Язык C#

Построение оконных приложений

Лекция #7

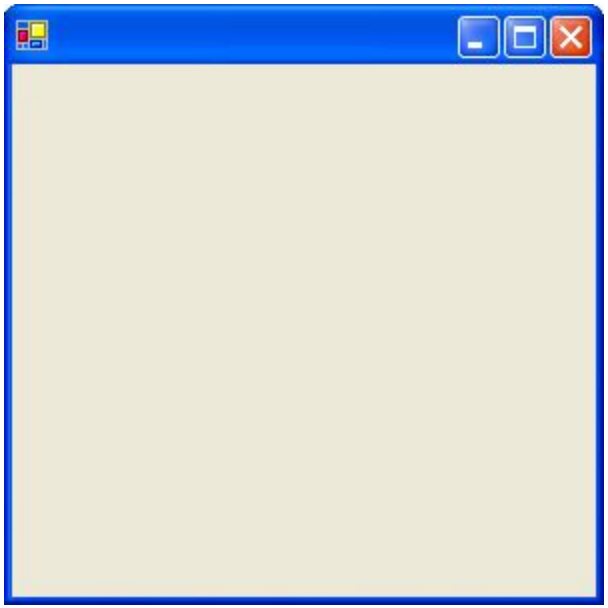
Способы создания приложения Windows

- Вручную и откомпилировать csc.exe
- WinDes.exe из .NET SDK
- Visual Studio.NET

Обзор пространства имен Windows.Forms

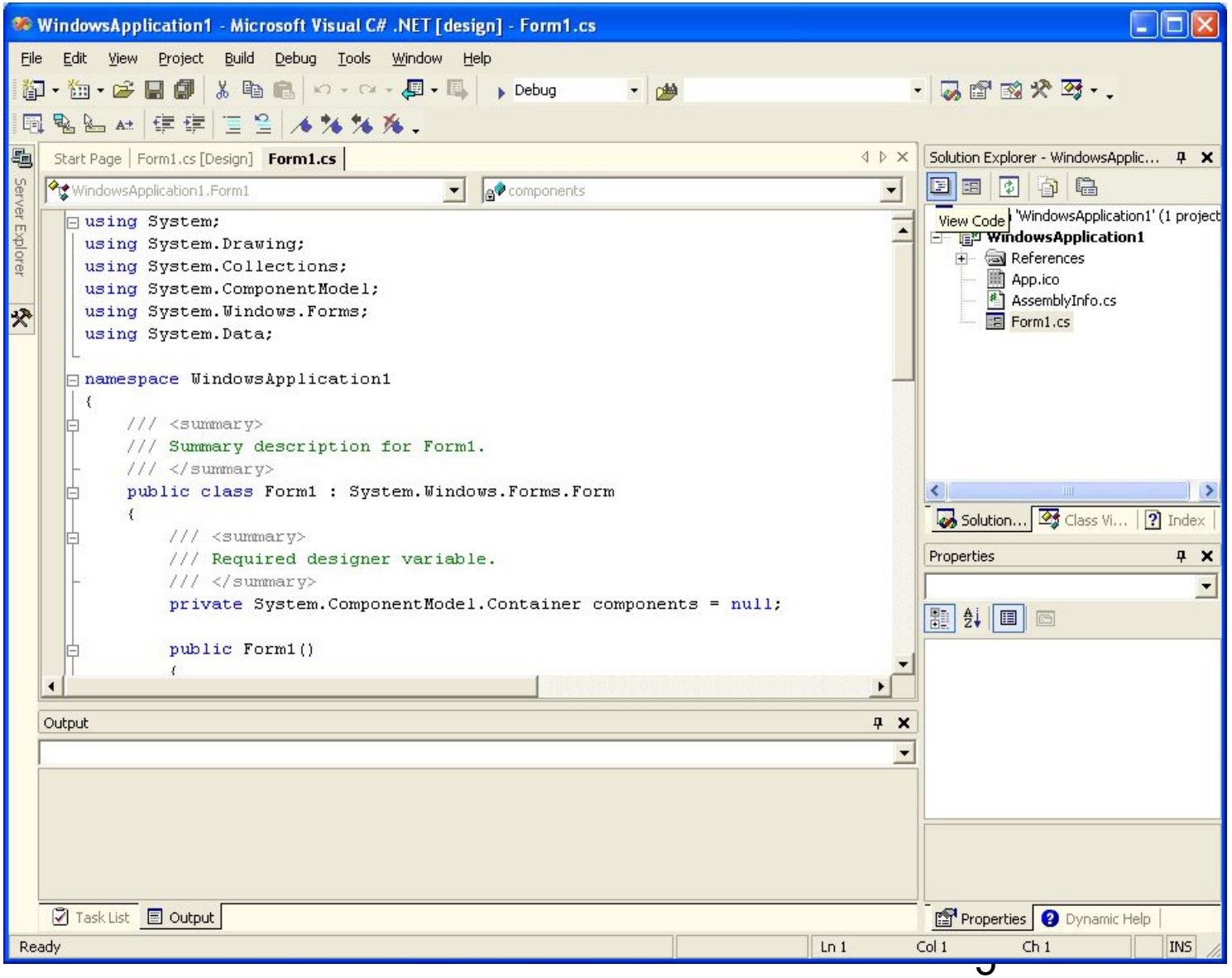
Application	Запуск, завершение. Обработка сообщений
Button, CheckBox, DataGrid …	Элементы графического интерфейса
Form	Главное окно приложения
ColorDialog, FontDialog, …	Готовые диалоговые окна
Menu, MainMenu, MenuItem, …	Ниспадающие и контекстные меню
Clipboard, Help, Timer, Screen, …	Разнообразные вспомогательные типы
StatusBar, ToolBar, ScrollBar, …	Дополнительные элементы управления

Создание главного окна приложения



```
namespace MyRawWindow
{
using System;
using System.Windows.Forms;

public class MainWindow : Form
{
    public MainWindow(){}

    // Запускаем приложение
    public static int Main (string[] args)
    {
        Application.Run(new MainWindow());
        return 0;
    }
}
}
```

Визуальная разработка



Фрагмент кода

```
public class Form1 : System.Windows.Forms.Form
    {
      private System.ComponentModel.Container components = null;

      public Form1()
      {
            InitializeComponent();
            // TODO here
      }

      protected override void Dispose( bool disposing )
      {  // Если удалять что-то, то здесь
            if( disposing )
            {       if (components != null)    { components.Dispose(); }
            }
            base.Dispose( disposing );
      }

      #region Windows Form Designer generated code
      private void InitializeComponent()
      {       this.components = new System.ComponentModel.Container();
            this.Size = new System.Drawing.Size(300,300);
            this.Text = "Form1";
      }
      #endregion

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

Методы объекта Application

AddMessageFilter RemoveMessageFilter	Позволяют приложению перехватывать сообщения. Необходимо создать класс, реализующий интерфейс IMessageFilter
DoEvents	Обеспечивает способность приложения обрабатывать сообщения из очереди сообщений во время длительной операции
Exit	Завершает работу приложения
ExitThreat	Прекращает обработку сообщений для текущего потока и закрывает все окна этого потока
OLERequired	Инициализирует библиотеки OLE
Run	Запускает стандартный цикл работы с сообщениями для текущего потока

Свойства объекта Application

CommonAppDataRegistry	Возвращает параметр системного реестра, который хранит общую для всех пользователей информацию о приложении
CompanyName	Возвращает имя компании
CurrentCulture	Позволяет получить и задать информацию о естественном языке
CurrentInputLanguage	Позволяет получить и задать информацию о естественном языке для ввода информации
ProductName ProductVersion	Имя и версия программного продукта, с которым ассоциировано приложение
StartupPath	Позволяет получить имя выполняемого файла и путь к нему

События объекта Application

ApplicationExit	Возникает в момент закрытия приложения
Idle	Возникает в момент, когда все текущие сообщения в очереди обработаны и приложение переходит в режим бездействия
ThreadExit	Возникает при завершении потока в приложении. Если работу завершает главный поток, это событие возникает до ApplicationExit

Добавление атрибутов в файл AssemblyInfo.cs

```
// Атрибуты для нашей сборки
[assembly:AssemblyCompany("Intertech, Inc.")]
[assembly:AssemblyProduct("A Better Window")]
[assembly: AssemblyCopyright("")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
```

Работа с классом Application

```
namespace AppClassExample
{
using System;
using System.Windows.Forms;
using System.Reflection;

public class MainForm : Form
{
    ...
    public MainForm()
    {
     GetStats();
    }

    private void GetStats()
    {
     MessageBox.Show(Application.CompanyName, "Company:");
     MessageBox.Show(Application.ProductName, "App Name:");
     MessageBox.Show(Application.StartupPath, "I live here:");
    }
}
}
```

Реагируем на событие ApplicationExit

```
public class MainForm : Form
{
  ...
  public MainForm()
  {
   ...
   // Перехватываем событие ApplicationExit
   Application.ApplicationExit += new EventHandler(Form_OnExit);
  }

   // Метод, запускаемый обработчиком событий
   private void Form_OnExit(object sender, EventArgs evArgs)
   {
   MessageBox.Show("See ya!", "This app is dead...");
   }
}
```

// Формат делегата EventHandler

public delegate void EventHandler(object sender, EventArgs evArgs)

Препроцессинг сообщений

```
// Мы должны указать это пространство имен!
using Microsoft.Win32;

// Создаем класс – фильтр сообщений
public class MyMessageFilter : IMessageFilter
{
  public bool PreFilterMessage(ref Message m)
  {
   // Перехватываем нажатие правой кнопки мыши
   if (m.Msg = = 513)      // WM_LBUTTONDOWN = 513
   {
    MessageBox.Show("WM_LBUTTONDOWN is: " + m.Msg);
    return true;
   }
   return false;   // Все остальные сообщения игнорируются
  }
}
```

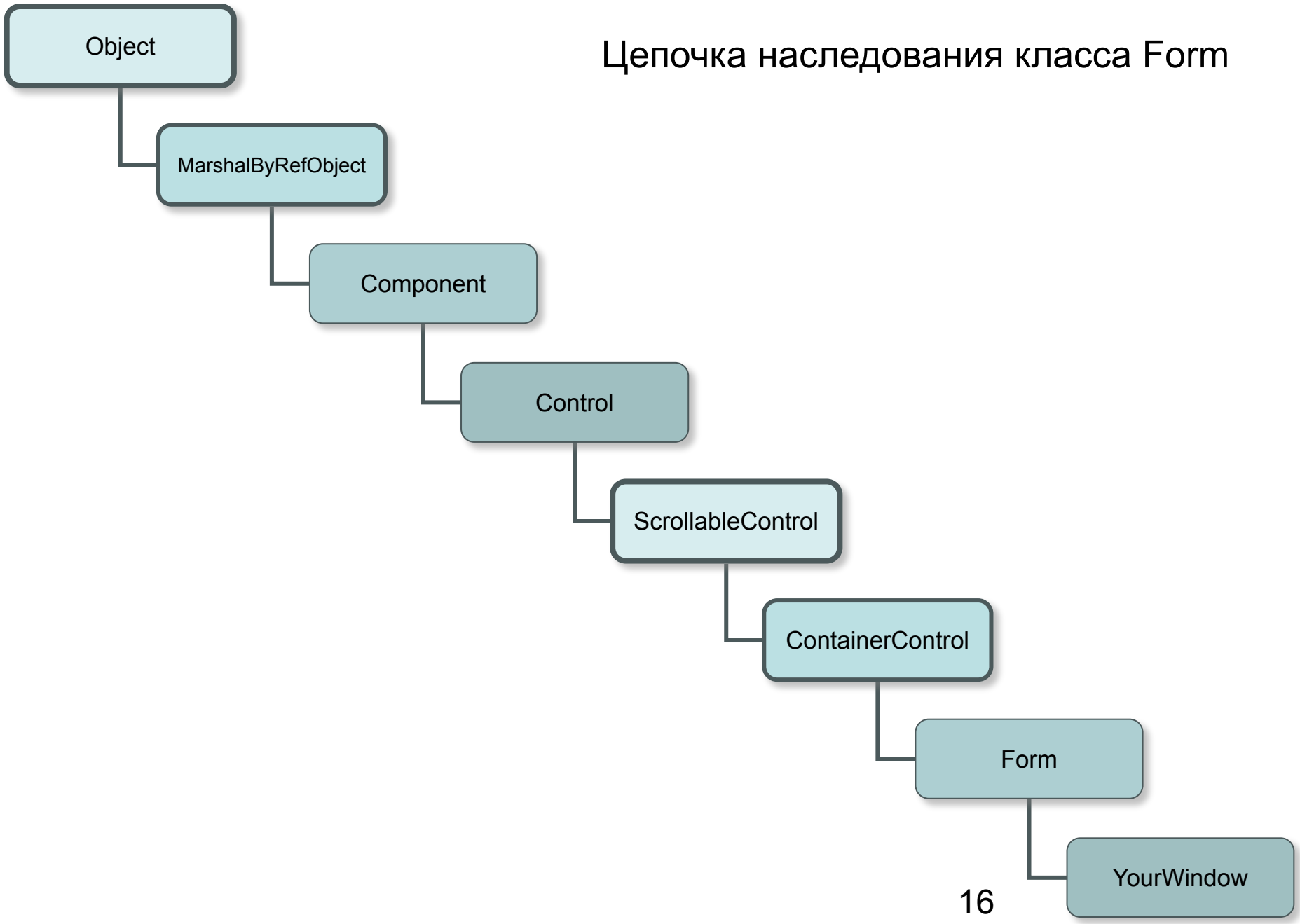
Добавление фильтра

```
public class MainForm : Form
{
   private MyMessageFilter msgFilter = new MyMessageFilter();
   ...
   public MainForm()
   {
    ...
    // Добавляем (регистрируем) фильтр сообщений
    Application.AddMessageFilter(msgFilter);
   }

   // Обработчик событий
   private void Form_OnExit(object sender, EventArgs evArgs)
   {
    MessageBox.Show("See ya!", "This app is dead...");
    // Удаляем фильтр сообщений
    Application.RemoveMessageFilter(msgFilter);
   }
}
```

Анатомия формы

Структура оконного приложения

Цепочка наследования класса Form



Класс Component
реализация интерфейса IComponent

```
public interface IComponent: IDisposable
{
  // Свойство Site
  public ISite Site { virtual get; vitural site;}

  // Событие Disposed
  public event EventHandler Disposed;

}
```

Класс Component

реализация интерфейса ISite, который позволяет элементу управления взаимодействовать с контейнером, в котором он расположен

```
public interface ISite: IServiceProvider
{
  // Свойства интерфейса ISite
 public IComponent Component { virtual get; }
 public IContainer Container { virtual get; }
 public bool DesignMode { virtual get; }
 public string Name { virtual get; virtual set; }
}
```

Как правило, свойства ISite необходимы при разработке собственных элементов управления

Переопределение метода Dispose

```
public override void Dispose()
{
   base.Dispose();

   // Здесь выполняем необходимые действия

}
```

Класс Component реализует метод Dispose, но если вы лучше знаете, как освободить ресурсы, можете его переопределить.

Класс Control

Класс Control определяет общие черты всех типов, относящихся к элементам графического интерфейса

Важные свойства объекта Control

Top, Left, Bottom, Right, Bounds, Height, Width, ClientRectangle	Определяют измерения объекта управления. Bounds и ClientRectangle возвращают объект Rectangle
Created, Disposed, Enabled, Focused, Visible	Возвращают bool, определяющее текущее состояние элемента управления
Exit	Завершает работу приложения
ExitThreat	Прекращает обработку сообщений для текущего потока и закрывает все окна этого потока
OLERequired	Инициализирует библиотеки OLE
Run	Запускает стандартный цикл работы с сообщениями для текущего потока

Важные методы объекта Control

GetStyle, SetStyle	Установка флагов управления стилем формы
Hide, Show	Управление свойством Visible
Invalidate	Заставляет элемент управления обновить собственное изображение
OnXXXX	Множество методов (OnMouseMove, OnKeyDown, OnResize и т.п.), которые могут быть замещены
Refresh	Обновляет элемент управления и все дочерние элементы
SetBounds, SetLocation, SetClientArea	Используются для управления измерениями элемента управления

Важные методы объекта Control

GetStyle, SetStyle	Установка флагов управления стилем формы
Hide, Show	Управление свойством Visible
Invalidate	Заставляет элемент управления обновить собственное изображение
OnXXXX	Множество методов (OnMouseMove, OnKeyDown, OnResize и т.п.), которые могут быть замещены
Refresh	Обновляет элемент управления и все дочерние элементы
SetBounds, SetLocation, SetClientArea	Используются для управления измерениями элемента управления

Настройка стиля формы

```
public enum ControlStyles
{
    AllPaintingToWmPaint,
    CacheText,
    ContainerControl,
    EnableNotifyMessage,
    FixedHeight,
    FixedWidth,
    Opaque,
    ResizeRedraw,
    Selectable,
    StandardClick,
    StandardDoubleClick,
    SupportsTransparentBackColor,
    UserMouse,
    UserPaint
}
```

```
public StyleForm()
{
   SetStyle(ControlStyles.ResizeRedraw, true)
}
```

Настройка стиля формы

```
public enum ControlStyles
{
    AllPaintingToWmPaint,
    CacheText,
    ContainerControl,
    EnableNotifyMessage,
    FixedHeight,
    FixedWidth,
    Opaque,
    ResizeRedraw,
    Selectable,
    StandardClick,
    StandardDoubleClick,
    SupportsTransparentBackColor,
    UserMouse,
    UserPaint
}
```

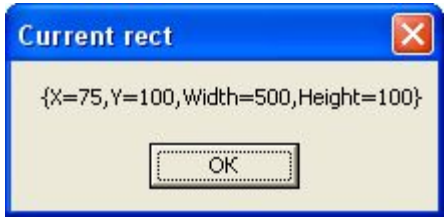
```
public StyleForm()
{
   SetStyle(ControlStyles.ResizeRedraw, true)
}
```

Важные события класс Control

Clickm DoubleClick, MouseEnter, MouseLeave, MouseDown, MouseUp, MouseMove, MouseHover, MouseWheel	События, связанные с мышью
KeyPress, KeyUp, KeyDown	События клавиатуры

Работаем с классом Control

```
// Необходимо для использования типа Rectangle
using System.Drawing;
...

public class MainForm : Form
{
    public static int Main(string[] args)
    {
     Application.Run(new MainForm());
     return 0;
    }

    public MainForm()
    {
     Top = 100;
     Left = 75;
     Height = 100;
     Width = 500;
     MessageBox.Show(Bounds.ToString(), "Current rect");
    }
}
```

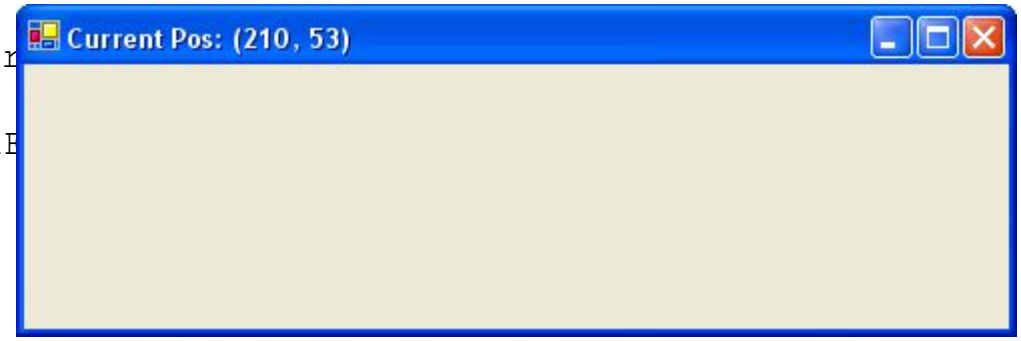


Завершение работы приложения

```
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
            components.Dispose();
    }
   base.Dispose( disposing );
   MessageBox.Show("Disposing this Form");
}
```

Перехват событий от мыши

```
public class MainForm : Form
{
    public static int Main(str
    {
     Application.Run(new MainF
     return 0;
    }

    public MainForm()
    {
     Top = 100;    Left = 75;
     Height = 100; Width = 500;
     MessageBox.Show(Bounds.ToString(), "Current rect");

     // Перехватываем событие MouseUp
     this.MouseUp += new MouseEventHandler(OnMouseUp);
    }

    // Метод, вызываемый при возникновении события MouseUp
    public void OnMouseUp(object sender, MouseEventArgs e)
    {
     this.Text = "Clicked at: (" + e.X + ", " + e.Y +")";
    }
...
}
```



Перехват событий от мыши

```
public class MainForm : Form
{
...
    public MainForm()
    {
     ...
     // Отслеживаем движения мыши (вместе с событием MouseUp
     this.MouseUp += new MouseEventHandler(OnMouseUp);
     this.MouseMove += new MouseEventHandler(OnMouseMove);
    }

    public void OnMouseUp(object sender, MouseEventArgs e)
    {
     if(e.Button = = MouseButtons.Left)
          MessageBox.Show("Left click!");
     if(e.Button = = MouseButtons.Right)
          MessageBox.Show("Right click!");
     if(e.Button = = MouseButtons.Middle)
          MessageBox.Show("Middle click!");
    }

    public void OnMouseMove(object sender, MouseEventArgs e)
    {
     this.Text = "Current Pos: (" + e.X + ", " + e.Y + ")";
    }
...
}
```

2-й вариант

```
public class MainForm : Form
{
     public MainForm()
     {
     // Сейчас обработчики событий нам уже нужны
     // this.MouseUp += new MouseEventHandler(OnMouseUp);
     // this.MouseUp += new MouseEventHandler(OnMouseMove);
     }

      protected override void OnMouseUp( //object sender,
                        MouseEventArgs e)
      {
     // Какая кнопка мыши нажата?
        if(e.Button == MouseButtons.Left)   MessageBox.Show("Left click!");
        if(e.Button == MouseButtons.Right) MessageBox.Show("Right click!");
        if(e.Button == MouseButtons.Middle) MessageBox.Show("Middle click!");
       }

    protected override void OnMouseMove(//object sender,
                        MouseEventArgs e)
    {
     this.Text = "Current Pos: (" + e.X + ", " + e.Y + ")";
    }
...
}
```

Реагируем на события клавиатуры

```
public class MainForm : form
{
...
   public MainForm()
   {
    Top = 100;     Left = 75;
    Height = 100;      Width = 500;
    MessageBox.Show(Bounds.ToString(), "Current rect");
    ...
    // Перехватываем событие KeyUp
    this.KeyUp += new KeyEventHandler(OnKeyUp);
   }

   public void OnKeyUp(object sender, KeyEventArgs e)
   {
    MessageBox.Show(e.KeyCode.ToString(), "KeyPressed!");
   }
...
}
```

Свойства класса KeyEventArgs

Alt	Инф. о том, нажата ли клавиша Alt
Control	То же для Control
Shift	То же для Shift
Handled	Позволяет получить или установить значение, которое говорит о том, обрабатывается ли данное событие
KeyCode	Код клавиши в событиях KeyDown и KeyDown
KeyData	Код сочетания клавиш
Modifiers	Состояние управляющих клавиш

Дополнительные свойства класса Control

AllowDrop	Разрешать/запретить drag-and-drop
BackColor, Font, BackGroundImage, ForeColor, Cursor	Определяют внешний вид формы
ContextMenu	Определяет контекстное меню по правой клавиши мыши
Anchor, Dock	Определяет способы привязки к родительскому объекту
Opacity	Степень прозрачности элемента
Region	Определение очертаний объекта

Дополнительные методы класса Control

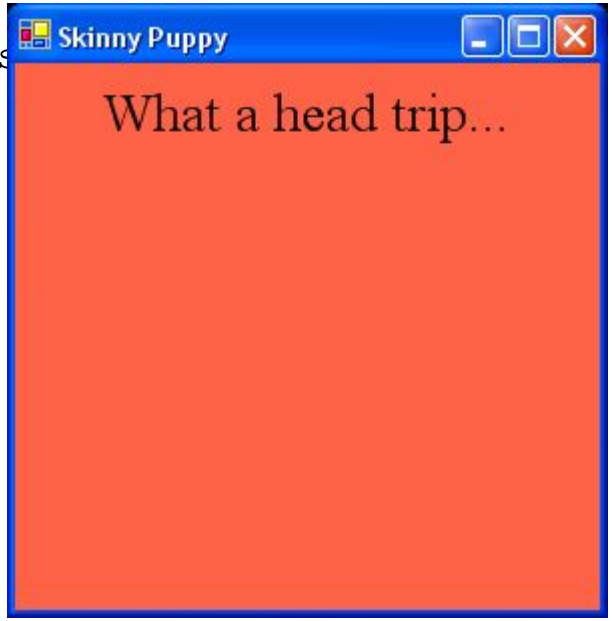
DoDragDrop, OnDragDrop, OnDragEnter, OnDragLeave, OnDragOver	Мониторинг операций drag-and-drop
ResetFont, ResetCursor, ResetForeColor, ResetBackColor	Сделать свойства элемента такими же, как у родительского контейнерного элемента
OnPaint	Метод для замещения в производных классах

Дополнительные события класса Control

DragDrop, DragEnter, DragLeave, DragOver	События в операциях drag-and-drop
Paint	Событие возникает в тех ситуациях, когда изображение элемента должно быть обновлено

Возможности класса Control

```
using System;
using System.Windows.Forms;
using System.Drawing;  // Для типов Color, Brus

public class MainForm : Form
{  ...
   public MainForm()
   {
   // Устанавливаем значения некоторых свойств
   BackColor = Color.Tomato;
   Opacity = 0.5d;
   Cursor = Cursors.WaitCursor;

   // Перехватываем событие Paint
   this.Paint += new PaintEventHandler(Form1_Paint);
   }

   private void Form1_Paint(object sender, PaintEventArgs e)
   {
    Graphics g = e.Graphics;
    g.DrawString("What a head trip...",
        new Font("Times New Roman", 20),
        new SolidBrush(Color.Black), 40, 10);
   }
}
```

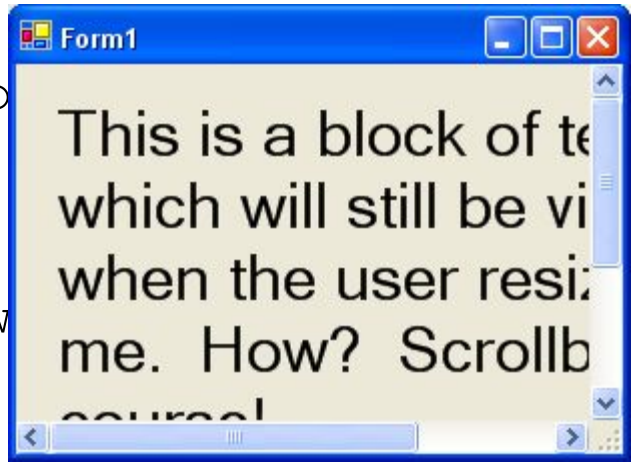


Событие Paint
и PaintEventArgs

- ClipRectangle – прямоугольная область вывода изображения
- Graphics – контекст вывода изображения

Класс ScrollableControl

```
// Этот код помещается в конструкто
// InitializeComponent()
// Для работы с классом Size также
// на пространство имен System.Draw

this.AutoScroll = true;
this.AutoScrollMiSize=new System.Drawing.Size(300,300);
```



Класс ContainerControl

ActiveControl	Это свойство позволяет получать информацию о том, какой элемент управления находится в фокусе, а также помещать в фокус элемент управления
ParentForm	Свойство Ссылка на форму-контейнер
ProcessTabKey()	Метод позволяет программным образом имитировать нажатие клавиши Tab для передачи фокуса следующему в очереди элементу управления

Некоторые свойства класса Form (2)

ControlBox	Установить/получить значение, определяющее будет ли присутствовать значок системного меню
Menu MergedMenu	Используется для установки и получения меню на форме
MaximizeBox MinimizeBox	Определяют присутствие стандартных значков «свернуть» и «восстановить»
ShowInTaskBar	Определяет, будет ли форма показываться в панели задач
StartPosition	Позволяет получить или установить значение, определяющее положение формы в момент старта программы (перечисление FormStartPosition)
WindowState	Определяет состояние отображаемой формы при запуске (перечисление FormWinowState)

Некоторые свойства класса Form

AcceptButton	Позволяет получить информацию или установить кнопку, которая будет активирована при нажатии пользователем Enter
ActiveMDIChild IsMDIChild IsMDIContainer	Предназначены для работы с MDI-приложениями
AutoScale	Позволяет установить или получить значение, определяющее, будет ли форма автоматически изменять свои размеры, чтобы наилучшим образом соответствовать высоте шрифта или размерам элементов управления
BorderStyle	Установить/получить стиль рамки вокруг формы
CancelButton	Позволяет получить информацию или установить кнопку, которая будет активирована при нажатии пользователем Esc

Некоторые методы класса Form

Activate()	Активирует указанную форму и помещает ее в фокус
Close()	Закрывает форму
CenterToScreen()	Помещает форму в центр экрана
LayoutMDI()	Размещает все дочерние формы на родительской (перечисление LayoutMDI)
OnResize()	Может быть замещен для реагирования на событие Resize
ShowDialog()	Отображает форму как диалоговое окно

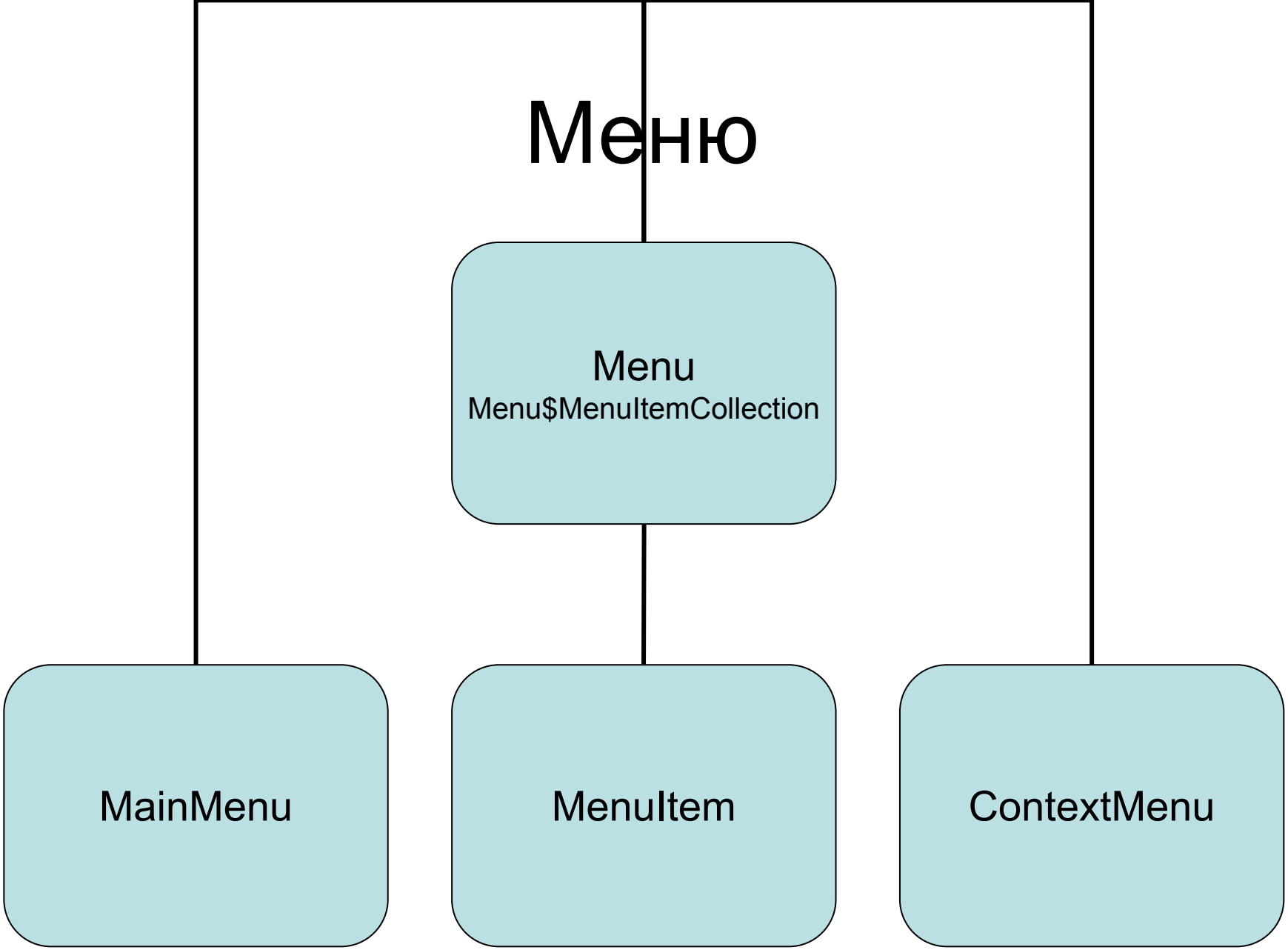
Некоторые события класса Form

Activate	Происходит при активизации формы
Close Closing	Происходят во время закрытия формы
MDIChildActive	Возникает при активизации дочернего окна

Иллюстрация возможностей класса Form

```
public class MainForm: Form

{
      ...
  public MainForm()
  {
     // Настраиваем исходный облик нашей формы
     BackColor = Color.LemonChiffon;       // Цвет фона:
     Text = "My Fantastic Form";           // Заголовок формы:
     Size = new Size(200, 200);            // Размер 200*200:
     CenterToScreen();               // Помещаем форму в центр экрана:

     // Перехватываем события
     this.Resize += new EventHandler(this.MainForm_Resize);
     this.Paint += new PaintEventHandler(this.MainForm_Paint);
  }

  private void MainForm_Resize(object sender, System.EventArgs e)
  {
   // При изменении размера формы ее изображение нужно обновить! Можно также
   // вместо вызова Invalidate просто установить поддержку стиля ResizeRedraw
   Invalidate();
}

  // Выводим текстовую строку.
  private void MainForm_paint(object sender, PaintEventArgs e)
  {
     Graphics g = e.Graphics;
     g.DrawString("Windows Forms is for building GUIs!",
     new Font("Times New Roman", 20), new SolidBrush(Color.Black),
     this.DisplayRectangle);   // Выводим в клиентской прямоугольной  области
  }
}
```

Меню


Menu
Menu$MenuItemCollection
MainMenu
MenuItem
ContextMenu

Члены класса Menu

Handle	Это свойство обеспечивает доступ к значению HMENU
IsParent	Это свойство определяет, содержит ли данное меню подменю или оно конечным
MdiListItem	Это свойство возвращает объект MenuItem, который содержит список дочерних окон MDI
MenuItem	Этой свойство возвращает объект вложенного класса Menu.MenuItemCollection
GetMainMenu()	Возвращает объект MainMenu, в котором содержится текущее меню
MergeMenu()	Объединяет элементы в единое меню согласно данным, содержащимся в свойствах mergeType и mergeOrder.
CloneMenu()	Создает меню, являющееся полной копией другого меню (создается полная локальная копия, а не ссылка)

Члены вложенного класса Menu$MenuItemCollection

Count	Возвращает текущее количество объектов класс MenuItem в коллекции
Add()	Добавляет новых объект класса MenuItem в коллекцию. Существует множество перегруженных вариантов этого метода, которые позволяют указывать клавиши быстрого доступа, делегаты и многое другое
Remove()	Это свойство удаляет объект MenuItem из коллекции
AddRange()	Позволяет добавить в коллекцию за один раз массив объектов MenuItem
Clear()	Удаляет все объекты MainMenu из коллекции
Contains()	Используется для того, чтобы определить, присутствует ли определенный объект класса MenuItem в коллекции

Создание меню программным способом

```
public class MainForm: Form
{
   private MainMenu mainMenu; // Главное меню для Form

   public MainForm()
   {
      mainMenu = new MainMenu();     // Создаем главное меню

     // Создаем меню File и добавляем его в MenuItemCollection
      MenuItem miFile = mainMenu.MenuItems.Add("&File");

     // Теперь создаем подменю Exit и добавляем его в меню File. Этот вариант Add()
     // принимает: (1) создаваемый объект MenuItem; (2) создаваемый делегат
     // (EventHandler); (3) необязательную клавиатурную комбинацию  быстрого доступа

     miFileMenuItems.Add(new MenuItem("E&xit",
            new
    EventHandler(this.FileExit_Clicked),Shortcut.CtrlX));

    // Присоединяем главное меню к объекту Form
     this.Menu = mainMenu;
    }
 ...
}
```

```
public class MainForm : Form // Простое приложение с главным меню
{
  // Главное меню для формы
  private MainMenu mainMenu;

  // Запускаем приложение
  [STAThread]
  public static int Main(string[] args)  { Application.Run(new MainForm());  }

  // Создаем форму
  public MainForm()
  {
    // Настраиваем исходный облик и местонахождение формы
    Text = "Simple Menu";CenterToScreen();

    mainMenu = new MainMenu(); // Создаем объект главного меню

    // Создаем меню File | Exit
    MenuItem miFile = mainMenu.MenuItems.Add("&File");
    miFile.MenuItems.Add(new MenuItem("E&xit",
     new EventHandler(this.FileExit_Clicked), Shortcut.CtrlX));

    this.Menu = mainMenu; // Присоединяем главное меню к объекту Form

    // MainMenu.GetForm() возвращает ссылку на форму, на которой расположено
    // меню. Поэтому мы можем сделать такой маленький фокус:
    mainMenu.GetForm().BackColor = Color.Black;
  }

  // И не забыть про обработчик событий для File-Exit
  private void FileExit_Clicked(object sender, EventArgs e)
  {
    this.Close();         // Выход из приложения
  }
}
```

Добавляем еще одно меню верхнего уровня


```
public class MainForm : Form
{
  private MainMenu mainMenu;
  ...
  public MainForm()
  {
    // Создаем меню File - Exit
    MenuItem miFile = mainMenu.MenuItems.Add("&F
    miFile.MenuItems.Add(new MenuItem("E&xit",
     new EventHandler(this.FileExit_Clicked), Sho

    // А теперь — еще и меню Help - About
    MenuItem miHelp = mainMenu.MenuItems.Add("He
   miHelp.MenuItems.Add(new MenuItem("&About",
   new EventHandler(this.HelpAbout_Clicked), Shortcut.CtrlA));
        ...
  }
   // И обработчик события для Help - About
   private void HelpAbout_Clicked(object sender, EventArgs e)
   {
   MessageBox.Show("The amazing menu app...");
   }
}
```

Создаем контекстное меню I

```
namespace MainForm
{
    // Вспомогательная структура для установки размера шрифта
    internal struct TheFontSize
    {
     public static int Huge = 30;
     public static int Normal = 20;
     public static int Tiny = 8;
    }

    public class MainForm : Form
    {
     // Исходный размер шрифта
     private int currFontSize = TheFontSize.Normal;

     // Контекстное меню формы
     private ContextMenu popUpMenu;

     public static void Main(string[] args)
     {
          Application.Run(new MainForm());
     }

     private void MainForm_Resize(object sender, System.EventArgs e)
     {
          Invalidate();
     }
```

Создаем контекстное меню II

```
public MainForm()
{
    // Прежде всего создаем контекстное меню
    popUpMenu = new ContextMenu();

    // Теперь добавляем в контекстное меню элементы
    popUpMenu.MenuItems.Add("Huge", new EventHandler(PopUp_Clicked));
    popUpMenu.MenuItems.Add("Normal", new EventHandler(PopUp_Clicked));
    popUpMenu.MenuItems.Add("Tiny", new EventHandler(PopUp_Clicked));

    // Теперь подключаем контекстное меню к форме
    this.ContextMenu = popUpMenu;

    // Ставим обработчики событий
    this.Resize += new System.EventHandler(this.MainForm_Resize);
    this.Paint += new PaintEventHandler(this.MainForm_Paint);
}
```

Создаем контекстное меню III

```
// Обработчик для PopUp_Clicked (всех трех пунктов)
private void PopUp_Clicked(object sender, EventArgs e)
{
// Ориентируемся на строковое имя выбранного пользователем элемента  меню
    MenuItem miClicked = (MenuItem)sender;
    string item = miClicked.Text;

    if(item = = "Huge")    currFontSize = TheFontSize.Huge;
    if(item = = "Normal")  currFontSize = TheFontSize.Normal;
    if(item = = "Tiny")    currFontSize = TheFontSize.Tiny;
    Invalidate();
}

private void MainForm_Paint(object sender, PaintEven
{
    Graphics g = e.Graphics;
    g.DrawString("Please click on me...",
    new Font("Times New Roman", (float)currFontSize),
     new SolidBrush(Color.Black),
     this.DisplayRectangle);
}
}
}
```

Дополнительные возможности меню

Checked	Установить/получить флажок
DefaultItem	Установить/получить пункт меню по умолчанию
Enabled	Сделать доступным или недоступным пункт меню
Index	Получить/установить позицию пункта меню
MergeOrder	Тоже при слиянии двух меню
MergeType	Установить/получить значение, определяющее поведение пункта меню при слиянии

Дополнительные возможности меню

OwnerDraw	Позволяет определить, кто будет ответственен за прорисовку пункта меню: Windows или ваш код
RadioCheck	Будет ли отображаться переключатель (вместо Checked)
Shortcut	Получить/установить клавиатурную комбинацию
ShowShortcut	Позволяет установить отображение клавиатурной комбинации рядом с пунктом меню
Text	Получить/установить название пункта меню

Установка флажка в контекстном меню I

```
public class MainForm : Form
{
   // Текущий размер шрифта
   private int currFontSize = TheFontSize.Normal;

   // Контекстное меню для формы
   private ContextMenu popUpMenu;

   // Дополнительные ссылки для отслеживания пункта меню,
   // напротив которого надо установить флажок

   private MenuItem currentCheckedItem;
       // Представляет выбранный в настоящий  момент пункт меню

    private MenuItem checkedHuge;
    private MenuItem checkedNormal;
    private MenuItem checkedTiny;
...
}
```

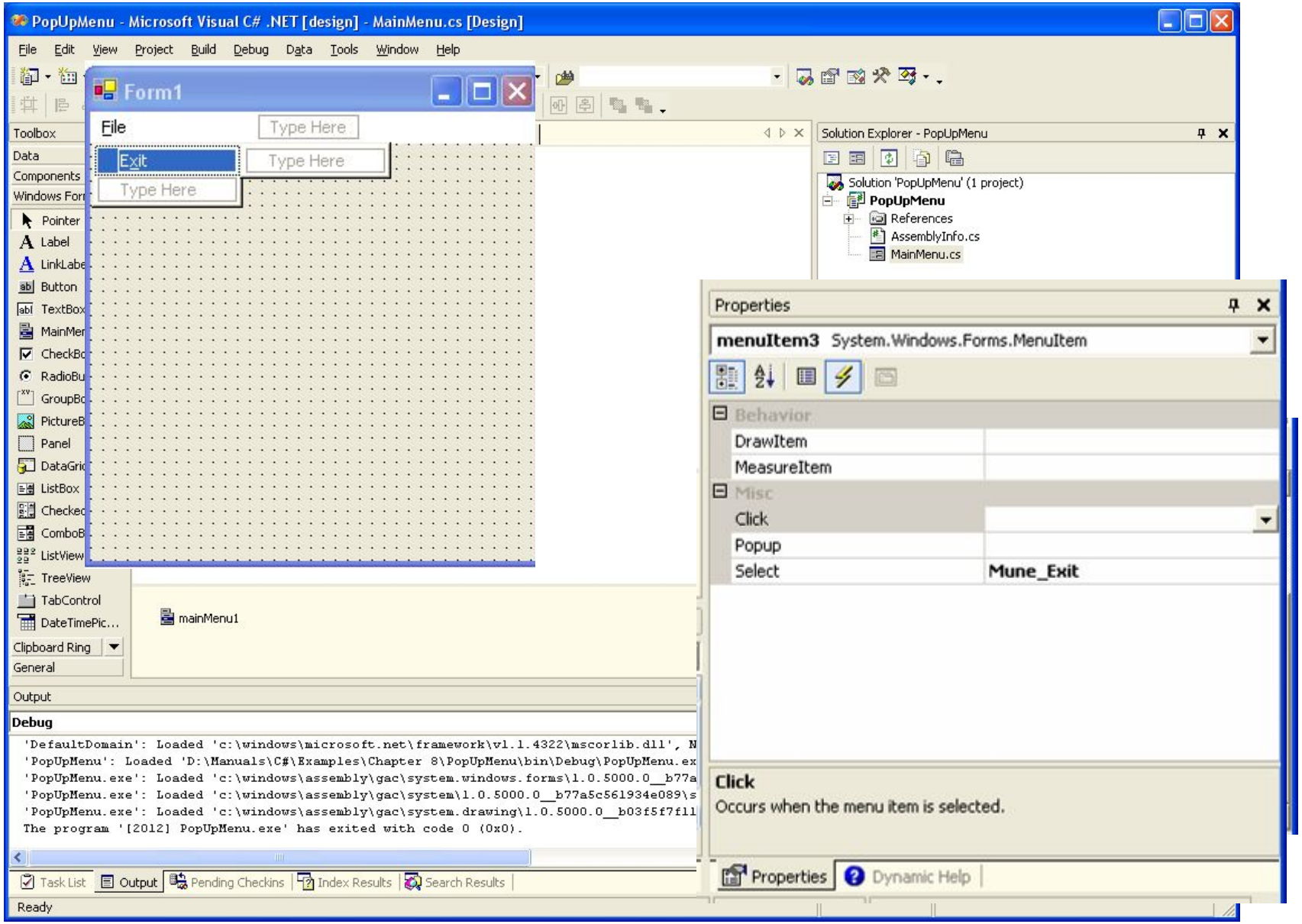
Установка флажка в контекстном меню II

```
// Конструктор формы
public MainForm()
{
    // Настраиваем исходный облик формы
    Text = "PopUp Menu";  CenterToScreen();

    // Прежде всего создаем контекстное меню
    popUpMenu = new ContextMenu();

    // А теперь добавляем пункты меню
    popUpMenu.MenuItems.Add("Huge", new EventHandler(PopUp_Clicked));
    popUpMenu.MenuItems.Add("Normal", new EventHandler(PopUp_Clicked));
    popUpMenu.MenuItems.Add("Tiny", new EventHandler(PopUp_Clicked));

    this.ContextMenu = popUpMenu;

    // Теперь привязываем каждую из наших ссылок на MenuItem к элементам
    // контекстного  меню
    checkedHuge = this.ContextMenu.MenuItems[0];
    checkedNormal = this.ContextMenu.MenuItems[1];
    checkedTiny = this.ContextMenu.MenuItems[2];

    // А теперь ставим флажок напротив Normal:
    currentCheckedItem = checkedNormal;
    currentCheckedItem.Checked = true;
}
```

Установка флажка в контекстном меню III

```
private void PopUp_Clicked(object sender, EventArgs e)
{
// Снимаем флажок с выбранного в настоящий момент пункта
currentCheckedItem.Checked = false;

// Определяем название выбранного пользователем пункта меню
MenuItem miClicked = (MenuItem)sender;
string item = miClicked.Text;

// В ответ на выбор пользователя устанавливаем нужный размер шрифта и
   флажок
// напротив пункта меню
if(item = = "Huge")   {    currFontSize = TheFontSize.Huge;
             currentCheckedItem = checkedHuge;
         }
if(item = = "Normal") {    currFontSize = TheFontSize.Normal;
             currentCheckedItem = checkedNormal;
         }
if(item = = "Tiny")   {    currFontSize = TheFontSize.Tiny;
             currentCheckedItem = checkedTiny;
         }

 // А теперь устанавливаем флажок
 currentCheckedItem.Checked = true;
 Invalidate();
}
```

Создание меню при помощи VS



Свойства объекта StatusBar

BackgroundImage	Получить/установить изображение, как фон строки состояния
Font	Получить/установить шрифт строки состояния
ForeColor	Цвет текста строки состояния
Panels	Возвращает вложенный тип StatusBarPanelCollection
ShowPanels	Получить/установить видимость отдельной панели
SizingGrip	Отображение «цеплялки» в правом нижнем углу

Свойства объекта StatusBarPanel

Свойство	Назначение	Значение по умолчанию
Alignment	Определяет, как будет выровнен текст на панели	HorizontalAligment.Left
AutoSize	Определяет, будет ли панель автоматически изменять свой размер	StatusBarPanelAutoSize.None
BorderStyle	Определяет стиль рамки вокруг строки состояния	StatusBarPanelBorderStyle.Sunken
Icon	Определяет, будет ли создан значок для этой панели	Нулевая ссылка

Свойства объекта StatusBarPanel

MinWidth	Минимальная ширина панели	10
Style	Определяет стиль, соответствующий содержанию панели	StatusBarPanelStyle.Text
Text	Заголовок панели	Пустая строка
ToolTipText	Подсказка, которая будет выдана, если установить над панелью мышь	Пустая строка
Width	Ширина строки состояния	100

Создаем строку состояния

```
public class MainForm : Form
{
   // Создаем объекты для всей строки состояния и ее
   // отдельных панелей
   private StatusBar statusBar = new StatusBar();
   private StatusBarPanel sbPnlPrompt = new StatusBarPanel();
   private StatusBarPanel sbPnlTime = new StatusBarPanel();

   public MainForm()
   {
    ...
        // Метод, который создаст нам строку состояния
    // и настроит ее нужным образом

    BuildStatBar();
   }
```

Создаем строку состояния

```
private void BuildStatBar()
    {
     // Настраиваем строку состояния
     statusBar.ShowPanels = true;
     statusBar.Size = new System.Drawing.Size(212, 20);
     statusBar.Location = new System.Drawing.Point(0, 216);

    // Панели добавляются в строку состояния при помощи метода AddRange()
    statusBar.Panels.AddRange(new StatusBarPanel[] {sbPnlPrompt, sbPnlTime});

     // Настраиваем левую панель
     sbPnlPrompt.BorderStyle = StatusBarPanelBorderStyle.None;
     sbPnlPrompt.AutoSize = StatusBarPanelAutoSize.Spring;
     sbPnlPrompt.Width = 62;
     sbPnlPrompt.Text = "Ready";

     // Настраиваем правую панель
     sbPnlTime.Alignment = HorizontalAlignment.Right;
     sbPnlTime.Width = 76;
```

Создаем строку состояния

```
   // Добавляем значок
   try
   {
   // Этот значок обязательно должен находиться в 
    Icon i = new Icon("status.ico");
    sbPnlPrompt.Icon = i;
   }
   catch(Exception e)
   {
        MessageBox.Show(e.Message);
   }

   // Теперь добавляем панель управления в коллекцию Controls
   // для формы
        this.Controls.Add(statusBar);
 }
}
```

Члены класса Timer

Enabled	Будет ли объект Timer сразу генерировать событие Tick
Interval	Промежуток между событиями Tick в мс
Start(), Stop()	Управляют генерацией Tick (аналогично Enabled)
OnTick()	Метод может быть замещен в потомках
Tick	Событие можно использовать для добавления обработчиков

Работа с таймером

```
public class MainForm : Form
{
...
    private Timer timer1 = new Timer();

    public MainForm()
    {
        // Настраиваем объект Timer
        timer1.Interval = 1000;
        timer1.Enabled = true;
        timer1.Tick += new EventHandler(timer1_Tick);
        ...
    }

    // Этот метод будет вызываться примерно каждую секунду
    private void timer1_Tick(object sender, EventArgs e)
    {
        DateTime t = DateTime.Now;
        string s = t.ToLongTimeString();

        // Выводим полученное строковое значение на правую панель
        sbPnlTime.Text = s;
    }
}
```

Отображение в строке состояния подсказок к пунктам меню

```
public class MainForm : Form
{
     ...
    public MainForm()
    {
          ...
    // Событие MenuComplete происходит при выходе пользователем из меню.
    // Мы будем реагировать на это событие, возвращая в левую панель строку
    // по умолчанию "Ready". Если мы этого не сделаем, в строке состояния
    // останется значение, которое изменилось при выборе пользователем пункта меню!
    this.MenuComplete += new EventHandler(StatusForm_MenuDone);
    BuildMenuSystem();
    }

    private void FileExit_Selected(object sender, EventArgs e)
    {
     sbPnlPrompt.Text = "Terminates this app";
    }

    private void HelpAbout_Selected(object sender, EventArgs e)
    {
     sbPnlPrompt.Text = "Displays app info";
    }

    private void StatusForm_MenuDone(object sender, EventArgs e)
    {
     sbPnlPrompt.Text = "Ready";
    }
```

Отображение в строке состояния подсказок к пунктам меню

```
// Вспомогательные функции
private void BuildMenuSystem()
{
    // Создаем главное меню
    mainMenu = new MainMenu();

    // Создаем меню File
    MenuItem miFile = mainMenu.MenuItems.Add("&File");
    miFile.MenuItems.Add(new MenuItem("E&xit",
                new EventHandler(this.FileExit_Clicked), Shortcut.CtrlX));

    // Обрабатываем событие Select для пункта меню Exit
    miFile.MenuItems[0].Select += new EventHandler(FileExit_Selected);

    // Теперь создаем меню Help | About
    MenuItem miHelp = mainMenu.MenuItems.Add("Help");
    miHelp.MenuItems.Add(new MenuItem("&About",
                new EventHandler(this.HelpAbout_Clicked), Shortcut.CtrlA));

    // Обрабатываем событие Select для пункта меню About
    miHelp.MenuItems[0].Select += new EventHandler(HelpAbout_Selected);

    // Присоединяем главное меню к объекту Form
    this.Menu = mainMenu;
}
...
}
```

Панель инструментов

- Класс ToolBar
- Класс ToolBarButton
- Поместить кнопки в ToolBarButtonCollection объекта ToolBar

Свойства класса ToolBar

BorderStyle	Определяет стиль рамки вокруг панели инструментов
Buttons	Для работы с набором кнопок на панели инструментов
ButtonSize	Определяет размер кнопок на панели инструментов
ImageList	Изображения, используемые на панели инструментов
ImageSize	Размер используемых изображений
ShowToolTips	Определяет, будут ли показываться подсказки для кнопок панели
Wrappable	Определяет возможность переноса части кнопок в следующий ряд

Свойства класса ToolBarButton

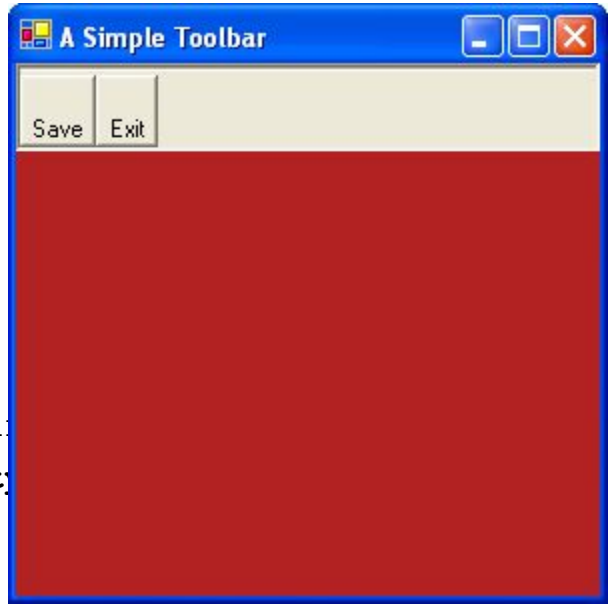
DropDownMenu	«Стрелочка» справа от кнопки. Доступно, если Style – DropDownButton
ImageIndex	Номер изображения, используемого для данной кнопки в коллекции ImageList
Style	Определяет стиль кнопки из перечисления ToolBarButtonStyle
Text	Надпись на кнопке
ToolTipText	Текст подсказки
Visible	Определяет видимость кнопки

Простейшая (без изображений) панель

```
public class MainForm : Form
{
    // Создаем панель инструментов и две кнопки
    private ToolBarButton tbSaveButton = new ToolBarButton();
    private ToolBarButton tbExitButton = new ToolBarButton();
    private ToolBar toolBar = new ToolBar();

    public MainForm()
    {
      ...
     BuildToolBar();       // Вспомогательная функция
    }
    ...
```

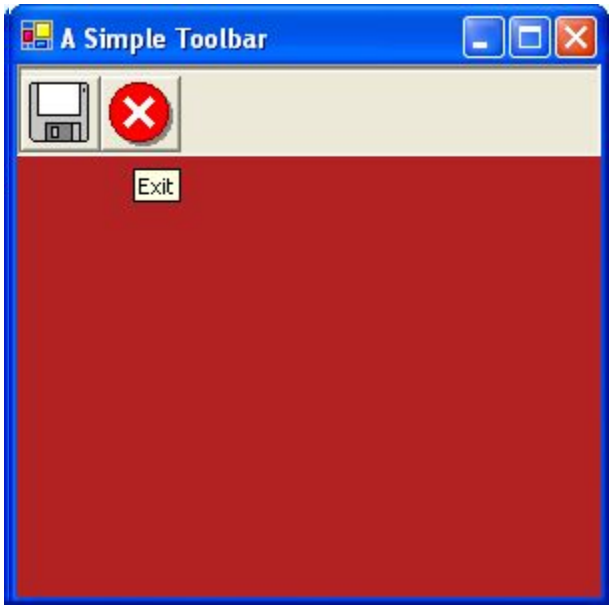
Простейшая (без изображений) панель

```
private void BuildToolBar()
{
    // Настраиваем каждую кнопку
    tbSaveButton.Text = "Save";
    tbSaveButton.ToolTipText = "Save";
    tbExitButton.Text = "Exit";
    tbExitButton.ToolTipText = "Exit";

// Настраиваем параметры панели инструментов и доба
toolBar.BorderStyle = System.Windows.Forms.BorderSt
toolBar.ShowToolTips = true;
toolBar.Buttons.AddRange(new ToolBarButton[]
        {tbSaveButton, tbExitButton});
toolBar.ButtonClick += new ToolBarButtonClickEventHandler(ToolBar_Clicked);
// Добавляем панель инструментов в коллекцию Controls
this.Controls.Add(toolBar);
}

// Обработчик событий для кнопок
private void ToolBar_Clicked(object sender, ToolBarButtonClickEventArgs e)
    { MessageBox.Show(e.Button.ToolTipText);}
}
```

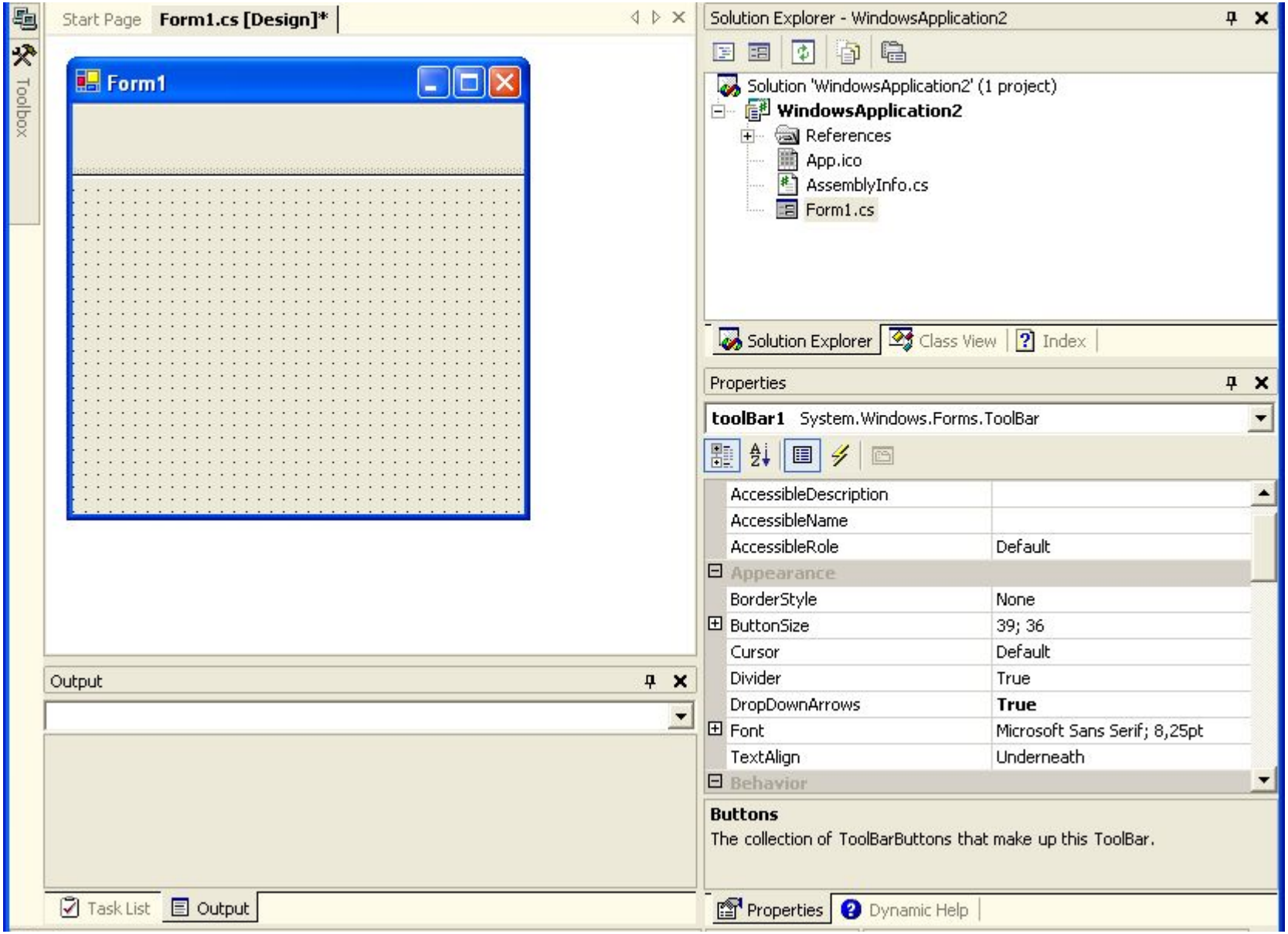


Добавление изображений

```
public class MainForm : Form
{
   // Объект для хранения набора изображений
   private ImageList toolBarIcons = new ImageList();
   ...
   private void BuildToolBar()
   {
    // Настраиваем кнопку "Save"
    tbSaveButton.ImageIndex = 0;
    tbSaveButton.ToolTipText = "Save";

    // Настраиваем кнопку "Exit"
    tbExitButton.ImageIndex = 1;
    tbExitButton.ToolTipText = "Exit";

    // Создаем панель инструментов и добавляем на нее кнопки
    toolBar.ImageList = toolBarIcons;
      ...
    // Загружаем значки (соответствующие файлы должны быть в каталоге приложения
    toolBarIcons.ImageSize = new System.Drawing.Size(32, 32);
    toolBarIcons.Images.Add(new Icon("filesave.ico"));
    toolBarIcons.Images.Add(new Icon("fileexit.ico"));
    toolBarIcons.ColorDepth = ColorDepth.Depth16Bit;
    toolBarIcons.TransparentColor = System.Drawing.Color.Transparent;
          ...
   }
}
```

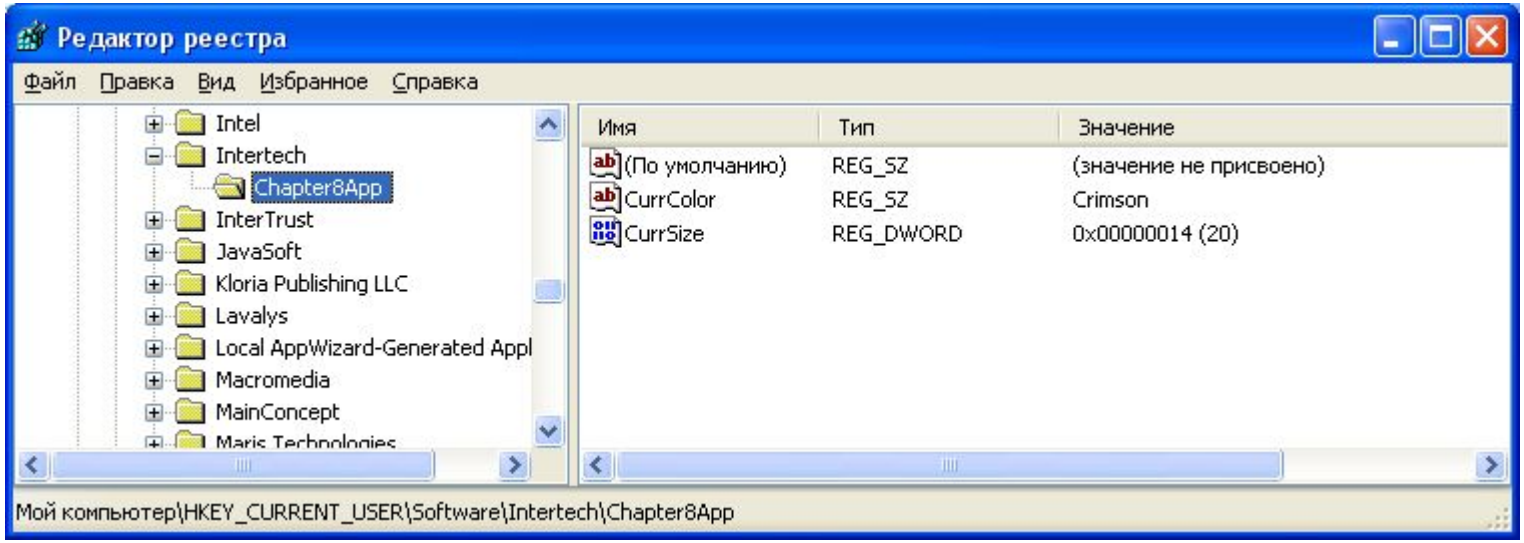


Создание панели инструментов в VS



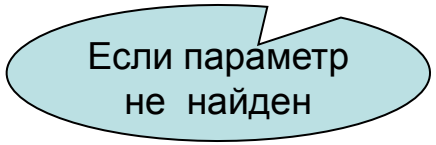
Пример работы с системным реестром

```
// Предположим, что у нас установлены следующие значения переменных:
// Color currColor = Color.MistyRose;
// private int currFontSize = TheFontSize.Normal;

private void FileSave_Clicked(object sender, EventArgs e)
{
    // Сохраняем настройки пользователя в реестре
    RegistryKey regKey = Registry.CurrentUser;
    regKey = regKey.CreateSubKey("Software\\Intertech\\Chapter8App");
    regKey.SetValue("CurrSize", currFontSize);
    regKey.SetValue("CurrColor", currColor.Name);
}
```



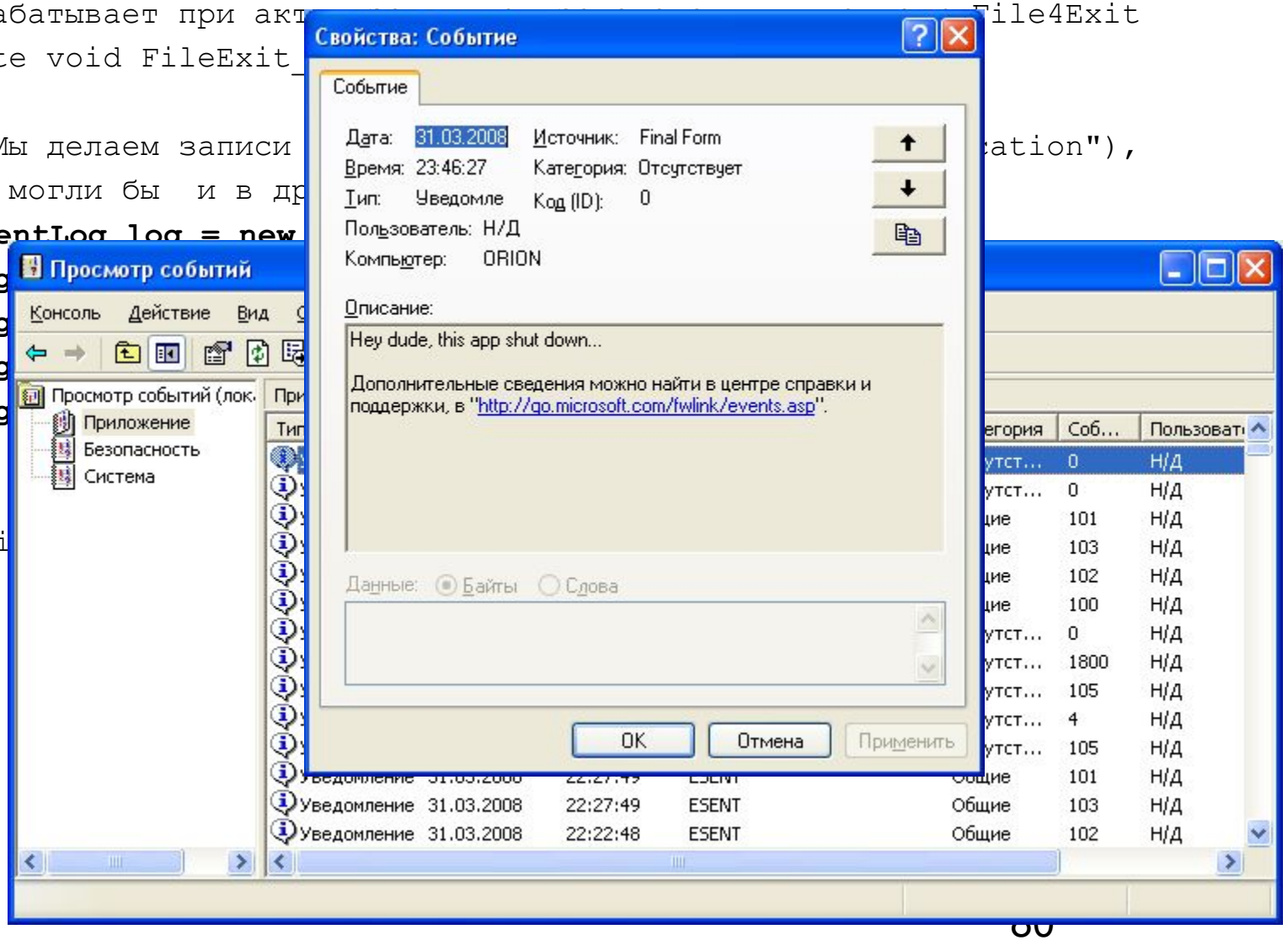
Пример работы с системным реестром

```
public MainForm()
{
    // Открываем параметр реестра
    RegistryKey regKey = Registry.CurrentUser;
    regKey = regKey.CreateSubKey("Software\\Intertech\\Chapter8App");

    // Считываем значения и присваиваем их соответствующим переменным
    currFontSize = (int)regKey.GetValue("CurrSize", currFontSize);
    string c = (string)regKey.GetValue("CurrColor", currColor.Name);
    currColor = Color.FromName(c);
    BackColor = currColor;

     ...
}
```



Запись в журнал событий

```
// Срабатывает при акт                                    File4Exit
private void FileExit_
{
  // Мы делаем записи                                     cation"),
  //  могли бы  и в др
    EventLog log = new
    log
    log
    log
    log

    //
    thi
}
```



Чтение журнала событий

```
private void FileExit_Clicked(object sender, EventArgs e)
{
     ...
// Отображаем первы
for(int i = 0; i <
{
try
{
MessageBox.Show("Me                                                    + "Box name: "
     + log.Entries[i
     log.Entries[i].
     log.Entries[i].
}
catch {}
}
}
```