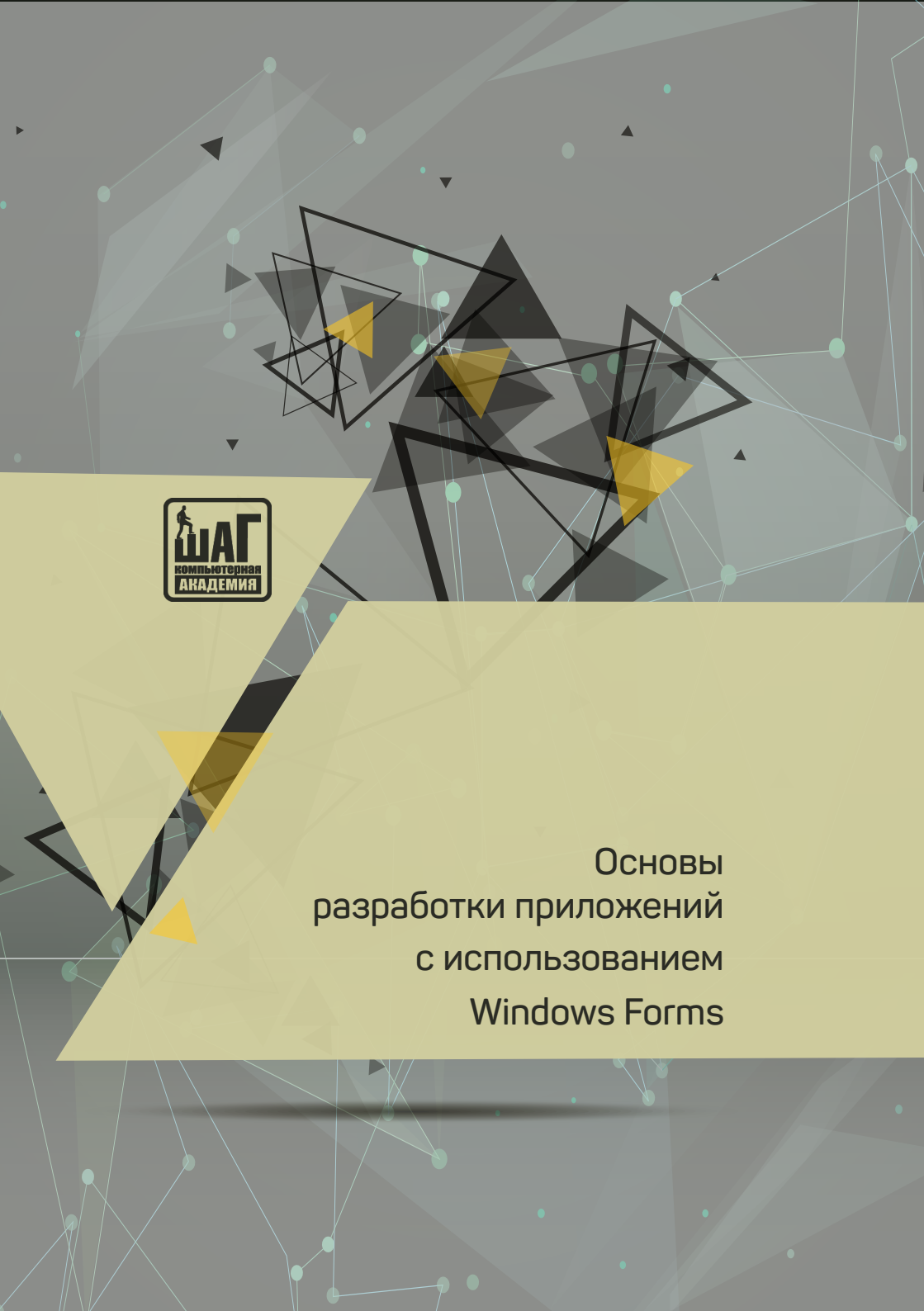

Основы разработки приложений с использованием Windows Forms

Урок №4

Списки, прокрутка, индикаторы

Содержание

1. Использование списков 3
2. Список. Класс ListBox 4
3. Список с селекторами. Класс CheckedListBox 8
4. Комбинированные списки. Класс ComboBox 13
5. Индикатор. Класс ProgressBar 14
6. Полосы прокрутки. Классы VScrollBar, HScrollBar 16
7. Счетчик. Класс NumericUpDown 17
8. Всплывающие подсказки. Класс ToolTip 19
9. Строка состояния. Класс StatusStrip 21
10. Слайдер. Класс TrackBar 23

Домашнее задание 25

Материалы урока прикреплены к данному PDF-файлу. Для доступа к материалам, урок необходимо открыть в программе Adobe Acrobat Reader.

1. Использование списков

В Windows и Web — приложениях широко используются элементы — списки.

Для внесения данных практически в любой форме есть предлагаемый перечень возможных вариантов. Это значительно облегчает введение данных вручную и при этом гарантирует, что выбранный нами вариант или значение будет корректно внесён в базу данных или записан в файл. Это своеобразный заменитель «валидации», где необходимость в непосредственной проверке данных отсутствует. Это удобно и для пользователя, и для разработчика, поскольку и одному, и другому экономит массу времени и создаёт определённый комфорт в работе с приложением.

2. Список. Класс ListBox

Этот элемент управления, как правило, используется для содержания и отображения нескольких элементов списка. Каждый элемент списка — это объект типа ListItem, обладающий набором собственных свойств (Text , Value, Selected).

Данный набор элементов списка является прокручиваемым. Пользователь может выбирать один и более элементов списка. По умолчанию можно выбрать один элемент (SelectionMode.One), но можно изменить эту возможность, выбрав одно из значений: SelectionMode.MultySimple — для выбора разрозненных элементов списка, SelectionMode.MultyExtended — для выбора нескольких элементов списка, расположенных подряд).

Для того чтобы обращаться к элементами списка программно, нужно работать с коллекцией элементов (Items) элемента управления ListBox. Это стандартная коллекция, к элементам которой мы имеем доступ по индексу и к которой мы можем добавлять элементы, удалять их, очищать коллекцию и т.д.

Для добавления элементов в список применяем метод Add():

```
this.listBox1.Items.Add("string");
```

Можно также добавить целый набор элементов:

```
this.listBox1.Items.AddRange(new object[]
{
    "Nastya",
```

```
    "Dima",
    "Sasha",
    "Ira",
    "Dasha",
    "Mitya",
    "Sonya"
});
```

Также добавление элементов можно осуществлять при помощи специального редактора добавления через окно свойства самого элемента (рис. 1).

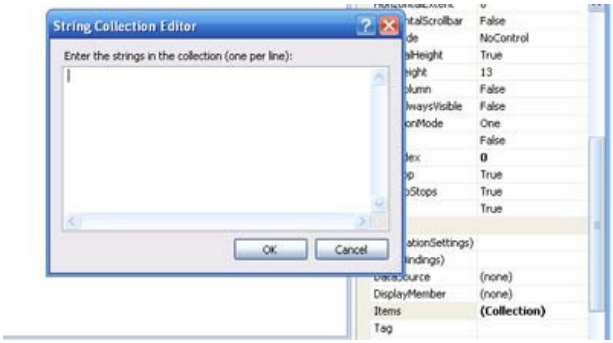

Рисунок 1

Для удаления всех элементов списка, применяем метод Clear():

```
this.listBox1.Items.Clear();
```

Причём у пользователя есть возможность менять настройки его внешнего вида, таких как:

- размер (свойство Size позволяет менять размеры ListBox, устанавливая ширину и высоту в пикселях).
- количество отображаемых колонок (свойство Multy-Column по умолчанию имеет значение false, при

установлении его в true отображает список внутри ListBox, разбитый на колонки).

Можно отсортировать элементы при помощи метода Sort().

События OnSelectedIndexChanged или OnSelectedValueChanged устанавливают для получения информации об изменениях выбранного элемента. Для примера мы рассматриваем приложение, в котором через текстовое поле добавляется по одному элементу в список и по нажатии кнопки всё содержимое списка копируется в другой ListBox.

Также есть возможность удалять по одному выделенные элементы из первого списка. Для удаления мы проверяем, чтобы в списке присутствовал хотя бы один элемент, а затем проверяем на наличие выделенного элемента (рис. 2).

```
private void btnRemoveSelected_Click(object sender,
        EventArgs e)
{
    //список не пуст
    if (this.listBox1.Items.Count != 0)
    {
        //есть выделенные
        if (this.listBox1.SelectedItems != null)
        {
            for (int i = 0; i < this.listBox1.
            SelectedItems.Count; i++)
        {
        this.listBox1.Items.Remove(this.listBox1.
            SelectedItems[i]);
        }
        }
    }
}
```

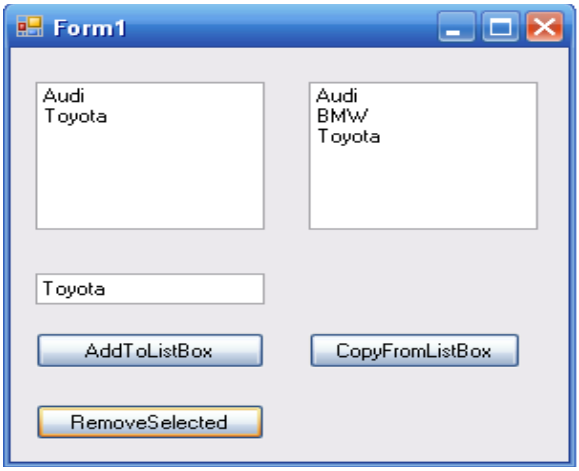
Form1
Audi
Toyota
Audi
BMW
Toyota
Toyota
AddToListBox
CopyFromListBox
RemoveSelected

Рисунок 2

3. Список с селекторами. Класс CheckedListBox

В том случае, когда нам необходимо выбрать из списка несколько элементов, лучше использовать элемент управления CheckedListBox, который фактически является расширенной версией ListBox. Он предназначен для множественного выбора элементов и снабжён чек-боксом для каждого элемента списка (рис. 3).

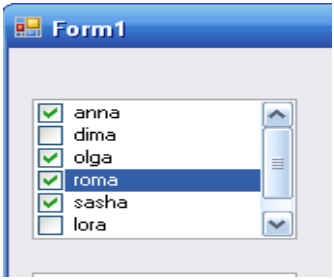

Рисунок 3

По умолчанию флажки расположены слева от элемента списка, при желании можно расположить их справа, поменяв значение свойства RightToLeft (рис. 4).

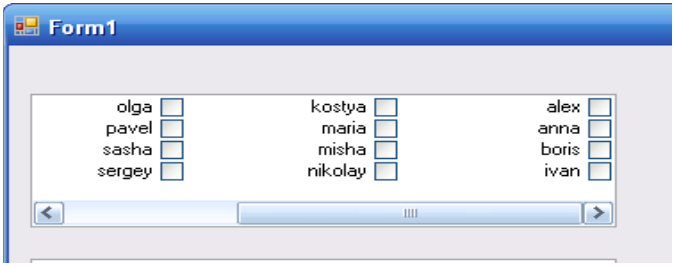

Рисунок 4

В список можно добавлять объекты при помощи методов Add() и AddRange() и удалять с помощью методов Remove() RemoveAt(), очищать список при помощи Clear().

```
this.checkedListBox1.Items.AddRange(new object[] {
    "anna",
    "dima",
    "olga",
    "peter",
    "jane"});
this.checkedListBox1.Items.Insert(3,"dana");
this.checkedListBox1.Items.Remove("dana");
this.checkedListBox1.Items.RemoveAt(1);
this.checkedListBox1.Items.Remove(checkedListBox1.
     SelectedItem);
this.checkedListBox1.Items.Clear();
```

Элементы могут быть отсортированы при помощи метода Sort(). Сортировка применяется в том случае, если нет привязки данных (DataBind).

Определить, какие из элементов в списке отмечены, можно при помощи метода GetItemChecked:

```
int i;
string str;
str = "Checked items:\n";
for (i = 0; i < this.checkedListBox1.Items.Count; i++)
{
   if (this.checkedListBox1.GetItemChecked(i))
   {
      str = str + "Item " + (i + 1).ToString() +
      " = " + this.checkedListBox1.Items[i].ToString() +
      "\n";
    }
}
MessageBox.Show(str);
```

Свойство CheckOnClick позволяет устанавливать флажок при выделении элемента списка. По умолчанию это свойство установлено в false и требуется двойное нажатие на элементе, чтобы отобразился флажок (при первом просто устанавливается выделение данной строки) (рис. 5).

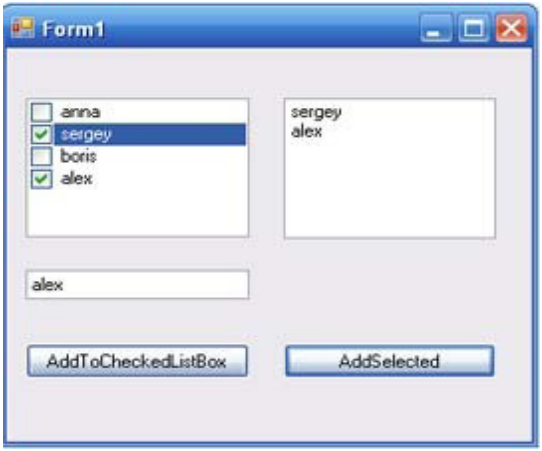

Рисунок 5

В качестве примера рассмотрим приложение, позволяющее добавлять элементы к CheckedListBox через текстовое поле, а затем при необходимости отображать все выделенные в данный момент времени элементы в ListBox.

При нажатии на кнопки нужно добавить необходимые проверки, на ввод пустой строки:

```
if (!String.IsNullOrEmpty(this.textBox1.Text))
```

На уникальность вводимого значения:

```
if (!this.checkedListBox1.Items.Contains(this.
    textBox1.Text))
```

В CheckedListBox мы добавляем по одному элементу из текстового поля:

```
this.checkedListBox1.Items.Add(this.textBox1.Text);
```

Для копирования всех выделенных значений в ListBox мы проходимся в цикле по каждому элементу CheckedListBox и проверяет, выделен ли он флажком:

```
for(int i =0 ;i < this.checkedListBox1.CheckedItems.
    Count;i++)
this.listBox1.Items.Add(this.checkedListBox1.
    CheckedItems[i]);

private void btnAddToCheckBox_Click(object sender,
             EventArgs e)
{
    // проверяет, чтобы строка не была пустой
    if (!String.IsNullOrEmpty(this.textBox1.Text))
    {
        // проверяет, чтобы значения были уникальными
        if (!this.checkedListBox1.Items.
            Contains(this.textBox1.Text))
            this.checkedListBox1.Items.Add(this.
                textBox1.Text);
        else
            MessageBox.Show("CheckedListBox already
                            contains this item");
    }
    else
        MessageBox.Show("Empty string");
}
private void btnAddSelected_Click(object sender,
             EventArgs e)
{
    // сбрасывает значение ListBox, чтобы оно каждый
    // раз перезаписывалось, а не добавлялось
```

```
    this.listBox1.Items.Clear();
    if (this.checkedListBox1.CheckedItems.Count != 0)
    {
        for(int i =0 ;i < this.checkedListBox1.
            CheckedItems.Count;i++)
        this.listBox1.Items.Add(this.checkedListBox1.
            CheckedItems[i]);
    }
    else
    MessageBox.Show("No items in CheckedListBox");
}
```

4. Комбинированные списки. Класс ComboBox

Этот элемент управления является комбинацией списка (ListBox) и текстового окна (TextBox). По умолчанию ComboBox отображается как однострочное текстовое окно со стрелочкой вниз. По щелчку на стрелочке список раскрывается и появляется окно с полным списком элементов. Пользователь может выбирать из списка один элемент.

Поведение ComboBox очень похоже на поведение элемента ListBox. Иногда они могут использоваться взаимозаменяемо. ComboBox предпочтительнее, когда предполагается расширение существующего списка.

В большинстве случаев редактировать текст комбинированного списка не разрешается. За это поведении е отвечает свойство DropDownStyle, имеющее значения Simple (список постоянно открыт), DropDown (список отображается по щелчку), DropDownList (текст не редактируется, список отображается по щелчку).

При установлении свойства Sorted в true строки текста списка отображаются в алфавитном порядке.

5. Индикатор. Класс ProgressBar

Этот элемент управления используется, как правило для индикации состояния длительности процесса. Он как будто сигнализирует о том, что какой-то процесс сейчас происходит и выдаёт информацию о том, на какой примерно стадии выполнения этот процесс находится. Представляет собой окошко, которое может заполняться (слева направо).

Самые основные свойства элемента ProgreessBar — это Minimum, Maximum, Step и Value. Minimum и Maximum соответствуют положению индикатора процесса (начиная от крайнего левого до крайнего правого положения). То есть они представляют собой диапазон, в котором происходят изменения значения свойства Value. Step указывает, на какое значение меняется положение индикатора при каждом вызове метода PerformStep():

```
this.progressBar1.PerformStep();
```

Можно также использовать метод Increment(), передавая ему в качестве параметра значение, на которое меняется состояние индикатора:

```
this.progressBar1.Increment(25);
```

В нашем примере представлен элемент ProgressBar и элемент Label, которые начинают динамически менять своё состояние по нажатию кнопки. На Label отображается значение свойства Value элемента ProgressBar.

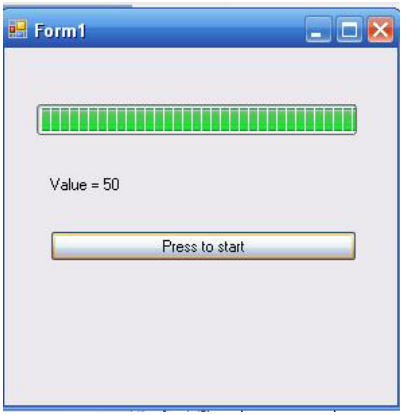

Рисунок 6

В обработчике события нажатия кнопки мы устанавливаем минимальное и максимальное начения ProgressBar, шаг и в цикле вызываем метод PerformStep() для выполнения шага. При этом каждый раз происходит обновление формы и задержка на 50 милисекунд, чтобы мы успевали следить за изменениями ProgressBar , а также постоянно меняющимися значениями текста на ярлыке.

```
private void button1_Click(object sender, EventArgs e)
{
    progressBar1.Minimum = 0;
    progressBar1.Maximum = 50;
    progressBar1.Step = 1;

    for (int i = 0; i <= 50; i++)
    {
        progressBar1.PerformStep();
        label1.Text = "Value = " +
            progressBar1.Value.ToString();
        this.Update(); Thread.Sleep(50);
    }
}
```

6. Полосы прокрутки. Классы VScrollBar, HScrollBar

Большинство элементов управления, которые нуждаются в полосе прокрутки, как правило, предоставляют их по умолчанию. Это касается, например, таких элементов, как TextBox, ListBox, ComboBox.

Элементы, производные от ScrollableControl, такие как Form и Panel, тоже отображают полосы прокрутки, если значение их свойства AutoScroll установлено в true. Можно также использовать полосы прокрутки в тех элементах-контейнерах, которые не предоставляют их по умолчанию.

```
VScrollBar vscroll = new VScrollBar();
vscroll.Dock = DockStyle.Bottom;
this.pictureBox1.Controls.Add(vscroll);
```

Или просто перетащим элемент управления VScrollBar на форму и установим его свойство Dock в нужное значение. Тогда при запуске формы размеры скрола меняются динамически, в зависимости от изменения размеров самого окна. Основными свойствами полосы прокрутки являются Minimum, Maximum, SmallChanged, LargeChanged и Value. Программа сама устанавливает начальные значения этих свойств. Value меняется в зависимости от положения бегунка на полосе прокрутки.

7. Счетчик. Класс NumericUpDown

Это элемент управления, внешне немного напоминающий верхнее поле ComboBox. Но он содержит единственное числовое значение, способное увеличиваться или уменьшаться при нажатии соответствующих кнопок со стрелочками с правой стороны поля контрола. Это кнопки UpButton и DownButton, при их нажатии инициируются методы, влияющие на значение счётчика (рис. 7).

Рисунок 7

Числовое значение данного контрола является десятичным, т.е. decimal. Это значит, что мы можем пользоваться не только целочисленными значениями в этом счётчике.

Числовые значения могут отображаться не только в десятичном виде, но и шестнадцатеричном, для этого нужно установить значение свойства Hexadecimal в true (рис. 8).

Рисунок 8

Свойство DecimalPaces позволяет определить, сколько знаков после десятичной точки мы хотим отбражать в значении счётчика.

Рисунок 9

Свойство Minimum и Maximum позволяют установить минимальное и максимальное значения счётчика соответственно. А свойство Increment — отрегулировать шаг изменения счётчика.

```
NumericUpDown updown = new NumericUpDown();
updown.Width = 50;
updown.Minimum = 10;
updown.Maximum = 30;
updown.Increment = 0.5M;
updown.DecimalPlaces = 2;
```

Установить значение, превышающее максимальное, пользователю не удастся. При попытке это сделать прозвучит звуковой сигнал, а значение будет установлено, равное максимально допустимому.

Свойство ThousandsSeparator ставит разделитель после разряда тысяч — это может быть пробел или запятая в зависимости от местных стандартов.

Можно позволить пользователю самостоятельно задавать значение счётчика, но для этого нужно установить значение поля ReadOnly в fasle.

Основным событием счётчика является событие ValueChanged. Оно происходит, если меняется значение счётчика в коде или при помощи нажатия кнопок Up и Down, или если пользователь ввёл новое значение в поле контроля.

8. Всплывающие подсказки. Класс ToolTip

ToolTip сам по себе не является элементом управления, однако работает с ними в тесном контакте. Он представляет собой способ передачи некоторой информации пользователю в качестве так называемой всплывающей подсказки.

Выгядят эти подсказки как небольшие окна, которые отображаются динамически при наведении курсора (возможно, с некоторой задержкой) на те элементы управления, для которых эти подсказки подключены. Текст на этих подсказках — это небольшая вспомогательная информация, содержание которой зависит от того, что в ней изначально заложено разработчиком.

Рисунок 10

Для того, чтобы установить ToolTip для какого-либо элемента управления, необходимо сначала добавить этот элемент на форму: перетянуть курсором или добавить программно:

```
ToolTip tip = new ToolTip();
```

Затем выделить курсором элемент, для которого эта подсказка формируется, и установить для него свойство ToolTip on, добавив в него необходимый текст.

То же самое можно сделать программно:

```
tip.SetToolTip(this.button1, "Information on button");
```

Можно настроить внешний вид всплывающей подсказки, например, используя свойство IsBalloon. По умолчанию оно установлено в false. Если мы меняем его на противоположное, вид подсказки меняется с обычного прямоугольного окошечка на ballon window.

Рисунок 11

Свойство InitialDelay определяет, через какой промежуток времени (в миллисекундах) после наведения курсора на элемент управления появляется подсказка.

Свойство AutoPopDelay определяет, в течение какого промежутка времени в миллисекундах всплывающая подсказка видна, при условии, что курсор остаётся на этом элементе (по умолчанию — 5000).

К всплывающие подсказке можно при необходимости добавить иконку — это свойство ToolTipIcon (None, Info,Warning, Error).

Рисунок 12

9. Строка состояния. Класс StatusStrip

StatusStrip предназначен для динамического отображения информации о состоянии других элементов управления. Он располагается в нижней части формы, по умолчанию слева. Часто используется для отображения времени и даты. Сам по себе элемент не содержит никаких панелей, но позволяет добавлять набор элементов (StatusLabel, ProgressBar, DropDownButton, SplitButton). Для их добавления можно использовать раскрывающееся меню элементов.

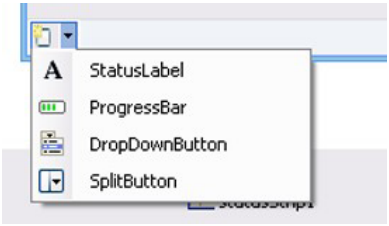

Рисунок 13

Мы добавим в строку состояния 3 основных элемента (2 StatusLabel и 1 DropDownButton). Далее в программе создаётся перечисление с двумя значениями — для отображения времени и для отображения даты.

```
public enum DateTimeFormat { ShowClock, ShowDate };
```

В конструкторе формы мы создаём экземпляр этого перечислимого типа:

```
DateTimeFormat format = DateTimeFormat.ShowClock;
```

Затем добавляем элемент Timer и в обработчике его события Tick делаем динамическое переключение для попеременного отображения времени и даты.

```
private void timer1_Tick(object sender, EventArgs e)
{
    string str;
    str = DateTime.Now.ToShortDateString();
    this.toolStripStatusLabel1.Text = str;
    str = DateTime.Now.DayOfWeek.ToString();
    this.toolStripMenuItem1.Text = str;

    if (format == DateTimeFormat.ShowClock)
    {
        this.toolStripStatusLabel2.Text =
            DateTime.Now.ToShortTimeString();
        format = DateTimeFormat.ShowDate;
    }
    else
    {
        this.toolStripStatusLabel2.Text =
            DateTime.Now.ToShortDateString();
        format = DateTimeFormat.ShowClock;
    }
}
```

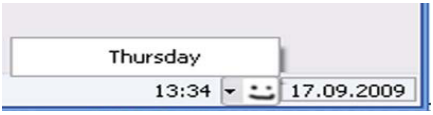

Рисунок 14

10. Слайдер. Класс TrackBar

Этот элемент управления по своей сути очень похож на элементы прокрутки, как ScrollBar, но выглядит иначе и действует немного иначе. По аналонии со ScrollBar,у элемента TrackBar есть свойства Minimum Maximum и Value, где диапазон значений свойства Value определяется значением предыдущих.

Свойство Orientation определяет вариант расположения ползунка (может быть установлено как Horizontal или Vertical).

По умолчанию ползунок выглядит как шкала с делениями, где точка отсчёта слева. Свойство RightToLeft со значением Yes меняет начальное расположение бегунка, и точка отсчёта идёт справа. Расположение шкалы может быть снизу, сверху и по обе стороны от индикатора ползунка, по которому он движется.

Свойство TickFrequency определяет количество делений, котые будут отображены на шкале. Чем больше значение свойства Maximum, тем больше вероятность, что вам не понадобится отображать все деления. В таком случае понадобится установить некий интервал значений, через который будут отображаться деления шкалы. Например, при максимальном значении диапазона, равном 20, а значении TickFrequency, уставновленном в 5, на шкале будет отображаться только 5 делений (включительно с минимальным и максимальным).

Свойство LargeChange устанавливает, на какой диапазон будет сдвинут бегунок при нажатии с левой или с правой стороны от него.

Рисунок 15

В нашем примере на форме присутствует 3 элемента TrackBar, которые управляют изменением 3-х основных цветов (RGB).

```
private void UpdateColor()
{
    Color c = Color.FromArgb(this.trackBar1.Value,
        this.trackBar2.Value, this.trackBar3.Value);
    this.BackColor = c;
}
```

Вычисление цвета происходит в методе:

```
Color.FromArgb();
```

Домашнее задание

1. Написать приложение, которое отображает количество текста, прочитанного из файла с помощью ProgressBar.
2. Написать приложение — анкету, которую предлагается заполнить пользователю, все данные отображаются на результирующем текстовом поле.

Урок №4

Списки, прокрутка, индикаторы