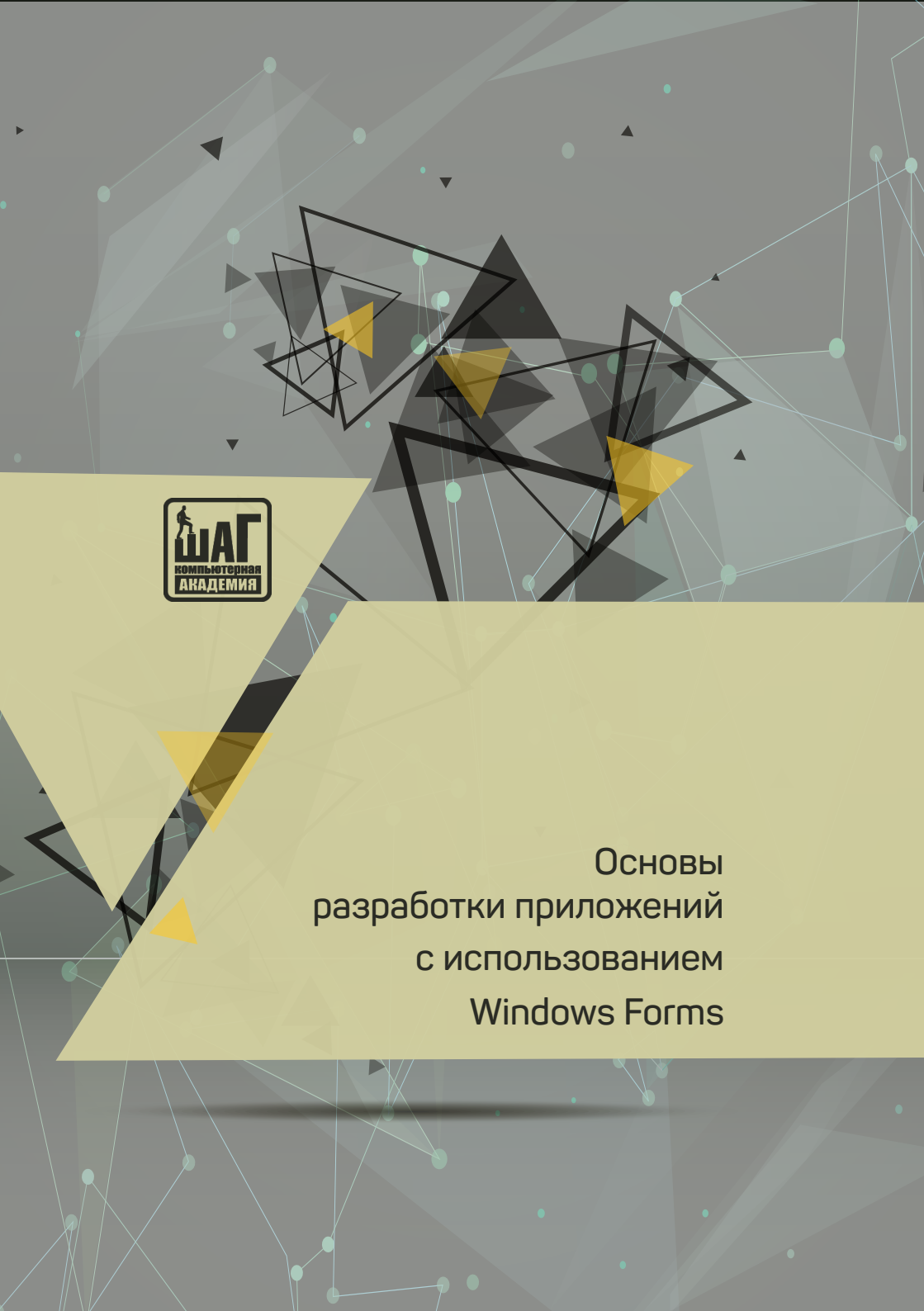

Основы разработки приложений с использованием Windows Forms

Урок №7

Использование расширенного текстового поля, дерева и списка

Содержание

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

1. Расширенное текстовое поле. Класс RichTextBox

Элемент управления RichTextBox используется для отображения, ввода и изменения текста с форматированием. Иерархия наследования класса RichTextBox:

System.Object
    System.MarshalByRefObject
        System.ComponentModel.Component
            System.Windows.Forms.Control
                System.Windows.Forms.TextBoxBase

Элемент управления RichTextBox выполняет те же функции, что и элемент управления TextBox, но помимо этого он позволяет отображать шрифты, цвета и ссылки, загружать текст и вложенные изображения из файлов, а также осуществлять поиск. Для текста элемента управления RichTextBox можно назначить формат символов и абзацев. Элемент управления RichTextBox обычно используется для предоставления возможностей изменения и отображения текста, схожих с возможностями текстовых редакторов, таких как Microsoft Word.

Элемент управления RichTextBox содержит множество свойств, которые можно использовать при применении форматирования к любой части текста в элементе управления. Перед тем как изменить форматирование текста, этот текст необходимо выделить. Только к выделенному тексту можно применить форматирование символов и абзацев. После того как выделенному тексту

был назначен какой-либо параметр, текст, введенный после выделенного, будет форматирован с учетом того же параметра, пока этот параметр не будет изменен, или не будет выделена другая часть документа элемента управления. Свойство SelectionFont позволяет выделять текст полужирным шрифтом или курсивом.Кроме того, с помощью этого свойства можно изменять размер и шрифт текста. Свойство SelectionColor позволяет изменять цвет текста. Для создания маркированных списков следует использовать свойство SelectionBullet. Настройка форматирования абзацев осуществляется также с помощью свойств SelectionIndent, SelectionRightIndent и SelectionHangingIndent.

Элемент управления RichTextBox, как и TextBox, позволяет отображать полосы прокрутки, однако в отличие от TextBox, он по умолчанию отображает и горизонтальную, и вертикальную полосы прокрутки в зависимости от необходимости, а также поддерживает дополнительные параметры их настройки. Как и для элемента управления TextBox, отображаемый текст задается свойством Text. в элементе управления RichTextBox содержится множество свойств для форматирования текста. Для управления файлами используются методы LoadFile и SaveFile:

```
LoadFile(String) /* Загружает файл в формате RTF
или стандартный текстовый файл в кодировке ASCII
в элемент управления RichTextBox.*/
SaveFile(String) /* Сохраняет содержимое элемента
управления RichTextBox в RTF-файл.*/
```

Методы LoadFile и SaveFile отображают и сохраняют множество форматов файлов, в том числе обычный текст,

обычный текст Юникод и форматируемый текст (RTF). с помощью метода Find выполняется поиск текстовых строк или определенных символов:

```
Find(Char[], Int32)
/* с заданной начальной позиции осуществляет поиск
первого экземпляра символа из списка символов по
тексту элемента управления RichTextBox. */

Find(String, RichTextBoxFinds)
/* Осуществляет поиск в элементе управления
RichTextBox текстовой строки с определенными
параметрами, примененными к поиску.*/

Find(Char(), Int32, Int32)
/* Осуществляет поиск первого экземпля ра символа из
списка символов по отрезку текста элемента управления
RichTextBox.*/

Find(String, Int32, RichTextBoxFinds)
/* Осуществляет поиск тексто вой строки
в определенном месте текста элемента управления
RichTextBox с примененными к поиску параметрами.*/

Find(String, Int32, Int32, RichTextBoxFinds)
/* Осуществляет поиск текстовой строки в определенном
отрезке текста элемента управления RichTextBox
с примененными к поиску параметрами.*/
```

Элемент управления RichTextBox можно также использовать для создания веб-ссылок; для этого надо задать для свойства DetectUrls значение true и создать код для обработки события LinkClicked. Можно запретить пользователю управлять частью текста или всем текстом в элементе управления, задав для свойства SelectionProtected значение true.

Большую часть операций редактирования в элементе управления RichTextBox можно отменить и восстановить с помощью вызова методов Undo и Redo. Метод CanRedo позволяет определить, можно ли заново применить выполненное последним и отмененное действие, к элементу управления.

Как известно, слово Rich можно перевести с английского языка как богатый. в контексте этого раздела нашего урока это означает богатство возможностей форматирования текста по сравнению с обычным элементом управления TextBox. с полным списком этих возможностей можно ознакомиться в MSDN.

Рассмотрим работу элемента управления RichTextBox на примере приложения SimplyNotepadCSharp (*исходный код приложения находится в архиве прикрепленном к PDF-файлу данного урока, папка SOURSE/SimplyNotepadCSharp*). Приложение SimplyNotepadCSharp напоминает приложение Windows notepad и позволяетформатировать текст. Общий вид приложения приведен ниже (рис. 1).

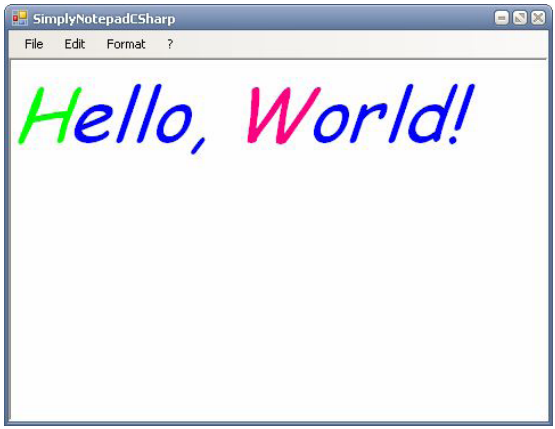

Рисунок 1

Создаем проект и добавляем элементы управления, как показано на рисунке ниже. Пункты главного меню имеет такие подменю (рис. 2).

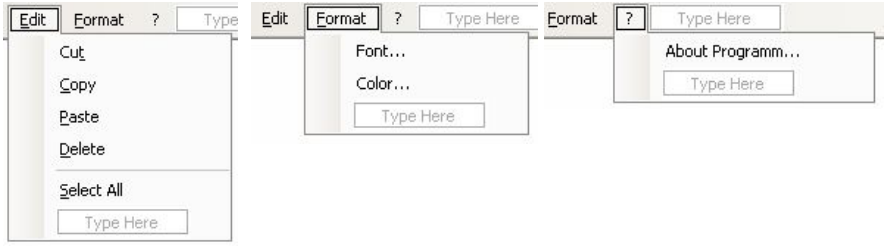

Рисунок 2

Для работы с файловыми потоками в коде главной формы подключаем пространство имен System.IO:

```
using System.IO;
```

1.1. OpenFileDialog

Добавьте на форму элемент управления OpenFileDialog из окна панели инструментов ToolBox. Подобно элементу MainMenu, он будет располагаться на панели невидимых компонент. Свойство FileName задает название файла, которое будет находиться в поле «Имя файла:» при появлении диалога. Название в этом поле — «Текстовые файлы». Свойство Filter задает ограничение файлов, которые могут быть выбраны для открытия — в окне будут показываться только файлы с заданным расширением. Через вертикальную разделительную линию можно задать смену типа расширения, отображаемого в выпадающем списке «Тип файлов». Здесь введено Text Files (*.txt)|*.txt|All Files(*.*)|*.* что означает обзор либо текстовых файлов, либо всех.

Добавим обработчик пункта меню Open формы

```
private void openToolStripMenuItem_Click(object sender,
        EventArgs e)
{
    // Можно программно задавать доступные для обзора
    // расширения файлов.
    openFileDialog1.Filter = "Text Files (*.txt)|*.
                             txt|All Files(*.*)|*.*";

    //Если выбран диалог открытия файла,
    //выполняем условие
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        //Если файл не выбран, возвращаемся
        //(появится встроенное предупреждение)
        if (openFileDialog1.FileName == "")
        {
            return;
        }
        else
        {
            // Создаем новый объект StreamReader и
            // передаем ему переменную OpenFileName
            StreamReader sr = new StreamReader
                        (openFileDialog1.FileName);

            // Читаем весь файл и записываем его
            // в richTextBox1
            richTextBox1.Text = sr.ReadToEnd();

            // Закрываем поток
            sr.Close();

            // Переменной DocName присваиваем
            // адресную строку.
            DocName = openFileDialog1.FileName;
         }
     }
}
```

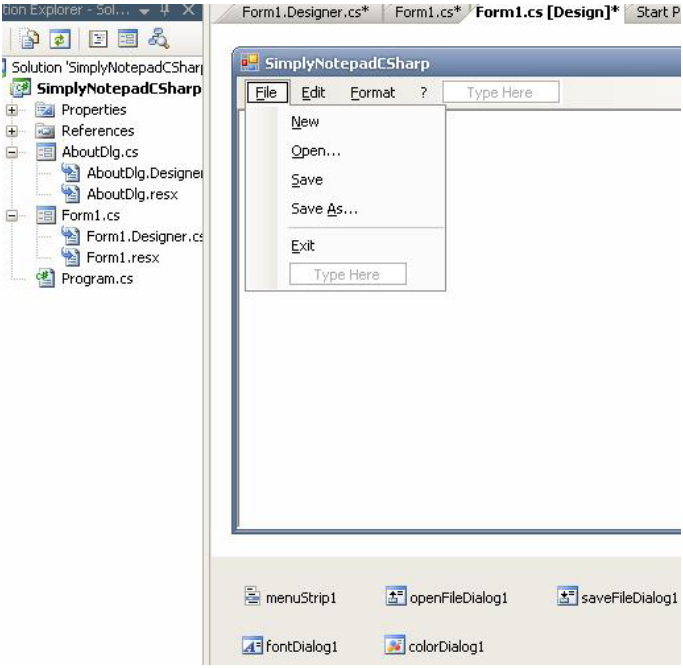

Рисунок 3

1.2. SaveFileDialog

Для сохранения файлов добавляем на форму элемент управления saveFileDialog1. Свойства этого диалога в точности такие же, как у OpenFileDialog.

Добавляем обработчик пункта меню Save формы:

```
private void saveToolStripMenuItem_Click(object
             sender, EventArgs e)
{
    // Если файл не выбран, возвращаемся
    // (появится встроенное предупреждение)
    if (DocName == "")
    {
       return;
```

```
        }
        else
        {
            // Создаем новый объект StreamWriter и
            // передаем ему переменную OpenFileName
            StreamWriter sw = new StreamWriter(DocName);
            //Содержимое richTextBox1 записываем в файл
            sw.WriteLine(richTextBox1.Text);
            // Закрываем поток
            sw.Close();
        }
}
```

Запускаем приложение. Теперь файлы можно открывать, редактировать и сохранять.

Запускаем приложение и открываем текстовый файл, сохраненный в формате блокнота (рис. 4).

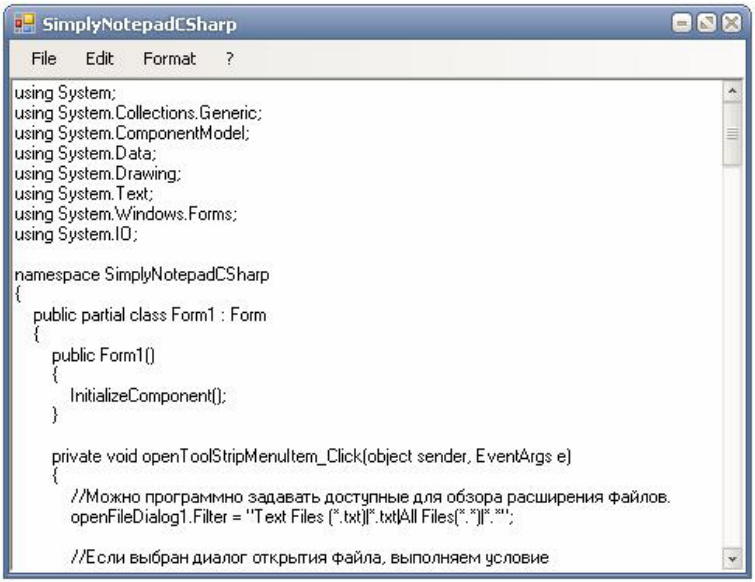

Рисунок 4

1.3. FontDialog

Добавим теперь возможность выбирать шрифт, его размер и начертание. в режиме дизайна перетащим на форму из окна ToolBox элемент управления FontDialog. Не изменяя ничего в свойствах этого элемента, переходим в обработчик пункта Font главного меню:

```
private void fontToolStripMenuItem_Click(object
             sender, EventArgs e)
{
    fontDialog1.ShowColor = true;
    // Связываем свойства SelectionFont и SelectionColor
    // элемента RichTextBox с соответствующими
    // свойствами диалога
    fontDialog1.Font = richTextBox1.SelectionFont;
    fontDialog1.Color = richTextBox1.SelectionColor;

    // Если выбран диалог открытия файла, выполняем
    // условие
    if (fontDialog1.ShowDialog() == DialogResult.OK)
    {
       richTextBox1.SelectionFont = fontDialog1.Font;
       richTextBox1.SelectionColor = fontDialog1.Color;
    }
}
```

1.4. ColorDialog

Диалоговое окно FontDialog содержит список цветов, которые могут быть применены к тексту, но предлагаемый список ограничен. Из окна ToolBox добавляем элемент управления ColorDialog и, вновь не изменяя его свойств, переходим к обработчику пункта Color главного меню формы:

```
private void colorToolStripMenuItem_Click(object
             sender, EventArgs e)
{
   colorDialog1.Color = richTextBox1.SelectionColor;
   if (colorDialog1.ShowDialog() == DialogResult.OK)
   {
      richTextBox1.SelectionColor = colorDialog1.Color;
   }
}
```

Остальные пункты меню не должны вызвать у вас затруднений. Изучите их обработчики самостоятельно.

2. Дерево просмотра. Класс TreeView.

Иерархия наследование для TreeView:

System.Object
    System.MarshalByRefObject
        System.ComponentModel.Component
            System.Windows.Forms.Control
                System.Windows.Forms.TreeView

Управляющий элемент «дерево просмотра» позволяет просмотреть иерархическую коллекцию объектов, представляющих собой узлы дерева — класс TreeNode (рис. 5):

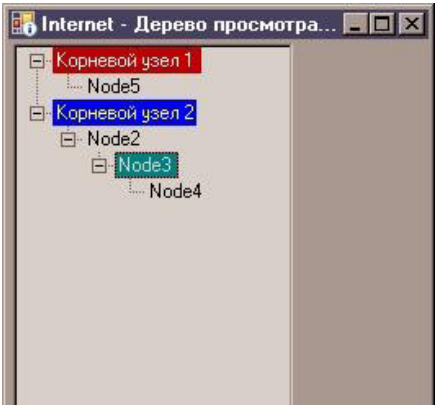

Рисунок 5

Иерархия наследование для TreeNode:

System.Object
    System.MarshalByRefObject
        System.Windows.Forms.TreeNode

«Дерево» содержит коллекцию узлов — Nodes. Каждый узел может быть добавлен на самый верхний уровень иерархии (root) или в качестве дочернего(child) к уже имеющимся узлам. Узел, к которому добавлены дочерние узлы, является для них родительским (parent). Каждый узел TreeNode тоже содержит коллекцию дочерних узлов Nodes**.** Именно в нее и добавляются новые дочерние узлы. Узел может отображать текст (свойство Text) или изображение, сопоставленное свойством (рис. 6)

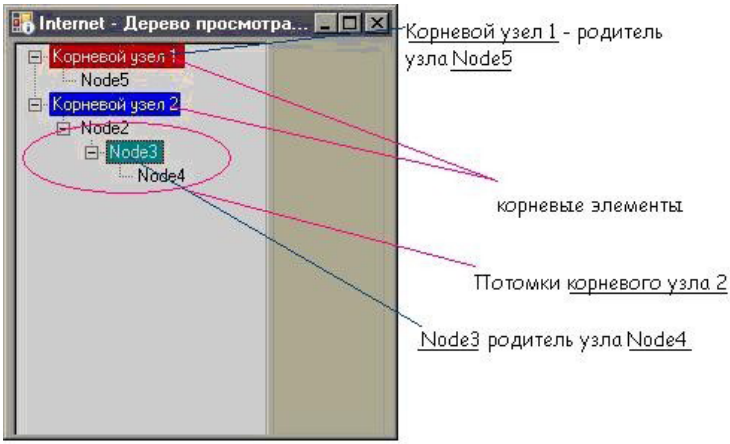

Рисунок 6

ImageIndex с номером элемента списка изображений ImageList.

2.1. Добавление узла к дереву

Продемонстрируем добавление дерева к проекту и узла к дереву:

```
// добавляем управляющий элемент
tv = new TreeView();
```

```
this.Controls.Add(tv);
tv.SetBounds(200, 30, 200, 200);
// создаем узел
TreeNode tn=new TreeNode("Новый узел");

// добавляем к коллекции узлов
tv.Nodes.Add(tn);
```

Вот как выглядит проект теперь (рис. 7).

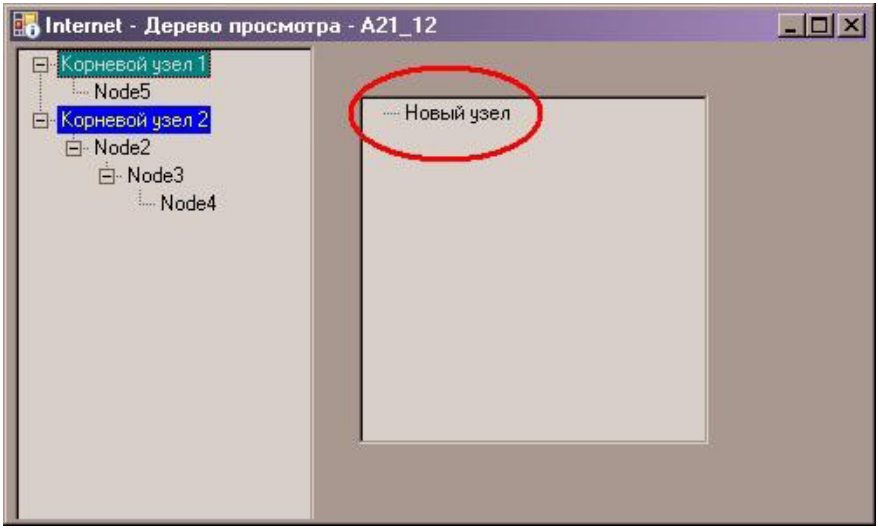

Рисунок 7

Теперь добавим к дереву изображения. Для этого нам потребуется список изображений ImageList , который используют многие управляющие элементы: TreeView, ListView, ToolBar etc. Коллекция ImageList имеет возможность трансформировать составляющие ее изображения, изменяя их размеры (свойство ImageSize) или количество цветов палитры (свойство ColorDepth). Пользуются этой коллекцией для хранения списка изображений и привязки

изображений к узлам управляющих элементов. Загрузим изображения из файлов:

```
try
{
    //создаем и привязываем список изображений
    galery = new ImageList();
    tree.ImageList = galery;
    //увеличиваем размеры изображений
    galery.ImageSize = new Size(65, 100);
    //добавляем изображения к списку
    Bitmap bmp = new Bitmap("bitmap13.bmp");
    galery.Images.Add(bmp);
    bmp = new Bitmap("bitmap14.bmp");
    galery.Images.Add(bmp);
    bmp = new Bitmap("bitmap15.bmp");
    galery.Images.Add(bmp);
    bmp = new Bitmap("bitmap16.bmp");
    galery.Images.Add(bmp);
    //добавляем еще 1узел
    node1 = new TreeNode("Изображение", 1,2);
    tree.Nodes.Add(node1);
    node1.Nodes.Add(new TreeNode("Изображение-2", 3, 2));
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

Изображения получились в виде иконок одного размера (рис. 8), при этом, после связывания ImageList и TreeView все узлы по умолчанию получили первое изображение в качестве иконки. Обратите внимание на искажение цветности (по умолчанию принимается 8-битная палитра, свойство ColorDepth = ColorDepth.Depth8bit). Последний добавленный узел в выбранном состоянии отображает другую иконку (в конструкторе TreeNode указаны

соответственно номера изображений 1 — не выбранный узел, 2 — выбранный узел):

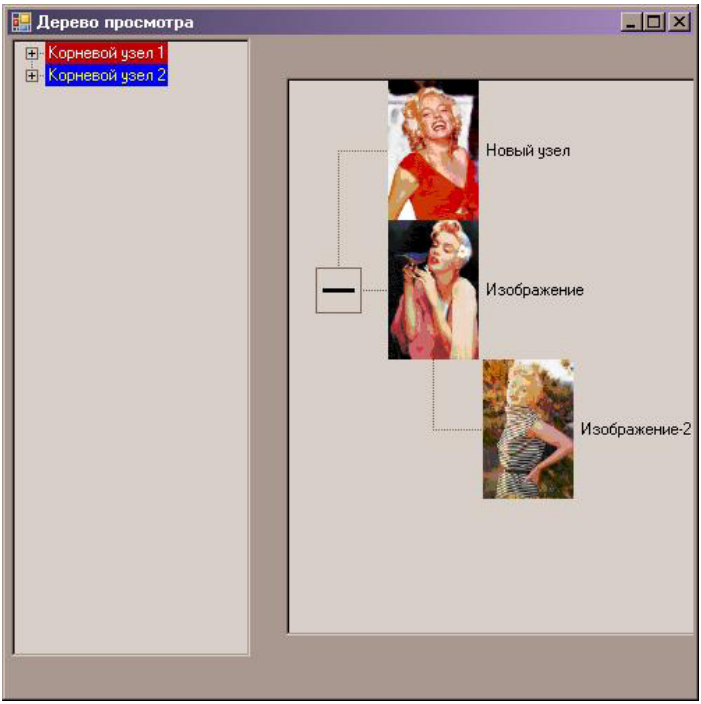

Рисунок 8

При выборе узлов, свертывании и развертывании ветвей «дерево» посылает соответствующие сообщения.

2.2. Удаление узла из дерева

Продемонстрируем удаление узла дерева, обработав событие выбора этого узла. Добавим обработчик события DoubleClick (у меня — в конструкторе формы):

```
// добавляем обработчик двойного щелчка
tree.DoubleClick += new EventHandler(tree_DoubleClick);
```

Создадим метод — обработчик:

```
private void tree_DoubleClick(object sender, EventArgs e)
{
    TreeView tree = (TreeView)sender;
    // удаляем выбранный(SelectedNode) узел из дерева
    tree.Nodes.Remove(tree.SelectedNode);
}
```

Теперь имеется возможность удалять узлы правого дерева двойным щелчком.

2.3. Обход дерева

Иногда возникает необходимость записать содержимое дерева с учетом всей существующей иерархии, например, список полных имен файлов из дерева каталога. Для этого придется совершить путешествие по всем узлам дерева с попыткой заглянуть «внутрь» — рекурсивный обход дерева. Суть такого обхода состоит в том, что каждая ветвь дерева представляет собой дерево в миниатюре — типичная характеристика рекурсивного объекта.

Напишем функцию для рекурсивного обхода дерева:

```
private void recurse_list(TreeNodeCollection nodes,
                          string QName)
{
    foreach (TreeNode i in nodes)
    {
        // добавляем элемент к списку
        lb1.Items.Add(QName+i.Text);
        if (i.Nodes.Count > 0)
        // добавляем все дочерние элементы к списку
        recurse_list(i.Nodes,QName + i.Text + ":");
    }
}
```

Теперь можно вызвать эту функцию и сохранить все дерево в виде квалифицированных имен в обычный список:

```
// добавляем листбокс
lb1 = new ListBox();
lb1.SetBounds(20, 200, 250, 300);
this.Controls.Add(lb1);
// обход дерева
recurse_list(treeView1.Nodes,"");
```

Результат показан на рис. 9. Узлы дерева, находящегося слева расписаны под ним в списке (ListBox) с указанием «полного пути». Обратите внимание на изменившееся качество изображений. Для этого при создании экземпляра коллекции ImageList указана 24-битная палитра:

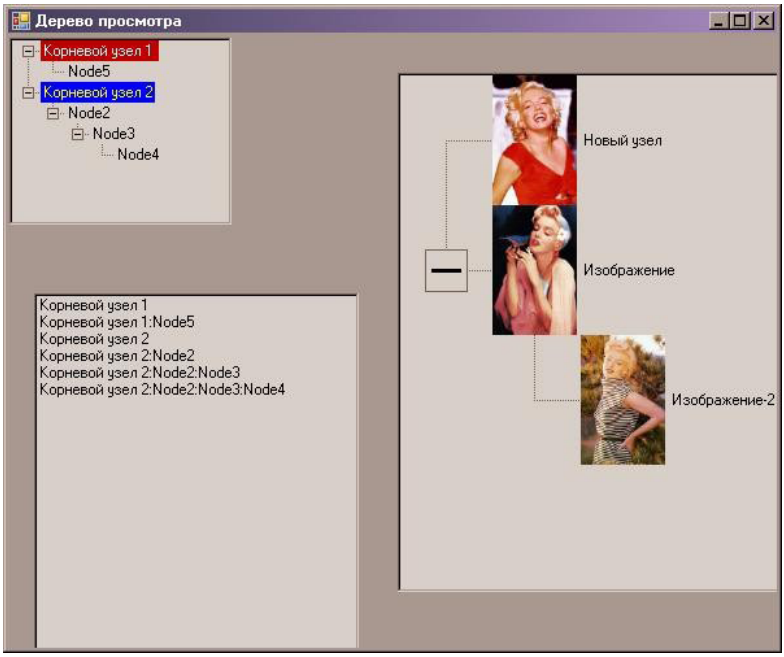

Рисунок 9

```
// создаем и привязываем список изображений
galery = new ImageList();
tree.ImageList = galery;
// увеличиваем размеры изображений
galery.ImageSize = new Size(65, 100);
// изменяем количество бит палитры
galery.ColorDepth = ColorDepth.Depth24Bit;
```

Код приложения находится в архиве прикрепленном к PDF-файлу данного урока, в каталоге Sources/TreeView-Example. В нем продемонстрированы только некоторые составляющие замечательного управляющего элемента «дерево просмотра». Множество других возможностей, такие как самостоятельная прорисовка узлов дерева, масштабирование, прокрутка, программное свертывание и развертывание как всего дерева так и его отдельных ветвей, добавление к элементам разных контекстных меню и всплывающих подсказок Вы сможете разобрать при помощи MSDN.

3. Управляющий элемент «просмотр списка». Класс ListView

Иерархия наследования:

System.Object
  System.MarshalByRefObject
    System.ComponentModel.Component
      System.Windows.Forms.Control
        System.Windows.Forms.ListView

Проще всего объяснить, что такое управляющий элемент «просмотр списка» , если вспомнить правую часть проводника — это как раз и есть просмотр списка. Вот варианты отображения элементов:

А) В виде списка изображений (LargeIcon)

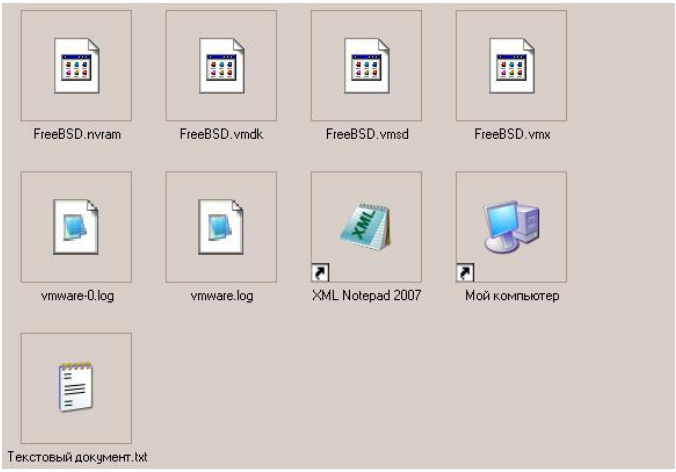

Рисунок 10

Б) В виде списка иконок(SmallIcon)

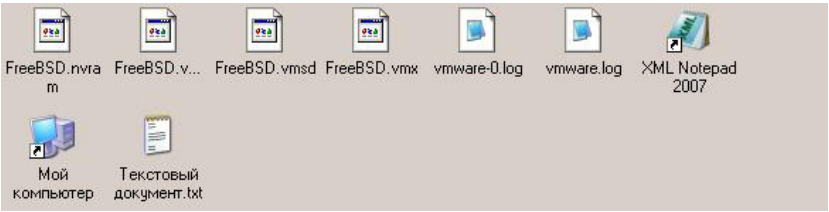

Рисунок 11

В) В виде «плитки» (Tile — появляются дополнительные сведения об элементах)

Рисунок 12

Г) в виде списка элементов(List)

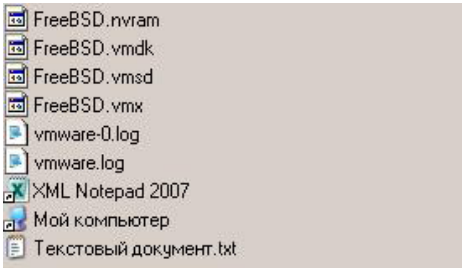

Рисунок 13

Д) в виде таблицы (Details — кроме дополнительных элементов появляются столбцы)

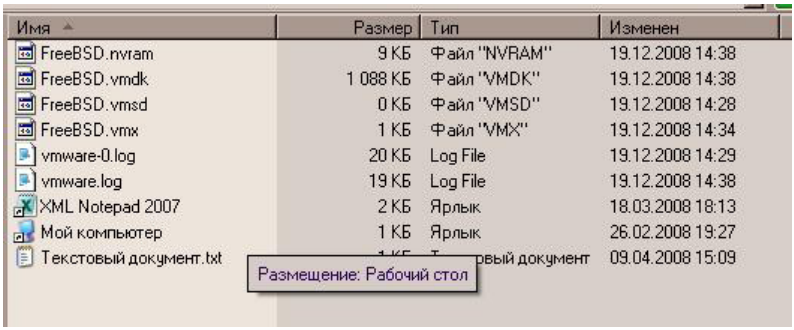

Рисунок 14

Управляется вариант отображения при помощи свойства View объекта. Теперь продемонстрируем создание и заполнение элемента «просмотр списка» из программы. Сначала создаем объект.

```
//создаем экземпляр ListView
table = new ListView();
table.SetBounds(400, 10, 300, 200);
this.Controls.Add(table);
```

3.1. Добавление элементов к ListView

Теперь добавим элементы

```
//добавляем элементы
table.Items.Add(new ListViewItem("Первый"));
table.Items.Add(new ListViewItem("Второй"));
table.Items.Add(new ListViewItem("Третий"));
table.Items.Add(new ListViewItem("Четвертый"));
table.Items.Add(new ListViewItem("Пятый"));
table.View = View.Details; //отображение в виде таблицы
```

Для добавления используется коллекция Items типа ListViewItemCollection, принадлежащая классу ListView. Каждый элемент коллекции ListViewItem содержит коллекцию подэлементов SubItems типа ListViewSubItemCollection. Подэлементы представляют собой подтип ListViewItem.ListViewSubitem, и содержат только дополнительные текстовые описания к элементу. Эти элементы видны при отображении в виде таблицы(View.Details) или в виде плитки(View.Tile).

Чтобы увидеть отображение элементов с подэлементами в виде таблицы необходимо создать нужное количество столбцов — 1 столбец для элемента и по одному для каждого подэлемента. Если столбцов нет, в виде таблицы элементы не отображаются. Добавим пару столбцов к ListView:

```
//Добавляем столбцы
table.Columns.Add("Столбец 1");
table.Columns[0].Width = 100;
table.Columns.Add("Столбец 2");
table.Columns[1].Width = 100;
```

Однако, для отображения информации во втором столбце нужно к элементам ListView добавить подэлементы:

```
//добавляем подэлементы
int k=1;
foreach (ListViewItem i in table.Items)
{
   i.SubItems.Add("подэлемент" + String.Format("{0}",
                  (k++)));
}
```

Внешний вид приложения:

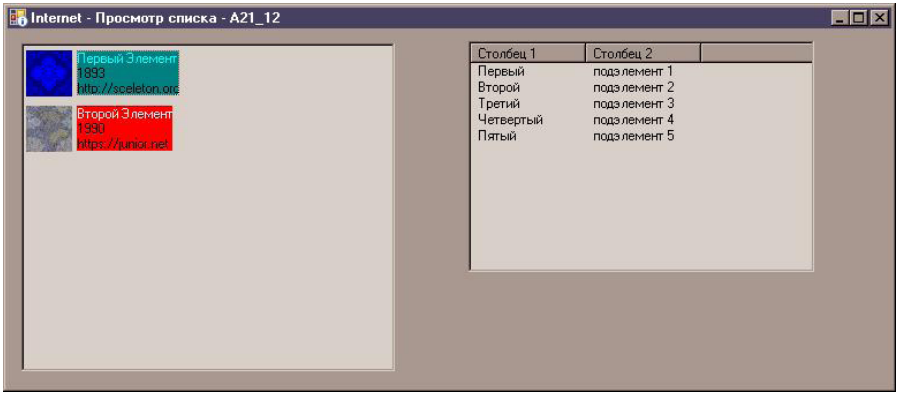

Рисунок 15

Элемент управления ListView получает множество событий: — выбор элемента(ItemSelectionChanged), нажатие на заголовок столбца(ColumnClick), позицию курсора, нажатие клавиш, изменение размеров или положения столбцов и другие. Продемонстрируем удаление выбранного элемента из просмотра списка.

3.2. Удаление элементов списка ListView

```
//добавляем обработчик DoubleClick
table.DoubleClick += new System.EventHandler(this.
                     listView_DoubleClick);
}
private void listView_DoubleClick(object sender, EventArgs e)
{
    ListView table = (ListView)sender;
    //MessageBox.Show(table.SelectedItems[0].Text);
    //удаляем выбранный элемент
    table.Items.Remove(table.SelectedItems[0]);
}
```

Отображение просмотра списка после удаления третьего элемента:

Рисунок 16

Для того, чтобы строку списка можно было выбирать в любом месте а не только в области первого столбца, следует установить свойство FullRowSelect = true. Для отображения сетки устанавливаем свойство GridLines = true.

Для формирования изображений следует использовать коллекцию ImageList и привязать ее к просмотру списка. Элементам ListView, которые должны отображать изображения, назначаются индексы соответствующих элементов из ImageList (свойство ListViewItem.ImageIndex).

3.3. Обход просмотра списка ListView

В заключение давайте выведем элементы списка в ListBox в виде строки с текстом элементов и подэлементов.

Добавим управляющий элемент в список и вызовем метод отображения:

```
// Добавляем listBox
box = new ListBox();

box.SetBounds(300, 300, 300, 200);
this.Controls.Add(box);
viewToBox();
```

Теперь сам метод — просматриваем коллекцию элементов и коллекции подэлементов в каждом:

```
//вывод содержимого ListView в виде ListBox
private void viewToBox()
{
    //просмотр коллекции элементов
    foreach (ListViewItem i in table.Items)
    {
        string text = i.Text;

        //просмотр подэлементов в коллекции элементов
        foreach (ListViewItem.ListViewSubItem si in
                i.SubItems)
        {
            text += ":" + si.Text;
        }

        box.Items.Add(text);
    }
}
```

При отображении обнаруживаем, что первым подэлементом в коллекции подэлементов является сам элемент:

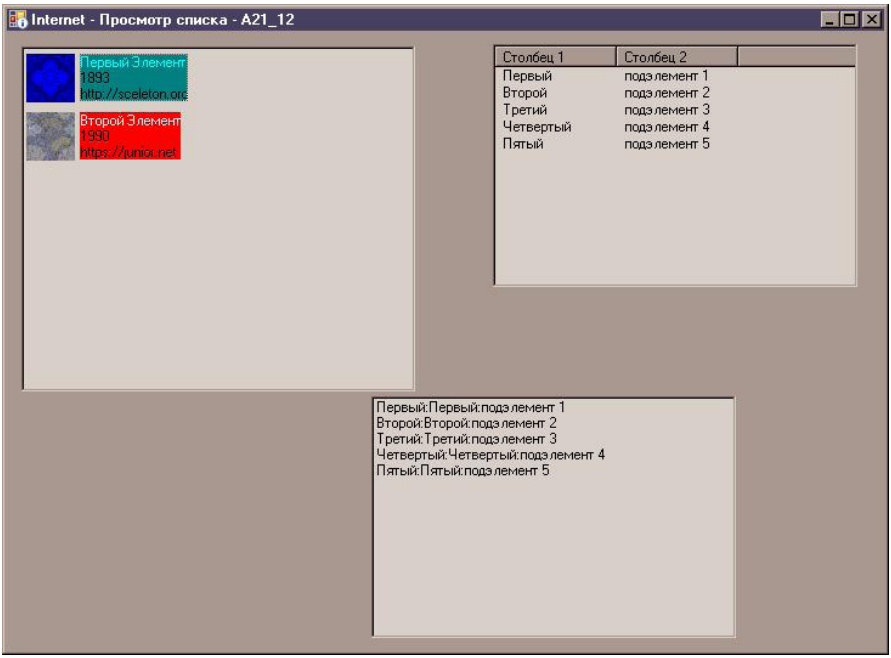

Рисунок 17

Полный текст примера находится в архиве прикрепленном к PDF-файлу данного урока Source/ListView-Example.

3.4. Drag-and-drop (перетаскивание)

Drag-and-drop — перетаскивание мышью объектов из одного приложения (источник перетаскивания — source) в другое (адресат перетаскивания — target) или внутри одного приложения. Приложение, в котором производится Drag-and-drop, полностью отвечает за соответствующие действия, и, если приложение не поддерживает Drag-and-drop, то курсор мыши над ним превращается в перечеркнутый круг. в качестве объектов перетаскивания чаще всего выступают файлы (можно «бросить» файл из

проводника на текстовый редактор, к примеру). Еще мы встречаем Drag-and-drop при использовании Resource Editor в Visual Studio.

Адресат Drag-and-drop должен иметь свойство AllowDrop = true. При попадании курсора с перетаскиваемым объектом на target возникает событие DragEnter, при выходе курсора из области управляющего элемента — DragLeave, при отпускании клавиши мыши — DragDrop. Обработчик событий принимает вторым параметром объект типа DragEventArgs, который содержит свойство Data типа IDataObject, содержащего методы GetFormats и GetDataPresent, которые позволяют определить возможность обработки перетаскиваемых данных программой. При обработке события DragDrop можно получить копию данных методом GetData. Продемонстрируем создание простейшего источника и простейшего адресата:

```
    // разрешаем списку стать адресатом буксировки
    listBox1.AllowDrop = true;
}

private void textBox1_MouseDown(object sender,
        MouseEventArgs e)
{
    // при опускании клавиши мыши выполняем
    // буксировку содержимого источника
    textBox1.DoDragDrop(textBox1.Text,
        DragDropEffects.Copy);
}

private void listBox1_DragEnter(object sender,
        DragEventArgs e)
{
    // при попадании на адресат формируем
    // соответствующую иконку для курсора
```

```
    if (e.Data.GetDataPresent(DataFormats.StringFormat))
        e.Effect = DragDropEffects.Copy;
    else
        e.Effect = DragDropEffects.None;
}

private void listBox1_DragDrop(object sender,
                               DragEventArgs e)
{
    // при отпускании кнопки производим
    // копирование данных вэлемент списка
    listBox1.Items.Add(e.Data.GetData(DataFormats.
            StringFormat).ToString());
}
```

Заполняем редактор:

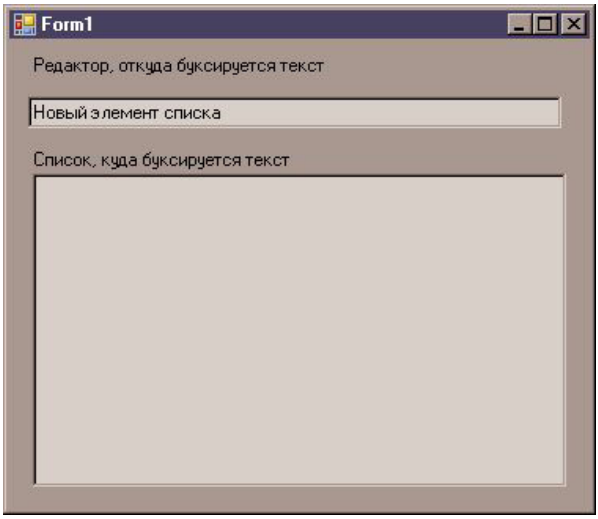

Рисунок 18

Производим Drag-and-drop:

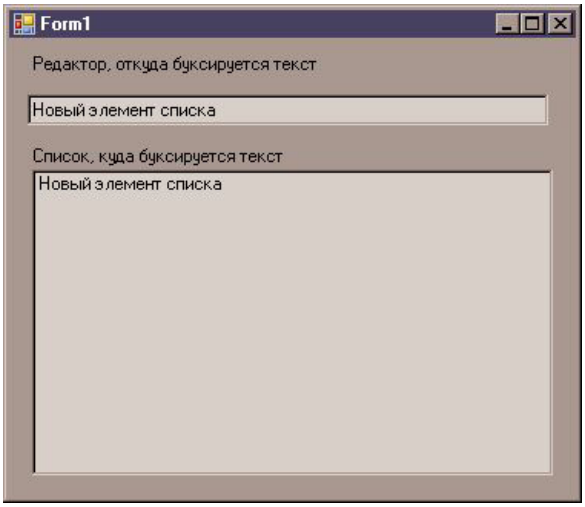

Рисунок 19

В процессе Drag-and-drop отмечаем, что на редакторе курсор принял форму перечеркнутого круга, а над списком — форму курсора с копируемым объектом. Из редактора этого приложения можно перетащить текст в Word или Wordpad, из других приложений перетащить текст в список. Текст примера находится в архиве прикрепленом к PDF-файлу данного урока (Sources\DragNDropExample).

Домашнее задание

1. Создайте приложение на базе TreeView для просмотра каталогов файлов с вложенными каталогами.
2. Создайте приложение на базе ListView для просмотра каталога файлов в 5-ти вариантах, для вариантов иконки-плитка-список_изображений отображать содержимое графических файлов.
3. Создайте редактор на базе RichTextBox и добавьте в него возможность выбора из каталога и просмотра содержимого файла перетаскиванием файла на поле редактора.

Урок №7

Использование расширенного текстового поля, дерева и списка