Лабораторная работа № 1
РАЗРАБОТКА ПРИЛОЖЕНИЙ НА ЯЗЫКЕ C#

1. Создание консольного приложения

Запустите систему *Visual Studio 2013*. Для создания нового проекта щелкните пункты меню *Файл* → *Создать* → *Проект*, затем в открывшемся окне выберите язык *Visual C#* и тип проекта – *Консольное приложение*. Дайте проекту имя *ConsoleHello* и назначьте папку для хранения проекта (рис.1).

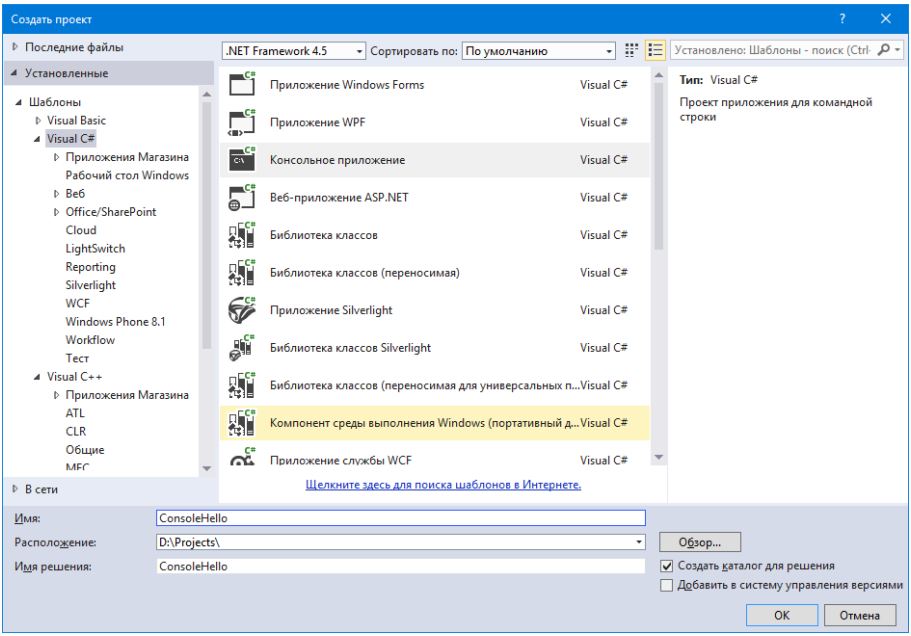


Рис. 1. Диалоговое окно создания нового проекта

Щелкните кнопку *OK*, и компилятор создаст решение (рис.2), имя которого совпадает с именем проекта.

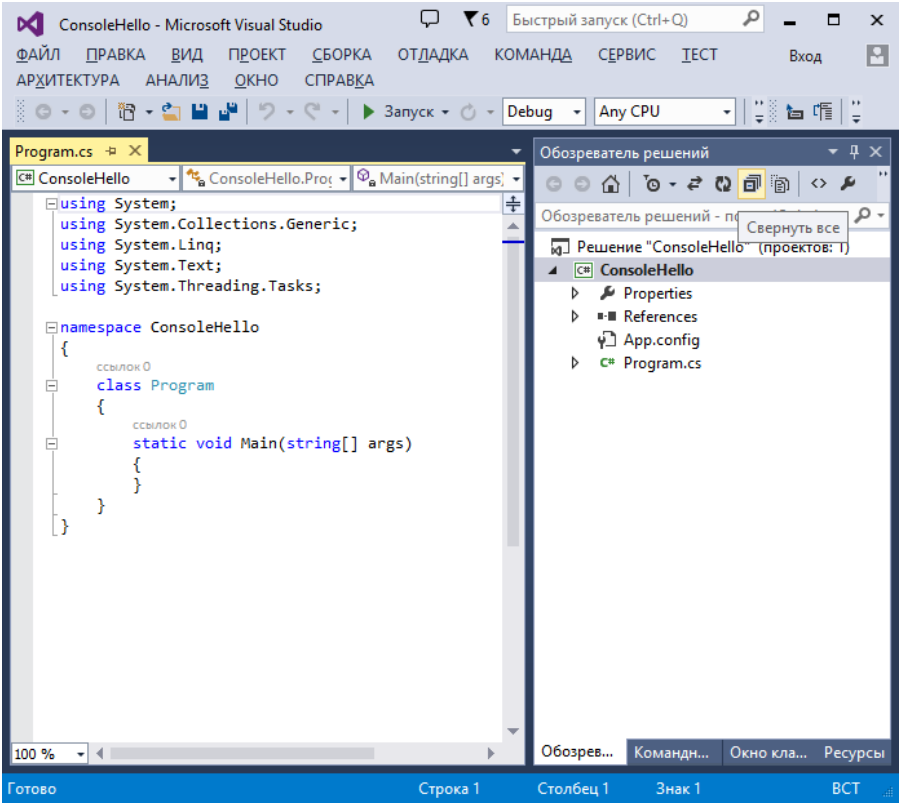


Рис. 2. Среда разработки с консольным приложением по умолчанию

В окне *Обозреватель решений* отображается структура создаваемого решения. В окне документов на вкладке *Program.cs* отображается выбранный документ, в данном случае программный код класса *Program*.

В проекте имеются папка *Properties*, включающая файл *AssemblyInfo.cs* с информацией для сборки, папка *References* со ссылками на системные пространства имен из библиотеки классов каркаса (*FCL – Framework Class Library*), конфигурационный файл

App.config и файл *Program.cs*, содержащий класс *Program*, в котором задается точка входа – метод *Main()*.

Строка кода *namespace ConsoleHello* показывает, что класс *Program* «погружен» в пространство имен *ConsoleHello*, имеющее по умолчанию то же имя, что и решение (проект). Этому пространству имен могут предшествовать предложения с ключевым словом *using*, после которого следует имя пространства имен из библиотеки *FCL* или из проектов, связанных с данным проектом. В нашем случае задано пространство имен *System* (основное пространство имен *FCL*). В частности, предложение *using System* упрощает в программе запись обращений к членам классов, входящих в пространство имен *System*, так как в этом случае не требуется в каждом обращении записывать имя *System* перед именем класса. Мы увидим это на примере работы с классом *Console* из пространства имен *System*. Запись полного имени класса может потребоваться, если в нескольких используемых пространствах имен имеются классы с одинаковыми именами.

В языке C# допускаются однострочные и многострочные комментарии в стиле языка C++. В методе *Main()* можно опустить аргументы, которые обычно не задаются. Они используются только при запуске приложения через командную строку.

Итак, мы рассмотрели консольный проект, построенный по умолчанию. Функций у него мало. Его можно скомпилировать, выбрав пункт меню *Сборка → Собрать решение*. Если компиляция прошла успешно, то выполнится сборка, и в папке *Debug* проекта появится так называемый *PE*-файл.

Компиляторы языков программирования в *Visual Studio .Net* создают модули на промежуточном языке (*MSIL – Microsoft Intermediate Language*). Фактически они создают так называемый переносимый исполняемый файл (*Portable Executable File* или *PE*-файл). *PE*-файл содержит код на языке *MSIL* и метаданные, то есть всю информацию, необходимую для *CLR* и конечных пользователей. В зависимости от типа проекта *PE*-файл может иметь расширения *exe*, *dll*, *mod* или *mdl*. Заметьте, хотя *PE*-файл с расширением *exe* и является исполняемым файлом, но это не совсем обычный, исполняемый файл *Windows*. При запуске он распознается как особый *PE*-файл и передается *CLR* для обработки. *CLR* начинает работать с кодом, в

котором нет специфики исходного языка. Этот код на языке *MSIL* называется управляемым кодом и выполняется под управлением *CLR*. Таким образом, *CLR* – это виртуальная машина для языка *MSIL*. Она «на лету» преобразует нужные для исполнения части кода в команды реального процессора, который фактически и выполняет код.

Наше приложение можно запустить, нажав комбинацию клавиш *Ctrl + F5* или выбрав пункт меню *Отладка → Запуск без отладки*. Оно будет выполнено *CLR*. В итоге появится консольное окно с приглашением нажать любую клавишу для закрытия окна.

Дополним наше приложение средствами ввода с консоли (клавиатуры) и вывода на консоль (экран) строки текста следующим образом:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleHello
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Введите Ваше имя");
            string name;
            name = Console.ReadLine();
            if (name == "")
                Console.WriteLine("Здравствуй, мир!");
            else
                Console.WriteLine("Здравствуй, " + name + "!");

        }
    }
}
```

Здесь использованы методы *ReadLine()* и *WriteLine()* класса *Console* из библиотеки *FCL*. Запустим приложение, нажав комбинацию

клавиш *Ctrl + F5* или выбрав пункт меню *Отладка → Запуск без отладки*. Пример выполнения этого приложения показан на рис. 3.

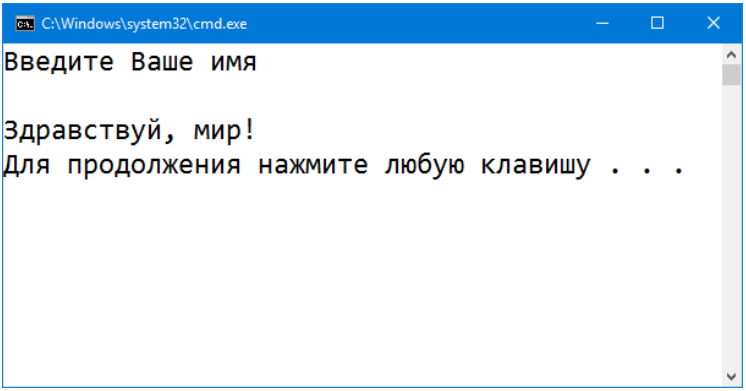


Рис. 3. Пример выполнения консольного приложения

2. Создание приложения Windows

Теперь создадим приложение Windows, изучим его вид по умолчанию, а затем дополним элементами управления Windows для ввода и вывода сообщений.

2.1. Создание приложения Windows и принцип его работы

Щелкнем пункты меню *Файл → Создать → Проект*, выберем язык *Visual C#* и тип проекта – *Приложение Windows Forms*. Дадим проекту имя *WindowsHello* и щелкнем кнопку *OK*. В итоге по умолчанию будет построено решение, содержащее единственный проект с пространством имен. Если в окне *Обозреватель решений* дважды щелкнуть узел *Program.cs*, то построенное по умолчанию решение отобразится в окне документов (рис. 4).

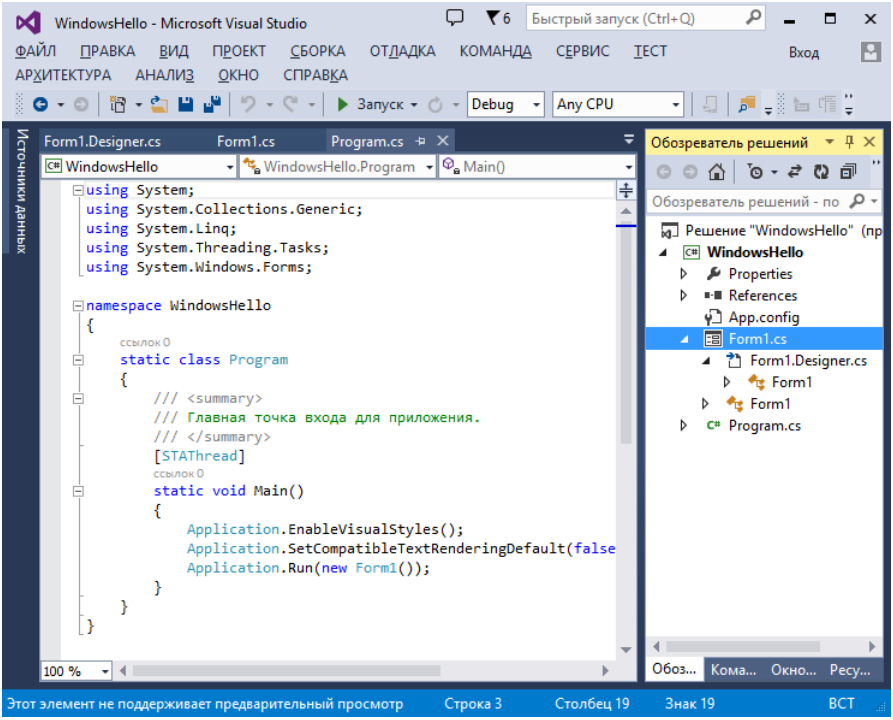


Рис. 4. Среда разработки с приложением Windows по умолчанию

Это приложение содержит следующие компоненты:

• папка *Properties*, содержащая файл *AssemblyInfo.cs* для сборки;

• папка *References* со ссылками на системные пространства имен из *FCL*;

• файл *Form1.cs*, содержащий класс *Form1*;

• файл *Program.cs*, содержащий класс *Program*, в котором определена функция *Main()*, являющаяся точкой входа.

Класс *Form1*, вложенный в пространство имен *WindowsHello*, определяет графический интерфейс пользователя.

Просмотрите исходные коды созданного приложения, раскрывая двойным кликом элементы в *Обозревателе решений*.

Откройте файл *Form1.cs*. Пространству имен предшествуют набор предложений *using*, т.е. используются множество классов из разных

пространств имен библиотеки *FCL*. Один из них – класс *Form* из пространства имен *System.Windows.Forms*. Класс *Form1*, построенный по умолчанию, является наследником класса *Form* и автоматически наследует его свойства, методы и события. *Visual Studio*, создавая объект класса *Form1*, одновременно создает его графический образ – окно с именем *Form1*. Для того, чтобы отобразить окно *Form1* в окне документов дважды щелкните узел *Form1.cs* в окне *Обозреватель решений* (рис. 5).

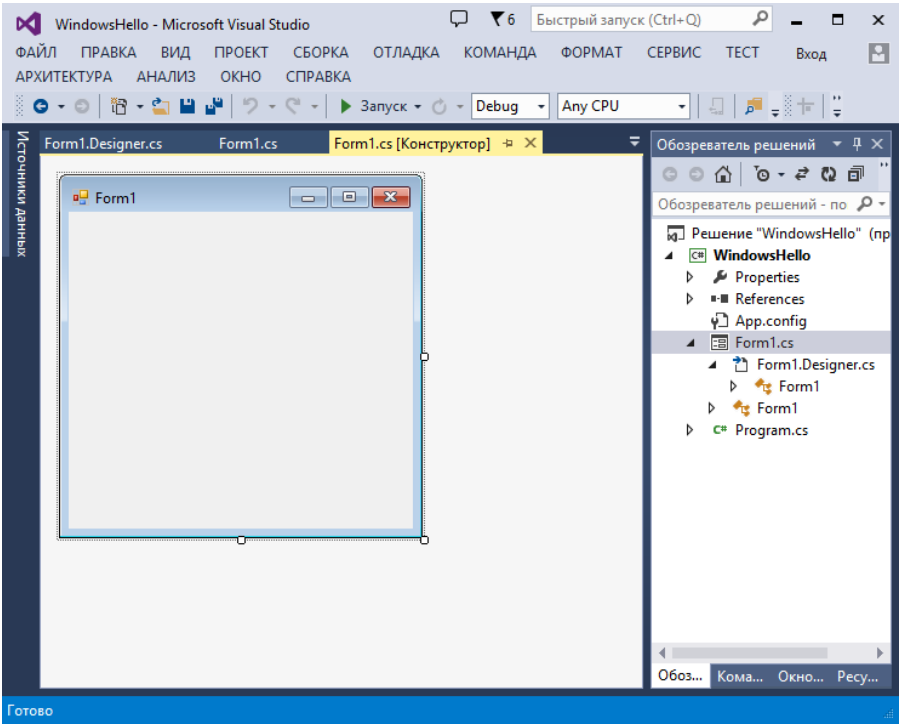


Рис. 5. Среда разработки с окном *Form1* по умолчанию

Программист заполняет окно *Form1* элементами управления *Windows*, перетаскивая их при помощи мыши из окна *Панель инструментов* в окно *Form1*. При этом программный код класса автоматически обновляется соответствующим образом. Появление в

нашем проекте формы *Form1*, открывающейся на экране при запуске приложения, означает, что мы перешли к визуальному программированию, т.е. к созданию приложений, управляемых событиями.

Класс *Form1* содержит закрытое (*private*) свойство в виде объекта *components* класса *IContainer* и конструктор, вызывающий закрытый метод *InitializeComponent*.

Когда система *CLR* получает сборку для выполнения, то в решении, входящем в сборку, отмечен стартовый проект, содержащий класс с методом *Main()*. Определенный объект системы *CLR* вызывает метод *Main()*. Обычно метод *Main()* создает один или несколько объектов, а затем вызывает для них методы и/или обработчики событий. В этих методах и обработчиках событий, в свою очередь, могут создаваться новые объекты и для них вызываться новые методы и обработчики событий. Таким образом, постепенно приходит в движение большой и разнообразный мир объектов приложения.

В нашем приложении метод *Main()*содержит строку

```
Application.Run(new Form1());
```

Здесь вызывается метод *Run()* класса *Application* из пространства имен *System.Windows.Forms*. Методу *Run()* в качестве аргумента передается выражение *new Form1()*, при вычислении которого создается первый объект класса *Form1*. Для его инициализации вызывается конструктор класса *Form1*, который, в свою очередь, вызывает метод *InitializeComponent()*. По завершении работы конструктора объект класса *Form1* передается в качестве аргумента методу *Run()*.

Метод *Run()* открывает на экране форму (графический образ объекта класса *Form1*), с которой теперь может работать пользователь. Но главная работа метода *Run()* состоит в том, что он запускает цикл обработки сообщений о событиях. Поступающие сообщения о событиях обрабатываются операционной системой согласно очереди и приоритетам при помощи соответствующих обработчиков событий.

Поскольку наша форма по умолчанию не снабжена никакими элементами управления, то возможных для нее событий немного. Действительно, запустим наше приложение, нажав комбинацию клавиш *Ctrl + F5* или выбрав пункт меню *Отладка → Запуск без*

отладки, и убедимся, что пользователь может делать с формой лишь следующие действия – перетаскивать ее по экрану, свертывать и изменять размеры. Наконец, он может закрыть форму, что приведет к завершению цикла обработки сообщений о событиях, завершению работы методов *Run()* и *Main()* и, соответственно, завершению работы приложения.

2.2. Дополнение приложения Windows элементами управления

Откроем нашу форму в режиме проектирования. Для этого нужно сделать двойной щелчок на узле *Form1.cs* в окне *Обозреватель решений* перейти на вкладку *Form1.cs [Конструктор]* в окне документов. Откройте окно *Свойства*, выполнив команду *Вид → Окно свойств* или нажав клавишу *F4*. Найдите свойство *Text* и установите значение *Hello*.

Откройте *Панель элементов* (*Вид → Панель элементов, Ctrl + Alt + X*). Найдите в группе *Стандартные элементы управления* и перенесите на форму при помощи мыши следующие элементы (рис. 6):

• Текстовое окно *TextBox*. По умолчанию оно получит имя *textBox1*. Это окно служит для ввода имени пользователя. Установите для него в свойстве *Multiline* значение *True*.

• Текстовое окно *TextBox*. По умолчанию оно получит имя *textBox2*. Это окно служит для вывода приветствия. Установим для него в свойствах *ReadOnly* и *Multiline* значение *True*.

• Метка *Label*. По умолчанию она получит имя *label1*.Установим для нее в свойстве *Text* значение *Your Name*.

• Метка *Label*. По умолчанию она получит имя *label2*.Установим для нее в свойстве *Text* значение *Hello Text*.

• Кнопка *Button*. По умолчанию она получит имя *button1*. Установим для нее в свойстве *Text* значение *OK*. Позже мы напишем для этой кнопки обработчик события «щелчок мыши» (*button1Click()*), который в ответ на щелчок мыши по кнопке будет считывать имя пользователя из окна *textBox1* и выводить текст приветствия в окно *textBox2*.

Вы можете изменить размеры элементов управления и формы. При выделении элемента вокруг него появляется рамка. Потянув мышью за ее углы или стороны, вы можете настроить размеры элемента.

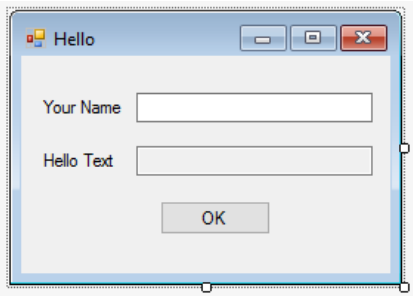


Рис. 6. Форма приложения с элементами управления

Важно, что все выполненные нами действия по проектированию графического интерфейса пользователя нашего приложения автоматически были преобразованы в программный код и добавлены в класс *Form1*. Как только мы вручную добавляем в форму элемент управления, в класс формы сразу же добавляется определяющее его закрытое свойство, а в методе *InitailizeComponent* – операторы его инициализации. Если мы потом изменяем свойство элемента управления, то это сразу отражается в программном коде в методе *InitailizeComponent*.

В нашем случае в классе формы, который отображается, если дважды щелкнуть узел *Form1.Designer.cs* в окне *Обозреватель решений*, появились следующие определения элементов управления:

```
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.TextBox textBox2;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.Button button1;
```

Теперь чуть выше этих определений дважды щелкните по рамке с надписью *Код, автоматически созданный конструктором форм Windows* (*Windows Form Designer generated code*), а затем – по рамке с многоточием. Тогда отобразится код метода *InitailizeComponent*. Он принял следующий вид:

```
#region Код, автоматически созданный конструктором форм Windows

/// <summary>
/// Обязательный метод для поддержки конструктора - не изменяйте
/// содержимое данного метода при помощи редактора кода.
/// </summary>
private void InitializeComponent()
{
  this.textBox1 = new System.Windows.Forms.TextBox();
  this.textBox2 = new System.Windows.Forms.TextBox();
  this.label1 = new System.Windows.Forms.Label();
  this.label2 = new System.Windows.Forms.Label();
  this.button1 = new System.Windows.Forms.Button();
  this.SuspendLayout();
  //
  // textBox1
  //
  this.textBox1.Location = new System.Drawing.Point(78, 25);
  this.textBox1.Multiline = true;
  this.textBox1.Name = "textBox1";
  this.textBox1.Size = new System.Drawing.Size(160, 20);
  this.textBox1.TabIndex = 0;
  //
  // textBox2
  //
  this.textBox2.Location = new System.Drawing.Point(78, 61);
  this.textBox2.Multiline = true;
  this.textBox2.Name = "textBox2";
  this.textBox2.ReadOnly = true;
  this.textBox2.Size = new System.Drawing.Size(160, 20);
  this.textBox2.TabIndex = 1;
  //
  // label1
  //
  this.label1.AutoSize = true;
  this.label1.Location = new System.Drawing.Point(12, 28);
  this.label1.Name = "label1";
  this.label1.Size = new System.Drawing.Size(60, 13);
  this.label1.TabIndex = 2;
  this.label1.Text = "Your Name";
  //
```

```
    // label2
    //
    this.label2.AutoSize = true;
    this.label2.Location = new System.Drawing.Point(12, 64);
    this.label2.Name = "label2";
    this.label2.Size = new System.Drawing.Size(55, 13);
    this.label2.TabIndex = 3;
    this.label2.Text = "Hello Text";
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(94, 98);
    this.button1.Name = "button1";
    this.button1.Size = new System.Drawing.Size(75, 23);
    this.button1.TabIndex = 4;
    this.button1.Text = "OK";
    this.button1.UseVisualStyleBackColor = true;
    //
    // Form1
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(250, 147);
    this.Controls.Add(this.button1);
    this.Controls.Add(this.label2);
    this.Controls.Add(this.label1);
    this.Controls.Add(this.textBox2);
    this.Controls.Add(this.textBox1);
    this.Name = "Form1";
    this.Text = "Hello";
    this.ResumeLayout(false);
    this.PerformLayout();

}

#endregion
```

Заметьте, тег <*summary*> сообщает нам, что метод *InitailizeComponent* нужен служебному блоку конструктора, и предостерегает от его изменения в редакторе кода. Генерирование и

изменение кода этого метода должно происходить только автоматически.

Обратите также внимание на то, что элементы управления после задания их свойств добавляются в коллекцию *Controls* при помощи метода *Add()*.

Теперь напишем код обработчика события для кнопки *OK* на нашей форме. Начать создание обработчика события любого элемента управления на форме можно разными способами. Стандартный способ состоит в том, чтобы выделить элемент управления на форме, щелкнуть в окне *Свойства* кнопку со значком молнии (*События*), и в открывшемся списке событий сделать двойной щелчок на нужном событии. В нашем случае можно поступить проще – сделать двойной щелчок на кнопке *OK*. В любом случае открывается окно редактора кода со следующей автоматически созданной пустой заготовкой обработчика события:

```
private void button1_Click(object sender, EventArgs e)
{

}
```

Впишем в тело этого обработчика события следующие строки:

```
string temp;
temp = textBox1.Text;
if (temp == "")
  textBox2.Text = "Hello World!";
else
  textBox2.Text = "Hello " + temp + "!";
```

Запустим приложение, нажав комбинацию клавиш *Ctrl + F5* или выбрав пункт меню *Отладка → Запуск без отладки*. Пример выполнения нашего приложения показан на рис. 7.

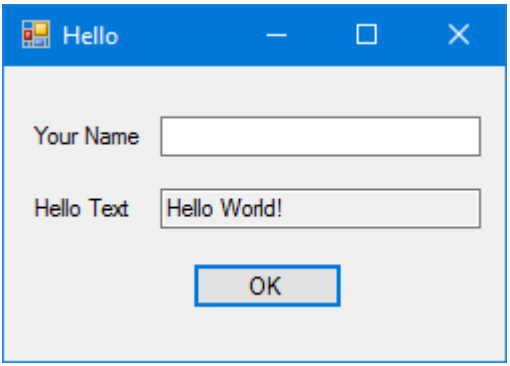


Рис. 7. Форма «Приветствие» в процессе работы

3. Разработка текстового редактора

Здесь вы изучите процесс создания текстового редактора, снабженного главным меню, панелью инструментов, строкой состояния и файлом справки.

3.1. Начало разработки нового приложения

1. Запустите *Visual Studio*. Создайте новый проект, выбрав пункты меню *Файл* → *Создать* → *Проект* (*Ctrl* + *Shift* + *N*). Также для создания проекта можно нажать ссылку «*Создать проект*» на *Начальной странице* (рис. 8). Откроется окно «*Создать проект*».

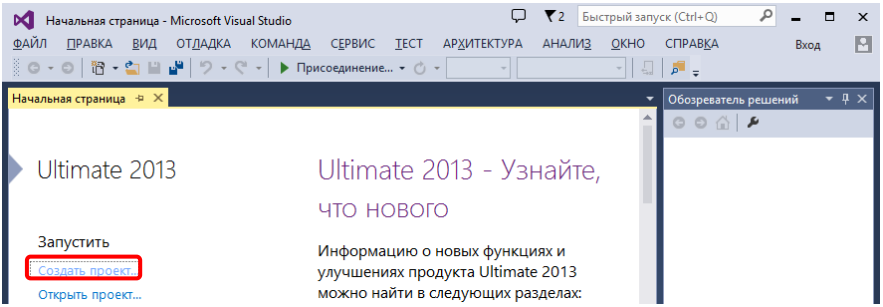


Рис. 8. Начальная страница *Visual Studio*

2. В окне «*Создать проект*» (рис. 9) выберите тип создаваемого приложения. Для этого в левой части окна перейдите в раздел «*Visual C#*» и выберите подраздел «*Рабочий стол Windows*». В центральной части окна щелкните на строке «*Приложение Windows Forms*».

Введите *Имя проекта*. Также можно указать *Расположение* – папку, в которой будут сохраняться файлы проекта. Нажмите кнопку *OK*.

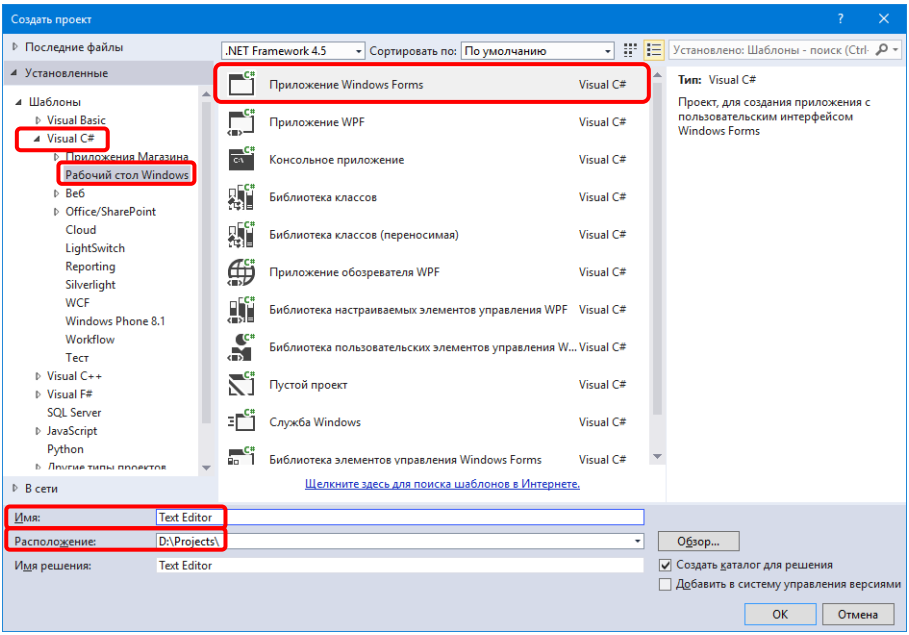


Рис. 9. Создание нового проекта

3. Когда проект будет создан, вы увидите пустое окно (форму) на вкладке *Form1.cs [Конструктор]*. На этой форме будем размещать различные элементы интерфейса – меню, панель инструментов, окно ввода текста и другие.

Заданная по умолчанию форма *Form1* имеет кнопки свертывания, восстановления и закрытия окна, а также управляющее меню. Если вы запустите эту форму, нажав клавишу *F5* (меню *Отладка* → *Начать*

отладку), то увидите, что все эти кнопки работают. Чтобы вернуться в режим разработки, щелкните кнопку закрытия окна(×).

4. Исходный код, в котором содержится описание формы, находится в файле *Form1.cs*. Чтобы перейти к исходному коду, описывающему форму, выберите в меню *Вид* → *Код* (*F7*). Для перехода к визуальному представлению выполните команду *Вид* → *Конструктор* (*Shift* + *F7*).

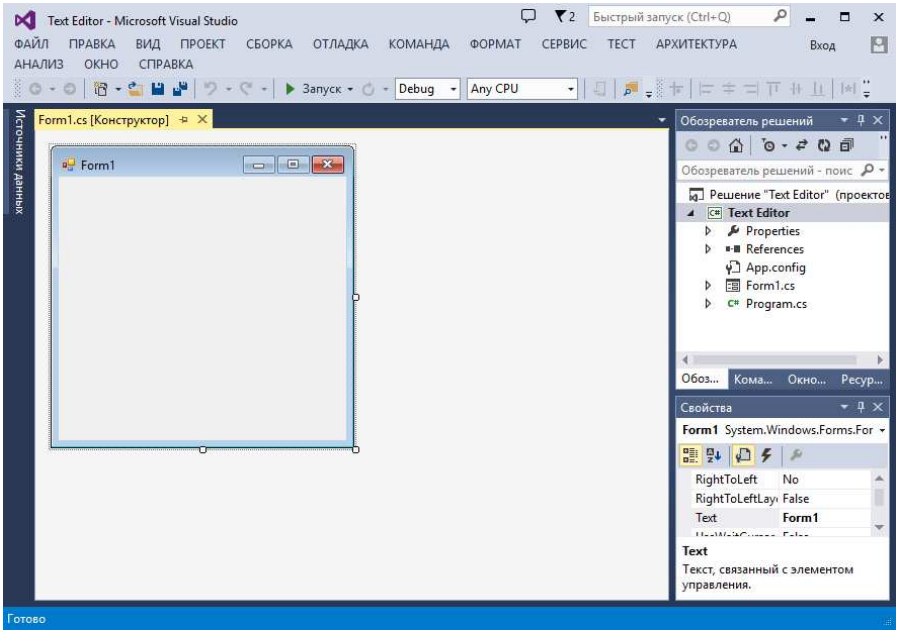


Рис. 10. Пустая форма разрабатываемого приложения

5.Посмотрите исходный код для формы. Приведем его здесь, дополнив комментариями.

```
// Директива using разрешает использование типов
// в пространстве имен (можно считать, что
// подключает необходимые библиотеки)
using System;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

// Пространство имен нашего приложения
namespace Text_Editor
{
    // Класс Form1 - это описание окна нашего приложения.
    // Он содержит только одну функцию - конструктор
    public partial class Form1 : Form
    {
        // Конструктор класса Form1 - это функция,
        // которая вызывается для создания окна.
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

6.Откройте обозреватель решений (*Вид* → *Обозреватель решений*, *Ctrl* + *Alt* + *L*).

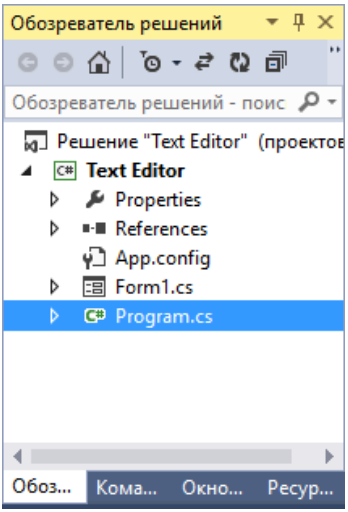


Рис. 11. Обозреватель решений

В обозревателе решений представлена структура проекта. Здесь вы можете найти файл *Form1.cs*. Также в проекте имеется файл *Program.cs*, в котором имеется точка входа в приложение – функция *Main()*. Откройте его.

7.Изучите исходный код *Program.cs*. Приведем его здесь.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Text_Editor
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
```

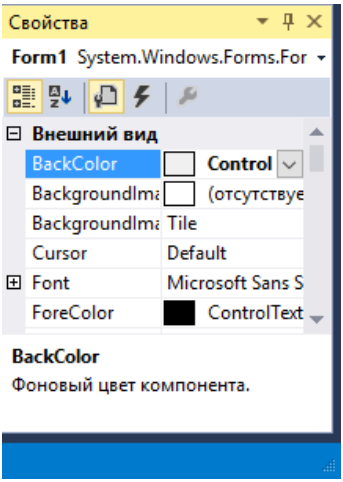
```
        static void Main()
        {
            Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Для исполнения программы запускается функция *Main()* из класса *Program*. Окно формы создается оператором new и передается методу *Run()* класса *Application*. Класс *Application* предоставляет методы и свойства для управления приложением (методы для запуска и остановки приложения, для обработки сообщений *Windows* и свойства для получения сведений о приложении).

8. Сохраните ваши файлы. Для этого выберите пункты меню *Файл* → *Сохранить все* (*Ctrl* + *Shift* + *S*).

3.2. Установка значений свойств

1. Перейдите к форме, дважды щелкнув строку *Form1.cs* в *Обозревателе решений*. Щелкните на визуальном представлении формы. Вызовите окно свойств (*Вид* → *Окно свойств*, *F4*).



Раскрывающийся список показывает выбранный объект. Здесь выбран объект *Form1* типа *System.Windows.Forms.Form*.

Для выбранного объекта отображаются все доступные свойства и события (для переключения отображения используйте кнопки). Найдите свойство *BackColor* и поменяйте его значение, выбрав из раскрывающегося списка справа от имени свойства.

Рис. 12. Настройка свойств формы

2. Просмотрите, какие свойства есть у формы. Описание свойств можно найти в *MSDN* (https://msdn.microsoft.com/ru-ru/library/system.windows.forms.form(v=vs.110).aspx). Попробуйте изменять значения свойств. Для корректной работы разрабатываемого приложения верните значения свойств, указанные по умолчанию. Для этого щелкните правой кнопкой мыши на имени свойства и выберите в контекстном меню пункт *Сброс*.

3. Измените заголовок формы (свойство *Text*) на *Text Editor Tutorial*. Убедитесь, что заголовок на визуальном представлении формы изменился.

3.3. Добавление объектов в форму

Прежде чем начать добавлять объекты в форму, вы должны тщательно спроектировать интерфейс пользователя вашего приложения. Он должен быть гибким, удобным и простым в использовании. С этой точки зрения ваше приложение (текстовый редактор) должно предоставлять пользователю окно редактирования, главное меню, панель инструментов для быстрого доступа к командам и строку состояния. Проектирование интерфейса пользователя в

конструкторе форм *Visual Studio* удобно тем, что вы можете экспериментировать с готовыми объектами и сразу наблюдать их работу, т.е. динамически моделировать интерфейс пользователя вашего приложения.

Visual Studio содержит обширную библиотеку визуальных компонентов, моделирующих разные части приложений. Например, есть компоненты, облегчающие программирование меню, панелей инструментов, диалоговых окон и т.п. *Панель элементов* представляет компоненты приложения в виде значков, сгруппированных по типам. Для открытия *Панели элементов* перейдите к визуальному представлению формы и выполните команду *Вид* → *Панель элементов* (*Ctrl* + *Alt* + *X*). Для добавления компонента в форму выберите его на палитре, а затем щелкните на форме точку, куда хотите его поместить. Вы можете также перетащить элемент из палитры в нужное место формы или просто дважды щелкнуть значок компонента, чтобы поместить его на форму. Чтобы получить справку по компоненту, выберите его (на панели элементов или форме) и нажмите клавишу `F1`.

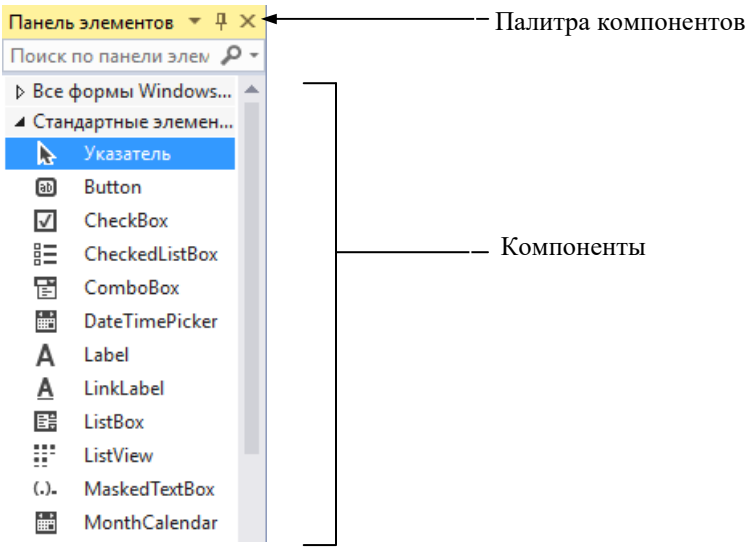


Рис. 13. Панель элементов

1. Начните разработку текстового редактора с создания окна редактирования текста. Для этого на *Панели элементов* раскройте группу *Стандартные элементы управления* и найдите на ней значок компонента *RichTextBox*. Установите на форму компонент *RichTextBox*:

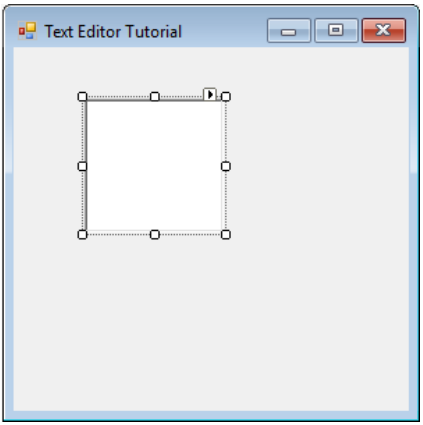


Рис. 14. Установка компонента *RichTextBox*

Каждый компонент – это класс. Когда вы добавляете компонент в форму, *Visual Studio* генерирует код, обеспечивающий создание экземпляра соответствующего класса в вашем приложении. Находится этот код в файле *Form1.Designer.cs*.

Откройте файл *Form1.Designer.cs* и убедитесь, что там появилась строка:

```
private System.Windows.Forms.RichTextBox richTextBox1;
```

2. Перейдите в конструктор формы и щелкните компонент *richTextBox1*. Найдите свойство *Dock* и установите значение *Fill*, как показано на рисунке 15.

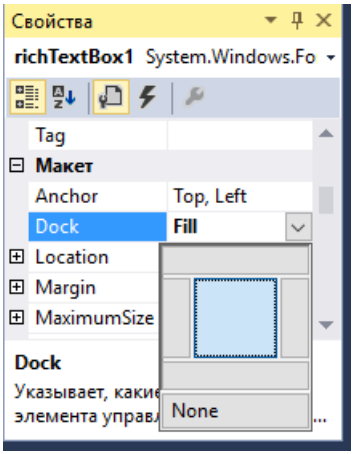


Рис. 15. Настройка параметров размещения компонента *RichTextBox*

Теперь компонент *richTextBox1* заполняет всю форму, поэтому у вас есть крупное окно редактирования текста. Выбор значения *Fill* для свойства *Dock* изменяет размер элемента управления *richTextBox1* так, что он заполняет отображаемое окно любого размера, даже если размеры формы изменены.

3. Найдите в панели элементов группу *Меню и панели инструментов* и дважды щелкните компонент *StatusStrip*. Тогда в нижней части формы добавится строка состояния.

Теперь надо создать место для отображения имени редактируемого файла. Щелкните на строке состояния *statusStrip1* в нижней части формы. Вы увидите кнопку с выпадающим меню: . Выберите в меню пункт *StatusLabel*:

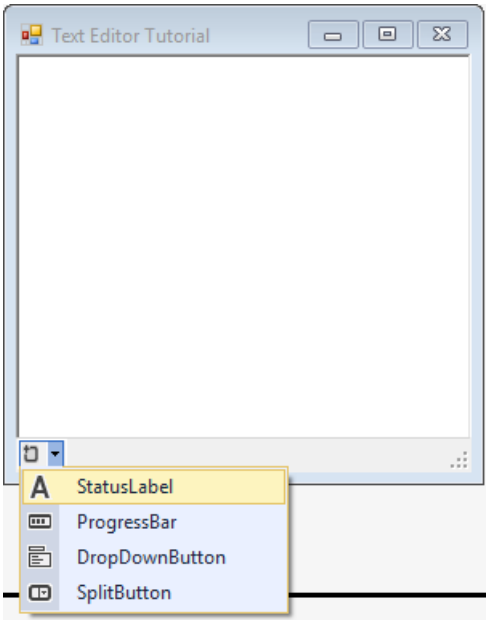


Рис. 16. Настройка строки состояния

На строке состояния появится новый компонент *toolStripStatusLabel1*. В свойстве *Text* вы можете установить любой текст, который вы хотите отображать. Программно вы можете вывести текст следующим образом:

```
toolStripStatusLabel1.Text = "untitled.txt";
```

Вы можете добавить на строку состояния несколько компонентов для вывода текста. Также можно установить индикатор выполнения процесса (*ProgressBar*) или различные кнопки (*DropDownButton SplitButton*).

3.4. Добавление поддержки меню и панели инструментов

Для работы с текстовым редактором пользователю нужны меню и команды, а для удобного запуска команд – панель инструментов (ПИ).

Планирование команд текстового редактора

Команда	Меню	ПИ	Описание
New	*File*	+	Создает новый файл.
Open	*File*	+	Открывает существующий файл для редактирования.
Save	*File*	+	Сохраняет текущий файл на диске.
Save As	*File*	–	Сохраняет файл с новым именем (позволяет сохранить новый файл с заданным именем).
Exit	*File*	+	Завершает работу текстового редактора.
Cut	*Edit*	+	Удаляет текст и сохраняет его в буфере обмена.
Copy	*Edit*	+	Копирует текст в буфер обмена.
Paste	*Edit*	+	Вставляет текст из буфера обмена.
Contents	*Help*	+	Отображает оглавление справки, по которому вы можете обратиться к разделам справки.
About	*Help*	–	Отображает информацию о приложении в окне.

3.4.1. Добавление меню

Добавьте в форму главное меню с тремя выпадающими меню – *File*, *Edit* и *Help*, а затем добавьте в каждое из них команды, используя список действий. Для этого выполните следующее.

1. Найдите в панели элементов группу *Меню и панели инструментов* и дважды щелкните компонент *MenuStrip*. Тогда в верхней части формы добавится строка меню.

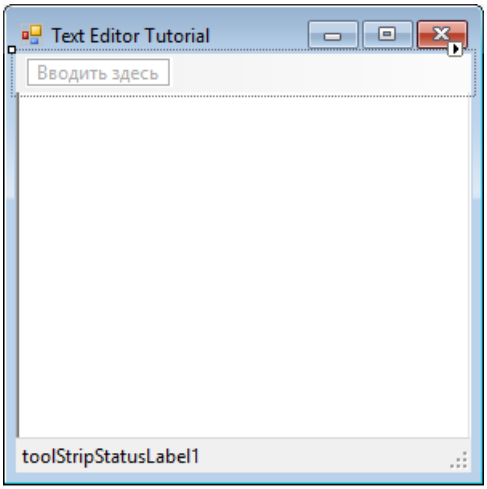


Рис. 17. Создание меню

2. Щелкните по полю «Вводить здесь» на строке меню. Введите название первого пункта меню «&*File*» и нажмите *Enter*. Вы увидите, что буква F стала подчеркнутой. Это нужно, чтобы быстро выполнять команды меню, используя клавиатуру.

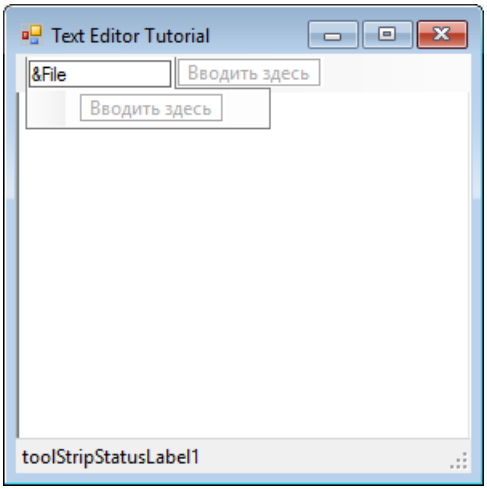


Рис. 18. Создание меню

3. Щелкните «*Вводить здесь*» справа от команды «*File*». Добавьте еще один пункт меню «*&Edit*». Аналогичным образом добавьте команду «*&Help*».

4. Щелкните по пункту меню «*File*», затем щелкните «Вводить здесь» под ним и добавьте команды «*&New*», «*&Open*», «*&Save*», «*Save &as*» и «*E&xit*».

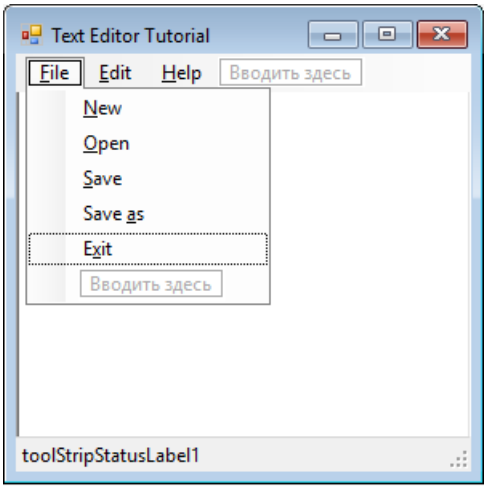


Рис. 19. Создание меню

5. Между командами «*Save as*» и «*Exit*» добавьте разделитель. Для этого введите «-» в качестве команды меню и нажмите *Enter*. Получится разделительная черта. Затем перетащите мышью эту черту вверх или пункт «*Exit*» вниз. Таким образом вы можете разделять и менять местами команды меню.

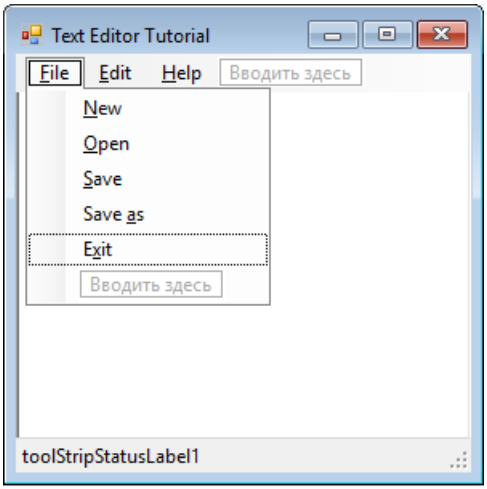


Рис. 20. Создание разделителей в меню

6. Добавьте в меню «*Edit*» команды «*C&ut*», «*&Copy*» и «*&Paste*».

7. Также добавьте в меню «*Help*» команды «*&Contents*» и «*&About*», а также разделитель между ними.

8. Запустите приложение. Нажмите клавишу *Alt*, а затем *H*. Откроется меню «*Help*». Нажмите клавишу *A*, и меню закроется. На самом деле мы вызвали команду «About», но пока она ничего не выполняет.

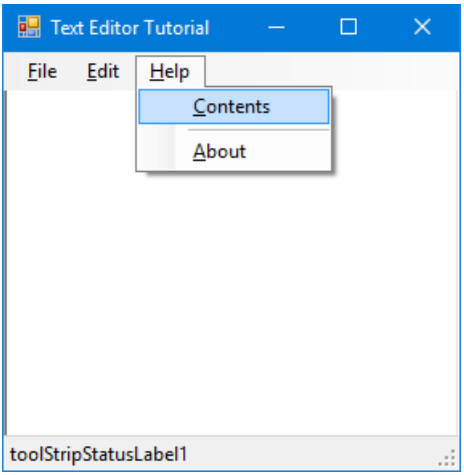


Рис. 21. Проверка работоспособности меню

9. Установите сочетания клавиш для быстрого вызова команд без использования меню. Для этого задайте для команд значение свойства *ShortcutKeys*, как указано в таблице. Обратите внимание, что в меню справа от имени команды отображаются сочетания клавиш.

Команда	Свойство *ShortcutKeys*
File → New	`Ctrl + N`
File → Open	`Ctrl + O`
File → Save	`Ctrl + S`
File → Exit	`Ctrl + Q`
Edit → Cut	`Ctrl + X`
Edit → Copy	`Ctrl + C`
Edit → Paste	`Ctrl + V`
Help → Contents	`Ctrl + H`

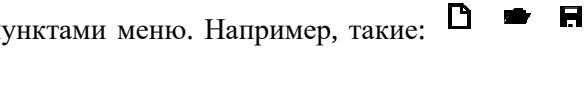
10. Добавьте всплывающие подсказки, отображающиеся при наведении курсора на пункты меню. Для этого установите свойство

ToolTipText, как указано в таблице. После этого запустите приложение и убедитесь, что подсказки работают.

Команда	Свойство *ToolTipText*
File → New	`Create a new file`
File → Open	`Open an existing file`
File → Save	`Save file`
File → Save as	`Save the file with a different name or in a different folder`
File → Exit	`Exit application`
Edit → Cut	`Cut the selected text`
Edit → Copy	`Copy the selected text`
Edit → Paste	`Paste text from clipboard`
Help → Contents	`Show help`
Help → About	`About the program`

3.4.2. Добавление изображений в список ресурсов приложения

Добавим несколько изображений, которые будут отображаться рядом с названиями пунктов меню и на кнопках панели инструментов. Такие изображения относятся к *ресурсам* приложения.

1. Подготовьте набор файлов с пиктограммами, которые будут отображаться рядом с пунктами меню. Например, такие:  .

2. Откройте свойства проекта, выполнив команду *Проект → Свойства: Text Editor…*

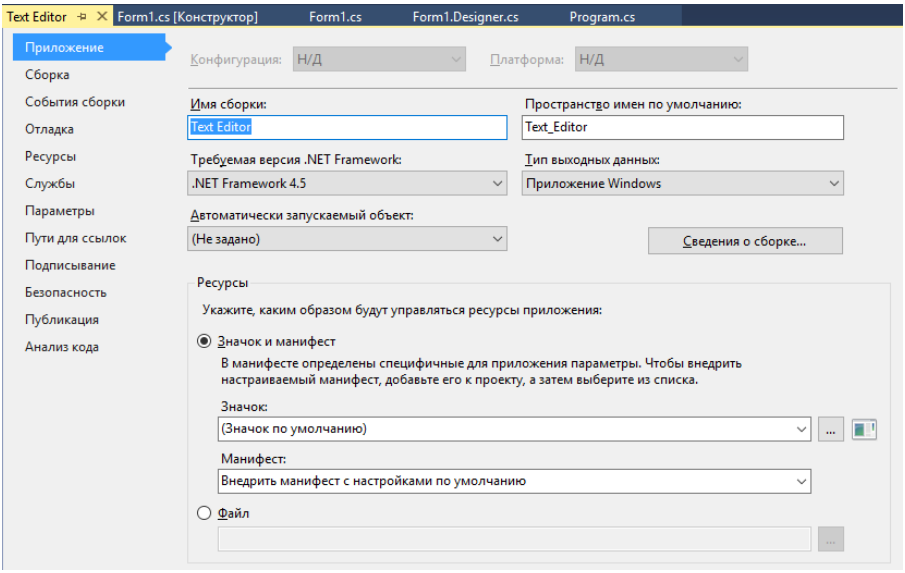


Рис. 22. Свойства проекта

3. Перейдите в раздел «*Ресурсы*». На кнопке «*Добавить ресурс*» имеется вложенное меню (раскрывается при нажатии на треугольник). Выберите пункт «*Добавить существующий файл*».

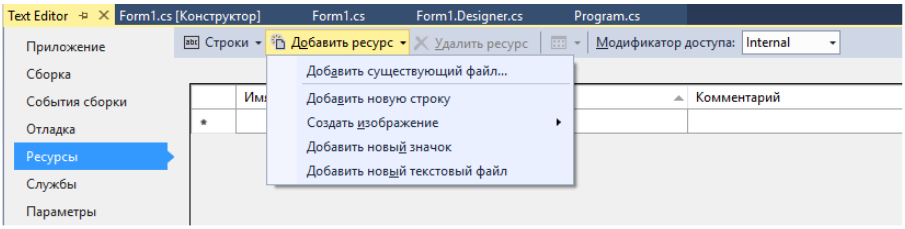


Рис. 23. Ресурсы проекта

4. Откроется окно выбора файлов. Выберите ранее подготовленные файлы.

5. Для того чтобы к команде меню добавить изображение, установите свойство *Image*. При нажатии на кнопку ... откроется окно

«*Выбор ресурса*». Выберите в списке файлов ресурсов проекта необходимое изображение и нажмите кнопку «*ОК*».

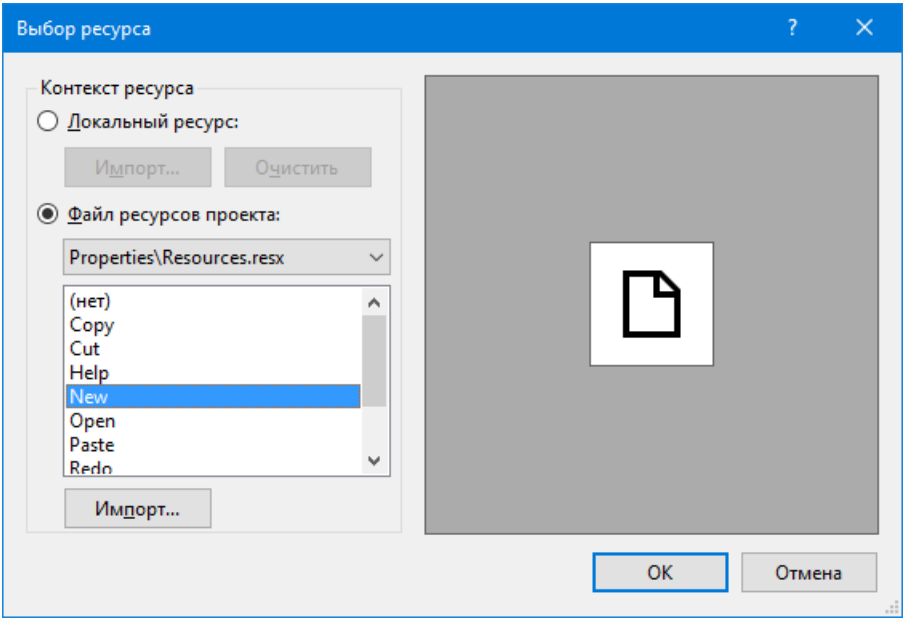


Рис. 24. Выбор ресурса (изображения)

6. Установите изображения для нескольких команд меню. Запустите приложение и убедитесь, что рядом с командами меню появились пиктограммы.

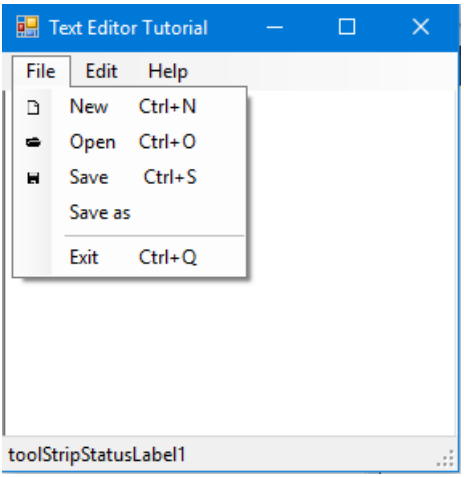


Рис. 25. Меню с пиктограммами

3.4.3. Добавление панели инструментов

1. Найдите в панели элементов группу *Меню и панели инструментов* и дважды щелкните компонент *ToolStrip*. Тогда в верхней части формы добавится пустая панель инструментов.
2. Щелкните панели инструментов *toolStrip1*. Вы увидите кнопку с выпадающим меню:  . Выберите в меню пункт *Button*, и на панели инструментов добавится кнопка.
3. Аналогичным образом создайте на панели инструментов еще две кнопки.
4. Добавьте разделитель групп кнопок, выбрав пункт «*Separator*».
5. Добавьте еще три кнопки, разделитель и еще одну кнопку:

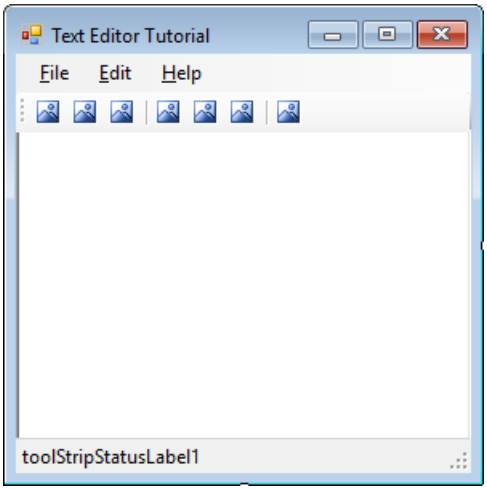


Рис. 25. Панель инструментов

6. Настройте свойства *Image*, *Text* и *ToolTipText* для кнопок панели инструментов. Первые три кнопки соответствуют командам для создания, открытия и сохранения файла. Следующие кнопки – для редактирования текста: копировать, вырезать и вставить фрагмент текста. Последняя кнопка вызывает справку.

7. Вы завершили расстановку компонентов пользовательского интерфейса на форме. Используя *Обозреватель решений*, откройте файл *Form1.Designer.cs* и просмотрите его содержимое. В классе *Form1* вы найдете объявление всех добавленных компонентов (приведем фрагмент кода):

```
private System.Windows.Forms.RichTextBox richTextBox1;
private System.Windows.Forms.StatusStrip statusStrip1;
private System.Windows.Forms.ToolStripStatusLabel toolStripStatusLabel1;
private System.Windows.Forms.MenuStrip menuStrip1;
private System.Windows.Forms.ToolStripMenuItem fileToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem editToolStripMenuItem;
```

8. Запустите приложение и убедитесь, что кнопки на панели инструментов имеют изображения и всплывающие подсказки.

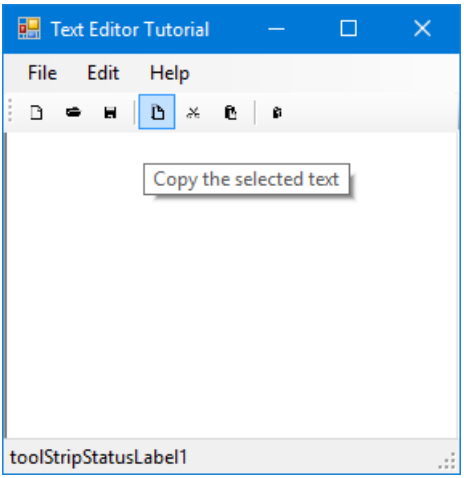


Рис. 26. Тестирование панели инструментов

3.5. Создание обработчиков событий

До сих пор вы проектировали приложение без написания программного кода. Размещая объекты на форме и задавая значения их свойств, вы извлекли полную выгоду из услуг *Visual Studio* по генерированию исходного кода. Теперь вы напишете на языке *C#* собственные функции, реагирующие на действия пользователя во время выполнения приложения. Такие функции называются *обработчиками событий*. Вы подключите их к командам меню и кнопкам панели инструментов так, чтобы в результате выбора команды или кнопки выполнялся код соответствующего обработчика события.

3.5.1. Создание обработчика событий для команды *New*

1. Команда *New* приводит к созданию нового файла. Для того чтобы хранить имя редактируемого файла, добавим переменную *fileName* типа *String*. Для этого выполните следующие действия:

- откройте файл *Form1.cs* и найдите строки

```
public partial class Form1 : Form
    {
```

- Добавьте объявление переменной в описание класса:

```
public partial class Form1 : Form
    {
        string fileName; // имя редактируемого файла
```

2. Теперь добавим переменную *defaultFileName* к ресурсам приложения, чтобы хранить имя файла по умолчанию. Откройте окно ресурсов приложения, выполните команду *Проект → Свойства: Text Editor…* и перейдите в раздел «*Ресурсы*». На кнопке «*Добавить ресурс*» откройте вложенное меню и выберите пункт «*Добавить новую строку*». Введите имя ресурса «*defaultFileName*» и значение «*untitled.rtf*».

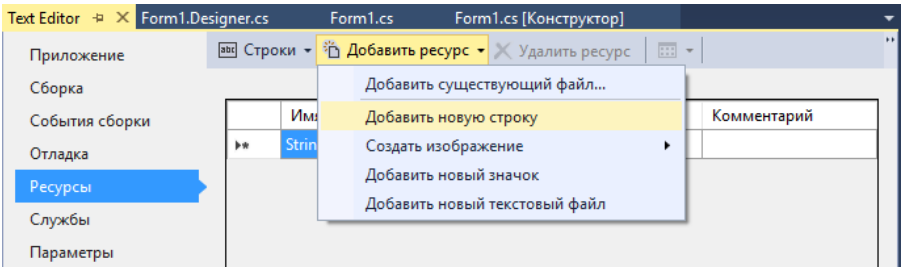


Рис. 27. Добавление ресурса (строки)

3. Перейдите к файлу *Form1.cs* и найдите конструктор класса *Form1*. Добавьте инициализацию переменной *fileName* и вывод имени файла на строку состояния (строки, выделенную жирным шрифтом):

```
        public Form1()
        {
            InitializeComponent();
            fileName = Properties.Resources.defaultFileName;
            toolStripStatusLabel1.Text = fileName;
        }
```

4. Перейдите к окну конструктора формы, нажмите пункт меню «*File*» и выполните двойной щелчок на команде «*New*». Откроется окно редактора кода с курсором в обработчике событий. Здесь вы можете описать действия, выполняемые в случае выбора пользователем команды «*New*». Введите строки, выделенные жирным шрифтом:

```
private void newToolStripMenuItem_Click(object sender, EventArgs
e)
{
  richTextBox1.Clear();
  fileName = Properties.Resources.defaultFileName;
  toolStripStatusLabel1.Text = fileName;
}
```

В приведенном коде сначала выполняется очистка окна редактирования текста. Затем в переменную *fileName* записывается имя файла по умолчанию. Оно берется из ресурсов приложения. После этого имя файла отображается на строке состояния.

3.5.2. Создание обработчика событий для команды *Open*

Когда вы в приложении выполняете команду *File* → *Open*, должно открываться диалоговое окно для выбора открываемого файла. Для этого необходимо добавить компонент *OpenFileDialog*.

1. Перейдите в конструктор формы, найдите в панели элементов группу *Диалоговые окна* и дважды щелкните компонент *OpenFileDialog*. На форму добавится компонент *openFileDialog1*. Место расположения компонента *openFileDialog1* не имеет значения, поскольку его визуальное представление – это новое окно, которое будет открываться после соответствующей команды.

2. Задайте свойства объекта *openFileDialog1*. Для этого найдите компонент в выпадающем списке окна «Свойства» и выберите его.

- Для свойства Filter введите строку:

```
Rich text files|*.rtf|All files|*.*
```

- Свойство DefaultExt – `*.rtf`
- Свойство Title – Open file
- Очистите значение свойства FileName.

3. Перейдите к окну конструктора формы, нажмите пункт меню «*File*» и выполните двойной щелчок на команде «*Open*». Откроется окно редактора кода с курсором в обработчике событий. Здесь вы можете описать действия, выполняемые в случае выбора пользователем команды «*Open*». Введите строки, выделенные жирным шрифтом:

```
private void openToolStripMenuItem_Click(object sender,
                                         EventArgs e)
```

```
{
  if (openFileDialog1.ShowDialog() ==
      System.Windows.Forms.DialogResult.OK)
  {
    richTextBox1.LoadFile(openFileDialog1.FileName);
    fileName = openFileDialog1.FileName;
    toolStripStatusLabel1.Text = fileName;
  }
}
```

Разберем этот код. Метод *ShowDialog()* отображает окно выбора файлов. Когда пользователь нажимает кнопку *OK*, выполняются следующие действия: в окно редактирования загружается текст из выбранного пользователем файла, а имя файла сохраняется и отображается в строке состояния.

3.5.3. Создание обработчика событий для команды *Save As*

Когда вы в приложении выполняете команду *File* → *Save as*, должно открываться диалоговое окно для сохранения файла. Для этого необходимо добавить компонент *SaveFileDialog*.

1. Перейдите в конструктор формы, найдите в панели элементов группу *Диалоговые окна* и дважды щелкните компонент *SaveFileDialog*. На форму добавится компонент *saveFileDialog1*. Место расположения компонента *saveFileDialog1* не имеет значения, поскольку его визуальное представление – это новое окно, которое будет открываться после соответствующей команды.

2. Задайте свойства объекта *saveFileDialog1*. Для этого найдите компонент в выпадающем списке окна «Свойства» и выберите его.

- Для свойства Filter введите строку:
  `Rich text files|*.rtf|All files|*.*`
- Свойство DefaultExt – `*.rtf`
- Свойство Title – Save file

3. Перейдите к окну конструктора формы, нажмите пункт меню «*File*» и выполните двойной щелчок на команде «*Save as*». Откроется окно редактора кода с курсором в обработчике событий. Здесь вы можете описать действия, выполняемые в случае выбора пользователем команды «*Save as*». Введите строки, выделенные жирным шрифтом:

```
private void saveAsToolStripMenuItem_Click(object sender,
                                           EventArgs e)
{
  saveFileDialog1.FileName = fileName;
  saveFileDialog1.InitialDirectory =
    Directory.GetCurrentDirectory();
  if (saveFileDialog1.ShowDialog() ==
      System.Windows.Forms.DialogResult.OK)
  {
    richTextBox1.SaveFile(saveFileDialog1.FileName);
    fileName = saveFileDialog1.FileName;
    toolStripStatusLabel1.Text = fileName;
  }
}
```

Перед вводом второй строки добавьте в начало файла директиву `using System.IO;`

Самостоятельно проанализируйте действия, которые выполняются в данном обработчике событий.

3.5.4. Создание обработчика событий для команды *Save*

Чтобы создать обработчик событий для команды *Save*, перейдите к окну конструктора формы, нажмите пункт меню «*File*» и выполните двойной щелчок на команде «*Save*». Откроется окно редактора кода с курсором в обработчике событий. Введите строки, выделенные жирным шрифтом:

```
private void saveToolStripMenuItem_Click(object sender,
                                         EventArgs e)
{
  if (fileName.Equals(Properties.Resources.defaultFileName))
  {
     saveAsToolStripMenuItem_Click(sender, e);
  }
  else richTextBox1.SaveFile(fileName);
}
```

Здесь проверяется, совпадает ли имя файла с именем по умолчанию. Если имена одинаковые, то файл не сохранялся ранее, и нужно вызвать диалог для сохранения файла. Такое действие уже предусмотрено командой *Save as*, поэтому мы вызываем

соответствующий обработчик событий. Если же файл был ранее сохранен, то сохранение выполнится с тем же именем.

3.5.5. Создание обработчика событий для команды *Exit*

Создайте обработчик команды Exit и впишите всего один вызов функции:

```
private void exitToolStripMenuItem_Click(object sender,
                                         EventArgs e)
{
  Close();
}
```

Метод *Close()* приводит к закрытию формы и завершению работы приложения.

Скомпилируйте и запустите текстовый редактор. Проверьте, как выполняются команды меню файл. Проверьте, что нажатие сочетаний клавиш тоже приводит к выполнению соответствующих команд.

Подумайте, как можно улучшить работу редактора. Например, при выполнении команды *Exit* можно предложить сохранить файл, если текст был изменен. Попробуйте самостоятельно доработать программу.

3.5.6. Создание обработчика событий для команд редактирования

Команды редактирования достаточно просто реализовать, поскольку компонент *RichTextBox* имеет соответствующие методы:

```
richTextBox1.Cut();  // скопировать в буфер обмена
                     // выделенный фрагмент текста
                     // и удалить этот фрагмент
                     // из поля ввода.
richTextBox1.Copy(); // скопировать в буфер обмена
                     // выделенный фрагмент текста.
richTextBox1.Paste();// вставить фрагмент текста
                     // из буфера обмена
                     // в позицию курсора
```

Создайте обработчики событий для команд меню и поместите в них вызовы соответствующих методов.

Вы можете добавить команды отмены (*Undo*) и повтора (*Redo*) последнего действия. Добавьте эти команды в меню, назначьте им сочетания клавиш и пиктограммы, поместите на панель инструментов соответствующие кнопки.

3.5.7. Создание окна «About»

Многие приложения позволяют открыть окно *About* [*О программе*], которое отображает информацию о программе (название, версия, эмблема) и другие данные, например сведения об авторском праве. В этих целях выше в меню *Help* была добавлена команда команда *About*.

1. Выполните команду меню *Проект* → *Добавить форму Windows*. В открывшемся окне «*Добавление нового элемента*» найдите «Окно «О программе»». При необходимости введите имя новой формы. Нажмите кнопку «Добавить».

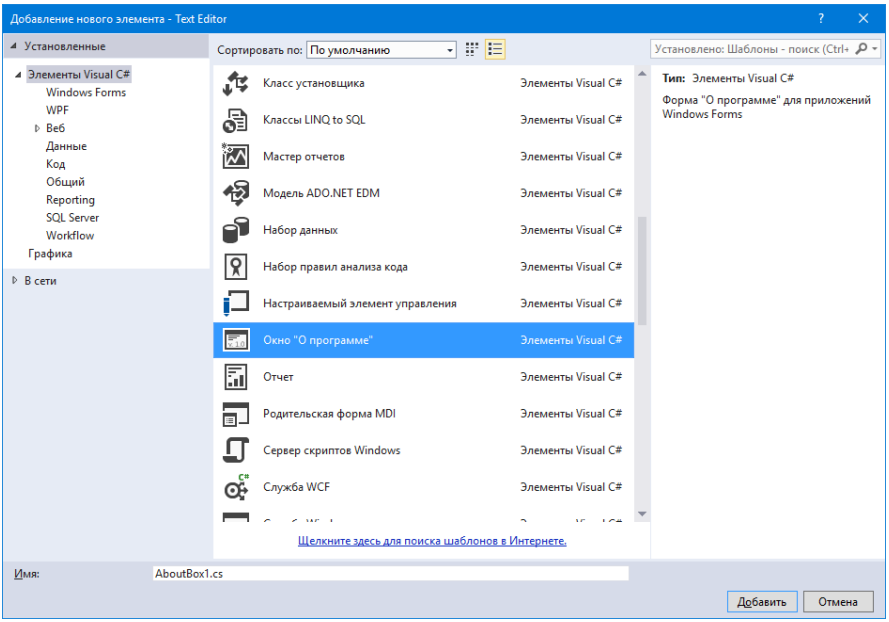


Рис. 28. Добавление новой формы

2. Вы увидите конструктор формы. В новой форме уже расставлены компоненты, и вам нужно настроить текст на них. При необходимости можно удалить или добавить новые копоненты. Например, как показано на рисунке 29.

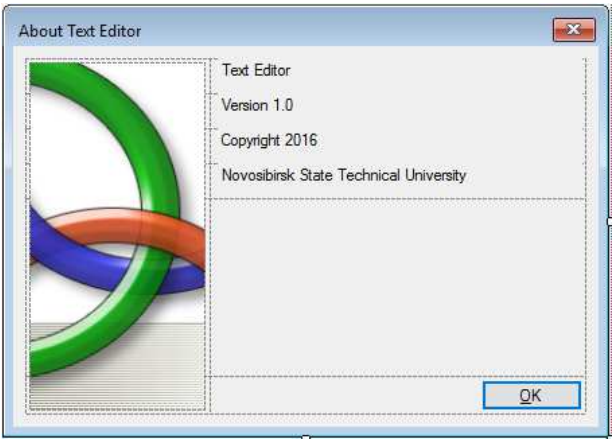


Рис. 29. Форма *About*

3. Добавьте обработчик событий для кнопки *OK*. В нем также должен вызываться метод *Close()*.

4. Сохраните форму *About*, выбрав пункты меню *Файл* → *Сохранить AboutBox1.cs*.

5. Создайте обработчик событий для команды меню *Help* → *About*:

```
private void aboutToolStripMenuItem_Click(object sender,
                                          EventArgs e)
{
  AboutBox1 aboutBox = new AboutBox1();
  aboutBox.ShowDialog();
}
```

6. Запустите приложение и убедитесь, что окно «About» отображается при выполнении соответствующей команды.

3.6. Доработка панели инструментов

Мы разместили на панели инструментов кнопки для ускорения вызова часто используемых функций, но сейчас они не работают. Можно создать для кнопок панели инструментов свои обработчики событий, как это сделано для меню. Однако чтобы избежать дублирования кода, можно назначить кнопкам ранее запрограммированные действия.

Щелкните по кнопке «Создать файл» на панели инструментов разрабатываемого приложения. В окне свойств перейдите к событиям, нажав кнопку  . Найдите событие *Click* и выберите из раскрывающегося списка команду `newToolStripMenuItem_Click`. Аналогичным образом установите действия для остальных кнопок панели инструментов.

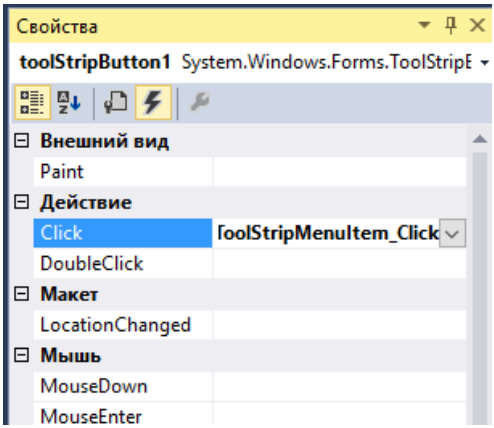


Рис. 30. Настройка действий для кнопок панели инструментов

Запустите приложение и убедитесь, что все кнопки правильно работают.