1. Выпадающие и обычные списки: LISTBOXES

Рис. 1. Окно приложения LISTBOXES

1.1. Создание и использование выпадающих списков

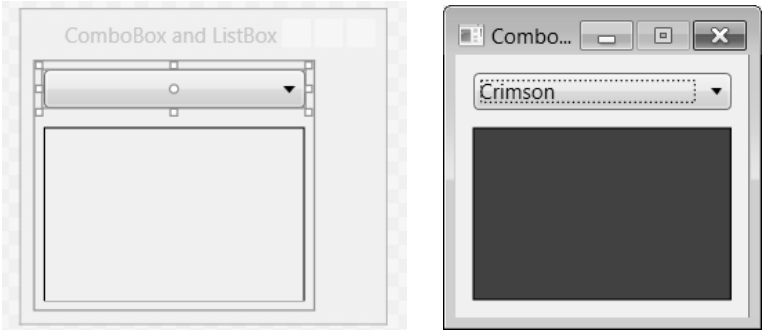

Рис. 2. Макет окна LISTBOXES (первый вариант) и его вид при запуске приложения

```
<Window x:Class="LISTBOXES.MainWindow"
    …
    Title="ComboBox and ListBox" SizeToContent="WidthAndHeight"
    WindowStartupLocation="CenterScreen"
    ResizeMode="CanMinimize">
  <StackPanel Margin="5" Orientation="Horizontal" >
    <StackPanel >
      <ComboBox x:Name="comboBox1" Margin="5" MinWidth="150"
          SelectionChanged="comboBox1_SelectionChanged" />
      <Rectangle x:Name="rect1" Margin="5" Height="100"
          Stroke="Black" />
    </StackPanel>
  </StackPanel>
</Window>
```

В классе MainWindow определите внутренний вспомогательный класс NamedBrush:

```
class NamedBrush
{
    static Dictionary<string, SolidColorBrush> colorNames =
      new Dictionary<string, SolidColorBrush>(141);
    string name;
    static NamedBrush()
    {
        foreach (System.Reflection.PropertyInfo pi in
          typeof(Colors).GetProperties())
            colorNames[pi.Name] = new
              SolidColorBrush((Color)pi.GetValue(null, null));
    }
    NamedBrush(string n)
    {
        name = n;
    }
    public SolidColorBrush Brush
    {
        get { return colorNames[name]; }
    }
    public string Name
    {
        get { return name; }
    }
    public override string ToString()
    {
        return name;
    }
    public static IEnumerable<NamedBrush> AllNamedBrushes()
    {
        return colorNames.Select(e => new NamedBrush(e.Key));
    }
    public static Brush GetBrush(string name)
    {
        return colorNames.ContainsKey(name ?? "") ?
          colorNames[name] : null;
     }
}
```

В конструктор класса MainWindow добавьте операторы:

```
comboBox1.ItemsSource = NamedBrush.AllNamedBrushes();
comboBox1.SelectedValuePath = "Brush";
comboBox1.SelectedIndex = 0;
comboBox1.Focus();
```

Определите обработчик события SelectionChanged для компонента comboBox1, уже указанный в xaml-файле:

```
private void comboBox1_SelectionChanged(object sender,
  SelectionChangedEventArgs e)
{
    rect1.Fill = (Brush)comboBox1.SelectedValue;
}
```

Результат. При запуске программы в выпадающем списке содержатся значения идентификаторов для всех именованных цветов. Идентификаторы указаны в алфавитном порядке. Выбор идентификатора из списка приводит к соответствующему изменению цвета прямоугольника rect1. Для перебора пунктов выпадающего списка можно использовать клавиши со стрелками, а также клавиши Home и End. Кроме того, нажатие буквенной клавиши приводит к выбору первого пункта с текстом, начинающимся с этой буквы. Для разворачивания списка можно использовать комбинацию Alt+↓ (однако данная комбинация срабатывает только при использовании той клавиши со стрелкой, которая *не находится* на цифровом блоке).

Следует заметить, что при выборе начального цвета AliceBlue (первого в списке comboBox1) прямоугольник rect1 остается практически белым, поскольку этот цвет является очень светлым.

1.2. Список: добавление и удаление элементов

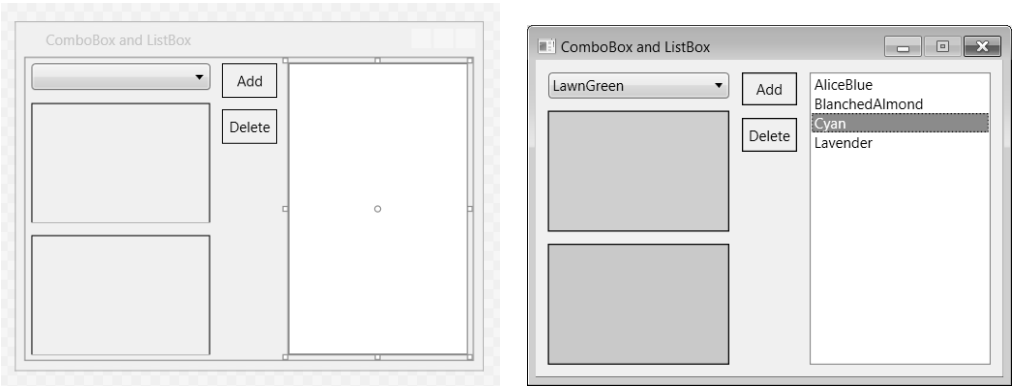

Рис. 3. Макет окна LISTBOXES (второй вариант) и его вид при запуске приложения

```
<Window x:Class="LISTBOXES.MainWindow"
    … >
  <StackPanel Margin="5" Orientation="Horizontal" >
```

```
    <StackPanel >
      <ComboBox x:Name="comboBox1" Margin="5" MinWidth="150"
          SelectionChanged="comboBox1_SelectionChanged" />
      <Rectangle x:Name="rect1" Margin="5" Height="100"
          Stroke="Black" />
      <Rectangle x:Name="rect2" Margin="5" Height="100"
          Stroke="Black" />
    </StackPanel>
    <StackPanel >
      <Label x:Name="label1" Content="Add"
          Padding="5" Margin="5" BorderBrush="Black"
          BorderThickness="1" HorizontalContentAlignment="Center"
          MouseDown="label1_MouseDown" />
      <Label x:Name="label2" Content="Delete"
          Padding="5" Margin="5" BorderBrush="Black"
          BorderThickness="1" HorizontalContentAlignment="Center"
          MouseDown="label2_MouseDown" />
    </StackPanel>
    <ListBox x:Name="listBox1" Margin="5" MinWidth="150"
          SelectionChanged="listBox1_SelectionChanged" />
  </StackPanel>
</Window>
```

Определите обработчики события SelectionChanged для списка listBox1 и событий MouseDown для меток label1 и label2 (уже указанные в xaml-файле):

```
private void listBox1_SelectionChanged(object sender,
  SelectionChangedEventArgs e)
{
    rect2.Fill =
      NamedBrush.GetBrush((string)listBox1.SelectedItem);
}
private void label1_MouseDown(object sender,
  MouseButtonEventArgs e)
{
    listBox1.SelectedIndex = listBox1.Items.Add(comboBox1.Text);
}

private void label2_MouseDown(object sender,
  MouseButtonEventArgs e)
{
    listBox1.Items.RemoveAt(listBox1.SelectedIndex);
```

```
}
```

Результат. При щелчке на метке «Add» в список listBox1 добавляется строка из выпадающего списка comboBox1. Фон прямоугольника rect2 соответствует выделенному элементу списка listBox1. Если список является пустым, то прямоугольник rect2 сохраняет белый фон. При щелчке на метке «Delete» из списка удаляется выделенный элемент.

Комментарий

Для изучения особенностей поведения текущего (обведенного рамкой) элемента списка при выполнении различных операций необходимо, чтобы список *не терял фокуса*. Поэтому выполнение операций мы связали с метками Label: эти компоненты, в отличие от обычных кнопок Button, не могут получать фокус (во всяком случае, таковы их свойства по умолчанию). Таким образом, в окне располагаются лишь два компонента, которые могут получать фокус: это comboBox1 и listBox1. Для переключения между ними достаточно использовать клавишу Tab.

Между прочим, указанная возможность позволила выявить недостаток в реализации компонента ListBox, связанный с взаимодействием выделенного и текущего элементов (напомним, что текущий элемент обводится пунктирной рамкой): при добавлении нового элемента выделение на него переносится, а пунктирная рамка нет (соответствующая ситуация приведена на рис. 49).

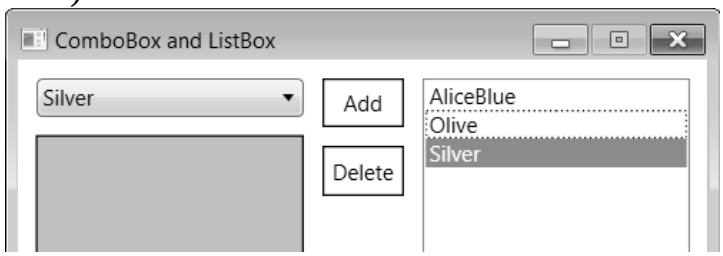

Рис. 4. Пример рассогласования текущего и выделенного элемента списка

Впрочем, подобная ситуация проявляется редко, поскольку для этого необходимо, во-первых, чтобы в момент добавления нового элемента список имел фокус, и, во-вторых, чтобы в нем отображалась пунктирная рамка, что происходит только если ранее для работы с программой (например, для изменения выделенного элемента или переключения между компонентами) использовалась клавиатура. Кроме того, эта ситуация никак не влияет на последующую работу со списком.

Недочет 1. Первая неприятность в нашей программе может возникнуть уже после добавления новых элементов в список listBox1. При использовании многих стандартных тем Windows 7 потеря фокуса списком listBox1 (например, в результате нажатия клавиши Tab или щелчка на выпадающем списке comboBox1) приводит к тому, что выделенный элемент отображается на таком светлом фоне, что на некоторых мониторах оказы-

вается очень сложно отличить его от других, невыделенных элементов (рис. 50, изображение левого окна).

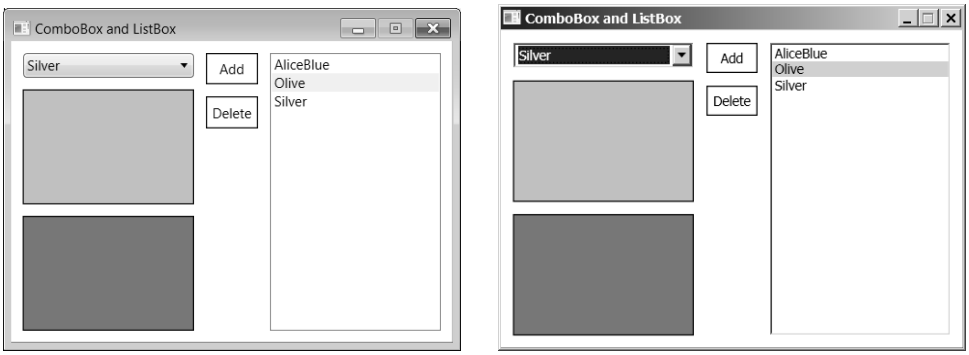

Рис. 5. Вид неактивного списка для различных тем Windows 7

Заметим, что при установке темы «Классическая» ситуация меняется: при потере фокуса списком выделенный элемент отображается на сером фоне более темного, и поэтому вполне различимого, оттенка (рис. 50, изображение правого окна).

Исправление. Добавьте в описание компонента listBox1 в xaml-файле новый атрибут:

```
<ListBox x:Name="listBox1" Margin="5" MinWidth="150"
    Background="LightGray"
    SelectionChanged="listBox1_SelectionChanged" />
```

Результат. Теперь элементы списка отображаются на сером фоне, а при потере фокуса выделенный элемент становится более светлым (рис. 51). Благодаря данному исправлению мы можем легко отличить выделенный элемент списка даже в случае, когда список не имеет фокуса.

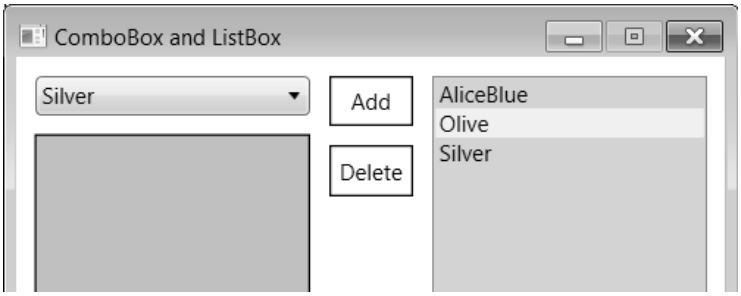

Рис. 6. Вид неактивного списка в случае использования серого фона

Недочет 2. После выполнения операции удаления в списке *исчезает выделенный элемент* (а фон прямоугольника rect2 становится белым). В этой ситуации свойство SelectedIndex равно –1, поэтому повторный щелчок на метке «Delete» приводит к ошибке времени выполнения, так как значение –1 не является допустимым для метода RemoveAt. По этой же причине к ошибке приводит щелчок на метке «Delete» в случае *пустого* списка.

Исправление. Измените метод label2_MouseDown:

```
private void label2_MouseDown(object sender,
  MouseButtonEventArgs e)
{
    int i = listBox1.SelectedIndex;
    if (i == -1)
    {
        Console.Beep();
        return;
    }
    listBox1.Items.RemoveAt(i);
    if (i == listBox1.Items.Count)
        i--;
    listBox1.SelectedIndex = i;
}
```

Результат. При удалении элемента в *середине* списка выделение сохраняется на текущей позиции (которую теперь занимает следующий элемент). При удалении элемента в *конце* списка выделяется предыдущий элемент. Таким образом, *если список не пуст, он всегда имеет выделенный элемент*. При нажатии кнопки «Delete» в случае пустого списка выдается звуковой сигнал.

1.3. Дополнительные операции для элементов списка. Использование стилей в xaml-файле

```
<Window x:Class="LISTBOXES.MainWindow"
    … >
  <StackPanel Margin="5" Orientation="Horizontal" >
    …
    <StackPanel >
      <StackPanel.Resources>
        <Style x:Key="label">
          <Setter Property="Control.Padding" Value="5" />
          <Setter Property="Control.Margin" Value="5" />
          <Setter Property="Control.BorderBrush" Value="Black" />
          <Setter Property="Control.BorderThickness" Value="1" />
          <Setter Property="Control.HorizontalContentAlignment"
              Value="Center" />
        </Style>
      </StackPanel.Resources>
      <Label x:Name="label1" Content="Add"
          Padding="5" Margin="5" BorderBrush="Black"
          BorderThickness="1" HorizontalContentAlignment="Center"
```

```
            MouseDown="label1_MouseDown"
            Style="{StaticResource label}" />
        <Label x:Name="label2" Content="Delete"
            Padding="5" Margin="5" BorderBrush="Black"
            BorderThickness="1" HorizontalContentAlignment="Center"
            MouseDown="label2_MouseDown"
            Style="{StaticResource label}" />
        <Label x:Name="label3" Content="Insert"
            MouseDown="label3_MouseDown"
            Style="{StaticResource label}" />
        <Label x:Name="label4" Content="Move Up"
            MouseDown="label4_MouseDown"
            Style="{StaticResource label}" />
        <Label x:Name="label5" Content="Move Down"
            MouseDown="label5_MouseDown"
            Style="{StaticResource label}" />
        <Label x:Name="label6" Content="Save To File"
            MouseDown="label6_MouseDown"
            Style="{StaticResource label}" />
        <Label x:Name="label7" Content="Load From File"
            MouseDown="label7_MouseDown"
            Style="{StaticResource label}" />
    </StackPanel>
    <ListBox x:Name="listBox1" … />
  </StackPanel>
</Window>
```

В конец списка директив using в файле MainWindow.xaml.cs добавьте директиву

```
using System.IO;
```

Обработчики событий MouseDown для новых меток label3–label7 определите следующим образом:

```
private void label3_MouseDown(object sender,
  MouseButtonEventArgs e)
{
    int i = listBox1.SelectedIndex;
    if (i == -1)
        label1_MouseDown(null, null);
    else
    {
        listBox1.Items.Insert(i, comboBox1.Text);
        listBox1.SelectedIndex = i;
```

```
    }
}
private void label4_MouseDown(object sender,
  MouseButtonEventArgs e)
{
    int i = listBox1.SelectedIndex;
    if (i <= 0)
    {
        Console.Beep();
        return;
    }
    var x = listBox1.Items[i];
    listBox1.Items[i] = listBox1.Items[i - 1];
    listBox1.Items[i - 1] = x;
    listBox1.SelectedIndex = i - 1;
}
private void label5_MouseDown(object sender,
  MouseButtonEventArgs e)
{
    int i = listBox1.SelectedIndex;
    if (i == -1 || i == listBox1.Items.Count - 1)
    {
        Console.Beep();
        return;
    }
    var x = listBox1.Items[i];
    listBox1.Items[i] = listBox1.Items[i + 1];
    listBox1.Items[i + 1] = x;
    listBox1.SelectedIndex = i + 1;
}
private void label6_MouseDown(object sender,
  MouseButtonEventArgs e)
{
    if (listBox1.Items.Count == 0)
    {
        Console.Beep();
        return;
    }
    File.WriteAllLines("LISTBOXES.dat",
      listBox1.Items.Cast<string>());
```

```
}
private void label7_MouseDown(object sender,
  MouseButtonEventArgs e)
{
    if (!File.Exists("LISTBOXES.dat"))
    {
        Console.Beep();
        return;
    }
    listBox1.Items.Clear();
    foreach (var e1 in File.ReadLines("LISTBOXES.dat"))
        listBox1.Items.Add(e1);
    listBox1.SelectedIndex = listBox1.Items.Count - 1;
}
```

Результат. Метка «Insert» обеспечивает вставку нового элемента перед выделенным и выделяет вставленный элемент (в случае пустого списка метка «Insert» действует аналогично метке «Add»). Метки «Move Up» и «Move Down» перемещают выделенный элемент вверх и вниз по списку, сохраняя его выделение (при попытке выполнить перемещение первого элемента вверх или последнего элемента вниз выдается звуковой сигнал). Метки «Save To File» и «Load From File» позволяют сохранить *непустой* список в файле LISTBOXES.dat и впоследствии загрузить список из этого файла (если файл отсутствует, то при попытке загрузить данные выдается звуковой сигнал).

1.4. Выполнение операций над списком с помощью мыши

```
<Window x:Class="LISTBOXES.MainWindow"
    … >
  <StackPanel Margin="5" Orientation="Horizontal" >
    <StackPanel >
      <ComboBox x:Name="comboBox1" Margin="5" MinWidth="150"
          SelectionChanged="comboBox1_SelectionChanged"
          PreviewMouseDown="comboBox1_PreviewMouseDown" />
      …
    </StackPanel>
    <StackPanel >
      …
    </StackPanel>
    <ListBox x:Name="listBox1" Margin="5" MinWidth="150"
        Background="LightGray"
        SelectionChanged="listBox1_SelectionChanged"
```

```
        MouseDoubleClick="label2_MouseDown" AllowDrop="True"
        PreviewMouseDown="listBox1_PreviewMouseDown"
        Drop="listBox1_Drop" />
  </StackPanel>
</Window>
```

Обратите внимание на то, что событие MouseDoubleClick для компонента listBox1 связывается с *уже имеющимся* обработчиком label2_MouseDown (и по этой причине мы не подчеркиваем имя обработчика).

В описание класса MainWindow добавьте новое поле, которое в режиме перетаскивания будет содержать индекс перетаскиваемого элемента списка:

```
int iSrc;
```

Определите добавленные в xaml-файл обработчики событий comboBox1_PreviewMouseDown, listBox1_PreviewMouseDown, listBox1_Drop, MouseDown, а также вспомогательные методы IsMouseOverTarget и IndexFromPoint:

```
static bool IsMouseOverTarget(Visual target, Point point)
{
    var bounds = VisualTreeHelper.GetDescendantBounds(target);
    return bounds.Contains(point);
}
static int IndexFromPoint(ListBox lb,
  Func<IInputElement,Point> getPos)
{
    for (int i = 0; i < lb.Items.Count; i++)
    {
        var lbi = lb.ItemContainerGenerator.ContainerFromIndex(i)
          as ListBoxItem;
        if (lbi == null) continue;
        if (IsMouseOverTarget(lbi, getPos((IInputElement)lbi)))
            return i;
    }
    return -1;
}
private void comboBox1_PreviewMouseDown(object sender,
        MouseButtonEventArgs e)
{
    if (e.ChangedButton == MouseButton.Right)
        DragDrop.DoDragDrop(comboBox1, comboBox1.Text,
          DragDropEffects.Copy);
}
```

```
private void listBox1_PreviewMouseDown(object sender,
        MouseButtonEventArgs e)
{
    if (e.ChangedButton == MouseButton.Right)
    {
        iSrc = IndexFromPoint(listBox1, e.GetPosition);
        if (iSrc != -1)
            DragDrop.DoDragDrop(listBox1,
              (string)listBox1.Items[iSrc],
              DragDropEffects.Move);
    }
}
private void listBox1_Drop(object sender, DragEventArgs e)
{
    string s = e.Data.GetData(typeof(string)) as string;
    int iTrg = IndexFromPoint(listBox1, e.GetPosition);
    if (e.AllowedEffects == DragDropEffects.Move)
        listBox1.Items.RemoveAt(iSrc);
    if (iTrg == -1)
        listBox1.SelectedIndex = listBox1.Items.Add(s);
    else
    {
        listBox1.Items.Insert(iTrg, s);
        listBox1.SelectedIndex = iTrg;
    }
}
```

Результат. При двойном щелчке на элементе списка происходит удаление этого элемента. Для перемещения элемента списка на новую позицию теперь можно использовать механизм Drag & Drop: достаточно «зацепить» любой (не обязательно выделенный) элемент *правой* кнопкой мыши и перетащить его на новое место. Текст из выпадающего списка comboBox1 также можно поместить в список с помощью перетаскивания правой кнопкой мыши. Если элемент перетаскивается на существующий элемент списка, то он *вставляется* в указанную позицию, а если элемент перетаскивается на свободную область списка, то он *добавляется* к списку. В любом случае он становится выделенным элементом.

При перетаскивании текста из выпадающего списка изображение курсора содержит символ «+», что является признаком режима перетаскивания Copy («Копировать»); при перетаскивании элемента списка на новое место курсор *не содержит* символа «+», что является признаком режима перетаскивания Move («Переместить»).

Недочет. В начале перетаскивания элемента из выпадающего списка курсор имеет вид запрещающего знака. В примере ZOO мы уже отмечали, что это нежелательно, так как может ввести в заблуждение пользователя, который решит, что он сделал что-то не так.

Исправление. Добавьте к элементу ComboBox в xaml-файле атрибут AllowDrop со значением true и определите *общий* обработчик для событий DravEnter и DragOver:

```
<ComboBox x:Name="comboBox1" Margin="5" MinWidth="150"
    SelectionChanged="comboBox1_SelectionChanged"
    PreviewMouseDown="comboBox1_PreviewMouseDown"
    AllowDrop="True" DragEnter="comboBox1_DragEnter"
    DragOver="comboBox1_DragEnter" />
```

```
private void comboBox1_DragEnter(object sender, DragEventArgs e)
{
    e.Effects = DragDropEffects.Copy;
    e.Handled = true;
}
```

Результат. Теперь при перетаскивании элемента из выпадающего списка курсор над данным списком является разрешающим (хотя, разумеется, отпускание элемента над выпадающим списком не приведет ни к каким результатам). Если перетаскивание начато из обычного списка listBox1, то над компонентом comboBox1 курсор будет иметь запрещающий вид (это связано с тем, что в обработчике comboBox1_DragEnter для событий DragEnter и DragOver в качестве допустимого режима перетаскивания указан только режим Copy).

2. Просмотр изображений: IMGVIEW

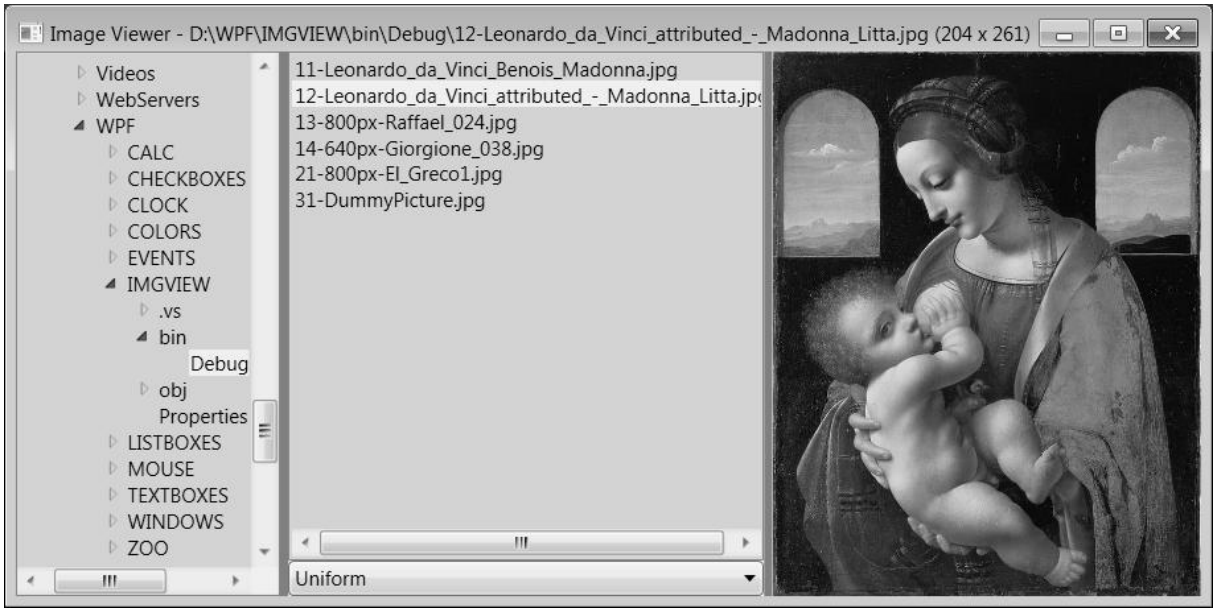

Рис. 7. Окно приложения IMGVIEW

2.1. Иерархический список каталогов

```
<Window x:Class="IMGVIEW.MainWindow"
    …
    Title="Image Viewer" Height="350" Width="700"
    WindowStartupLocation="CenterScreen" >
  <Grid x:Name="grid1" >
      <TreeView x:Name="dirList1" />
  </Grid>
</Window>
```

К списку директив using в начале файле MainWindow.xaml.cs добавьте новую директиву:

```
using System.IO;
```

В классе MainWindow определите два вспомогательных метода:

```
void ExpandItem(TreeViewItem item)
{
    item.Items.Clear();
    if (item.Tag == null)
        foreach (var drive in DriveInfo.GetDrives())
        {
            if (!drive.IsReady)
```

```
                continue;
            TreeViewItem newItem = new TreeViewItem();
            newItem.Tag = drive.RootDirectory;
            newItem.Header = drive.Name;
            if (drive.VolumeLabel != "")
                newItem.Header += " [" + drive.VolumeLabel + "]";
            if (drive.RootDirectory.GetDirectories().Length > 0)
                newItem.Items.Add("*");
            item.Items.Add(newItem);
        }
    else
    {
        try
        {
            foreach (var subDir in (item.Tag as
              DirectoryInfo).GetDirectories())
            {
                try
                {
                    TreeViewItem newItem = new TreeViewItem();
                    newItem.Tag = subDir;
                    newItem.Header = subDir.Name;
                    if (subDir.GetDirectories().Length > 0)
                        newItem.Items.Add("*");
                    item.Items.Add(newItem);
                }
                catch
                { }
            }
        }
        catch
        { }
    }
    item.IsExpanded = true;
}
private void TreeViewItem_Expanded(object sender,
  RoutedEventArgs e)
{
    ExpandItem(e.Source as TreeViewItem);
}
```

В конструктор класса MainWindow добавьте следующие операторы:

```
TreeViewItem item = new TreeViewItem();
dirList1.Tag = item;
item.Tag = null;
item.Header = "Компьютер";
item.Items.Add("*");
dirList1.Items.Add(item);
item.IsSelected = true;
dirList1.Focus();
dirList1.AddHandler(TreeViewItem.ExpandedEvent,
  new RoutedEventHandler(TreeViewItem_Expanded));
```

Результат. При запуске программы в окне отображается начальный пункт «Компьютер» иерархического списка. При его разворачивании появляются все доступные в данный момент диски, а при разворачивании любого диска – все содержащиеся в нем каталоги первого уровня. Если диск имеет метку, то она отображается в квадратных скобках рядом с именем диска.

Если каталог содержит подкаталоги, то его тоже можно развернуть. Рядом с теми элементами списка, которые можно развернуть, отображается специальный *маркер разворачивания*, вид которого зависит от текущей темы Windows: в случае классической темы это квадратик с символом «+», в случае тем Windows 7 – белый треугольник (рис. 58). При разворачивании элемента списка вид маркера изменяется (соответственно на квадратик с символом «–» или на черный наклонный треугольник – см. рисунки, приведенные ниже). Для разворачивания/сворачивания элемента списка достаточно выполнить щелчок мышью на маркере разворачивания или двойной щелчок на имени требуемого элемента.

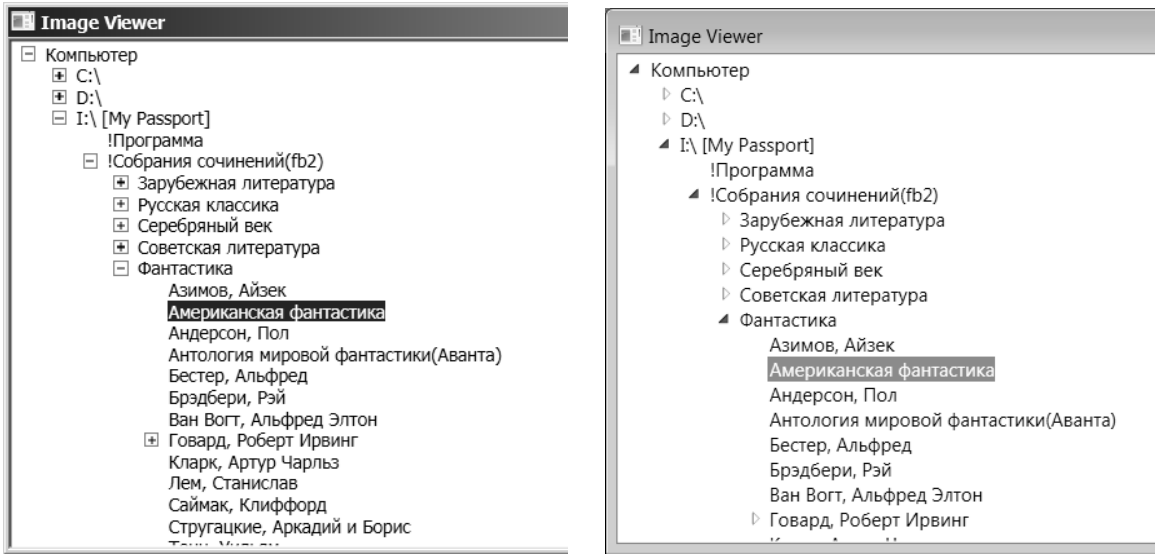

Рис. 8. Вид иерархического списка каталогов для разных тем Windows

Недочет 1. Хотя для компонента TreeView предусмотрена возможность перемещения по элементам с помощью клавиш со стрелками, эта возможность является практически бесполезной, если одновременно не поддерживаются клавиатурные действия по разворачиванию/сворачиванию элементов (а они по умолчанию не поддерживаются).

Исправление. Определите обработчик события PreviewKeyDown для компонента dirList1:

```
<TreeView x:Name="dirList1"
    PreviewKeyDown="dirList1_PreviewKeyDown" />
```

```
private void dirList1_PreviewKeyDown(object sender,
  KeyEventArgs e)
{
    if (e.Key == Key.Space || e.Key == Key.Return)
    {
        var tv = e.Source as TreeViewItem;
        tv.IsExpanded = !tv.IsExpanded;
    }
}
```

Результат. Теперь в программе обеспечена полная поддержка перемещения по списку каталогов с помощью клавиатуры, причем для сворачивания или разворачивания текущего элемента можно использовать либо клавишу Enter, либо клавишу пробела.

Недочет 2. При каждом запуске программы пользователю приходится выполнять одни и те же действия для перехода в нужный каталог.

Исправление. Добавьте в класс MainWindow новый метод:

```
TreeViewItem InitialExpanding(string fullPath)
{
    if (!Directory.Exists(fullPath))
        return null;
    var paths = fullPath.Split('\\');
    paths[0] += "\\";
    TreeViewItem rootItem = dirList1.Items[0] as TreeViewItem;
    ExpandItem(rootItem);
    TreeViewItem item = rootItem;
    foreach (var e in paths)
    {
        item = item.Items.Cast<TreeViewItem>()
          .FirstOrDefault(e1 => (e1.Tag as
            DirectoryInfo).Name.ToUpper() == e.ToUpper());
        if (item == null)
```

```
            return null;
        ExpandItem(item);
    }
    return item;
}
```

И *после* имеющегося в конструкторе класса MainWindow оператора

```
item.IsSelected = true;
```

добавьте новые операторы:

```
item = InitialExpanding(Directory.GetCurrentDirectory());
if (item != null)
    item.IsSelected = true;
```

Результат. Теперь при запуске программы выполняется автоматический переход к *текущему каталогу* (по умолчанию это подкаталог Debug, входящий в каталог, содержащий разрабатываемый проект, рис. 59).

Рис. 9. Окно приложения IMGVIEW с выделенным текущим каталогом

2.2. Список файлов. Компоненты-разделители

```
<Window x:Class="IMGVIEW.MainWindow"
    … >
  <Grid x:Name="grid1">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="160"/>
      <ColumnDefinition Width="Auto"/>
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <TreeView x:Name="dirList1"
        PreviewKeyDown="dirList1_PreviewKeyDown"
        SelectedItemChanged="dirList1_SelectedItemChanged" />
```

```
    <GridSplitter Grid.Column="1" MinWidth ="5"
        HorizontalAlignment="Center" />
    <ListBox x:Name="fileList1" Grid.Column="2" />
  </Grid>
</Window>
```

К списку директив using в начале файла MainWindow.xaml.cs добавьте новую директиву:

```
using IOPath = System.IO.Path;
```

В описание класса MainWindow добавьте новое поле:

```
string[] imageExts = { ".bmp", ".jpeg", ".jpg", ".png", ".gif",
  ".ico", ".wmf", ".emf" };
```

Определите добавленный в xaml-файл обработчик события SelectedItemChanged для компонента dirList1:

```
private void dirList1_SelectedItemChanged(object sender,
  RoutedPropertyChangedEventArgs<object> e)
{
    var dirInfo = (dirList1.SelectedItem as TreeViewItem).Tag as
      DirectoryInfo;
    if (dirInfo == null)
        fileList1.ItemsSource = null;
    else
    {
        try
        {
            var src = dirInfo.GetFiles().Select(e1 => e1.Name)
              .Where(e1 => imageExts.Contains(IOPath
                .GetExtension(e1).ToLower()));
            fileList1.ItemsSource = src;
            if (src.Count() > 0)
                fileList1.SelectedIndex = 0;
        }
        catch
        {
            fileList1.ItemsSource = null;
        }
    }
}
```

В подкаталог Debug, связанный с проектом IMGVIEW, скопируйте несколько графических файлов.

Результат. При выборе какого-либо каталога в иерархическом списке каталогов dirList1 в списке файлов fileList1 отображаются имена *графиче-*

ских файлов (с расширениями bmp, jpeg, jpg, png, gif, ico, wmf, emf), находящихся в выбранном каталоге, причем имя первого файла из списка становится текущим (рис. 60).

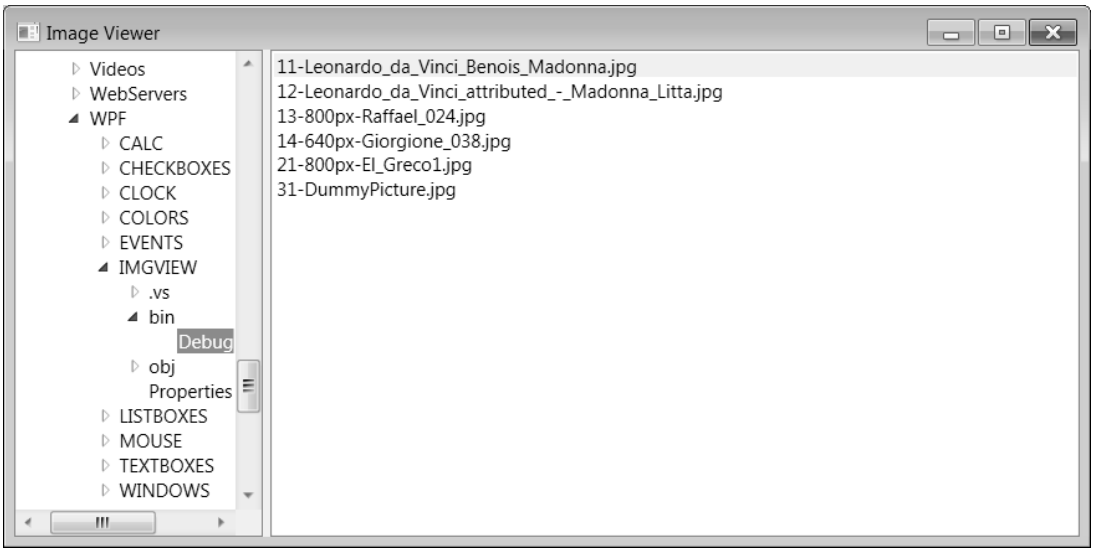

Рис. 10. Окно приложения IMGVIEW со списком файлов

Между списками каталогов и файлов содержится специальный компонент-разделитель GridSplitter, зацепив мышью за который можно изменить ширину одного списка за счет ширины другого (рис. 61).

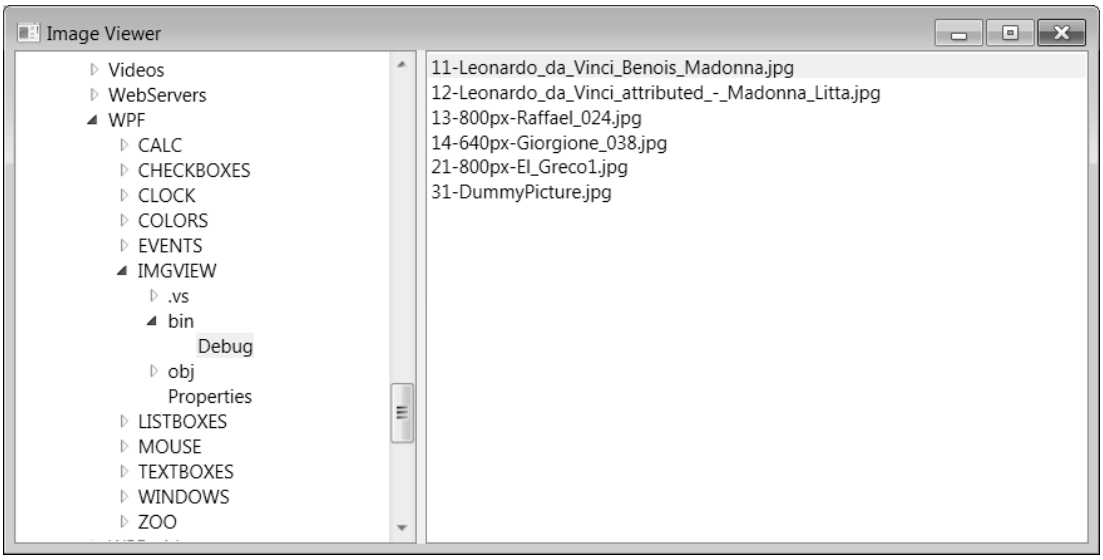

Рис. 11. Окно приложения IMGVIEW после перемещения разделителя

Недочеты. В стандартных темах Window 7 в случае, если список не имеет фокуса, сложно определить, какой из его элементов является текущим (поскольку фон текущего элемента почти не отличается от фона остальных элементов). Кроме того, после перетаскивания разделителя GridSplitter сам разделитель остается текущим компонентом (имеющим фокус), что является неудобным. При использовании клавиши Tab для переключения между компонентами фокус со списка каталогов вначале пе-

реходит на разделитель, а уже после повторного нажатия Tab – на список файлов, что также является неудобным. Наконец, разделитель в окне никак не выделяется, и догадаться о его наличии можно только после наведения на него курсора мыши (курсор при этом изменит вид на горизонтальную двунаправленную стрелку).

Исправления. Все отмеченные недочеты исправляются с помощью дополнительных настроек в xaml-файле:

```
<TreeView x:Name="dirList1"
    PreviewKeyDown="dirList1_PreviewKeyDown"
    SelectedItemChanged="dirList1_SelectedItemChanged"
    Background="LightGray" />
<GridSplitter Grid.Column="1" MinWidth ="5"
    HorizontalAlignment="Center"
    Background="Gray" Focusable="False" />
<ListBox x:Name="fileList1" Grid.Column="2"
    Background="LightGray" />
```

Результат. Теперь оба списка отображаются на сером фоне, на котором выделяется текущий компонент неактивного списка, поскольку он имеет более светлый фон (рис. 62). Разделитель GridSplitter теперь имеет темно-серый фон и, кроме того, не может получать фокус, поэтому клавиша Tab обеспечивает немедленное переключение между списком каталогов и списком файлов.

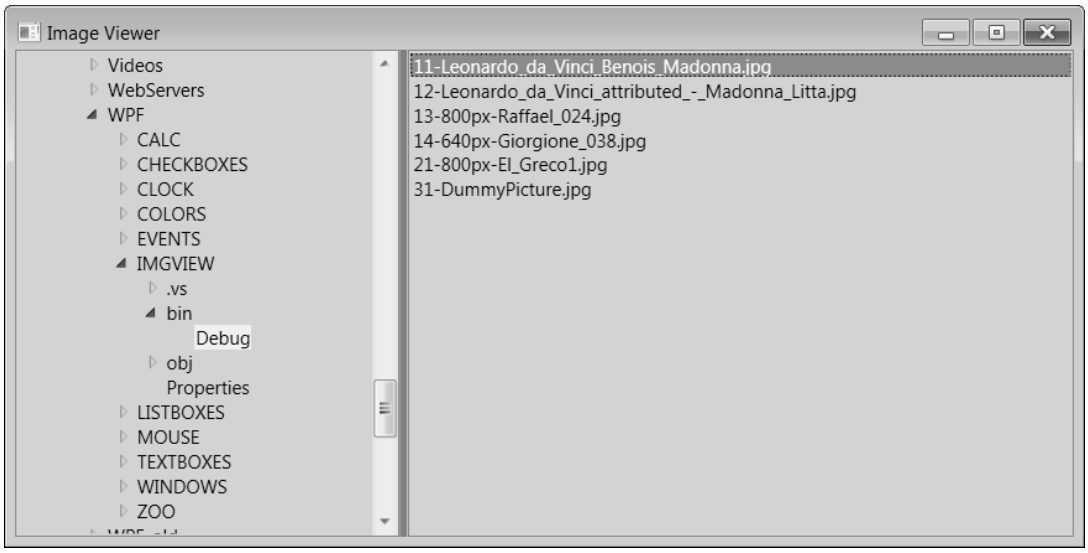

Рис. 12. Окно приложения IMGVIEW со списками каталогов и файлов на сером фоне

2.3. Компоненты для просмотра изображений и прокрутки содержимого

```
<Window x:Class="IMGVIEW.MainWindow"
    … >
  <Grid x:Name="grid1">
```

```
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="160"/>
      <ColumnDefinition Width="Auto"/>
      <ColumnDefinition Width="160"/>
      <ColumnDefinition Width="Auto"/>
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <TreeView x:Name="dirList1" … />
    <GridSplitter Grid.Column="1" … />
    <ListBox x:Name="fileList1" Grid.Column="2"
        Background="LightGray"
        SelectionChanged="fileList1_SelectionChanged" />
    <GridSplitter Grid.Column="3" MinWidth ="5"
        HorizontalAlignment="Center" Background="Gray"
        Focusable="False" />
    <ScrollViewer x:Name="scrollViewer1" Grid.Column="4"
        VerticalScrollBarVisibility="Auto"
        HorizontalScrollBarVisibility="Auto" >
      <Image x:Name="image1" Stretch="None" />
    </ScrollViewer>
  </Grid>
</Window>
```

Определите указанный в xaml-файле обработчик события SelectionChanged для компонента fileList1:

```
private void fileList1_SelectionChanged(object sender,
     SelectionChangedEventArgs e)
{
    string name = (string)fileList1.SelectedValue;
    if (name != null)
    {
        name = ((dirList1.SelectedItem as TreeViewItem).Tag as
          DirectoryInfo).FullName + "\\" + name;
        Title = "Image Viewer - " + name;
        Mouse.OverrideCursor = Cursors.Wait;
        try
        {
            image1.Source = new BitmapImage(new Uri(name));
            Title += " (" + (int)image1.Source.Width + " x " +
              (int)image1.Source.Height + ")";
        }
        catch
```

```
        {
            Title += " (WRONG FORMAT)";
            image1.Source = null;
        }
        Mouse.OverrideCursor = null;
    }
    else
    {
        Title = "Image Viewer";
        image1.Source = null;
    }
}
```

Результат. В правой части окна загружается изображение из текущего графического файла (рис. 63). При этом полное имя файла и размер изображения (в аппаратно-независимых единицах без дробных частей) выводятся в заголовке окна. Если размер изображения больше размера области просмотра, то в области просмотра отображаются полосы прокрутки.

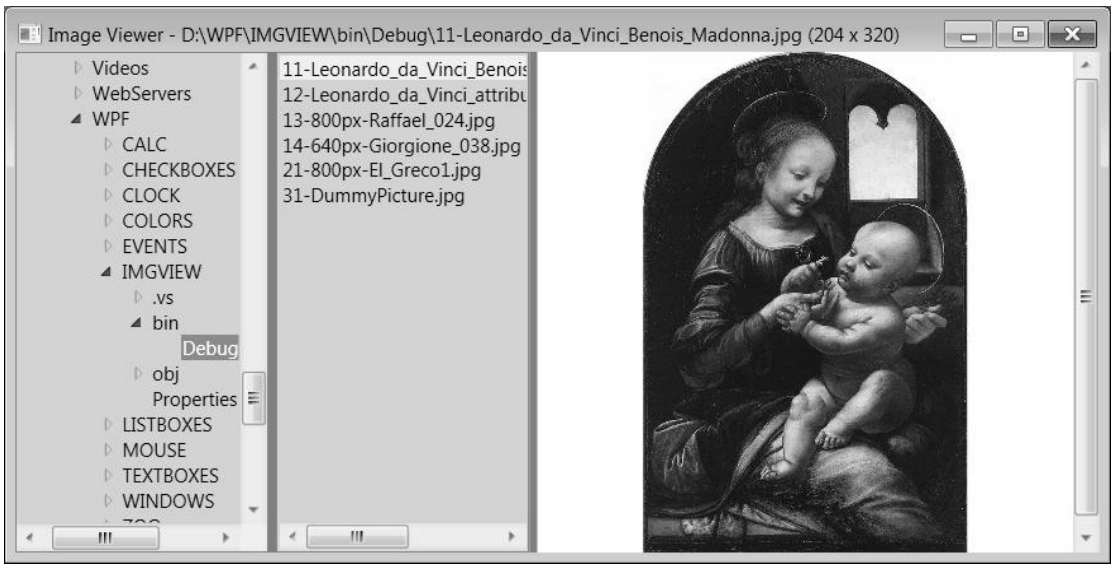

Рис. 13. Окно приложения IMGVIEW с загруженным изображением

Если текущий каталог не содержит графических файлов или если текущий графический файл имеет неверный формат, то область просмотра остается пустой, причем в случае неверного формата соответствующая информация выводится в заголовке окна (для проверки этой возможности в каталог Debug был добавлен пустой файл DummyPicture.jpg, рис. 64).

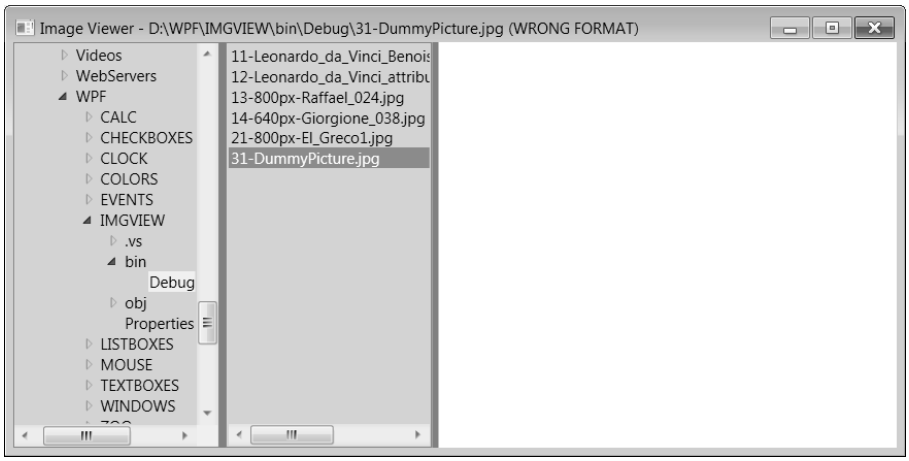

Рис. 14. Вид окна приложения IMGVIEW при выборе ошибочного графического файла

При загрузке изображения вид курсора для всего приложения изменяется на курсор ожидания, после завершения загрузки вид курсора восстанавливается (работа с курсорами подробно обсуждалась ранее в проекте CURSORS).

При перемещении левого разделителя изменяется ширина списка каталогов и области просмотра (ширина списка файлов остается неизменной, однако изменяется его положение). При перемещении правого разделителя изменяется ширина списка файлов и области просмотра (ширина списка каталогов остается неизменной). При изменении ширины окна изменяется только ширина области просмотра изображения.

Недочет. Прокрутку изображения можно выполнять с помощью клавиатуры, однако только после щелчка мышью на изображении. Таким образом, хотя компонент ScrollViewer может принимать фокус, перейти на него клавишей Tab не удается.

Исправление. Добавьте в метод fileList1_SelectionChanged следующие операторы:

```
scrollViewer1.Focusable = image1.Source != null;
scrollViewer1.IsTabStop = image1.Source != null;
```

Результат. Теперь на изображение можно перейти клавишей Tab, после чего использовать клавиши со стрелками (а также Home, End, PgUp и PgDn) для прокрутки большого изображения. Таким образом, нажатие Tab приводит к циклическому перебору списка каталогов, списка файлов и области изображения. В случае пустого списка файлов или пустой области изображения они не могут быть сделаны активными.

2.4. Масштабирование изображений

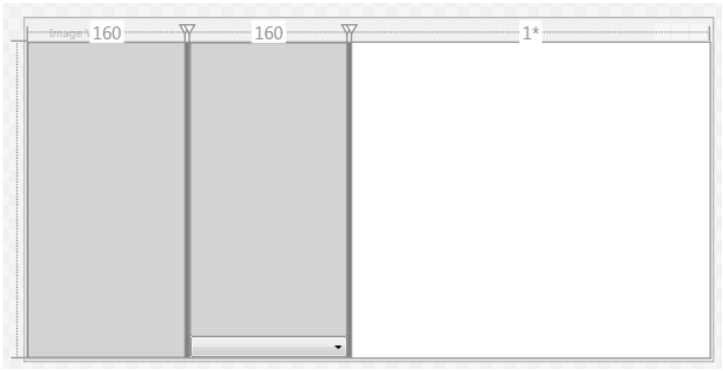

Рис. 15. Макет окна приложения IMGVIEW с добавленным выпадающим списком

```
<Window x:Class="IMGVIEW.MainWindow"
    … >
  <Grid x:Name="grid1">
    …
    <TreeView x:Name="dirList1" … />
    <GridSplitter Grid.Column="1" … />
    <DockPanel Grid.Column="2">
      <ComboBox x:Name="comboBox1" DockPanel.Dock="Bottom"
          SelectionChanged="comboBox1 SelectionChanged" />
      <ListBox x:Name="fileList1" Background="LightGray"
          Grid.Column="2"
          SelectionChanged="fileList1_SelectionChanged" />
    </DockPanel>
    <GridSplitter Grid.Column="3" … />
    <ScrollViewer x:Name="scrollViewer1" Grid.Column="4"
        VerticalScrollBarVisibility="Auto"
        HorizontalScrollBarVisibility="Auto" >
      <Image x:Name="image1" Stretch="None" />
    </ScrollViewer>
  </Grid>
</Window>
```

В конструктор класса MainWindow добавьте операторы:

```
comboBox1.ItemsSource = Enum.GetValues(typeof(Stretch));
comboBox1.SelectedIndex = 0;
```

Определите добавленный в xaml-файл обработчик события SelectionChanged для компонента comboBox1:

```
private void comboBox1_SelectionChanged(object sender,
  SelectionChangedEventArgs e)
{
    image1.Stretch = (Stretch)comboBox1.SelectedValue;
```

```
    if (image1.Stretch == Stretch.None)
    {
        scrollViewer1.HorizontalScrollBarVisibility =
          scrollViewer1.VerticalScrollBarVisibility =
          ScrollBarVisibility.Auto;
        image1.Width = image1.Height = double.NaN;
    }
    else
    {
        scrollViewer1.HorizontalScrollBarVisibility =
          scrollViewer1.VerticalScrollBarVisibility =
          ScrollBarVisibility.Disabled;
        image1.Width = scrollViewer1.ActualWidth;
        image1.Height = scrollViewer1.ActualHeight;
    }
}
```

Результат. С помощью выпадающего списка comboBox1 можно настраивать режим *масштабирования* изображения (определяемый свойством Stretch). Выпадающий список содержит четыре значения – None, Fill, Uniform и UniformToFill, которые соответствуют вариантам значений свойства Stretch компонента Image. При значении None изображение не масштабируется, при Fill оно масштабируется по размеру компонента Image без сохранения пропорций, при Uniform – с сохранением пропорций, а при UniformToFill – с сохранением пропорций, но при полном заполнении компонента (при этом часть изображения – нижняя или правая – будет отсекаться). На рис. 66 приведены варианты представления одного и того же изображения в разных режимах.

Недочет 1. При изменении размеров области просмотра размер отмасштабированного изображения не изменяется.

Исправление. Для реакции на изменение размера области просмотра необходимо определить обработчик события SizeChanged компонента scrollViewer1:

```
<ScrollViewer x:Name="scrollViewer1" Grid.Column="4"
    SizeChanged="scrollViewer1_SizeChanged" >
```

```
private void scrollViewer1_SizeChanged(object sender,
  SizeChangedEventArgs e)
{
    if (image1.Stretch != Stretch.None)
    {
        image1.Width = scrollViewer1.ActualWidth;
        image1.Height = scrollViewer1.ActualHeight;
```

```
    }
}
```

Рис. 16. Варианты режимов представления изображения

После определения данного обработчика можно *заменить* операторы настройки размеров компонента image1 в конце метода comboBox1_SelectionChanged на вызов этого обработчика:

```
image1.Width = scrollViewer1.ActualWidth;
image1.Height = scrollViewer1.ActualHeight;
scrollViewer1_SizeChanged(null, null);
```

Результат. Теперь при любом изменении размеров области просмотра (либо за счет изменения размеров окна, либо в результате перетаскивания одного из разделителей) отмасштабированное изображение подстраивается под новые размеры области просмотра.

Недочет 2. Порядок перехода по нажатию клавиши Tab является не вполне естественным: после списка каталогов происходит переход на выпадающий список comboBox1 (расположенный в нижней части второго столбца), затем – на список файлов fileList1 (расположенный в верхней части этого же столбца) и затем – на область просмотра изображения. Это обусловлено тем, что в xaml-файле при перечислении дочерних компонентов панели DockPanel вначале указан компонент comboBox1, а затем – компонент fileList1.

Заметим, что изменить порядок следования компонентов в xaml-файле нельзя, так как дочерний компонент, занимающий всю оставшуюся область панели DockPanel, должен быть указан последним.

Можно было бы явно задать значения свойства TabIndex для компонентов fileList1 (TabIndex="1") и comboBox1 (TabIndex="2"). При этом потребовалось бы определить это свойство и для компонента scrollViewer1 (TabIndex="3"); если этого не сделать, то нажатие Tab будет переводить фокус со списка каталогов сразу на область просмотра и только потом на компоненты второго столбца.

Однако еще более удобным для пользователя вариантом было бы *исключение* выпадающего списка из последовательности компонентов, которые обходятся по нажатию клавиши Tab, особенно если в программе предусмотрен быстрый способ изменения варианта масштабирования с помощью клавиатуры.

Исправление. Добавьте в xaml-файл два новых атрибута

```
<ComboBox x:Name="comboBox1" DockPanel.Dock="Bottom"
    SelectionChanged="comboBox1_SelectionChanged"
    IsTabStop="False" />
…
<ScrollViewer x:Name="scrollViewer1" Grid.Column="4"
    SizeChanged="scrollViewer1_SizeChanged"
    KeyDown="scrollViewer1_KeyDown" >
```

и определите указанный в xaml-файле обработчик события KeyDown:

```
private void scrollViewer1_KeyDown(object sender, KeyEventArgs e)
```

```
{
    if (e.Key == Key.Space || e.Key == Key.Return)
        comboBox1.SelectedIndex = (comboBox1.SelectedIndex + 1) %
          comboBox1.Items.Count;
    else if (e.Key == Key.Back)
        comboBox1.SelectedIndex = (comboBox1.SelectedIndex +
          comboBox1.Items.Count - 1) % comboBox1.Items.Count;
}
```

Результат. Теперь при нажатии клавиши Tab перебираются только списки каталогов, файлов и область просмотра. Кроме того, при активной области просмотра нажатие клавиши Enter или клавиши пробела обеспечивает перебор вариантов масштабирования в прямом порядке, а нажатие клавиши Backspace – в обратном.

2.5. Сохранение в реестре Windows информации о состоянии программы

```
<Window x:Class="IMGVIEW.MainWindow"
    …
    Closed="Window Closed" >
```

В список директив using файла MainWindow.xaml.cs добавьте директиву

```
using Microsoft.Win32;
```

В описание класса MainWindow добавьте поле

```
string regKeyName = "Software\\WPFExamples\\IMGVIEW";
```

и определите добавленный в xaml-файл обработчик:

```
private void Window_Closed(object sender, EventArgs e)
{
    RegistryKey rk = null;
    try
    {
        rk = Registry.CurrentUser.CreateSubKey(regKeyName);
        if (rk == null)
            return;
        rk.SetValue("Width", (int)ActualWidth);
        rk.SetValue("Height", (int)ActualHeight);
        rk.SetValue("DirList",
          (int)grid1.ColumnDefinitions[0].ActualWidth);
        rk.SetValue("FileList",
          (int)grid1.ColumnDefinitions[2].ActualWidth);
        rk.SetValue("Stretch", comboBox1.SelectedIndex);
        var dirInfo = (dirList1.SelectedItem as TreeViewItem).Tag
```

```
          as DirectoryInfo;
        rk.SetValue("Path", dirInfo == null ? "" :
          dirInfo.FullName);
        rk.SetValue("File", fileList1.SelectedIndex);
    }
    finally
    {
        if (rk != null)
            rk.Close();
    }
}
```

Результат. Теперь при завершении работы программы сведения о ее текущем состоянии записываются в реестр Windows. Для того чтобы в этом убедиться, следует запустить программу regedit (Редактор реестра). Проще всего это сделать, выбрав в главном меню Windows команду «Выполнить…» и указав в появившемся окне имя программы: regedit. В редакторе реестра надо выбрать раздел «HKEY_CURRENT_USER», а в нем подраздел «Software\WPFExamples\IMGVIEW». В результате на правой панели редактора будут отображены поля данного подраздела и их значения. Подраздел должен содержать (помимо поля «По умолчанию», которое не будет использоваться в нашей программе) семь полей в алфавитном порядке: DirList, File, FileList, Height, Path, Stretch, Width (поле Path является строковым, остальные – целочисленными). В следующем пункте к программе будет добавлен фрагмент, позволяющий получать из реестра эти данные.

2.6. Восстановление из реестра Windows информации о состоянии программы

Измените конструктор класса MainWindow следующим образом:

```
public MainWindow()
{
    InitializeComponent();
    TreeViewItem item = new TreeViewItem();
    item.Tag = null;
    item.Header = "Компьютер";
    item.Items.Add("*");
    dirList1.Items.Add(item);
    item.IsSelected = true;
    item = InitialExpanding(Directory.GetCurrentDirectory());
    if (item != null)
        item.IsSelected = true;
```

```
    comboBox1.ItemsSource = Enum.GetValues(typeof(Stretch));
    comboBox1.SelectedIndex = 0;
    string s = "";
    int i = 0;
    RegistryKey rk = null;
    try
    {
        rk = Registry.CurrentUser.OpenSubKey(regKeyName);
        if (rk != null)
        {
            Width = (int)rk.GetValue("Width", (int)Width);
            Height = (int)rk.GetValue("Height", (int)Height);
            grid1.ColumnDefinitions[0].Width =
              new GridLength((int)rk.GetValue("DirList",
              (int)grid1.ColumnDefinitions[0].Width.Value));
            grid1.ColumnDefinitions[2].Width =
              new GridLength((int)rk.GetValue("FileList",
              (int)grid1.ColumnDefinitions[2].Width.Value));
            comboBox1.SelectedIndex = (int)rk.GetValue("Stretch",
              comboBox1.SelectedIndex);
            s = (string)rk.GetValue("Path", "");
            i = (int)rk.GetValue("File", 0);
        }
    }
    finally
    {
        if (rk != null)
            rk.Close();
    }
    if (!Directory.Exists(s))
        s = Directory.GetCurrentDirectory();
    item = InitialExpanding(s);
    if (item != null)
        item.IsSelected = true;
    if (fileList1.Items.Count == 0)
        i = -1;
    else
        if (i >= fileList1.Items.Count || i == -1)
            i = 0;
    fileList1.SelectedIndex = i;
```

```
    dirList1.Focus();
    dirList1.AddHandler(TreeViewItem.ExpandedEvent,
        new RoutedEventHandler(TreeViewItem_Expanded));
    comboBox1.ItemsSource = Enum.GetValues(typeof(Stretch));
    comboBox1.SelectedIndex = 0;
}
```

Результат. При запуске программы данные из реестра Windows используются для восстановления ее состояния. Если данные в реестре отсутствуют, то устанавливаются настройки по умолчанию. Поскольку с момента последнего сохранения данных в реестре состояние дисковой системы могло измениться, при считывании поля Path проверяется наличие указанного в нем каталога, и если он не найден, то используется текущий каталог. Кроме того, прочитанное поле File при необходимости корректируется таким образом, чтобы в случае непустого списка файлов обязательно был выделен какой-либо его элемент.

При отображении на экране окно центрируется с учетом его новых размеров, считанных из реестра.