

## **Лабораторная работа 3. Создание элементов управления**

### **Цель работы**

Изучение способов разработки элементов управления и получение навыков по их настройке и применению в дальнейшей работе.

#### **Упражнение 1. Создание составного элемента управления**

В дополнение к уже существующим элементам управления можно разрабатывать собственные, чтобы обеспечить для своих приложений специализированную функциональность.

Существует три вида разрабатываемых пользователем элементов управления:

- **составные (composite)**, которые создаются при объединении других элементов управления Windows Forms;
- **специализированные (custom)**, создаваемые с нуля и предоставляющие собственный код для прорисовки;
- **расширенные (extended)**, которые добавляют функциональность к уже существующему элементу управления Windows Forms.

Составные элементы управления наследуются от класса **UserControl**. Он предоставляет базовый уровень функциональности, обеспечивающий добавление других элементов управления, а также свойств, методов и событий. Класс **UserControl** имеет собственный конструктор, позволяющий использовать в Visual Studio IDE перетаскивание дополнительных элементов управления из **Toolbox** на поверхность конструктора и настраивать их.

В этом упражнении вы создадите составной элемент управления, действующий как цифровые часы. В него вы добавите элемент управления **Label**, отображающий правильное время, и компонент **Timer**, каждую

секунду обновляющий **Label**. Предоставив свойство **Enabled** элемента управления **Timer**, вы дадите пользователям возможность включать и отключать часы.

### **Разработка составного элемента управления**

1. Создайте в Visual Studio новое приложение Windows Forms. Назовите его **WinTimer1**.

2. В меню **Project** выберите **Add User Control** и щелкните **Add** в диалоговом окне **Add New Item**. Укажите имя **UserControlTimer** и нажмите **Добавить**. К вашему проекту будет добавлен пустой пользовательский элемент управления, который откроется в конструкторе.

3. Из **Toolbox** перетащите **Label** в пользовательский элемент управления.

4. Удалите данные из свойства **Text** ЭУ **Label**.

5. Измените размеры пользовательского элемента управления так, чтобы он был приблизительно равен размеру элемента управления **Label**.

6. Из **Toolbox** перетащите в пользовательский элемент управления компонент **Timer**.

7. В окне **Properties** компонента **Timer** присвойте свойству **Interval** значение **1000** и свойству **Enabled** значение **True**.

8. Дважды щелкните компонент **Timer**, чтобы открыть в окне кода обработчик события **Timer.Tick** по умолчанию и добавьте следующую строку программы:

```
label1.Text = DateTime.Now.ToString("T");
```

9. В окне кода добавьте следующее объявление свойства:

```
public bool TimeEnabled
{
    get { return timer1.Enabled; }
    set { timer1.Enabled = value; }
}
```

10. В меню **File** выберите **Save All**, чтобы сохранить ваше решение.

11. В меню **Build** выберите **Build Solution**.

### **Применение составного элемента управления**

12. Выберите вкладку конструктора **Form1**. Из **Toolbox** перетащите **UserControlTimer** в форму. К форме добавится экземпляр вашего пользовательского элемента управления и начнет отсчитывать каждую секунду. Обратите внимание, что вы можете приостановить это, присвоив свойству **TimeEnabled** значение **False** в окне **Properties**.

13. Постройте и запустите приложение. Обратите внимание, что пользовательский элемент управления во время выполнения действует так же, как в конструкторе.

### **Разработка библиотеки классов элементов управления**

14. Чтобы разработать библиотеку классов элементов управления, выполните следующие действия:

а. Создайте в Visual Studio новое приложение – **Библиотека элементов управления Windows**.

- b. По умолчанию проект содержит составной элемент управления.
- c. Добавьте элементы управления и код, как описано в упражнении 1.
- d. Выберите элемент управления, который не должен изменяться при наследовании классов (в данном случае это и **label1**, и **timer1**), и задайте для свойства **Modifiers** (Модификаторы) этого элемента управления значение **Private**.
- e. Создайте библиотеку DLL, построив проект.

## **Упражнение 2. Создание специализированного элемента управления**

Для специализированных элементов управления не существует пользовательского интерфейса по умолчанию, они должны предоставлять весь код, необходимый для отображения своего графического представления. Элемент управления разрабатывается для того, чтобы получить точно такое визуальное представление, которого вам необходимо, и закодировать любую требуемую функциональность для взаимодействия с пользователем.

Специализированные элементы управления наследуются от класса **Control**. Этот класс обеспечивает функциональность, требуемую для элемента управления. Он предоставляет базовую функциональность, необходимую для взаимодействия элемента управления с остальной частью приложения.

Ключевой задачей при разработке специализированного элемента управления является реализация видимого пользовательского интерфейса. Его можно создать реализацией метода **OnPaint**, вызываемого каждый раз, когда элемент управления отображается на экране.

Выполнив это упражнение, вы научитесь создавать специализированный элемент управления, также представляющий собой цифровые часы. Подобно элементу управления из предыдущего упражнения, они включают компонент **Timer** для обновления пользовательского интерфейса на регулярной основе. Однако теперь вы сами создадите отображение для этого элемента управления вместо того, чтобы использовать элемент управления **Label**.

### **Разработка специализированного элемента управления**

1. Создайте в Visual Studio новое приложение Windows Forms. Назовите его **WinTimer2**.

2. В меню **Project** выберите **Add New Item**. Выберите **Custom Control** в диалоговом окне **Add New Item** и щелкните **Add**. Укажите имя **UserControlTimer2** и нажмите **Add** (**Добавить**). К проекту будет добавлен новый специализированный элемент управления. Откройте и просмотрите его код.

3. Перейдите на вкладку конструктора. В конструкторе **UserControlTimer2** перетащите компонент **Timer** из **Toolbox** на поверхность конструктора.

4. В окне **Properties** присвойте свойству **Interval** компонента **Timer** значение **1000** и свойству **Enabled** значение **True**.

5. Дважды щелкните компонент **Timer** чтобы открыть окно кода в обработчике события **Timer.Tick** по умолчанию и добавьте следующую строку программы:

```
this.Refresh();
```

6. Переопределите метод **OnPaint** и укажите код для отображения прямоугольника, залитого синим цветом, заполняющего весь элемент управления:

```
protected override void OnPaint(PaintEventArgs pe)
{
    base.OnPaint(pe);
    Graphics g = pe.Graphics;
    g.FillRectangle(Brushes.Blue, 0, 0, this.Width,
    this.Height);
}
```

7. Для отображения времени добавьте следующий код к методу **OnPaint** ниже определения прямоугольника:

```
pe.Graphics.DrawString(DateTime.Now.ToString(),
this.Font, new SolidBrush(this.ForeColor), 0, 0);
```

8. В меню **File** выберите **Save All**, чтобы сохранить ваше решение.

9. В меню **Build** выберите **Build Solution**.

### **Применение специализированного элемента управления**

10. Выберите вкладку конструктора **Form1**. Из **Toolbox** перетащите **UserControlTimer2** в форму. К ней будет добавлен экземпляр вашего специализированного элемента управления, который начнет отсчитывать каждую секунду.

11. Постройте и запустите приложение. Обратите внимание, что пользовательский элемент управления действует во время выполнения таким же способом, как в конструкторе.

### **Упражнение 3. Создание расширенных элементов управления**

Расширенные элементы управления – это разработанные пользователем элементы управления, расширяющие уже существующий элемент управления .NET Framework. Сохраняется вся функциональность существующих элементов управления, но при этом добавляются свойства и методы, а при необходимости изменяется и внешнее представление элемента управления.

#### **Разработка расширенного элемента управления**

1. Создайте в Visual Studio новое приложение Windows Forms. Назовите его **WinButNum**.

2. В меню **Project** выберите **Add Class**. Назовите этот класс **ClickButton** и щелкните **Add**.

3. Измените объявление класса, чтобы **ClickButton** наследовал класс **Button**:

```
public class ClickButton : System.Windows.Forms.Button
```

4. Добавьте следующее поле и свойство в окно кода с целью создания свойства **Clicks**:

```
int mClicks;  
public int Clicks  
{  
    get { return mClicks; }  
}
```

5. Переопределите метод **OnClick**, чтобы инкрементировать закрытую переменную **mClicks** каждый раз, когда щелкается кнопка:

```
protected override void OnClick(EventArgs e)  
{  
    mClicks++;  
    base.OnClick(e);  
}
```

6. Переопределите метод **OnPaint**, чтобы отобразить количество щелчков в правом нижнем углу элемента управления:

```
protected override void  
OnPaint(System.Windows.Forms.PaintEventArgs pevent)  
{  
    base.OnPaint(pevent);  
    System.Drawing.Graphics g = pevent.Graphics;  
    System.Drawing.SizeF stringsize;  
    stringsize = g.MeasureString(Clicks.ToString(),  
this.Font, this.Width);  
    g.DrawString(Clicks.ToString(), this.Font,  
System.Drawing.SystemBrushes.ControlText,  
this.Width - stringsize.Width - 3, this.Height -  
stringsize.Height - 3);  
}
```

7. Сохраните и постройте решение.

### **Применение расширенного элемента управления**

8. Выберите вкладку конструктора **Form1**.

9. Из **Toolbox** перетащите экземпляр **ClickButton** в форму и измените его размеры в сторону увеличения.

10. Постройте и запустите приложение.

11. В форме щелкайте **ClickButton1**. Обратите внимание, что количество щелчков отображается в правом нижнем углу.

## **Лабораторная работа 4. Использование окон диалога в формах**

### **Цель работы**

Изучение способов использования компонентов, представляющие диалоговые окна и получение навыков по работе с окнами диалога.

#### **Упражнение 1. Использование компонента SaveFileDialog**

Чтобы пользователи могли сохранять файлы, можно использовать встроенный компонент **SaveFileDialog**.

В этом упражнении Вы отобразите диалоговое окно, используя метод **ShowDialog**. Затем с помощью поля **DialogResult.OK** проверите, нажал ли пользователь кнопку **OK**.

Для реализации отображения диалогового окна обозревателя папок выполните:

1. Создайте приложение Windows Forms, укажите имя **TestStandartDialog**.

2. Добавьте элемент **MenuItem**, задайте имя первого пункта меню **Файл** и команду **Сохранить как...**

3. Добавьте в форму элемент управления **richTextBox**, оставив имя по умолчанию **richTextBox1**. Свойству **Dock** установите **Fill**.

4. Добавьте в форму компонент **SaveFileDialog**. Проверьте, что в области компонентов появился компонент **saveFileDialog1**.

5. Дважды щелкните кнопку, чтобы добавить в редактор кода обработчик событий по умолчанию.

6. В обработчике событий добавьте следующий код для отображения диалогового окна **Сохранение файла**. Этот код сохраняет текст, введенный в элемент управления **richTextBox**, в текстовый файл в указанной папке.

```
saveFileDialog1.Filter = "txt files (*.txt)|*.txt";
if(saveFileDialog1.ShowDialog() ==
System.Windows.Forms.DialogResult.OK
&& saveFileDialog1.FileName.Length > 0)
{
    richTextBox1.SaveFile(saveFileDialog1.FileName,
    RichTextBoxStreamType.PlainText);
}
```

7. Постройте и протестируйте приложение.

8. В открывшейся форме введите какой-либо текст в текстовое поле.

9. Выберите команду **Сохранить как...** и сохраните файл (имя и место для сохранения файла выберите по своему усмотрению).

10. Убедитесь, что текстовый файл находится в указанном месте.

## **Упражнение 2. Использование компонента *ColorDialog***

Для отображения диалогового окна цветовой палитры можно использовать встроенный компонент **ColorDialog** вместо того, чтобы создавать свое собственное диалоговое окно.

В этом упражнении Вы дополните приложение **TestStandartDialog**, чтобы дать пользователям возможность выбрать цвет и применить его в форме Windows после указания соответствующей команды меню.

Для реализации отображения диалогового окна цветовой палитры выполните:

1. Для элемента **MenuItem** задайте имя второго пункта меню – **Формат** и команду – **Цвет фона**.

2. Добавьте в форму компонент **ColorDialog**.

3. Проверьте, что в области компонентов появился компонент **colorDialog1**.

4. Дважды щелкните кнопку **Цвет фона**, чтобы создать обработчик событий по умолчанию в редакторе кода.

5. В обработчике событий добавьте следующий код для отображения диалогового окна выбора цвета и изменения фонового цвета в соответствии с выбором пользователя:

```
if (colorDialog1.ShowDialog() == DialogResult.OK)
{
    richTextBox1.BackColor = colorDialog1.Color;
}
```

6. Постройте и протестируйте приложение.

### **Упражнение 3. Использование компонента *FontDialog***

Для отображения диалогового окна выбора шрифтов можно использовать встроенный компонент **FontDialog** вместо того, чтобы создавать свое собственное диалоговое окно.

В этом упражнении Вы дополните приложение **TestStandartDialog**, чтобы дать пользователям возможность выбрать шрифт в диалоговом окне и затем применить его к тексту.

Для реализации отображения диалогового окна выбора шрифтов выполните:

1. Задайте в меню **Формат** новую команду – **Шрифт**.

2. Перетащите в форму компонент **FontDialog**.

3. Проверьте, что в области компонентов появится компонент **fontDialog1**.

4. Дважды щелкните команду **Шрифт**, чтобы создать в редакторе кода обработчик событий по умолчанию.

5. В обработчик событий добавьте следующий код для отображения диалогового окна выбора шрифта текста в окне и изменения шрифта текста в соответствии с выбором пользователя:

```
if (fontDialog1.ShowDialog() == DialogResult.OK)
{
    richTextBox1.Font = fontDialog1.Font;
}
```

6. Постройте и протестируйте приложение.

### **Упражнение 4. Использование компонента *OpenFileDialog***

Чтобы пользователи могли выбрать текстовый файл и загрузить его в элемент управления **RichTextBox** в форме Windows Forms, можно использовать компонент **OpenFileDialog**.

В этом упражнении Вы дополните приложение **TestStandartDialog**, чтобы дать пользователям возможность открыть текстовый файл.

1. Задайте в меню **Файл** новую команду **Открыть...**

2. Перетащите в форму компонент **OpenFileDialog**.

В области компонентов появился компонент **openFileDialog1**.

3. Дважды щелкните команду **Открыть...**, чтобы создать в редакторе кода обработчик событий по умолчанию.

4. В обработчик событий добавьте следующий код для отображения диалогового окна открытия файла:

```
Stream myStream = null;
OpenFileDialog openFileDialog1 = new OpenFileDialog();
openFileDialog1.InitialDirectory = @"c:\";
openFileDialog1.Filter = "txt files (*.txt)|*.txt|All
files (*.*)|*.*";
openFileDialog1.FilterIndex = 2;
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
    try
    {
        if ((myStream = openFileDialog1.OpenFile()) != null)
        {
            using (myStream)
            {

richTextBox1.LoadFile(openFileDialog1.FileName,
                    RichTextBoxStreamType.PlainText);
                }
            }
        catch (Exception ex)
        {
            MessageBox.Show("Error: Could not read file from disk: "
+ ex.Message);
        }
    }
}
```

5. Постройте и протестируйте приложение.