Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

О. В. Герман, А. В. Заяц

СОВРЕМЕННЫЕ СИСТЕМЫ ПРОГРАММИРОВАНИЯ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО вузов Республики Беларусь
по образованию в области информатики и радиоэлектроники
в качестве учебно-методического пособия для студентов учреждений,
обеспечивающих получение высшего образования по специальности
«Автоматизированные системы обработки информации»

Минск БГУИР 2012

УДК 004.42(075)
ББК 32.973.26-018.2я73
Г38

Р е ц е н з е н т ы:
профессор, заведующий кафедрой «Информационные системы
и технологии» учреждения образования
«Белорусский государственный технологический университет»,
доктор технических наук П. П. Урбанович

доцент кафедры «Системы автоматизированного проектирования»
учреждения образования
«Белорусский национальный технический университет»,
кандидат технических наук В. А. Кочуров

Г38 **Герман** О. В. Современные системы программирования. Лабораторный практикум: учебно-методическое пособие / О. В. Герман, А. В. Заяц – Минск : БГУИР, 2012. – 77 с.: ил.
ISBN 978-985-488-843-9.

Практикум содержит описание десяти лабораторных работ. При выполнении работ предполагается использование системы Visual Studio.NET (2008/2010). Работы содержат теоретическую часть, описание средств .NET для решения поставленных задач и указания к порядку выполнения работ. Приведен список литературы по курсу.

УДК 004.42(075)
ББК 32.973.26-018.2я73

ISBN 978-985-488-843-9

СОДЕРЖАНИЕ

Лабораторная работа №1
РАЗРАБОТКА ПРИЛОЖЕНИЙ НА ОСНОВЕ ФОРМЫ

Цель: научиться писать оконные приложения.

Краткие теоретические сведения

Для создания приложений на основе формы выбираем пункты основного меню File→New→Project→Windows Application

На экране монитора появится пустая форма, на которую разместим компонент Label и компонент TrackBar. Для этого откройте палитру элементов ToolBox, а если ее нет, то используйте пункт основного меню View. Для того, чтобы придать элементам требуемый вид, нажмите на элемент правой кнопкой мыши и выберите пункт Properties. Откроется окно свойств данного элемента. Например, свойство Value компонента TrackBar определяет позицию ползунка. Свойство Text компонента Label определяет его текст. Свойство BackColor компонентов Label и Form определяет цвет этих компонентов. Свойство TextAlign компонента Label определяет расположение текста и т.д. Пусть цель нашей программы состоит в том, чтобы фиксировать в ярлыке Label положение ползунка. Для этого дважды щелкнем по компоненту TrackBar и введем следующий код с позиции, где устанавливается курсор.

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

private void trackBar1_Scroll(object sender, System.EventArgs e)
{
    label1.Text=""+trackBar1.Value;//добавлено нами
}
```

Запустим программу на выполнение. Ее результат поясняется экранной формой, приведенной на рис. 1.1.

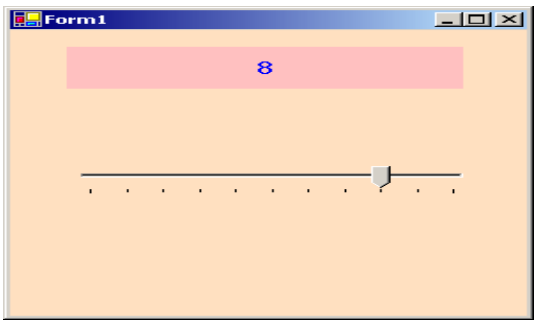

Рис. 1.1. Экранная форма приложения

Итак, концепция программирования в C# имеет много общего с Delphi и Visual Basic: с компонентами связан набор свойств и событий. События обрабатываются стандартными методами. Задача пользователя – создать свой код обработки события. Познакомимся более подробно с работой списков.

Расположим на форме список ComboBox и кнопку Button. По нажатию на кнопку значение ползунка будет записываться в список. Запрограммируем кнопку таким образом:

```
private void button1_Click(object sender, System.EventArgs e)
{
    string s=""+trackBar1.Value;
    int i=comboBox1.FindString(s);
    if (i<0)
        comboBox1.Items.Add(s);
    else
        MessageBox.Show("Элемент уже есть");
}
```

Добавление нового элемента в список реализуется через конструкцию comboBox1.Items.Add(s). Команда int i=comboBox1.FindString(s) присваивает переменной i значение номера строки, где в списке находится строка s. Если строки s в списке нет, то функция FindString(s) возвращает значение минус 1. Теперь пусть по щелчку на элементе списка устанавливается новая координата ползунка.

```
private void comboBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{
    string s=comboBox1.Items[comboBox1.SelectedIndex].ToString();
    trackBar1.Value=Convert.ToInt32(s);
    label1.Text=s;
}
```

Чтобы записать этот метод, просто щелкнем мышью по элементу comboBox1.

Ссылка comboBox1.SelectedIndex определяет номер выделенного элемента списка. Значение comboBox1.Items[comboBox1.SelectedIndex]. ToString(); определяет сам выделенный элемент. Метод Convert.ToInt32(s); преобразует строку в целое число.

Теперь сделаем так, чтобы по двойному щелчку мыши на форме список очищался. Прежде всего откройте список свойств формы. Найдите значок молнии и откройте его;

Найдите слово DoubleClick и щелкните по нему. Запишите следующий код:

```
private void Form1_DoubleClick(object sender, System.EventArgs e)
{
    comboBox1.Items.Clear();
}
```

Наконец, научимся удалять элемент из списка путем щелчка по нему правой кнопкой мыши. Для этого выделим комбосписок, откроем его окно свойств, щелкнем по значку молнии и найдем свойство MouseDown. Щелкнем по нему и в окне редактора кода кода напишем следующие строки:

```
private void comboBox1_MouseDown(object sender,
                                        System.Windows.Forms.MouseEventArgs e)
{
    if (e.Button==MouseButtons.Right)
        comboBox1.Items.RemoveAt(comboBox1.SelectedIndex);
}
```

Обратим внимание на то, что сначала необходимо выбрать элемент списка щелчком левой кнопки мыши, а затем щелкнуть правой кнопкой мыши по значению элемента в текстовом поле списка. Команда comboBox1.Items.RemoveAt(comboBox1.SelectedIndex); удаляет выделенный элемент списка. Команда comboBox1.Items.Insert(string s,int i); вставляет строку s в позицию i. Для того чтобы сделать список невидимым, используйте код:

```
comboBox1.Hide();
```

Чтобы сделать его обратно видимым: comboBox1.Show(). Чтобы вовсе удалить список: comboBox1.Dispose();

Команды выделяют элемент списка, совпадающий со строкой textBox2.Text.

```
int index = comboBox1.FindString(textBox2.Text);
comboBox1.SelectedIndex = index;
```

Имеется возможность связать с событием свой собственный обработчик. Тем не менее список аргументов этого обработчика должен соответствовать стандартному. Так, изменим обработчик события comboBox1_MouseDown на свой собственный. Для этого в модуль инициализации формы нужно поместить строку:

```
this.comboBox1.MouseDown += new System.EventHandler(this.myownmethod);
```

Тем самым указывается, что событие OnMouseDown закрепляется за обработчиком по имени myownmethod.

Реализуем этот метод так:

```
private void myownmethod(object sender,   System.Windows.Forms.MouseEventArgs e)
{
    if (e.Button==MouseButtons.Right)
        comboBox1.Items.RemoveAt(comboBox1.SelectedIndex);
}
```

Вставим его следующим образом. Выберем компонент Form1 откроем метод InitializeComponent. Заменим строку

```
// this.comboBox1.MouseDown +=
// new System.Windows.Forms.MouseEventHandler(this.comboBox1_MouseDown);
```

на строку

```
this.comboBox1.MouseDown +=
        new System.Windows.Forms.MouseEventHandler(this.myownmethod);
```

Теперь изменим содержимое класса Form1:

```
… …. … … … … … .. …

static void Main()
{
    Application.Run(new Form1());
}
private void trackBar1_Scroll(object sender, System.EventArgs e)
{
    label1.Text=""+trackBar1.Value;
}
private void button1_Click(object sender, System.EventArgs e)
{
    string s=""+trackBar1.Value;
    int i=comboBox1.FindString(s);
    if (i<0)
        comboBox1.Items.Add(s);
    else
    MessageBox.Show("Элемент уже есть");
}
private void comboBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{
    string s=comboBox1.Items[comboBox1.SelectedIndex].ToString();
    trackBar1.Value=Convert.ToInt32(s);
    label1.Text=s.Insert(0,"hie");
}
private void Form1_DoubleClick(object sender, System.EventArgs e)
{
    comboBox1.Items.Clear();
}

// этот метод  теперь бесполезен
private void comboBox1_MouseDown(object sender,
                                    System.Windows.Forms.MouseEventArgs e)
{
    if (e.Button==MouseButtons.Right)
    comboBox1.Items.RemoveAt(comboBox1.SelectedIndex);
}
```

```
//добавлено нами
private void myownmethod(object sender,
                                    System.Windows.Forms.MouseEventArgs e)
{
    if (e.Button == MouseButtons.Right)
        comboBox1.Items.RemoveAt(comboBox1.SelectedIndex);
 }
```

Индивидуальное задание

Вариант 1. Написать оконное приложение, иллюстрирующее работу со списками. Из списка выбирается название книги. Для выбранной книги из файла считывается информация по ней и отображается в текстовом поле.

Вариант 2. Написать оконное приложение, иллюстрирующее работу с элементом «слайдер» (см. рис. 1).

Изменяя положение ползунка добиться изменения высоты прямоугольника, изображенного на форме. Другой ползунок должен растягивать или сжимать круг, изображенный на форме. Наконец, третий ползунок должен растягивать или сжимать синусоиду (график $y = \sin(x)$).

Вариант 3. Создать приложение-калькулятор с операциями сложения, вычитания, умножения и деления. Калькулятор должен быть представлен панелью с кнопками и циферблатом для отображения результата.

Вариант 4. Создать приложение для управления выбором шрифта – тип, размер, цвет, жирность, наличие наклона. Продемонстрировать вывод различных шрифтов в текстовом поле.

Вариант 5. Создать приложение для управления выбором рисунков. Рисунки должны выбираться из галереи (типа имеющейся на компьютере папки «Мои рисунки»).

Вариант 6. Создать приложение-словарь для получения перевода с английского на русский и наоборот. Число слов в словаре порядка 50. Добиться перевода слов с одиночными ошибками.

Вариант 7. Создать приложение для просмотра анимаций (файлов мультимедиа).

Лабораторная работа №2
ФАЙЛОВЫЙ ВВОД-ВЫВОД В C#

Цель: изучить простые способы работы с файлами в C#.

Краткие теоретические сведения

В данной работе будут изучены не только механизмы файлового ввода-вывода, но и основные программистские принципы C#.

Прежде всего запустите Visual Studio Net и выберите пункт New Project , а затем Window Application. На экране появится форма, на которую следует перетащить элементы управления:ComboBox, PictureBox и Button, причем через свойство Items/Collection окна свойств Combobox1 занесите начальные значения в ComboBox . Собственно это и будет нужной формой проекта. Смысл работы таков. Имеется (будет создан) некоторый файл, содержащий информацию о расположении рисунков. Содержимое этого файла, например, таково:

Мотоцикл:mot.bmp,Автомобиль:avto.bmp,Трактор:lorry.bmp **(*)**

Пользователь будет выбирать из комбобокса название элемента, например **мотоцикл**. Затем, нажимая кнопку Show Picture, он получает изображение мотоцикла (рис. 2.1).

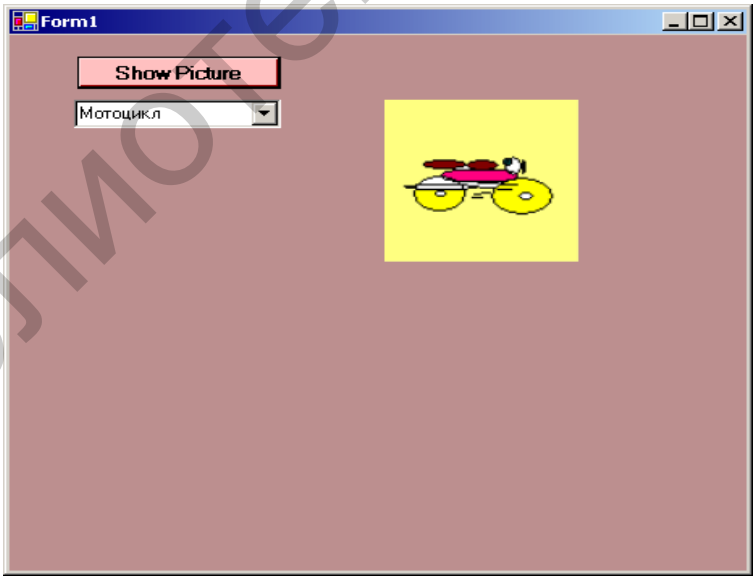

Рис. 2.1. Оконная форма приложения

Для решения этой задачи нужно научиться работать с ComboBox и PictureBox, выполнять работу с файлами и уметь обрабатывать строки. Сначала о работе с элементами. Выбор элементов осуществляется из окна ToolBox слева на экране, но если его нет, выберите данный пункт из основного меню View.

Выберите вкладку Window Forms окна ToolBox. Элементы добавляются так же, как и в Delphi. Взгляните на конструкцию:

```
if (comboBox1.SelectedIndex<0)
{
    MessageBox.Show("No items selected");
}
```

Это чистая конструкция языка C. Терм comboBox1.SelectedIndex получает индекс выделенного элемента ComboBox. Отметим, что терм comboBox1.SelectedItem имеет значение выделенного элемента ComboBox. Чтобы добавить элемент в список, используйте команду comboBox1.Items.Add("NewElement"), а чтобы удалить элемент - comboBox1.Items.Remove("NewElement") .

Работу с файлами лучше пояснить прямо из текста программы:

```
StreamReader sr= File.OpenText(test.path);
string s="";
s=sr.ReadLine();
if (s!=null)
{.
…
```

Объявляется потоковая переменная sr типа StreamReader. Этой переменной присваивается значение файловой переменной File.OpenText(имя файла). Переменная sr определяет текстовый файл для ввода, хотя заметим, что представление информации в этом файле не соответствует ASCII (будет использоваться кодировка UTF8Encoding). Чтение строки из файла осуществляет команда s=sr.ReadLine(). Если прочитан хотя бы один байт, то это устанавливает проверка if(s!=null). Таким образом, если в нашем файле одна единственная строка, а пока это так и есть, то переменная s имеет именно вид (*) (см. с. 9).

Программирование в C# выполняется как и в JAVA, на основе классов. В каком-то одном из этих классов должен быть реализован метод Main(). В программе создаются объекты классов, как обычно, – с помощью конструкторов. Предварительно на форме разместим кнопку, по нажатию на которую будем выдавать сообщение. В обработчике события от кнопки создадим объект, а для вывода сообщения воспользуемся методом этого объекта. Таким образом:

```
using System;
class Ex
{
    public void ShowStr()
    {
        MessageBox.Show("Ну, что, получили?");
        return;
```

```
    }
}
 public class Form1 : System.Windows.Forms.Form
{
    .... .... ... ...  ... ...
    static void Main()
    {
        Application.Run(new Form1());
    }
    private void button1_Click(object sender, System.EventArgs e)
    {
        Ex myObj= new Ex();
        myObj.ShowStr();
    }
```

Многоточие заменяет все производимые системой подстановки, в данный момент не востребованные нами. Главное, что мы объявили свой собственный класс:

```
class Ex
{
    public void ShowStr()
    {
        MessageBox.Show("Ну, что, получили?");
        return;
    }
}
```

И далее используем общепринятым образом:

```
Ex  myObj= new Ex();
myObj.ShowStr();
```

Теперь возвращаемся снова к нашей программе и приводим ее полный текст:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Text;

namespace ComboandFiles
{

    public class Form1 : System.Windows.Forms.Form
    {
```

```
private System.Windows.Forms.ComboBox comboBox1;
private System.Windows.Forms.PictureBox pictureBox1;
private System.Windows.Forms.Button button1;
private System.ComponentModel.Container components = null;
private Bitmap MyImage;
public Form1()
{
    InitializeComponent();
}

protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

class Test
{
    public string path;
    public Test()
    {
        path="d:/msdev/test.txt";
    }

    public string getinfo()
    {
        if (!File.Exists(path))
        {
            return ("File does not exists");
        }
        else
        {
            return ("OK");
        }
    }
}

#region Windows Form Designer generated code

private void InitializeComponent()
{
    this.comboBox1 = new System.Windows.Forms.ComboBox();
    this.pictureBox1 = new System.Windows.Forms.PictureBox();
    this.button1 = new System.Windows.Forms.Button();
    this.SuspendLayout();
```

```
        this.comboBox1.Items.AddRange(new object[] {
                        "Мотоцикл","Автомобиль",
                        "Трактор"});
        this.comboBox1.Location = new System.Drawing.Point(40, 48);
        this.comboBox1.Name = "comboBox1";
        this.comboBox1.Size = new System.Drawing.Size(128, 21);
        this.comboBox1.TabIndex = 0;
        this.comboBox1.Text = "comboBox1";
        this.pictureBox1.BackColor = System.Drawing.SystemColors.Control;
        this.pictureBox1.Location = new System.Drawing.Point(232, 48);
        this.pictureBox1.Name = "pictureBox1";
        this.pictureBox1.Size = new System.Drawing.Size(208, 232);
        this.pictureBox1.TabIndex = 1;
        this.pictureBox1.TabStop = false;
        this.button1.BackColor = System.Drawing.Color.FromArgb(
                                ((System.Byte)(255)),
                                ((System.Byte)(192)), ((System.Byte)(192)));
        this.button1.Font = new System.Drawing.Font("Microsoft Sans Serif", 9.75F,
                                System.Drawing.FontStyle.Bold,
                                System.Drawing.GraphicsUnit.Point,
                                ((System.Byte)(204)));
        this.button1.Location = new System.Drawing.Point(42, 16);
        this.button1.Name = "button1";
        this.button1.Size = new System.Drawing.Size(126, 24);
        this.button1.TabIndex = 2;
        this.button1.Text = "Show Picture ";
        this.button1.Click += new System.EventHandler(this.button1_Click);
        this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
        this.BackColor = System.Drawing.Color.RosyBrown;
        this.ClientSize = new System.Drawing.Size(456, 397);
        this.Controls.Add(this.button1);
        this.Controls.Add(this.pictureBox1);
        this.Controls.Add(this.comboBox1);
        this.Name = "Form1";
        this.Text = "Form1";
        this.ResumeLayout(false);
    }
    #endregion

    [STAThread]
    static void Main()
    {
        Application.Run(new Form1());
    }

    private void button1_Click(object sender, System.EventArgs e)
    {
        if (comboBox1.SelectedIndex<0)
        {
            MessageBox.Show("No items selected");
        }
```

```
else
{
    Test test=new Test();
    string stest=test.getinfo();
    if (stest=="OK")
    {
        // MessageBox.Show(stest);
        if (MyImage!=null)
        {
            MyImage.Dispose();
        }
        StreamReader sr= File.OpenText(test.path);
        string s="";
        s=sr.ReadLine();
        if (s!=null)
        {
            string s1 = comboBox1.SelectedItem.ToString();
            int k= s.IndexOf(s1,0,s.Length);
            if (k<0)
            {
                MessageBox.Show("ComBoItem not Found");
                goto labelfin;
            }
            k=s.IndexOf(':',k,s.Length-k);
            if (k<0)
            {
                MessageBox.Show(":not found after specified combo Item");
                goto labelfin;
            }
            s=s.Substring(k+1,s.Length-k-1);
            k=s.IndexOf(',',0,s.Length);
            if (k>=0)
            {
                s=s.Substring(0,k);
            }
            s=@"d:\msdev\"+s;
            // MessageBox.Show("File found:"+s);
            pictureBox1.SizeMode= PictureBoxSizeMode.StretchImage;
            MyImage= new Bitmap(s);
            pictureBox1.ClientSize=new Size(120,120);
            pictureBox1.Image=(Image) MyImage;
            labelfin: ;
        }
    }
    else
    {
        MessageBox.Show(stest);
        FileStream fs=File.Create(test.path,1024);
```

```
                    Byte [] info=new UTF8Encoding(true).GetBytes(
                            "Мотоцикл:mot.bmp,
                            Автомобиль:avto.bmp,
                            Трактор:lorry.bmp");
                    fs.Write(info,0,info.Length);
                    MessageBox.Show("File Created.Repeat then");
                }
            }
        }
    }
}
```

Опишем некоторые фрагменты этой программы подробнее. Фрагмент

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Text;
```

используется для подключения библиотек, содержащих описания системных функций. Например, функции для работы с файлами содержатся в библиотеке System.IO. Строка namespace ComboandFiles используется для объявления пространства имен. Это – новшество, впервые введенное в C++. Пространство имен содержит определения имен, используемых в некотором контексте. Например, в программе может встретиться имя функции ReadLine(), которое конфликтует с именем стандартной функции. В этом случае для системы важно, какое пространство имен используется в качестве контекста. Далее идет длинное описание класса public class Form1: System.Windows.Forms.Form. Это стандартное описание, так как система создает его сама. Но мы уже поместили внутрь данного описания следующее определение встроенного класса:

```
class Test
{
    public string path;
    public Test()
    {
        path="d:/msdev/test.txt";
    }
    public string getinfo()
    {
        if (!File.Exists(path))
        {
            return ("File does not exists");
        }
```

```
            else
            {
                return ("OK");
            }
        }
    }
```

Это описание включает объявление переменной класса path, конструктора Test(), который данную переменную инициализирует, и метода getinfo(), который проверяет, имеется ли данный файл в указанном каталоге (рекомендуем запомнить эту простую команду: File.Exists(path)). Остается привести обработчик события от нажатия кнопки и прояснить его, что сделано с помощью помещенных рядом с командами комментариев.

```
private void button1_Click(object sender, System.EventArgs e)
{
    if (comboBox1.SelectedIndex<0)  //проверяет, выделен ли элемент в комбосписке
    {
        MessageBox.Show("No items selected");   //выдает сообщение, если не выделен
    }
    else
    {
        Test test=new Test();  //иначе создает объект класса Test
        // метод getinfo()класса Test сообщает о наличии файла данных
        string stest=test.getinfo();
        if (stest=="OK")
        {
            //  MessageBox.Show(stest);
            //проверяет наличие рисунка и удаляет текущий рисунок
            if (MyImage!=null)
            {
                MyImage.Dispose();
            }
            //создает файловую переменную для ввода
            StreamReader sr= File.OpenText(test.path);
            string s="";
            // читает строку из файла, имя которого помещено в классовой
            // переменной test.path
            s=sr.ReadLine();
            if (s!=null)
            {
                //преобразует в подстроку выделенный элемент списка
                string s1=comboBox1.SelectedItem.ToString();
                //определяет местонахождение данной подстроки в файле
                int k= s.IndexOf(s1,0,s.Length);
                if (k<0)
                {
                    MessageBox.Show("ComBoItem not Found");
                    goto labelfin;
                }
```

```
                    //определяет позицию : вслед за именем подстроки
                    k=s.IndexOf(':',k,s.Length-k);
                    if (k<0)
                    {
                        MessageBox.Show(": not found after specified combo Item");
                        goto labelfin;
                    }
                    s=s.Substring(k+1,s.Length-k-1);//определяет адрес файла с рисунком
                    k=s.IndexOf(',',0,s.Length);
                    if (k>=0)
                    {
                        s=s.Substring(0,k);
                    }
                    //добавляет к имени файла полный путь;
                    //в вашей работе этот оператор следует изменить
                    s=@"d:\msdev\"+s;
                    pictureBox1.SizeMode= PictureBoxSizeMode.StretchImage;
                    MyImage= new Bitmap(s); //Создает битовый образ картинки из файла
                    pictureBox1.ClientSize=new Size(120,120);//Задает размеры картинки
                    pictureBox1.Image=(Image) MyImage;//Преобразует BitMap в Image
                    labelfin: ;
                }
            }
            else  //файл с адресами картинок не существует и он создается
            {
                MessageBox.Show(stest);
                FileStream fs=File.Create(test.path,1024);
                Byte [] info = new UTF8Encoding(true).GetBytes(
                        "Мотоцикл:mot.bmp,Автомобиль:avto.bmp,Трактор:lorry.bmp");
                fs.Write(info,0,info.Length);//Занесение информации в файл из буфера
                MessageBox.Show("File Created.Repeat then");

            }
        }
    }
```

Индивидуальное задание

Нужно смоделировать простой SQL-запрос к базе данных в текстовом файле. Следует реализовать задачу поиска автомобиля, например, не старше 1990 года выпуска и стоимостью не выше 1000$. Для этого нужно включить в текстовый файл поля :

avto: ... , price: ..., year: ... , photo : ... , ... avto: ... , price: ..., year: ... , photo : ...

Далее потребуется несколько дополнительных возможностей. Для этого, во-первых, показываем, как преобразовать строку в целое число:

```
string s= textBox1.Text;
int i=Convert.ToInt32(s,10);
```

Во-вторых, рассмотрим, как осуществлять запись и чтение в текстовый файл не одной единственной строки, а нескольких строк.

В C# для этих целей используют классы StreamWriter и StreamReader. Покажем это на примере:

```
using System;
using System.IO;
class Ex2
{
    public static void Main()
    {
        FileStream ous=File.Create(@"c:\msdev\data..txt");
        StreamWriter sw= new StreamWriter(ous);
        sw.WriteLine("one");
        sw.WriteLine("two");
        sw.Flush();
        sw.Close();
        StreamReader sr=new StreamReader(ous);
        string s1;
        while((s1=sr.ReadLine())!=null)
        {
            Console.WriteLine(s1);
        }
        sr.Close();
    }
}
```

Итак, создаем объект StreamWriter для записи в файл и StreamReader – для чтения. Предварительно порождаем файловую переменную FileStream ous. Запись в файл реализуем командой WriteLine, чтение – ReadLine(). Признаком достижения конца файла является sr.ReadLine()==null.

Теперь самостоятельно постройте простейшее приложение для поиска автомобиля по цене и году.

Лабораторная работа №3
ОСНОВЫ РАБОТЫ С БАЗАМИ ДАННЫХ В C#

Цель: познакомиться с принципами работы с базами данных в C#.

Краткие теоретические сведения

В настоящей работе рассматривается работа с базами данных на основе классов ADO.NET . Имеется три способа использования этих классов: через соединения SqlConnection, OleDbConnection и OdbcConnection.Соединение SqlConnection предназначено для работы с сервером баз данны MS SQL-server 2000. В этой работе он не рассматривается. Соединение OleDbConnection используется для связи «напрямую» с локальной базой данных (поддерживаются только базы данных MicroSoft). Соединение OdbcConnection использует механизм драйверов ODBC для связи с широким диапазоном баз данных.

Начнем с OleDbConnection. Рассмотрим работоспособный пример и затем прокомментируем его.

```
using System;
using System.Data;
using System.Data.OleDb;

namespace lessonbasaconsole
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            string connectingString =
        @"provider=Microsoft.Jet.OLEDB.4.0;data source=c:\disk_d\msdev\mydb.mdb";
            OleDbConnection myConn = new OleDbConnection (connectingString);
            //Сформировать строку запросов
            string selectString ="Select name,group from stud";
            OleDbCommand myCommand= myConn.CreateCommand();
            myCommand.CommandText= selectString;
            OleDbDataAdapter oda= new OleDbDataAdapter();
            oda.SelectCommand=myCommand;
            DataSet myDataset= new DataSet();
            myConn.Open();
            string dataTableName="studA";
            oda.Fill(myDataset,dataTableName);
            DataTable myDataTable= myDataset.Tables[dataTableName];

            foreach (DataRow dr in myDataTable.Rows)
            {
                Console.WriteLine("Name="+dr["name"]);
```

```
                Console.WriteLine("Group="+dr["group"]);
            }
            string s= Console.ReadLine();
            myConn.Close();
        }
    }
}
```

Обратим внимание на подключаемые библиотеки, рассмотрим фрагмент, выделенный жирным шрифтом. Сначала строим строку соединения с локальной базой:

```
string connectingString =
@"provider=Microsoft.Jet.OLEDB.4.0;data source= c:\disk_d\msdev\mydb.mdb";
```

Эта база построена в ACCESS. Она содержит таблицу stud, которая в свою очередь содержит всего два поля: name и group. Следующая строка

```
OleDbConnection myConn = new OleDbConnection (connectingString);
```

создает новое соединение с этой базой. Затем с помощью трех строк

```
string selectString ="Select name,group from stud";
OleDbCommand myCommand= myConn.CreateCommand();
myCommand.CommandText= selectString;
```

строим SQL-команду для выполнения над таблицей stud. Затем нужно подключить адаптер для связи с базой:

```
OleDbDataAdapter oda= new OleDbDataAdapter();
oda.SelectCommand=myCommand;
```

Созданную выше SQL-команду присваиваем полю SelectCommand объекта-адаптера. Далее создаем объект типа DataSet, который хранит как выбираемые наборы данных из таблиц, так и сами таблицы. После этого открываем соединение:

```
DataSet myDataset= new DataSet();
myConn.Open();
```

Теперь по команде

```
string dataTableName="studA";
oda.Fill(myDataset,dataTableName);
```

осуществляем выборку данных из базы и помещение их в условную таблицу studA (заметим, что имя реальной таблицы – stud).

Итак, OleAdapter располагает командой для связи с базой и через команду

связи соединен с объектом-соединением(connection). Таким образом, при открытии соединения данные могут быть считаны адаптером в память. Адаптер выбирает данные и помещает их в таблицу studA командой oda.Fill(myDataset,dataTableName);

Теперь остается только прочитать данные и отобразить их на консоли:

```
DataTable myDataTable= myDataset.Tables[dataTableName];
foreach (DataRow dr in myDataTable.Rows)
{
    Console.WriteLine("Name="+dr["name"]);
    Console.WriteLine("Group="+dr["group"]);
}
```

Приводим пример той же программы для работы с ODBC:

```
using System;
using System.Data;
using System.Data.Odbc;
namespace odbcexample
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            string connectingString ="DSN=myaccess";
            OdbcConnection myConn = new OdbcConnection (connectingString);
            string mySelectQuery = "SELECT name,group FROM stud";
            OdbcCommand myCommand = new OdbcCommand(mySelectQuery);
            myCommand.Connection= new OdbcConnection(connectingString);
            OdbcDataAdapter oda= new OdbcDataAdapter();
            oda.SelectCommand=myCommand;
            //Сформировать строку запросов
            DataSet myDataset= new DataSet();
            myConn.Open();
            string dataTableName="studA";
            oda.Fill(myDataset,dataTableName);
            DataTable myDataTable= myDataset.Tables[dataTableName];
            foreach (DataRow dr in myDataTable.Rows)
            {
                Console.WriteLine("Name="+dr["name"]);
                Console.WriteLine("Group="+dr["group"]);
            }
            string s= Console.ReadLine();
            myConn.Close();
        }
    }
}
```

Обратим внимание на строки

```
OdbcCommand myCommand = new OdbcCommand(mySelectQuery);
myCommand.Connection= new OdbcConnection(connectingString);
```

которые отличают этот способ от предыдущего, а именно: нужно определять свойство Connection у объекта OdbcCommand. Запросы, отличные от SELECT, реализуются, как показано на примере ниже, для INSERT:

```
using System;
using System.Data;
using System.Data.Odbc;
namespace odbcexample
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            string myConnectionString="DSN=myaccess";
            string myExecuteQuery="insert into stud values('han',2,null)";
            OdbcConnection myConnection =
                                    new OdbcConnection(myConnectionString);
            OdbcCommand myCommand =
                                        new OdbcCommand(myExecuteQuery, myConnection);
            myConnection.Open();
            myCommand.ExecuteNonQuery();
            myConnection.Close();
            string s=Console.ReadLine();
        }
    }
}
```

Таким образом, достаточно воспользоваться в этом случае только объектами OdbcConnection и OdbcCommand. Сам запрос выполняется на открытом соединении командой myCommand.ExecuteNonQuery();

Индивидуальное задание

Создать консольное приложение, которое позволяло бы не только выбирать записи из базы данных, но и добавлять и удалять их.

Лабораторная работа №4
КЛИЕНТ-СЕРВЕРНОЕ ВЗАИМОДЕЙСТВИЕ НА ОСНОВЕ ПРОТОКОЛОВ TCP И HTTP

Цель: изучить работу с файлами в сети Internet.

Краткие теоретические сведения

Надлежит написать программу сервера и программу клиента и запустить их как раздельные приложения. Программа-клиент вызовет метод программы-сервера и получит результат работы последней. Теперь приведем текст программы-сервера:

```
using System.Net;
using System.Text;
using System.Net.Sockets;

namespace ExampleServer
{
    public class ServerExample
    {
        //это – единственный используемый нами метод сервера
        private void Listen()
        {
            //создаем прослушиватель порта 50001
            TcpListener tcpl = new TcpListener(new IPAddress(new byte[] {127, 0, 0, 1}),
                                               50001);
            tcpl.Start(); //запускаем прослушиватель

            // ОЖИДАНИЕ ПОДКЛЮЧЕНИЯ КЛИЕНТА
            //прослушиватель ожидает подключения клиента
            Socket newSocket = tcpl.AcceptSocket();
            //Если соединение установлено, то пересылаем данные клиенту
            if (newSocket.Connected)
            {
                NetworkStream ns = new NetworkStream(newSocket);

                //пересылка данных
                string sendString = "SO MUST BE I SAY";
                //преобразование строки в массив байтов
                byte[] sendBuf = Encoding.Unicode.GetBytes(sendString);
                //отправляем данные клиенту
                ns.Write(sendBuf, 0, sendBuf.Length);

                //Очистка
                ns.Flush();
                ns.Close();
            }
        }
```

```
        static void Main()
        {
            ServerExample se = new ServerExample();
            se.Listen();
        }
    }
}
```

Итак, чтобы сервер возвратил строку «SO MUST BE I SAY» , необходимо подключение к нему клиента через порт 50001. Номер порта можно использовать и другой.

Теперь нужно создать новый проект – приложение клиента. Клиент должен знать сетевой адрес сервера и номер порта. Вот его текст:

```
using System;
using System.Net.Sockets;
using System.Text;

class ExampleClient
{
    public static void Main()
    {
        //Создаем объект типа TcpClient для связи с сервером
        TcpClient newSocket = new TcpClient("localhost", 50001);
        //ns –потоковая переменная на базе сокета
        NetworkStream ns = newSocket.GetStream();
        //чтение массива байтов от сервера из входного потока:
        byte[] byteBuf = new byte[100];
        int readByteCount = ns.Read(byteBuf, 0, 100);
        // теперь выполним преобразование байтов в тип string:
        string stringBuf = Encoding.Unicode.GetString(byteBuf, 0, readByteCount);
        // выводим строку от сервера на консоль клиента;
        Console.WriteLine(stringBuf);
        ns.Close();
        newSocket.Close();
        Console.Read();
    }
}
```

Скомпилируем клиента.

В результате компиляции клиента и сервера строятся файлы с расширением exe. Сначала из командной строки запускаем программу-сервер. Затем отдельно запускаем клиента (убедитесь в правильности работы программы). Обратимся теперь к взаимодействию на основе протокола HTTP.

HTTP (hyper text transport protocol) – это протокол сетевого уровня (упрощенный), который размещается поверх TCP. Этот протокол используем для обмена WEB-ресурсами (документами, файлами, картинками). Сообщение

в формате HTTP состоит из заголовков, несущих служебную информацию, и данных. Примерами заголовков являются

ACCEPT – определяет, какие виды документов может принимать клиент (текст, бинарные файлы, xml, картинки);

REFERER – определяет интернет-адрес источника сообщения;

USER-AGENT – определяет информацию о браузере и т.д.

HTTP-соединение задействует серверную сторону и клиентскую сторону.

Имеется возможность связи клиента и сервера через протокол HTTP. Рассмотрим самый простой способ осуществить такую связь. В рассматриваемо варианте соединения нет необходимости указывать номер порта. Сервер реализуется как удаленный объект (remote object), поэтому требуется посредник между сервером и клиентом – *интерфейс*. В интерфейсе нужно объявить методы сервера, которые следует сделать доступными клиенту. Приведем пример сравнительно простого интерфейса:

```
using System;
public interface ISimpleObject
{
    string ToUpper(String inString);
}
```

Единственным методом сервера является ToUpper. Этот метод принимает в качестве входного аргумента строку inString. Реализацию данного метода следует перенести в сервер. Само приложение сервера таково:

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;

namespace Server
{
    public class SimpleObject: MarshalByRefObject, ISimpleObject
    {
        public String ToUpper(String inString)
        {
            return(inString.ToUpper());
        }
    class Example8_8
    {
        static void Main(string[] args)
        {
            HttpChannel channel = new HttpChannel(54321);
            ChannelServices.RegisterChannel(channel);
```

```
            RemotingConfiguration.RegisterWellKnownServiceType(
                                typeof(SimpleObject),
                                "SOEndPoint",
                                WellKnownObjectMode.Singleton);
            Console.WriteLine("Press To Stop Server");
            Console.ReadLine();
        }
    }
}
```

Обратим внимание на то, что в объявление класса сервера подключены созданный нами интерфейс и еще один интерфейс: MarshalByRefObject. Дальше следует выполнить реализацию метода ToUpper(). Метод Main() сервера осуществляет соединение с клиентом:

```
// Создаем канал связи снова через порт
HttpChannel channel = new HttpChannel(54321);

// Регистрируем канал  в операционной системе
ChannelServices.RegisterChannel(channel);

// определяем тип соединения "Simple Object End Point" и режим
// его работы Singleton (обслуживание одного клиента одним удаленным объектом)
RemotingConfiguration.RegisterWellKnownServiceType(
                                typeof(SimpleObject),
                                "SOEndPoint",
                                WellKnownObjectMode.Singleton);
```

Теперь рассмотрим сторону клиента:

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;

class client
{
    public static void Main()
    {
        HttpChannel h1=new HttpChannel(0);
        ChannelServices.RegisterChannel(h1);
        Object remoteOb= RemotingServices.Connect(
                                    typeof(ISimpleObject),
                                    "http://localhost:54321/SOEndPoint");
        ISimpleObject so=remoteOb as ISimpleObject;
```

```
            Console.WriteLine(so.ToUpper("so must be I say"));
        }
    }
```

Клиент должен получить доступ к удаленному объекту сервера. Это делается с помощью команды RemotingServices.Connect(typeof(ISimpleObject), "http://localhost:54321/SOEndPoint"); Здесь использовано *сетевое имя* localhost; через двоеточие указан номер порта. Сетевое имя уникально идентифицирует компьютер. Следующая команда ISimpleObject so = remoteOb as ISimpleObject; создает у клиента экземпляр объекта сервера. Отметим при этом, что используется имя интерфейса, а имя класса сервера вовсе не задействуется. Обращение к методу ToUpper сервера выполняется таким образом: Console.WriteLine(so.ToUpper("so must be I say"));

Теперь рассмотрим считывание файла на основе протокола HTTP. Следующий пример демонстрирует простое скачивание файла с отображением на консоль:

```
using System;
using System.Text;
using System.Net;
using System.IO;

namespace HtmlReading
{
    class MyHtml
    {
        static void Main(string[] args)
        {
            string query = @"e:\samko\excel_guide2.txt";
            FileWebRequest req = (FileWebRequest)FileWebRequest.Create(query);
            FileWebResponse resp = (FileWebResponse)req.GetResponse();
            StreamReader sr = new StreamReader(resp.GetResponseStream(),
                                                Encoding.ASCII);
            Console.WriteLine(sr.ReadToEnd());
            resp.Close();
            sr.Close();
            Console.ReadLine();
        }
    }
}
```

Строка query представляет URI скачиваемого файла. Усложним этот пример следующим образом. Будем считывать html-файл и открывать его броузером. Наша программа имеет следующий вид:

```
using System;
using System.Text;
using System.IO;
```

```
using System.Net;
using System.Diagnostics;

namespace HtmlReading2
{
    class MyHtml2
    {
        static void Main(string[] args)
        {
            string query = @"e:\samko\excel_guide2.txt";
            string newFileName = @"e:\samko\new_new.html";

            FileWebRequest req = (FileWebRequest)FileWebRequest.Create(query);
            FileWebResponse resp = (FileWebResponse)req.GetResponse();

            long contentLength = resp.ContentLength;
            byte[] byteBuffer = new byte[contentLength];
            Stream respStream = resp.GetResponseStream();
            //чтение содержимого файла из потока в буферный массив
            respStream.Read(byteBuffer, 0, (int)contentLength);

            using (FileStream fs = new FileStream(newFileName,
                                        FileMode.OpenOrCreate, FileAccess.Write))
            {
                fs.Write(byteBuffer, 0, byteBuffer.Length);
            }
            //Запуск считанного файла
            Process.Start(newFileName);
            resp.Close();
        }
    }
}
```

Покажем теперь, как адаптировать приведенный вариант программы к считыванию файла в Интернете. Наш новый вариант имеет следующий вид

```
namespace HtmlReading2
{
    class MyHtml2
    {
        static void Main(string[] args)
        {
            string webstr = @"file://localhost/";
            Uri baseUri = new Uri(webstr);
            //второй параметр - относительный путь (относительно baseUri)
            Uri query = new Uri(baseUri, @"e:\samko\excel_guide2.txt");

            string newFileName = @"e:\samko\new_new.html";

            FileWebRequest req = (FileWebRequest)FileWebRequest.Create(query);
            FileWebResponse resp = (FileWebResponse)req.GetResponse();
```

```
                long contentLength = resp.ContentLength;
                byte[] byteBuffer = new byte[contentLength];
                Stream respStream = resp.GetResponseStream();
                //чтение содержимого файла из потока в буферный массив
                respStream.Read(byteBuffer, 0, (int)contentLength);

                using (FileStream fs = new FileStream(newFileName,
                                        FileMode.OpenOrCreate, FileAccess.Write))
                {
                    fs.Write(byteBuffer, 0, byteBuffer.Length);
                }

                Process.Start(newFileName);
                resp.Close();
            }
        }
    }
```

Строки

```
string webstr = @"file://localhost/";
Uri baseUri = new Uri(webstr);
Uri query=new Uri(baseUri,@"e:\samko\new.html");
```

задают адрес файла в Интернете. При считывании создается новый файл

```
FileStream fs = new FileStream(@"e:\samko\new_new.html",
                                FileMode.OpenOrCreate, FileAccess.Write);
```

Индивидуальное задание

1. Реализовать примеры, описанные в теоретической части
2. На основе представления сервера как удаленного объекта объявить интерфейс с методами для доступа к базе данных и чтения из базы информации на основании переданной в запрос фамилии (читать номер группы и оценки по сданным экзаменам). Эту информацию вернуть клиенту.
3. Считать файл с картинкой (.bmp, .jpg) и открыть ее в клиентском сайте.

Лабораторная работа №5
СОЗДАНИЕ СОБСТВЕННЫХ КОМПОНЕНТОВ

Цель: целью настоящей лабораторной работы является изучение способов создания собственных компонентов в среде C#.

Краткие теоретические сведения

В C# можно создавать следующие компоненты: COM, компоненты на основе визуальных классов и пользовательские компоненты. Компоненты COM могут использоваться в других средах программирования, например в JAVASCRIPT. В этой работе мы их не рассматриваем. Компоненты на основе визуальных классов позволяют придать некоторую новую функциональность стандартным визуальным компонентам, например, на основе класса файлового диалога дополнительно предусмотреть выдачу информацию о свойствах файла. Пользовательские компоненты позволяют Вам создавать любые функциональности для своего класса. Наконец, в C# можно внедрить чужой элемент ACTIVEX, например календарь.

Итак, начнем с компонентов, которые строятся на основе визуальных классов. Рассмотрим создание компонента, представляющего текстовое поле и полосу вертикальной прокрутки. При перемещении ползунка в текстовом поле будет отображаться его положение.

Для создания такого компонента следует выполнить следующие действия.

1.Создать новый проект на базе шаблона WINDOWS CONTROL LIBRARY (File→ New Project→Windows Control Library)

2. Присвойте имя создаваемому компоненту, например, MYSCROLL.

3. Разместите на форме текстовое поле (задайте его свойство ReadOnly= true) и полосу вертикальной прокрутки из панели элементов Window Forms.

4. Создайте следующий код:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;

namespace WindowsControlLibrary1
{
    public class MYSCROLL : System.Windows.Forms.UserControl
    {
        private int m_min=int.MinValue;
        private int m_max=int.MaxValue;
        private int m_current=0;
```

```
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.VScrollBar vScrollBar1;
private System.ComponentModel.Container components = null;

public int Min
{
    get{ return m_min; }
    set{ m_min=value; }
}
public int Max
{
    get{ return m_max; }
    set{ m_max=value; }
}
public int Current
{
    get{ return m_current; }
    set
    {
        if((value>m_max)||(value<m_min))
        {
            throw new ArgumentOutOfRangeException("Current");
        }
        m_current=value;
        textBox1.Text=m_current.ToString();
    }
}
```

В классе MYSCROLL мы определили три свойства: Min, Max и Current, определяющие максимальную позицию, минимальную позицию и текущую позицию ползунка. Эти свойства являются публикуемыми. Это значит, что они будут доступны в окне свойств объекта класса MYSCROLL, размещенного на форме обычным образом. Обратите внимание, как для каждого из этих свойств задаются методы get и set, используемые для получения (get) и установки (set) значения данного свойства у объекта. Теперь надо запрограммировать реакцию на перемещение ползунка. Дважды щелкните мышью на ползунке на форме и вставьте следующий код:

```
private void vScrollBar1_Scroll(object sender, System.Windows.Forms.ScrollEventArgs e)
{
    if (e.Type==ScrollEventType.SmallIncrement)
    {
        try
        {
            Current-=1;;
        }
        catch
        {
        }
```

```
        }
        else if (e.Type==ScrollEventType.SmallDecrement)
        {
            Current+=1;
        }
    }
```

Событие SmallIncrement происходит при смещении ползунка вниз, событие SmallDecrement – при смещении ползунка вверх.

5. Скомпилируйте данный файл. В результате будет построен выходной библиотечный dll-файл.

6. Закройте проект.

Теперь выясним, как разместить созданный компонент на форме. Для этого создайте проект на основе шаблона Windows APPLICATION. Создайте далее в окне элементов новую закладку, для чего нажмите правую кнопку мыши на этом окне и выберите пункт ADD TAB. Введите имя закладки : MYCOMPONENTS. Выделите эту закладку щелчком мыши и нажмите снова правую кнопку мыши. Появится окно наподобие представленного на рис. 5.1.

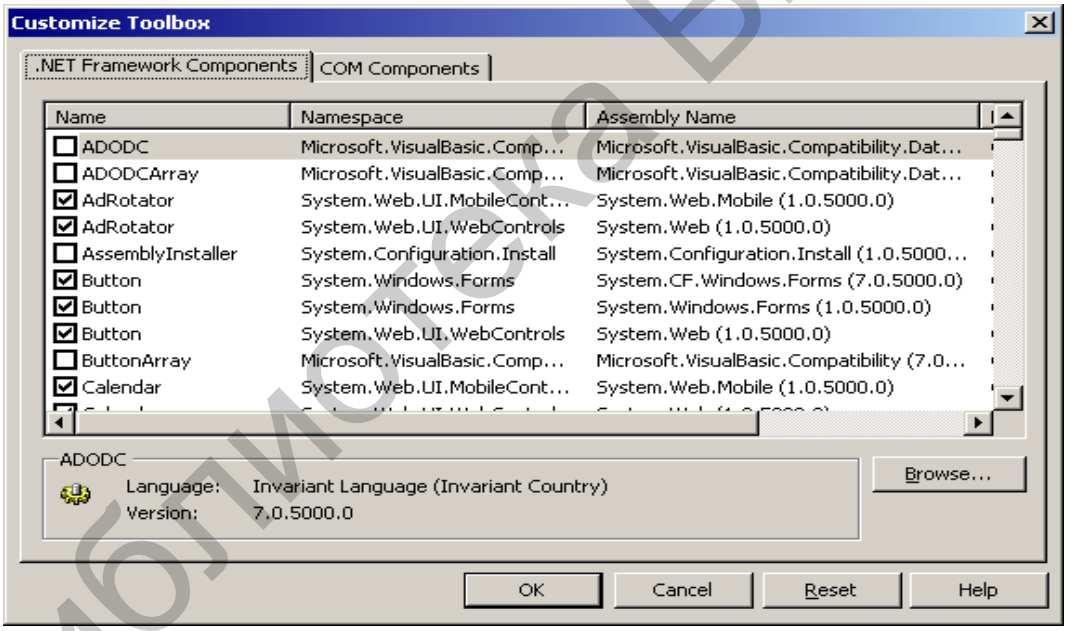

Рис. 5.1. Окно выбора компонентов

Нажмите закладку .NET Framework Components, затем кнопку Browse. Найдите, где был сохранен ваш dll-файл. Он сохраняется в том месте, где вы указали при открытии пустого проекта в каталоге папка_с_проектами/debug/bin/... Когда dll-файл найден, нажмите OK. Ваш класс появится в списке, приведенном на рисунке выше. Выделите имя этого класса и нажмите OK. Теперь уже класс будет размещен в окне элементов. С ним можно работать, как с обычной кнопкой. То, что вы создали, иллюстрируется рисунком 5.2.

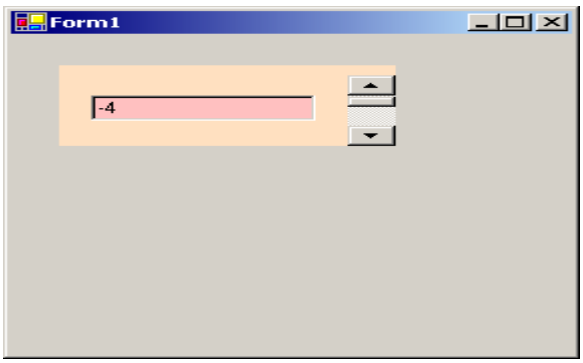

Рис. 5.2. Созданный компонент

Теперь рассмотрим, как создавать собственные невизуальные компоненты. Технология в целом почти такая же.

Итак, продолжаем с компонентов, которые строятся на основе собственных пользовательских классов. Рассмотрим создание компонента, представляющего закрашенный эллипс и имеющего редактируемое свойство Message.

Для создания такого компонента следует выполнить следующие действия.

1. Создайте новый проект на базе шаблона WINDOWS CONTROL LIBRARY (File→ New Project→Windows Control Library)

2. Присвойте имя создаваемому компоненту, например, MYCOMP.

3. В окне Solution Explorer щелкните правой кнопкой мыши на узле UserControl1 и удалите его из проекта.

4. Щелкните правой кнопкой мыши в окне Solution Explorer на названии библиотеки и выберите команду Add→Add Inherited Control. Выберите шаблон Custom Control и присвойте файлу имя, например WindowsControlLibrary2.cs.

5. Создайте, например, следующий код :

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;

namespace WindowsControlLibrary2
{
    public class CustomCTRL1 : System.Windows.Forms.Control
    {
        private string m_mes;
        public string Message
        {
            get{ return m_mes; }
            set{ m_mes=value; }
        }
```

```
        private System.ComponentModel.Container components = null;

        public CustomCTRL1()
        {
            InitializeComponent();
            ResizeRedraw =true;
            m_mes="INITIALLY";
            // TODO: Add any initialization after the InitComponent call
        }

            protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if( components != null )
                    components.Dispose();
            }
            base.Dispose( disposing );
        }

        #region Component Designer generated code
           private void InitializeComponent()
        {
            components = new System.ComponentModel.Container();
        }
        #endregion

        protected override void OnPaint(PaintEventArgs pe)
        {
            SolidBrush b= new SolidBrush (Color.SteelBlue);
            pe.Graphics.FillEllipse(b,0,0,50,50);
            base.OnPaint(pe);
        }
    }
}
```

Прорисовку эллипса помещаем в обработчик события OnPaint.

```
protected override void OnPaint(PaintEventArgs pe)
{
    // TODO: Add custom paint code here
    // Calling the base class OnPaint
    SolidBrush b= new SolidBrush (Color.SteelBlue);
    pe.Graphics.FillEllipse(b,0,0,50,50);
    base.OnPaint(pe);
}
```

Объявление редактируемого свойства представлено следующим блоком:

```
private string m_mes;
public string Message
{
    get{  return m_mes; }
    set{  m_mes=value; }
}
```

Теперь достаточно просто скомпилировать проект. При этом создается файл – dll, который нужно установить на палитру элементов так же как, и ранее созданный компонент на основе визуальных классов.

Индивидуальное задание

Создайте собственный невизуальный элемент, который имеет два метода: один используется для получения головной части строки, второй – хвостовой. Например, первый метод объявлен как

```
String Head(String str, String mask)
```

Этот метод получает исходную строку, например «Hello, friends» и строку-разделитель «,». Данный метод отыскивает в строке «Hello, friends» первое вхождение строки-разделителя и возвращает часть исходной строки, стоящую до разделителя. Если же разделитель не встретится, то возвращается вся исходная строка.

Метод, возвращающий хвостовую часть, отличается от описанного только тем, что возвращает часть строки, стоящую после разделителя.

Для выполнения задания Вам потребуются методы для работы со строками:

```
public int indexOf(String s)
```

– возвращает номер позиции, где встречается первое вхождение строки s, например:

```
string s1="Hello, Friends";
int i =s1.indexOf(",");
```

```
public String Substring(int i1,int2)
```

– выделяет из строки подстроку в диапазоне позиций от i1 до i1+i2.

Лабораторная работа №6
ИЗУЧЕНИЕ ТЕХНОЛОГИИ WCF

Цель: изучить возможности и функциональные задачи технологии сервисов WCF.

Краткие теоретические сведения

WCF – это технология для построения распределенных приложений.

Ключевым понятием в WCF является служба. В основе своей служба – это множество оконечных точек (endpoints), которые предоставляет клиентам некие полезные возможности. Оконечная точка – это просто сетевой ресурс, которому можно посылать сообщения. Чтобы воспользоваться предоставляемыми возможностями, клиент посылает сообщения оконечным точкам в формате, который описывается контрактом между клиентом и службой. Службы ожидают поступления сообщений на адрес оконечной точки, предполагая, что сообщения будут записаны в оговоренном формате.

Чтобы клиент мог передать службе осмысленную информацию, он должен знать адрес, привязку и контракт.

Адрес (Address, A) определяет, куда следует отправлять сообщения, чтобы оконечная точка их получила. В случае протокола HTTP адрес будет выглядеть так: http://myserver/myservice/.

Привязка (Binding, B) определяет канал для коммуникаций с оконечной точкой. По каналам передаются все сообщения, циркулирующие в приложении WCF. Канал состоит из отдельных элементов привязки (binding element), которые определяют транспортный механизм, способ кодирования данных, способы обеспечения безопасности и др. WCF позволяет заменять данные модули на другие (например, протокол HTTP заменить на TCP) и при этом логика приложения не изменится.

Контракт (Contract, C) определяет набор функций, предоставляемых оконечной точкой, то есть операции, которые она может выполнять, и форматы сообщений для этих операций. Описанные в контракте операции отображаются на методы класса, реализующего оконечную точку, и включают, в частности, типы параметров, передаваемых каждому методу и получаемых от него.

WCF-служба может состоять из нескольких оконечных точек, каждая из которых описывается собственным адресом, привязкой и контрактом. Поскольку поток сообщений обычно двунаправленный, клиенты не явно также оказываются контейнерами оконечных точек.

На рис. 6.1 показаны компоненты, принимающие участие в коммуникациях WCF.

Рис. 6.1. Компоненты, принимающие участие в коммуникации WCF

Клиент вызывает какой-либо метод прокси-класса. Прокси-класс предлагает методы, предоставляемые службой, но их вызов преобразуется в сообщение, которое затем передается по каналу. Канал имеет клиентскую и серверную части, взаимодействующие между собой через сетевой протокол. Из канала сообщение передается диспетчеру, который преобразует его обратно в вызов метода, после чего указанный метод вызывается в службе.

В WCF поддерживается несколько коммуникационных протоколов: SOAP, WSDL (Web Services Description Language – язык описания веб-служб), REST (Representational State Transfer – передача репрезентативного состояния) JSON (JavaScript Object Notation – представление объектов JavaScript).

Создание службы

В данном разделе будет рассмотрено создание простого WCF сервиса. В качестве хоста для данного сервиса было выбрано консольное приложение.

Для того чтобы консольное приложение могло работало с WCF, необходимо добавить ссылку на сборку System.ServiceModel.

Создание службы включает в себя несколько этапов.

Определение контракта. Пишется класс, который делает нечто полезное, после чего он снабжается атрибутами WCF. Атрибут [ServiceContract] помечает класс как контракт. Атрибут [OperationContract] определяет методы класса, которые можно вызывать через интерфейс службы. Одновременно он определяет, какие сообщения можно передать этим методам и получить от них.

Определение оконечной точки. В определении оконечной точки адрес, привязка и контракт задаются с помощью метода AddServiceEndpoint класса ServiceHost. Адрес мы оставляем пустым, так как адрес оконечной точки такой же, как адрес самой службы. В качестве привязки указывается basicHttpBinding, обеспечивающая совместимость с большинством систем, в которых реализованы Web-службы на базе XML.

Размещение службы в процессе, чтобы она могла прослушивать входящие запросы. В приведенном примере служба размещается в консольном

приложении. Служба ожидает поступления запросов на адрес http://localhost:8000/FirstWCFService:

```
using System;
using System.ServiceModel;

namespace FirstWCFService
{
    [ServiceContract]
    public interface ITestService
    {
        [OperationContract]
        string SayHello(string ticker);
    }

    public class TestService : ITestService
    {
        public string SayHello(string clientName)
        {
            return string.Format("Hello {0}", clientName);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ServiceHost serviceHost = new ServiceHost(typeof(TestService),
            new Uri("http://localhost:8000/FirstWCFService"));
            serviceHost.AddServiceEndpoint(typeof(ITestService),
            new BasicHttpBinding(), "");
            serviceHost.Open();
            Console.WriteLine("Для выхода нажмите ENTER.\n");
            Console.ReadLine();
            serviceHost.Close();
        }
    }
}
```

В WCF имеется развитая поддержка для определения атрибутов службы в конфигурационных файлах. В конфигурационные файлы можно перенести задание адресов, привязок и поведений оконечных точек. Задание оконечных точек и поведения в конфигурационных файлах обеспечивает большую гибкость по сравнению с заданием в коде.

В случае применения конфигурационного файла функцию main из нашего примера можно переписать в следующем виде:

```
static void Main(string[] args)
{
    ServiceHost serviceHost = new ServiceHost(typeof(TestService));
    serviceHost.Open();
    Console.WriteLine("Для выхода нажмите ENTER.\n");
    Console.ReadLine();
    serviceHost.Close();
}
```

Файл конфигурации проекта app.config в этом случае должен выглядеть следующим образом:

```
<?xml version="1.0"?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="FirstWCFService.TestService">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8000/FirstWCFService"/>
          </baseAddresses>
        </host>
        <endpoint address=""
         binding="basicHttpBinding"
         contract="FirstWCFService.ITestService" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Как видно, описание службы находится в секции system.serviceModel. Внутри нее определяются службы, привязки, поведение службы, клиенты, способы диагностики, расширения и др. Как минимум, должна существовать подсекция <services>, в которой описаны оконечные точки, а в ней хотя бы один узел <endpoint>, описывающий не инфраструктурную точку.

Атрибут address определяет URI, на который клиенты будут посылать сообщения оконечной точке. Атрибут binding описывает коммуникационные детали, необходимые для соединения со службой. В частности, определяется весь стек каналов, который должен как минимум включать канал сетевого адаптера. Помимо этого, можно включить в стек каналы шифрования, сжатия и другие.

Атрибут contract ссылается на тип, определенный для оконечной точки службы. WCF инспектирует этот тип и раскрывает его метаданные в виде оконечной точки MEX, если таковая входит в состав службы.

Для работы с конфигурационными файлами в Visual Studio предусмотрена специальная утилита (рис. 6.2). Для редактирования нашего конфигурационного файла нужно выполнить следующие действия:

1. Выполнить команду Tools→WCF Service Configuration Editor.

2. В появившемся окне выполнить File→Open→Config File и в диалоговом окне выбрать конфигурационный файл app.config проекта.

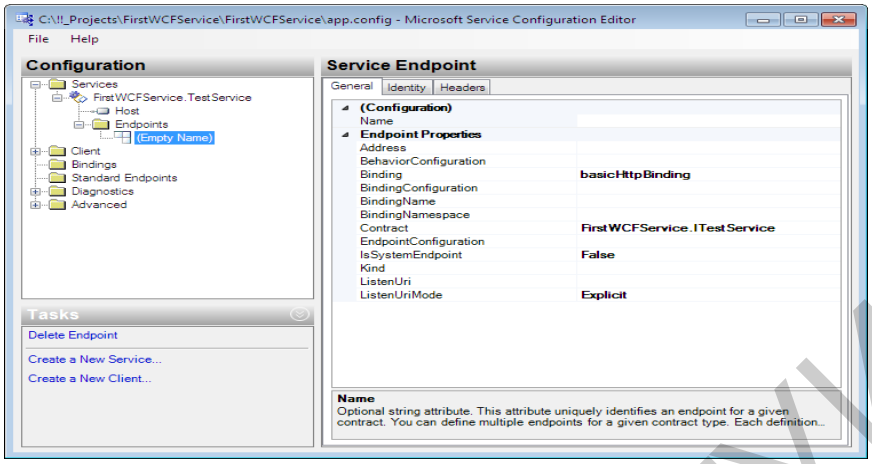

Рис. 6.2. Диалоговое окно конфигуратора службы WCF

Добавление оконечных точек. Включение оконечной точки обмена метаданными (MEX). Метаданные в WCF содержат информацию, точно описывающую, как следует обращаться к службе. Запросив у работающей службы метаданные, клиент может узнать о ее оконечных точках и требуемых форматах сообщений. На этапе проектирования клиенты посылают такой запрос в виде сообщения, определенного в стандарте WSMetadataExchange, и получают в ответ WSDL документ. Этот документ клиент может использовать для генерации прокси-класса и конфигурационного файла, которые впоследствии будут использоваться для доступа к службе во время выполнения.

По умолчанию WCF-службы не раскрывают оконечную точку MEX. Это означает, что никто не сможет узнать у службы, как с ней взаимодействовать. Не зная адреса, привязок и контракта, очень трудно обратиться к службе, которая не занесена в реестр. Однако WCF позволяет очень просто раскрыть оконечную точку MEX, чтобы клиенты могли корректно общаться со службой.

Для начала необходимо подключить в исходном файле пространство имен System.ServiceModel.Description.

Далее будет рассмотрено создание оконечной точки MEX при помощи конфигуратора.

Для создания оконечной точки MEX при помощи конфигуратора необходимо выполнить следующую последовательность действий.

1. В окне конфигуратора выделить подраздел Services→ FirstWCFService.TestService →EndPoints и выполнить команду File→Add New Item Service Endpoint.

2. В появившемся окне мастера в поле Contract ввести IMetadataExchange и нажать Next.

3. Выбрать протокол HTTP и нажать 2 раза Next.

4. В поле Address ввести mex. Нажать Next а затем Finish.

В результате будет создана новая оконечная точка. У только что созданной оконечной точки нужно изменить параметр Binding (в правой панели) на mexHttpBinding.

Далее необходимо настроить поведение сервиса (Service Behavior), чтобы он мог передавать метаданные по запросу клиента, выполнив следующую последовательность действий:

1. Выделить подраздел Advanced→Service Behaviors и из контекстного меню выбрать пункт New Service Behavior Configuration.

2. В правой панели нажать Add, в появившемся окне выбрать из списка serviceMetadata и нажать OK.

3. В стеке щелкнуть по вновь созданному элементу (NewBehavior0) и в настройках элемента установить параметру HttpGetEnabled значение True.

4. Перейти в подраздел Services→FirstWCFService.TestService и в правой панели параметру Behavior Configuration установить значение NewBehavior0.

5. Сохранить изменения в файле конфигурации.

Теперь клиент сможет получить описание нашей службы, отправив соответствующий MEX-запрос.

В результате файл конфигурации будет иметь следующий вид:

```
<?xml version="1.0"?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="NewBehavior0">
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="FirstWCFService.TestService">
        <endpoint address="" binding="basicHttpBinding"
                  contract="FirstWCFService.ITestService"/>
        <endpoint address="mex" binding="mexHttpBinding"
                  contract="FirstWCFService.ITestService"/>
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8000/FirstWCFService" />
          </baseAddresses>
        </host>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Реализация клиента

Для программирования клиента, обращающегося к службе, необходимо, во-первых, сгенерировать конфигурационный файл и прокси-класс, и, во-вторых, написать код, который будет с помощью прокси-класса обращаться к службе.

Создадим новый проект консольного приложения под названием FirstWCFClient и добавм ссылку на сборку System.ServiceModel. Далее добавим ссылку на созданный сервис при помощи команды Add Service Reference. Данная операция в Visual Studio применяется для получения метаданных от WCF-службы и генерации прокси-класса и конфигурационного файла. Прокси-класс позволяет клиенту обращаться к операциям службы так, будто они являются методами локального класса.

Для добавления ссылки на сервис необходимо выполнить следующие действия:

1. Выполнить команду Project→Add Service Reference.
2. В поле Address ввести адрес http://localhost:8000/FirstWCFService/mex и нажать GO (предварительно убедившись, что сервис запущен).
3. В случае удачи в левой панели отобразится список доступных сервисов. В этом случае необходимо ввести в поле NameSpace имя пространства имен (ServiceReference1), в котором будет определен прокси-класс и после нажатия OK будет создан прокси-класс и файл конфигурации клиента.

Напишем код, который будет вызывать метод SayHello службы.

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

namespace FirstWCFClient
{
    class Program
    {
        static void Main(string[] args)
        {
            FirstWCFClient.ServiceReference1.TestServiceClient proxy =
                                    new ServiceReference1.TestServiceClient();
            string request = proxy.SayHello("Client");
            Console.WriteLine(request);
            proxy.Close();
            Console.Read();
        }
    }
}
```

В результате выполнения данного кода на экран будет выведено сообщение, приведенное на рис. 6.3.

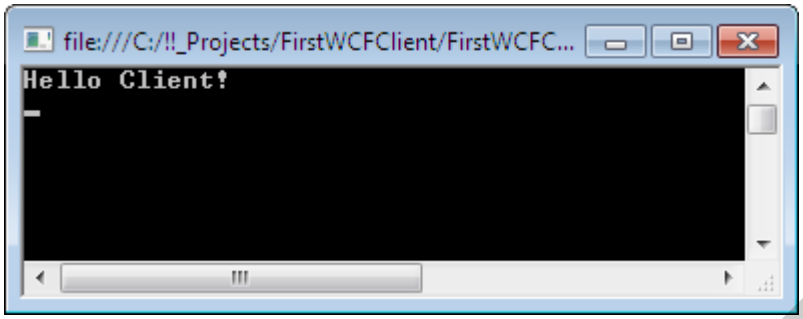

Рис. 6.3. Сообщение, переданное сервером

Индивидуальное задание

При помощи технологии WCF создайте распределенное приложение-чат, которое включает в себя две составляющие: службу и клиента.

Служба должна выполнять следующие функции:

– добавление сообщения;

– получение сообщений.

В качестве хранилища сообщений должна использоваться база данных.

В качестве хоста для службы можно выбрать консольное приложение.

В качестве приложения-клиента должно выступать оконное приложение. В приложении-клиенте должен быть реализован просмотр сообщений, добавление сообщений, а также получение сообщений.

Просмотр сообщений должен осуществляться посредством компонента ListBox. Для каждого сообщения должен отображаться текст сообщения, имя автора, а также дата добавления.

При запуске приложение-клиент должно запрашивать имя пользователя и адрес службы, без ввода которых приложение не должно позволять выполнять какие-либо действия.

Получение сообщений должно быть реализовано следующим образом: если приложение-клиент еще не загружало сообщения, то оно должно запрашивать у службы все сообщения, имеющиеся в базе, иначе должны быть загружены только те сообщения, которые были добавлены после последнего запроса списка сообщений.

Лабораторная работа №7
РАБОТА С ASP.NET В СРЕДЕ VISUAL STUDIO

Цель: изучить на практике основные принципы разработки ASP.NET приложений.

Краткие теоретические сведения

Как известно, при программировании в Web строят отдельно клиентские и серверные приложения. Задача серверного приложения – обработать данные сайта, переданные браузером. C# позволяет создавать серверные приложения.

Для создания Web-приложений необходимо установить и запустить Web-сервер IIS (Internet Information Server). Алгоритм создания Web-приложений сводятся следующий.

– Создать новый проект на базе шаблона ASP.NET Web Application и присвоить ему имя, например, MyWebApplication1. При вводе имени приложения указать также в поле Location адрес: http://localhost/MyWebApplication1. Создаваемые файлы будут помещены в папку MyWebApplication1 каталога wwwroot. По умолчанию IIS использует каталог c:\InetPub\wwwroot, однако, этот каталог можно изменить, используя окно установки свойств IIS.

– Расположить на форме визуальные компоненты, например, Label (либо TextBox) и Button. Запрограммировать кнопку, к примеру:

```
{ label1.Text = "Hello From Server! ";}
```

– Чтобы проверить работу формы, необходимо просто вызвать ее на выполнение командой **Debug→Start** либо использовать комбинацию клавиш <CTRL>+<F5>. В результате компиляции проекта система создаст необходимые файлы Web-приложения в указанном каталоге MyWebAppication1.

Система построит для созданного Web-приложения два файла: WebForm1.aspx, содержащий код HTML и ASP.NET, и файл WebForm1.aspx.cs, содержащий код C#. Файл с кодом HTML используется для возврата клиенту формы документа с результатом действия нашей кнопки. Файл с кодом C# выполняет программу для кнопки.

Содержимое файла WebForm1.aspx будет иметь следующий вид:

```
<%@ Page language="c#" Codebehind="WebFormsApp1.aspx.cs"
        AutoEventWireUp="false"Inherits="MyWebApplication.WebForm1" %>
<!DOCTYPE HTML PUBLIC "~//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <title> WebForm1</title>
```

```
        <meta content="Microsoft Visual Studio 7.0" name="Generation">
        <meta content="C#" name="CODE_LANGUAGE">
        <meta content="JavaScript" name="vs_defaultClientScript">
        <meta content=http://schemas.microsoft.com/intellisense/ie5 name="vs_targetSchema">
    </HEAD>
    <body MS_POSITIONING="GridLayout">
        <form id="Form1" method=”post” runat="server">
            <asp:TextBox id="TextBox1" style="Z-INDEX: 101; LEFT: 24px;
                POSITION: absolute; TOP: 20px" runat="server" Width="411px"
                Height="187px" TextMode="MultiLine"> </asp:TextBox>
            <asp:Button id="Button1" style="Z-INDEX: 102; LEFT: 40px;
                POSITION: absolute; TOP: 312px" runat="server" Width="104px"
                Height="27px" Text=" PRESS" > </asp:Button>
        </form>
    </body>
</HTML>
```

Приведенный документ – это ASP.NET-страница (Active Server Pages – активные серверные страницы). ASP-страница содержит вперемежку теги HTML и ASP-теги. ASP-тег

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
AutoEventWireUp="false"
Inherits="MyWebApplication.WebForm1" %>
```

указывает, что рабочим языком страницы является C#, а имя программного модуля, поддерживающего Web-форму, – WebForm1.aspx.cs. Указывается, что классом формы является Inherits="MyWebApplication.WebForm1".

Наконец, сообщается через AutoEventWireUp="false", что процессор ASP.NET не будет автоматически вызывать методы обработки событий инициализации и загрузки ASP-страницы. Чтение остальных тегов не вызывает затруднений.

Теперь, чтобы обратиться из клиентской стороны (сайта) к данному серверному приложению, нужно в окне браузера ввести адрес:

```
http://localhost/MyWebApplication1
```

Если обращение к ASP-странице должно производиться с другого компьютера, то вместо localhost следует указать сетевой адрес компьютера, содержащего ASP-страницу.

Для активизации созданных нами ASP-страниц на стороне сервера, должно быть активным приложение Web-сервера. Для C# используем в качестве Web-сервера IIS. Этот Web-сервер входит в состав Windows и его следует инсталлировать и запустить.

Для инсталляции IIS выберите Пуск→Настройка→Панель управления→ Установка/удаление программ→Установка/удаление компонентов Windows. Затем отметьте окошко IIS и нажмите кнопку Next.

Система пригласит вас вставить компакт-диск с версией Windows, что и надлежит сделать.

После инсталляции IIS этот сервер нужно запустить. Для этого снова выберите Пуск→Настройка→Панель управления→Сервисы. Найдите в списке сервисов Internet Information Services и активизируйте контекстное меню щелчком правой кнопки мыши на имени IIS. Выберите пункт Start.

Попытайтесь связаться с главной страницей IIS, открыв окно Internet Explorer и указав в качестве URL в поле адреса http://www.localhost. Если IIS успешно инсталлирован и запущен, то откроется домашняя страница этого Web-сервера.

Имейте в виду, что Web-сервер IIS требует размещать создаваемые ASP-страницы в каталоге C:\Inetpub\wwwroot. Запомните это. Следовательно, создаваемые визуальной средой C# страницы ASP следует размещать в этом каталоге.

Другая возможность создания Web-приложений состоит в использовании шаблонов Empty Web Project. Эта возможность является аналогом сервлетов языка Java. Создадим подобное простейшее приложение:

– Выберите шаблон **Empty Web Project** и присвойте ему имя webHandle. Включите в этот проект ссылку на библиотеку System.Web, для чего следует щелкнуть правой кнопкой мыши в строке **References**, выбрать в контекстном меню пункт **Add Reference**, а затем выбрать из выпадающего списка опцию System.Web.dll, щелкнув по ней дважды.

– Откройте контекстное меню проекта и выберите **Add→Add New Item→Text File**. Присвойте новому файлу имя webHandlerC.ashx. Введите в него следующий код

```
<%@ WebHandler Language="c#"
      Codebehind="webHandlerC.ashx.cs" class ="webHandle.webHandlerC" %>
```

– Теперь нужно добавить в проект файл на языке C#, именованный как webHandlerC. Для этого щелкните правой кнопкой мыши на проекте и выберите Add→Add New Item→Code File. Присвойте файлу имя webHandlerC.ashx.cs. Введите следующий код

```
using System.Web;
namespace webHandle
{
    public class webHandlerC : IHttpHandler
    {
        public void ProcessRequest (HttpContext context)
        {
            HttpRequest request=context.Request;
```

```
            HttpResponse resp=context.Response;
            resp.ContentType="text/plain";
            resp.Write(" Hello from C#-script");
            resp.End();
        }

        public bool IsReusable
        {
            get{ return true; }
        }
    }
}
```

– Откройте панель свойств `webHandlerC.ashx` и установите свойство Build Action в значение Content. После этого выберите команду Set as Start Page в контекстном меню `webHandlerC.ashx`.

– Добавьте в проект конфигурационный файл:

Add→Add New Item→Web Configuration File

– Запустите проект на выполнение: <CTRL>+<F5>.

Построенный нами проект есть аналог сервлета языка Java. Обращение к созданному Web-приложению осуществляется через URL: http://localhost/webHandle.

В приведенном примере ключевую роль играют переменные resp и request.

Переменная request позволяет получить данные из формы клиентского сайта. Наиболее просто это сделать, используя по аналогии приведенные далее команды:

```
int i = int.Parse(context.Request["tfage"]); // Получаем значение
// элемента формы HTML-документа с именем tfage и преобразуем его
// к целому числу
s= context.Request["Uname"];
```

Здесь в квадратных скобках после Request указаны имена текстовых полей, размещенных на форме клиента и переданных браузером нашему Web-приложению. Команда int.Parse преобразует строку в целое число.

Переменная resp позволяет вернуть на сторону клиента HTML-документ. Пояснение дает следующий пример.

```
int i1, i2; //индексы для массивов
NameValueCollection params=Request.QueryString; // Получаем
// коллекцию имен и значений параметров
// В массив arr заносим имена параметров
String[] arr = params.AllKeys;
for (i1 = 0; i1 < arr.Length; i1++)
{
```

```
        // Выводим на сторону клиента имя параметра в цикле
        Response.Write("Key: " + Server.HtmlEncode(arr[i1]) + "<br>");
        // Получаем значения параметра в массиве arr2
        String[] arr2 = params.GetValues(arr[i1]);
        for (i2 = 0; i2 < arr2.Length; i2++)
        {
            // Выводим значения параметра на сторону клиента
            Response.Write("Value " + i2 + ": " + Server.HtmlEncode(arr2[i2]) + "<br>");
        }
    }
```

Теперь покажем, как создавать ASP.NET-страницы, содержащие пользовательские компоненты. Создадим заготовку приложения WEB ASP. Вручную напишем следующий код Web-страницы:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
                    Inherits="web2._Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server">
        <title>Untitled Page</title>
    </head>
    <body bgcolor=#aabbcc>
        <center>
            <h2>
                <font color=red>LESSON 1</font>
                <hr width=800 />
            </h2>
        </center>
        <form id="form1" runat="server">
            <div></div>
        </form>
    </body>
</html>
```

Этот код набираем в окне Source. Откроем этот документ в броузере через Run→Run Without debugging. Броузер отобразит созданное окно на рис 7.1.

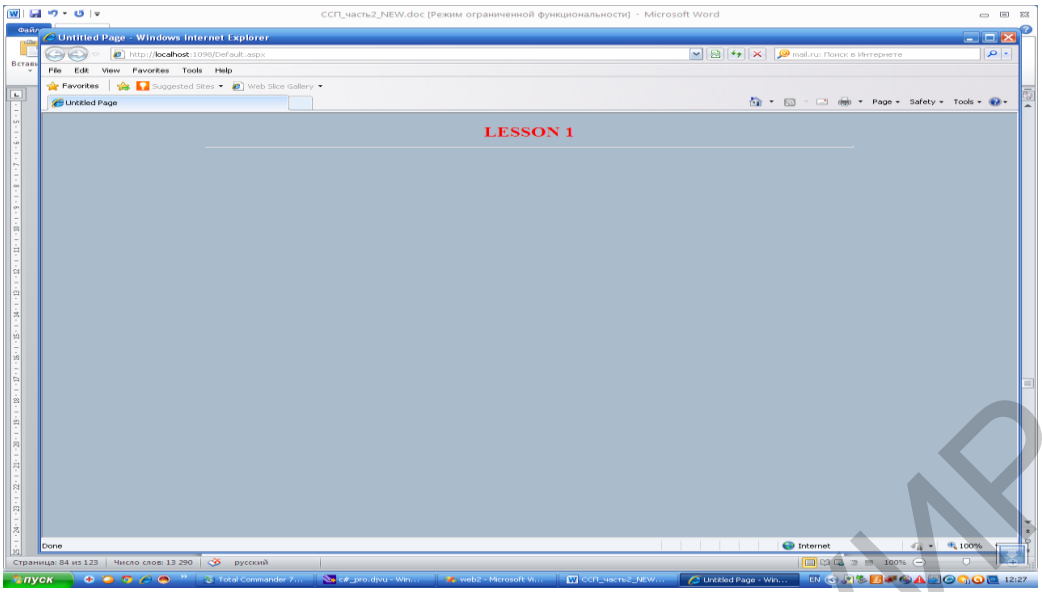

Рис.7.1. Снимок экрана приложения

Теперь добавим в сайт пользовательский элемент управления, вызвав контекстное меню щелчком правой кнопкт мыши на элементе web2 (название проекта) и выбра пункт Add→New Item→Web user control:В поле Name введем myuserctrl1. Система отобразит окно с текстом

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="myctrl1.ascx.cs"
                          Inherits="web2.myctrl1" %>
```

Расширим этот текст следующим образом:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="myctrl1.ascx.cs"
                    Inherits="web2.myctrl1" %>
<table cellspacing="4">
    <tr valign=middle>
        <td>
            <asp:Image runat="server" ID="suitPic" ImageUrl="e:\work\toad1.bmp" />
        </td>
        <td>
            <asp:Label runat="server" ID="suitLabel">To do something</asp:Label>
        </td>
    </tr>
</table>
```

После этого с помощью мыши перетащим данный элемент на форму (рис.7.2):

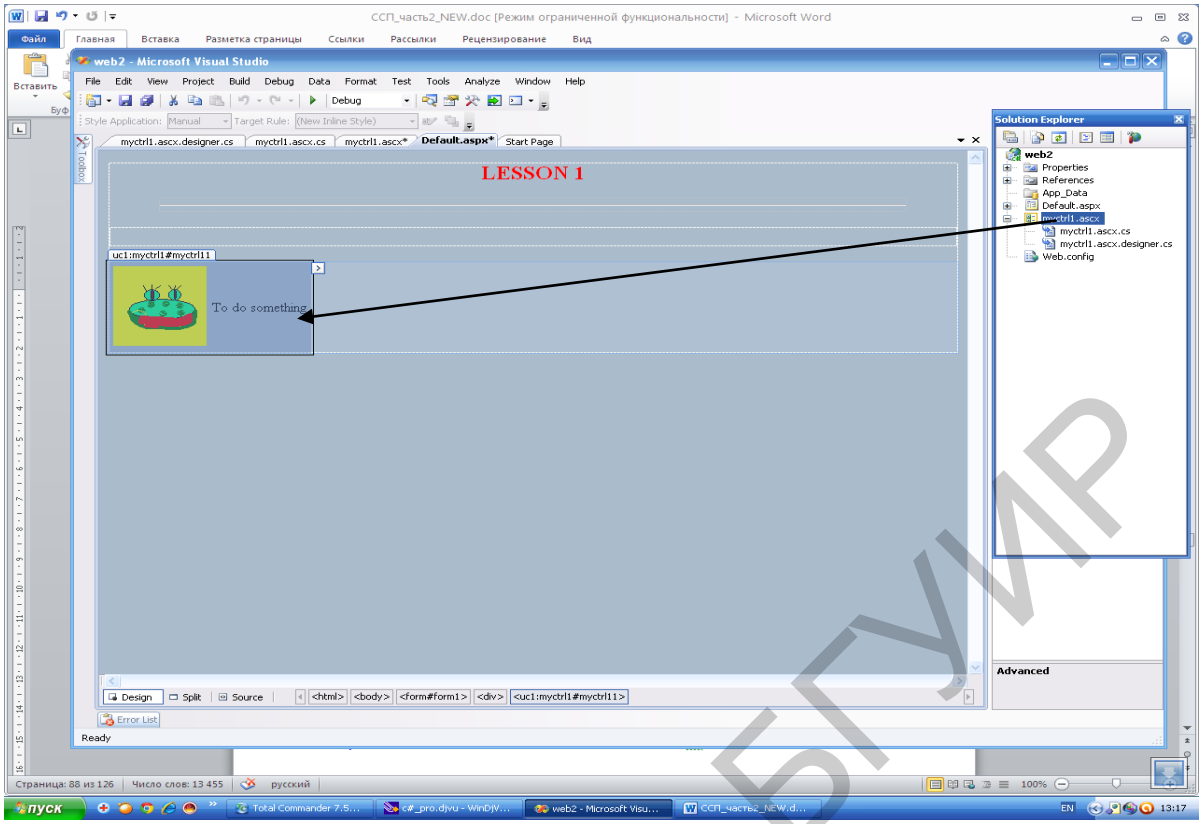

Рис.7.2. Модифицированный пример приложения

Система создаст следующий код в основном документе – default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
                Inherits="web2._Default" %>
<%@ Register src="myctrl1.ascx" tagname="myctrl1" tagprefix="uc1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server">
        <title>Untitled Page</title>
    </head>
    <body bgcolor=#aabbcc>
        <form id="form1" runat="server">
            <center>
                <h2><font color=red>LESSON 1</font><hr width="800" /></h2>
            </center>
            <div>
                <uc1:myctrl1 ID="myctrl11" runat="server" />
            </div>
        </form>
    </body>
</html>
```

Оттестируем приложение. Теперь надо придать нашему элементу некоторую функциональность. Сначала добавим список радиокнопок:

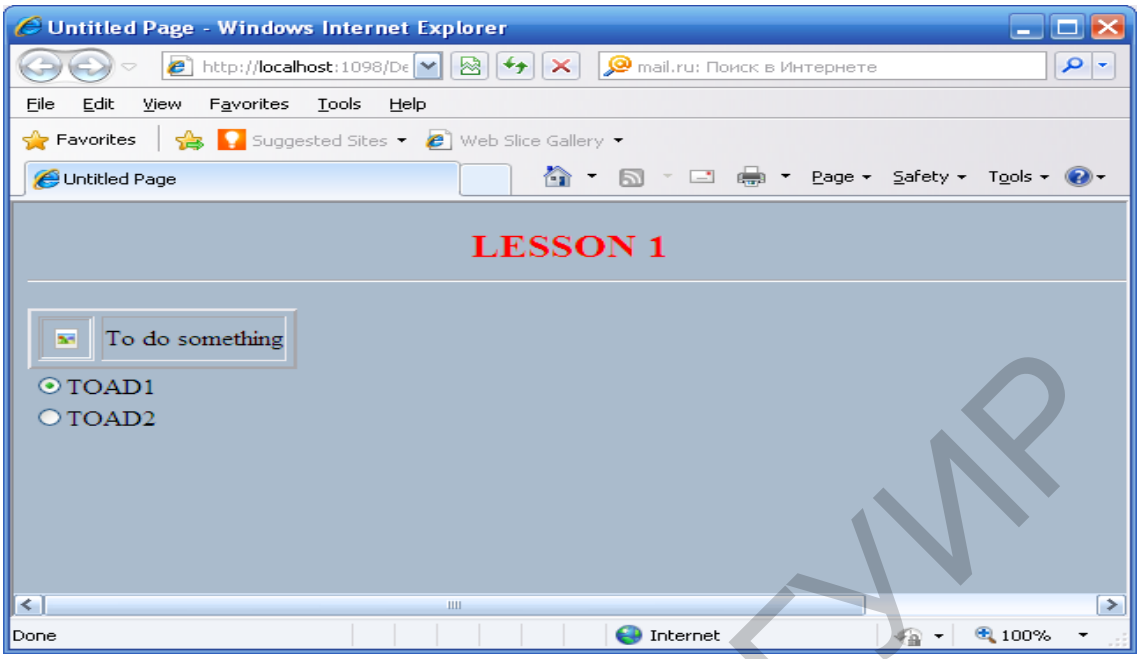

Рис.7.3. Окончательный вариант приложения

Исходный текст документа претерпит следующие изменения:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
                Inherits="web2._Default" %>
<%@ Register src="myctrl1.ascx" tagname="myctrl1" tagprefix="uc1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server">
        <title>Untitled Page</title>
    </head>
    <body bgcolor=#aabbcc>
        <form id="form1" runat="server">
            <center>
                <h2><font color=red>LESSON 1</font><hr width="800" /></h2>
            </center>
            <div>
                <uc1:myctrl1 ID="myctrl11" runat="server" />
                <asp:RadioButtonList runat="server" ID="suitList"
                            AutoPostBack="true">
                    <asp:ListItem Value="e:\work\toad1.bmp" Selected="True">
                            TOAD1</asp:ListItem>
                    <asp:ListItem Value="e:\work\toad2.bmp" >TOAD2</asp:ListItem>
                </asp:RadioButtonList>
            </div>
        </form>
    </body>
</html>
```

Для написания кода реакции на радиокнопку щелкнем по радиокнопке дважды. Откроется следующий редактор

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

namespace web2
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }

        protected void suitList_SelectedIndexChanged(object sender, EventArgs e)
        {
        }
    }
}
```

Нас интересует событие SelectedIndexChanged. Чтобы это событие обработать, расширяем определение класса нашего элемента myCtrl:

```
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

namespace web2
{
    public partial class myctrl1 : System.Web.UI.UserControl
    {
        protected string currentpic = @"e:\work\toad1.bmp";
        public string Suit
        {
            get
            {
                return currentpic;
            }

            set
```

```
            {
                currentpic = value;
                suitLabel.Text = value;
                suitPic.ImageUrl = value;
            }
        }

        protected void Page_Load(object sender, EventArgs e)
        {
        }
    }
}
```

Теперь нетрудно запрограммировать и сам обработчик события web2.default.aspx.cs:

```
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

namespace web2
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }

        public void suitList_SelectedIndexChanged(object sender, EventArgs e)
        {
            myctrl11.Suit = (string) suitList.SelectedItem.Value.ToString();
        }
    }
}
```

Тем самым в сайт добавлен собственный элемент и запрограммирована его обработка.

Наконец, вспомним работу с базами данных (см. в ASP.NET-приложение). Следующее консольное приложение показывает работу с OleDB-источником данных (разберитесь в нем самостоятельно).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.OleDb;
```

```
namespace database1
{
    class Program
    {
        static void Main(string[] args)
        {
            string connectingString =
                @"provider=Microsoft.Jet.OLEDB.4.0;data source= e:\work\mydb.mdb";
            OleDbConnection myConn = new OleDbConnection (connectingString);
            OleDbCommand myCommand= myConn.CreateCommand();
            myCommand.Connection = myConn;
            myConn.Open();
            string cmdString ="Select * from stud";
            myCommand.CommandText= cmdString;
            DataSet myDataset= new DataSet();
            OleDbDataAdapter oda= new OleDbDataAdapter(myCommand);
            oda.Fill(myDataset);
            Console.WriteLine(myDataset.Tables[0].Rows[0]["fio"].ToString());
            string s= Console.ReadLine();
            myConn.Close();
        }
    }
}
```

Здесь используется таблица stud с полями fio, rating.

Индивидуальное задание

1. Выполнить все примеры, приведенные в теоретической части и разобраться в неясных местах.
2. Создать собственную базу данных в ACCESS.
3. Разработать серверный ASP.NET-сценарий, который обращается к вашей базе данных (например, временя начала лекции или какая-то иная справочная информация (наличие или цена товара, характеристика человека и пр.)).
4. В ASP-сценарии использовать собственный элемент пользователя в качестве интерфейса с базой данных.
5. Содать клиентский сайт (средствами HTML) для доступа к ASP.NET-приложению.
6. Подготовить отчет и защитить работу.

Лабораторная работа № 8
WEB-СЕРВИСЫ

Цель: изучить механизмы WEB-сервисов и их использование в NET.

Краткие теоретические сведения

Отличие WEB-сервисов от ASP.NET-приложений состоит в том, что WEB-сервисы используют компоненты COM, а не скрипты. Таким образом, можно включить в приложение класс, объявленный в компоненте COM, и использовать его методы. Создание WEB-сервиса производится на основе конфигурационных файлов XML. Рассмотрим простой пример. Предполагается, что сервер IIS запущен.

1. Выберем File→ New Project → ASP.NET Web Service

2. Выберем закладку, соответствующую файлу .cs. Из меню выберем View Code по щелчку правой кнопкой мыши. При этом откроется окно редактора кода:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace WebService2
{
    /// <summary>
    /// Summary description for Service1.
    /// </summary>
    public class Service1 : System.Web.Services.WebService
    {
        public Service1()
        {
            //CODEGEN: This call is required by the ASP.NET Web Services Designer
            InitializeComponent();
        }

        #region Component Designer generated code

        //Required by the Web Services Designer
        private IContainer components = null;

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
```

```
        /// </summary>
        private void InitializeComponent()
        {
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if(disposing && components != null)
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #endregion


//      [WebMethod]
//      public string HelloWorld()
//      {
//          return "Hello World";
//      }
    }
}
```

Откроем комментарии (//), начиная со строки [WebMethod]. .Запустим этот Web-сервис на выполнение (<CTRL>+F5). Откроется окно, показанное на рис. 8.1.

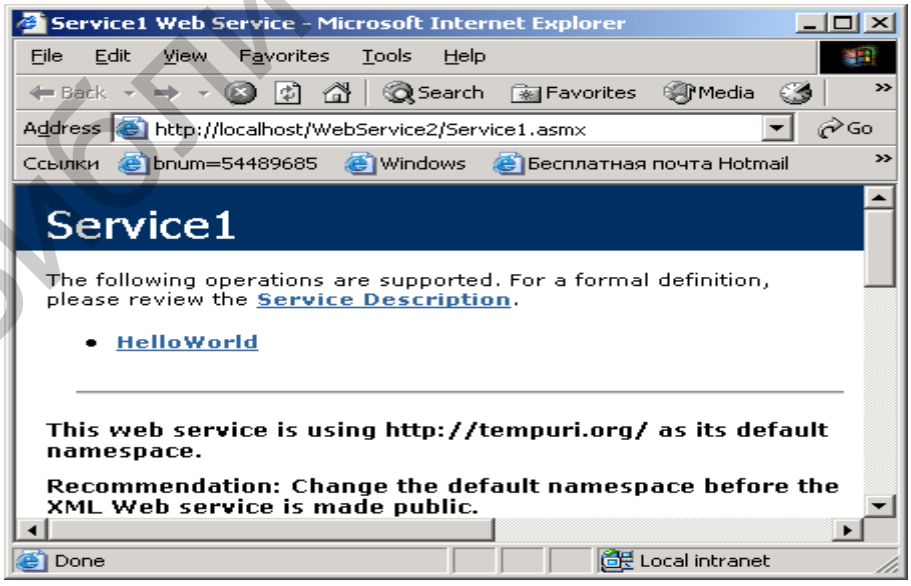

Рис.8.1. Установление соединения с Web-сервисом

4. Перейдем по гиперссылке HelloWorld.

5. В окне 2 увидим результат. При нажатии на кнопку Invoke получим результат в форме XML-документа.

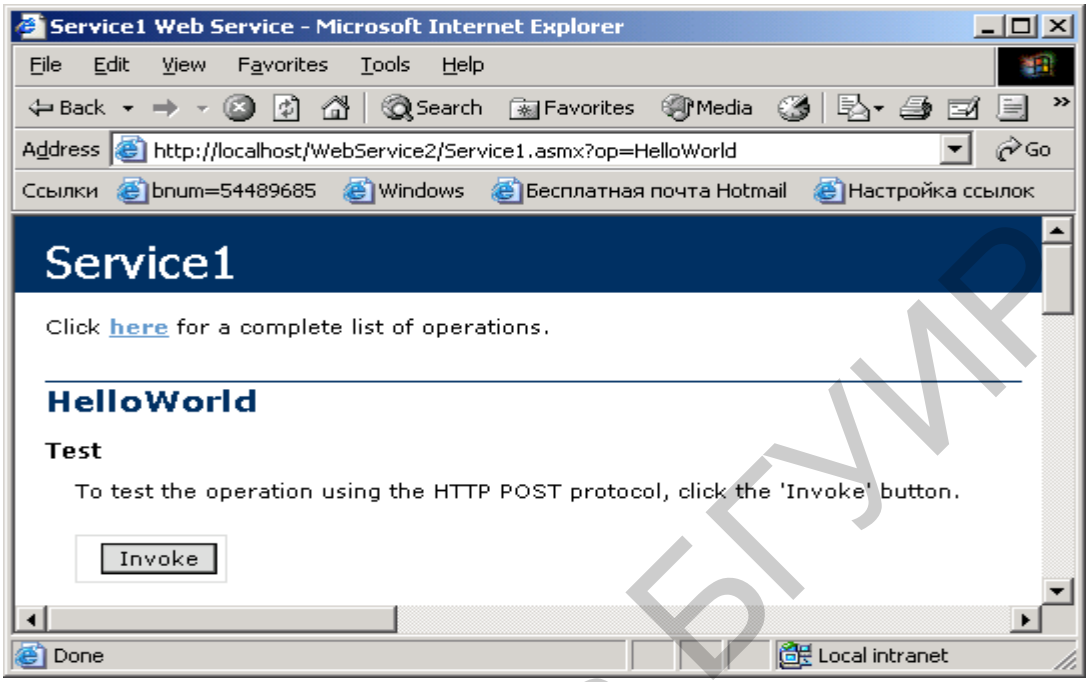

Рис.8.2. Пример работы приложения

Реализуем более сложный вариант сервиса. Именно, пусть web-сервис возвращает строку, переданную клиентом. Изменим несколько текст нашего сервиса

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace WebService2
{
    /// <summary>
    /// Summary description for Service1.
    /// </summary>
    public class Service1 : System.Web.Services.WebService
    {
        public Service1()
        {
            //CODEGEN: This call is required by the ASP.NET Web Services Designer
            InitializeComponent();
        }
```

```
        #region Component Designer generated code

        //Required by the Web Services Designer
        private IContainer components = null;

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if(disposing && components != null)
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #endregion

        // WEB SERVICE EXAMPLE
        // The HelloWorld() example service returns the string Hello World
        // To build, uncomment the following lines then save and build the
        //project
        // To test this web service, press F5
        [WebMethod]
        public string HelloWorld(String s)
        {
            return ("Received "+s);
        }
    }
}
```

Видим, что в метод HelloWorld передается в качестве параметра текстовая строка. Эта строка возвращается методом. Скомпилируем сервис с помощью комбинации клавиш <Ctrl>+F5. Теперь создадим обычное приложение на основе формы (Рис.8.3).

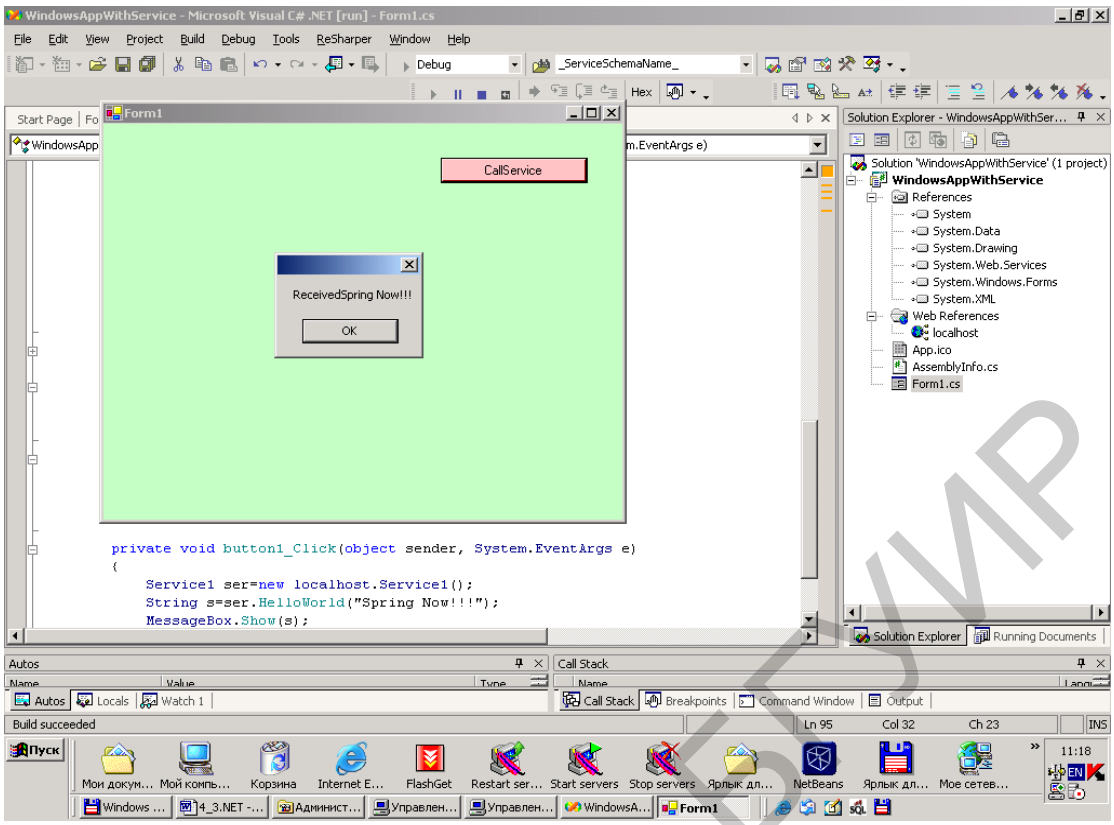

Рис.8.3. Экранная форма приложения

На этом рисунке показан результат обращения к сервису по нажатию на кнопку. Наше Windows-приложение имеет следующий вид:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using WindowsAppWithService.localhost;

namespace WindowsAppWithService
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Button button1;
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;
```

```
public Form1()
{
	//
	// Required for Windows Form Designer support
	//
	InitializeComponent();

	//
	// TODO: Add any constructor code after InitializeComponent call
	//
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
	if( disposing )
	{
		if (components != null)
		{
			components.Dispose();
		}
	}
	base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
	this.button1 = new System.Windows.Forms.Button();
	this.SuspendLayout();
	//
	// button1
	//
	this.button1.BackColor = System.Drawing.Color.FromArgb(
			((System.Byte)(255)), ((System.Byte)(192)), ((System.Byte)(192)));
	this.button1.Location = new System.Drawing.Point(312, 32);
	this.button1.Name = "button1";
	this.button1.Size = new System.Drawing.Size(136, 24);
	this.button1.TabIndex = 0;
	this.button1.Text = "CallService";
	this.button1.Click += new System.EventHandler(this.button1_Click);
	//
	// Form1
	//
	this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
```

```
            this.BackColor = System.Drawing.Color.FromArgb(
                    ((System.Byte)(192)), ((System.Byte)(255)), ((System.Byte)(192)));
            this.ClientSize = new System.Drawing.Size(480, 373);
            this.Controls.Add(this.button1);
            this.Name = "Form1";
            this.Text = "Form1";
            this.ResumeLayout(false);
        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.Run(new Form1());
        }

        private void button1_Click(object sender, System.EventArgs e)
        {
            Service1 ser=new localhost.Service1();
            String s=ser.HelloWorld("Spring Now!!!");
            MessageBox.Show(s);
        }
    }
}
```

В это приложение необходимо добавить ссылку (reference) на метод HelloWorld построенного сервиса. Для добавления ссылки в окне Solution Explorer активизируем контекстное меню на элементе References и выберем п. Add Web Reference. Затем в новом окне выберем пункт Web Services On Local Machine. Затем появится новое окно, в котором выберем имя нашего сервиса и нажмем кнопку Add Referernce. После этого правомочно адресоваться к классу Service1 в программе обработки события от кнопки:

```
private void button1_Click(object sender, System.EventArgs e)
{
    Service1 ser=new localhost.Service1();
    String s=ser.HelloWorld("Spring Now!!!");
    MessageBox.Show(s);
}
```

Индивидуальное задание

Реализовать сервис для выборки информации из базы данных по передаваемому SQL-запросу. Для простоты реализации обеспечить отправку только одной записи, выбранной по запросу.

Лабораторная работа №9
РАБОТА С XML-ДОКУМЕНТАМИ

Цель: освоить основные принципы работы с документами XML.

Краткие теоретические сведения

Продемонстрируем сначала создание документа вручную. Для этой цели будем использовать класс XmlDocument, XmlElement. Можно воспользоваться следующим кодом:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml;
using System.Xml.XPath;
using System.IO;

namespace xml1c
{
    class myxml1c
    {
        public static void Main(string[] args)
        {
            XmlDocument doc = new XmlDocument();
            XmlElement bookEl = doc.CreateElement("Book");
            bookEl.SetAttribute("Year", "2009");
            XmlElement titleEl = doc.CreateElement("Title");
            titleEl.InnerText = "Billi Timm";
            bookEl.AppendChild(titleEl);

            XmlElement titleEl2 = doc.CreateElement("Title");
            titleEl2.InnerText = "Brother Grimms";
            bookEl.AppendChild(titleEl2);
            doc.AppendChild(bookEl);

            StringBuilder sb = new StringBuilder();
            using (StringWriter sw= new StringWriter(sb))
            using (XmlTextWriter xtw = new XmlTextWriter(sw))
            {
                xtw.Formatting = Formatting.Indented;
                doc.WriteContentTo(xtw);
                Console.WriteLine(sb.ToString());
                Console.ReadLine();
            }
        }
    }
}
```

Создание тегов выполняется командой CreateElement. Присоединение тегов к родительскому узлу выполняет метод AppendChild.

Команда xtw.Formatting = Formatting.Indented необходима, чтобы вывод на экран выполнялся строка за строкой, а не в одну строку.

Теперь преобразуем программу таким образом, чтобы направить вывод в файл

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml;
using System.Xml.XPath;
using System.IO;

namespace xml1c
{
    class myxml1c
    {
        public static void Main(string[] args)
        {
            XmlDocument doc = new XmlDocument();
            XmlElement bookEl = doc.CreateElement("Book");
            bookEl.SetAttribute("Year", "2009");
            XmlElement titleEl = doc.CreateElement("Title");
            titleEl.InnerText = "Billi Timm";
            bookEl.AppendChild(titleEl);

            XmlElement titleEl2 = doc.CreateElement("Title");
            titleEl2.InnerText = "Brother Grimms";
            bookEl.AppendChild(titleEl2);
            doc.AppendChild(bookEl);

            StringBuilder sb = new StringBuilder();
            using (StringWriter sw= new StringWriter(sb))
            using (XmlTextWriter xtw = new XmlTextWriter(sw))
            {
                xtw.Formatting = Formatting.Indented;
                doc.WriteContentTo(xtw);
                FileStream fstm = new FileStream(@"e:\work\xml3.xml",
                                                 FileMode.OpenOrCreate);

                doc.Save(fstm);
                fstm.Close();
                Console.WriteLine(sb.ToString());
                Console.ReadLine();
            }
        }
    }
}
```

Собственно, вывод в файл выполняет здесь фрагмент

```
FileStream fstm = new FileStream(@"e:\work\xml3.xml",
FileMode.OpenOrCreate);
doc.Save(fstm);
fstm.Close();
```

Чтение XML- документа из файла выполняет такой код:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml;
using System.Xml.XPath;
using System.IO;

namespace xml1c
{
    class myxml1c
    {
        public static void Main(string[] args)
        {
            XmlDocument doc = new XmlDocument();
            XmlElement bookEl = doc.CreateElement("Book");
            bookEl.SetAttribute("Year", "2009");
            XmlElement titleEl = doc.CreateElement("Title");
            titleEl.InnerText = "Billi Timm";
            bookEl.AppendChild(titleEl);

            XmlElement titleEl2 = doc.CreateElement("Title");
            titleEl2.InnerText = "Brother Grimms";
            bookEl.AppendChild(titleEl2);

            doc.AppendChild(bookEl);

            StringBuilder sb = new StringBuilder();
            using (StringWriter sw= new StringWriter(sb))
            using (XmlTextWriter xtw = new XmlTextWriter(sw))
            {
                xtw.Formatting = Formatting.Indented;
                doc.WriteContentTo(xtw);
                FileStream fstm = new FileStream(@"e:\work\xml3.xml",
                                        FileMode.OpenOrCreate);

                doc.Save(fstm);
                fstm.Close();
                Console.WriteLine(sb.ToString());
                Console.ReadLine();
            }
```

```
            XmlDocument doc2 = new XmlDocument();
            doc2.Load(@"e:\work\xml4.xml");
            Console.WriteLine("Year {0}",
                  doc2.GetElementsByTagName("Root")[0].Attributes["Year"].Value);
            Console.WriteLine("Title ={0}",
                  doc2.GetElementsByTagName("Title")[0].InnerText);
            Console.ReadLine();
        }
    }
}
```

На скриншоте (рис. 9.1) видим результат работы этой программы.

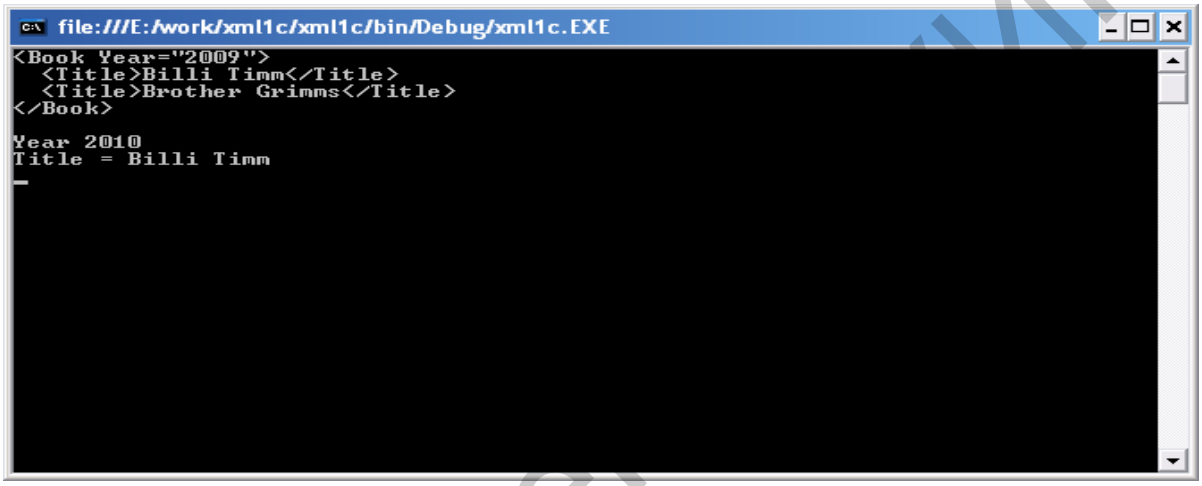

Рис.9.1. Результат работы программы

Выбор значений нужных тегов и атрибутов делает следующий фрагмент

```
doc2.GetElementsByTagName("Root")[0].Attributes["Year"].Value);
Console.WriteLine("Title ={0}",
doc2.GetElementsByTagName("Title")[0].InnerText);
```

Рассмотрим теперь преобразование документа XML в документ HTML. Для начала научимся запускать WEB-броузер. Следующий код показывает, как это сделать:

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

```
namespace XML_HTML
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            WebBrowser wb = new WebBrowser();
            wb.Navigate(@"e:\work\my.html", true);
        }
    }
}
```

Это приложение создает экземпляр броузера и открывает файл my.html (рис. 9.2).

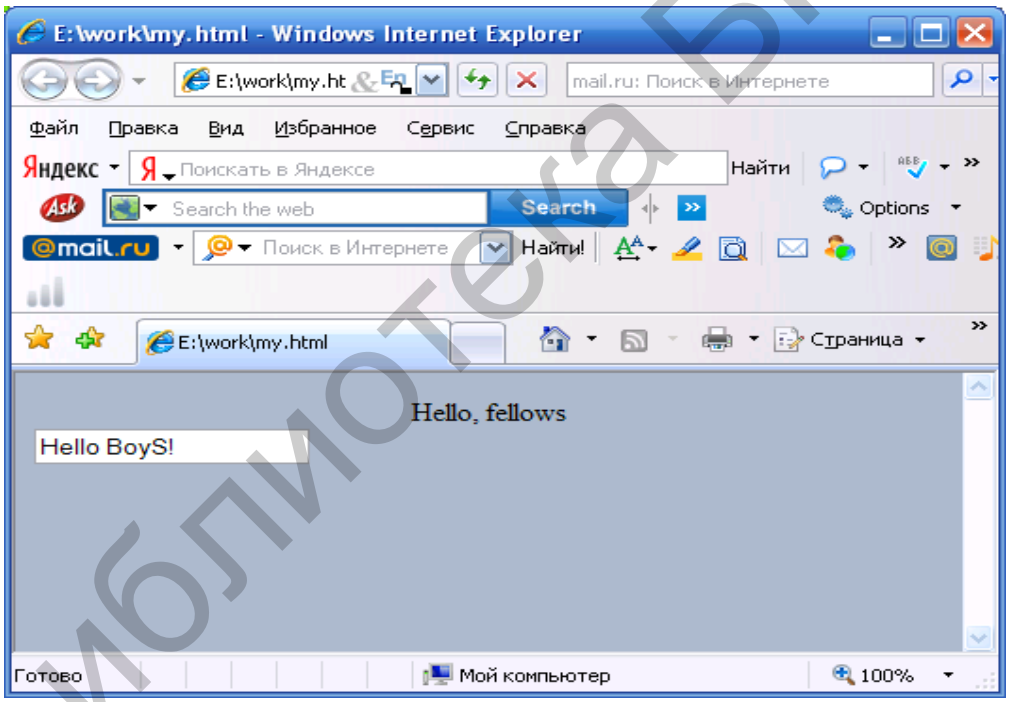

Рис.9.2. Результат работы программы

Пусть файл xml имеет следующий вид:

```
<?xml version="1.0" ?>
<root>
    <book>
        <title>Gold Key</title>
        <author>folk</author>
    </book>
```

```
    <book>
        <title>Cinderela</title>
        <author>Coster</author>
    </book>
    <book>
        <title>Three bears</title>
        <author>L.Tolstoy</author>
    </book>
</root>
```

Создадим новый html-файл для отображения в табличной форме содержимого приведенного выше xml-файла:

```
<html>
    <body bgcolor=#aabbcc>
        <center>
            <h2>XML in HTML</h2>
            <hr size=4 width=800><br>
            <xml src="e:\work\table.xml" id="myxml"></xml>
            <table id="tb" border=2 width="80%" datasrc="#myxml">
                <thead>
                    <th> title</th>
                    <th> author</th>
                </thead>
                <tr>
                    <td><span datafld="title"/></td>
                    <td><span datafld="author"/></td>
                </tr>
            </table>
        </center>
    </body>
</html>
```

Теперь в C#-программе нужно сделать ссылку на этот файл:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        WebBrowser wb = new WebBrowser();
        wb.Navigate(@"e:\work\table.html", true);
    }
}
```

Результат работы программы иллюстрируется следующим скриншотом:

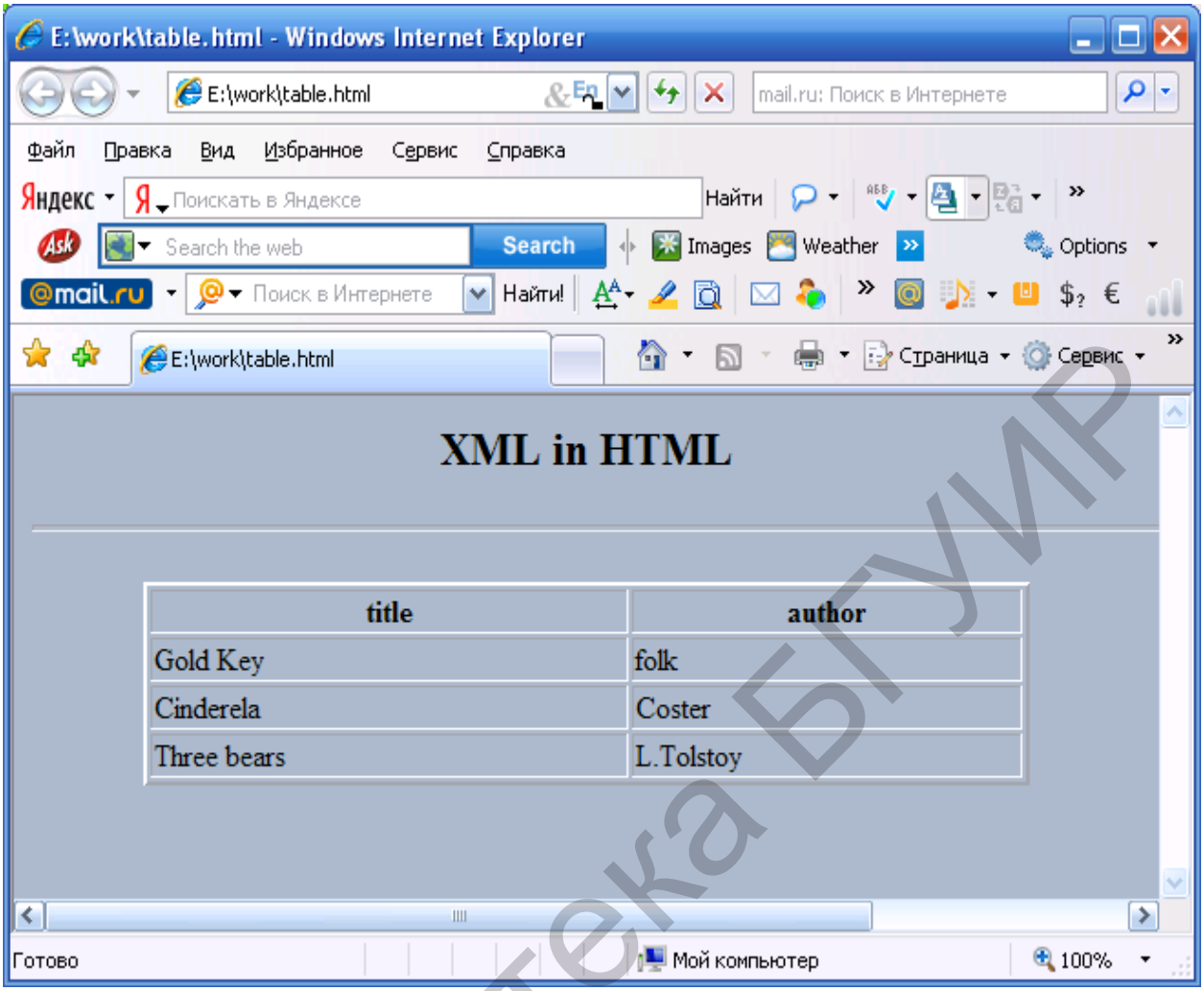

Рис. 9.3. Результат работы программы

Индивидуальное задание

1. Создать документ XML
2. Выполнить поиск в документе требуемого тега по его содержанию (или атрибуту)

Лабораторная работа №10
ТЕХНОЛОГИЯ WPF

Цель: изучить принципы технологии Windows Presentation Foundation.

Краткие теоретические сведения

Windows Presentation Foundation (WPF) – новая технология создания приложений на основе графического интерфейса. Код приложения параллельно создается в файле XAML (eXtended Application Markup Languge). Этот язык используется также в SilverLight, WFC. Чтобы быстро войти в курс этой технологии, создадим простое приложение. Создадим новый проект в NET 2008(2010) – Windows – WPF Application – (имя проекта). Система создаст следующую заготовку (рис.10.1)

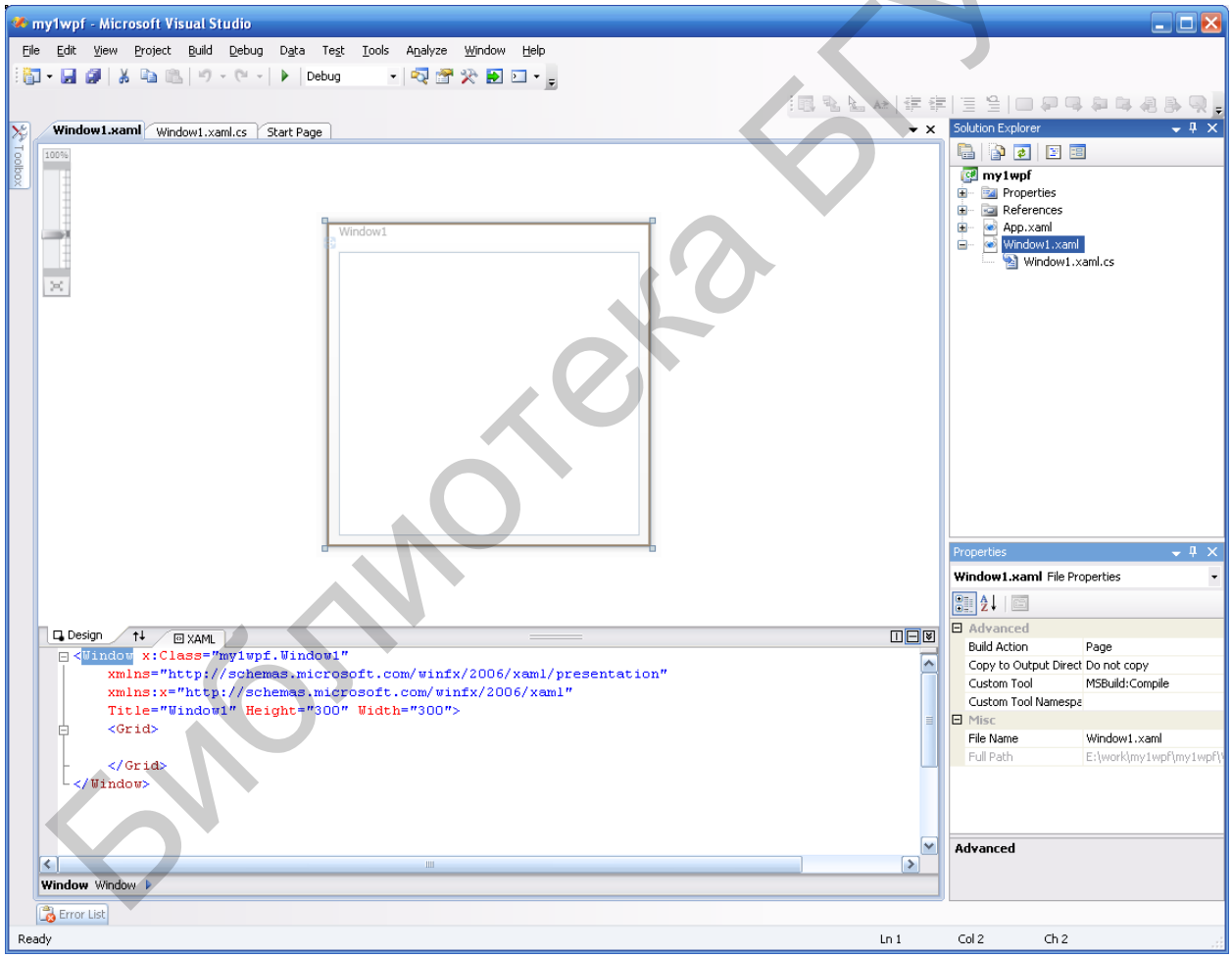

Рис.10.1. Рабочая среда Visual Studio

В нижнем окне мы видим код XAML:

```
<Window x:Class="my1wpf.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Window1" Height="300" Width="300">
   <Grid>
   </Grid>
</Window>
```

Этот код представляет заготовку. Он напоминает язык xml. Тэги описывают элементы, размещаемые в окне.

Первоначально разместим в окне панель (DockPanel), а на ней элементы меню. Текст XAML такой:

```
<Window x:Class="my1wpf.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Window1" Height="300" Width="300">
   <DockPanel Name="dp1" VerticalAlignment="Stretch"
         HorizontalAlignment="Stretch">
      <Menu DockPanel.Dock="Top" Height="Auto">
         <MenuItem Header="_Cut"/>
         <MenuItem Header="_Paste"/>

         <MenuItem Header="_Exit"  />

      </Menu>
   </DockPanel>
</Window>
```

Теперь окно принимает следующий вид:

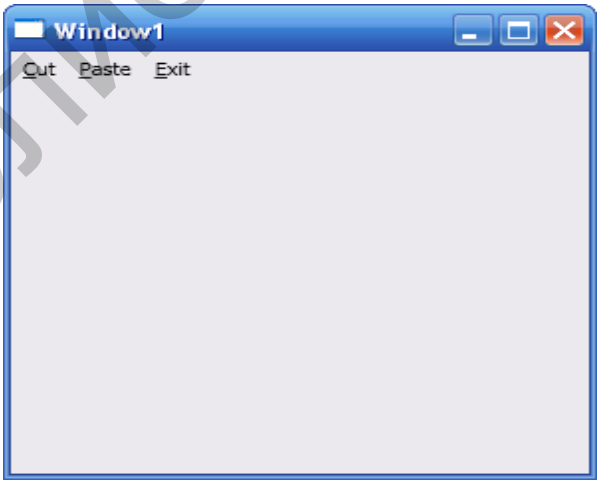

Рис.10.2. Результат работы приложения

Напомним, что запуск программы можно реализовать через меню Debug → Start Debugging.

Далее добавим текстовое поле:

```
<Window x:Class="my1wpf.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Window1" Height="300" Width="300">
   <DockPanel Name="dp1" VerticalAlignment="Stretch"
         HorizontalAlignment="Stretch">
      <Menu DockPanel.Dock="Top" Height="Auto">
         <MenuItem Header="_Cut"/>
         <MenuItem Header="_Paste"/>

         <MenuItem Header="_Exit"  />

      </Menu>
      <TextBox AcceptsReturn="True" SpellCheck.IsEnabled="True"
            HorizontalScrollBarVisibility="Auto"
            VerticalScrollBarVisibility="Auto"
            Name="txt1"

      />
   </DockPanel>
</Window>
```

Теперь можем набирать текст в текстовом поле (рис.10.3).

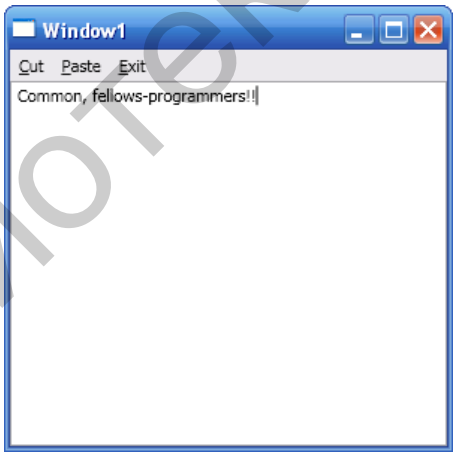

Рис.10.3. Пример работы приложения

Добавим кнопку:

```
<Window x:Class="wpf2.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Window1" Height="300" Width="300">
   <DockPanel Name="dp1" VerticalAlignment="Stretch"
         HorizontalAlignment="Stretch">
      <Menu DockPanel.Dock="Top"  Height="Auto" Name="menu1" Width="Auto">
```

```
        <MenuItem Header="_Cut" Command="Cut"/>
        <MenuItem Header="_Paste" Command="Paste"/>

      </Menu>
      <ToolBarTray DockPanel.Dock="Top">
        <ToolBar>
          <Button Name="btn"  Content="Exit" Background="Coral"  Click="myClick">
          </Button>
        </ToolBar>
      </ToolBarTray>
      <TextBox AcceptsReturn="True"
          SpellCheck.IsEnabled="True"

          HorizontalScrollBarVisibility="Auto"
          VerticalScrollBarVisibility="Auto"
         Name="txt1">
      </TextBox>
    </DockPanel>
</Window>
```

Окно приложения примет такой вид (рис. 10.4)

Рис.10.4. Экранная форма приложения

Необходимо запрограммировать реакцию на элементы меню и кнопку. Реакция может описываться стандартными действиями и нестандартными действиями, предложенными пользователями. Действия для элементов меню задаются в параметрах Command:

```
<MenuItem Header="_Paste" Command="Paste"/>
```

Для кнопки действие задается в параметре Click:

```
<Button Name="btn"  Content="Exit" Background="Coral" Click="myClick">
```

Стандартные действия не программируются. Чтобы запрограммировать событие от кнопки, просто щелкнем на ней дважды в режиме редактирования и введем код на языке C#:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace wpf2
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class Window1 : Window
    {
        public Window1()
        {
            InitializeComponent();
            btn.Click+=new RoutedEventHandler(myClick);
        }
        private void myClick(object sender, RoutedEventArgs e)
        {
            Application.Current.Shutdown();
        }
    }
}
```

Здесь в качестве обработчика события для кнопки закреплен метод myClick. Аналогичным образом поступаем и для меню. В результате получим следующий окончательный вариант:

```
        private void MenuItem_Click(object sender, RoutedEventArgs e)
        {
            if (sender.Equals(mit3))
                Application.Current.Shutdown();
        }
    }
}
```

```
<Window x:Class="wpf2.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Window1" Height="300" Width="300">
   <DockPanel Name="dp1" VerticalAlignment="Stretch"
         HorizontalAlignment="Stretch">
      <Menu DockPanel.Dock="Top"  Height="Auto" Name="menu1" Width="Auto">
         <MenuItem  Name="mit1"  Header="_Cut" Command="Cut"/>
         <MenuItem  Name="mit2" Header="_Paste" Command="Paste"/>
          <MenuItem Name="mit3" Header="_Exit" Click="MenuItem_Click" />
      </Menu>
      <ToolBarTray DockPanel.Dock="Top">
         <ToolBar>
            <Button Name="btn"  Content="Exit" Background="Coral" Click="myClick">

            </Button>
         </ToolBar>

      </ToolBarTray>
      <TextBox AcceptsReturn="True"
            SpellCheck.IsEnabled="True"

            HorizontalScrollBarVisibility="Auto"
            VerticalScrollBarVisibility="Auto"
         Name="txt1">
      </TextBox>
   </DockPanel>
</Window>
```

Можно несколько изменить свойства элементов, например кнопки, сделав следующее ее представление:

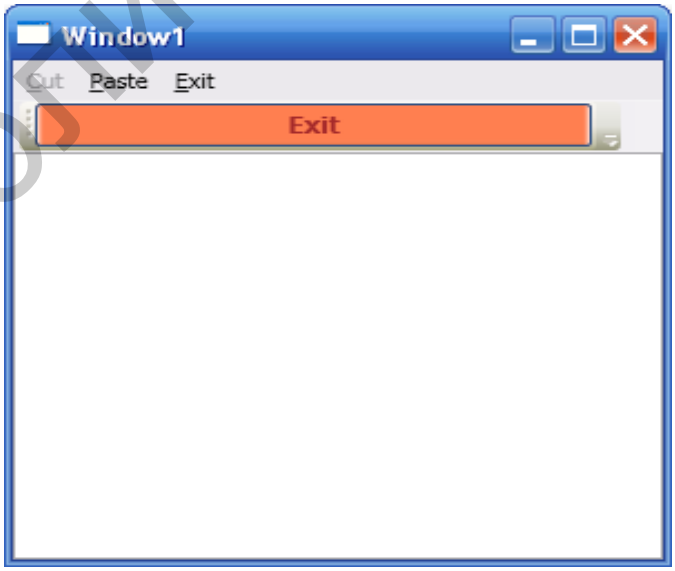

Рис.10.5. Экранная форма приложения

```
<Window x:Class="wpf2.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Window1" Height="300" Width="300">
   <DockPanel Name="dp1" VerticalAlignment="Stretch"
         HorizontalAlignment="Stretch">
      <Menu DockPanel.Dock="Top"  Height="Auto" Name="menu1" Width="Auto">
         <MenuItem  Name="mit1"  Header="_Cut" Command="Cut"/>
         <MenuItem  Name="mit2" Header="_Paste" Command="Paste"/>
          <MenuItem Name="mit3" Header="_Exit" Click="MenuItem_Click" />
      </Menu>
      <ToolBarTray DockPanel.Dock="Top">
         <ToolBar>
            <Border BorderBrush="Blue" Width="250">
            <Button Name="btn"  Content="Exit" Background="Coral"
                  Foreground="Brown"
                  FontSize="12"
                  FontWeight="Bold"
                  Click="myClick">

               </Button>
          </Border>
         </ToolBar>

      </ToolBarTray>

      <TextBox AcceptsReturn="True"
            SpellCheck.IsEnabled="True"

            HorizontalScrollBarVisibility="Auto"
            VerticalScrollBarVisibility="Auto"
          Name="txt1">
      </TextBox>
   </DockPanel>
</Window>
```

Более подробные сведения по работе с WPF и XAML можно найти в [10].

Индивидуальное задание

1. Выполнить все задания в теоретической части
2. Реализовать свой проект WPF, который загружает документ (html) с помощью броузера.

Список использованных источников

1. Герман, О. В. Программирование на JAVA и C# / О. В. Герман, Ю. О. Герман – Спб: «БХВ», 2005 – 510 с.
2. Лабор, В. В. Си Шарп. Создание приложений для Windows / В. В. Лабор – Мн.: Харвест, 2003 – 382 с.
3. Гиббонз, П. Платформа NET для Java программистов / П. Гиббонз – Спб.: Питер, 2001 – 326 с.
4. Прайс, Дж. Visual C#.NET. Полное руководство / Дж. Прайс, М. Гандэрлой – Киев: Век, 2004 – 958 с.
5. Нейгел, К. C# 4 и платформа NET 4.0 для профессионалов / К. Нейгел [и др.] – М.: Вильямс, 2011 –1440 с.
6. Шапошников, И. В. ASP.NET / И. В. Шапошников – Спб.: БХВ, 2002 – 358 с.
7. Кровчик, Э. .NET. Сетевое программирование для профессионалов / Э. Кровчик [и др.] – М.: Лори, 2005, – 417 с.
8. Пауэрс, Л. MicroSoft Visual Studio 2008 / Л. Пауэрс, М. Снелл – Спб.: БХВ, 2009 – 1100 с.
9. Андерсон, Р. Доказательство правильности программ / Р. Андерсон – М.: Мир, 1982 – 168 с.
10. Ватсон, Б. C# 4.0 на примерах / Б. Ватсон – Спб.: БХВ, 2011 – 608 с.

Учебное издание

Герман Олег Витольдович
Заяц Андрей Витальевич

СОВРЕМЕННЫЕ СИСТЕМЫ ПРОГРАММИРОВАНИЯ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Пособие
для студентов специальности
«Автоматизированные системы обработки информации»
дневной и дистанционной форм обучения

Редактор Т. П.Андрейченко
Корректор А. В. Тюхай

Подписано в печать	2012г.	Формат 60x84 1/16	Бумага офсетная
Гарнитура «Таймс»		Отпечатано на ризографе	Усл.печ.л. 5,0
Уч.-изд. л. 5,0		Тираж 100 экз.	Заказ 833

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009
220013, Минск, П.Бровки,6