс, 2010. – 592 с.
3. Снелл, М. Microsoft Visual Studio 2008 / М.Снелл, Л.Пауэрс. – СПб: БХВ-Петербург, 2009. – 1200 с.
4. Троелсен, Э. Язык программирования С# 5.0 и платформа .NET 4.5 / Э.Троелсен. – 6-е изд. – М.: Вильямс, 2015. – 1312 с.
5. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж.Рихтер. – СПб.: Питер, 2016. – 896 с.

Программирование в среде Microsoft .Net. Создание приложений Windows Forms для взаимодействия с базами данных [Текст] + [Электронный ресурс]: Методические указания к выполнению лабораторной работы №12 для студентов очной формы обучения по направлениям подготовки 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.01 – «Информатика и вычислительная техника» и 09.03.04 – «Программная инженерия». – Брянск: БГТУ, 2017. – 20 с.

КОРОСТЕЛЁВ ДМИТРИЙ АЛЕКСАНДРОВИЧ

Научный редактор Д.И. Булатицкий

Компьютерный набор Д.А. Коростелёв

Иллюстрации Д.А. Коростелёв

Подписано в печать 07.07.2017 г. Формат 60х84 1/16 Бумага офсетная. Офсетная печать. Усл.печ.л. 0,63 Уч.-изд.л. 0,63 Тираж 1 экз.

Брянский государственный технический университет

Кафедра «Информатика и программное обеспечение», тел. 56-09-84

241035, Брянск, бульвар 50 лет Октября, 7 БГТУ