Бесплатная электронная книга

УЧУСЬ

winforms

Free unaffiliated eBook created from
Stack Overflow contributors.

#winforms

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: winforms

It is an unofficial and free winforms ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official winforms.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с winforms

замечания

Windows Forms («WinForms») - это библиотека классов GUI, включенная в .NET Framework. Это сложная объектно-ориентированная оболочка вокруг Win32 API , позволяющая разрабатывать настольные и мобильные приложения Windows, ориентированные на .NET Framework .

WinForms в первую очередь управляется событиями . Приложение состоит из нескольких *форм* (отображается как окна на экране), которые содержат *элементы управления* (метки, кнопки, текстовые поля, списки и т. Д.), С которыми пользователь взаимодействует напрямую. В ответ на взаимодействие с пользователем эти элементы управления вызывают события, которые могут быть выполнены программой для выполнения задач.

Как и в Windows, все в WinForms - это элемент управления, который сам по себе является типом окна. Базовый класс Control предоставляет базовые функции, включая свойства для установки текста, местоположения, размера и цвета, а также общий набор событий, которые можно обрабатывать. Все элементы управления получают из класса Control, добавляя дополнительные функции. Некоторые элементы управления могут содержать другие элементы управления, например, для повторного использования (`Form` , `UserControl`) или макета (`TableLayoutPanel` , `FlowLayoutPanel`).

WinForms поддерживается с оригинальной версии .NET Framework (v1.0) и по-прежнему доступна в современных версиях (v4.5). Тем не менее, он больше не находится в активной разработке, и новые функции не добавляются. Согласно 9 разработчикам Microsoft на конференции Build 2014 :

> Windows Forms продолжает поддерживаться, но в режиме обслуживания. Они будут исправлять ошибки по мере их обнаружения, но новая функциональность отключена от таблицы.

Кросс-платформенная библиотека с открытым исходным кодом Mono обеспечивает базовую реализацию Windows Forms, поддерживая все функции, реализованные Microsoft в .NET 2.0. Однако WinForms не активно разрабатывается в Mono, и полная реализация считается невозможной, учитывая, как неразрывно связана инфраструктура с родным Windows API (который недоступен на других платформах).

Смотрите также:

- Документация Windows Forms на MSDN

Examples

Создание простого приложения WinForms с помощью Visual Studio

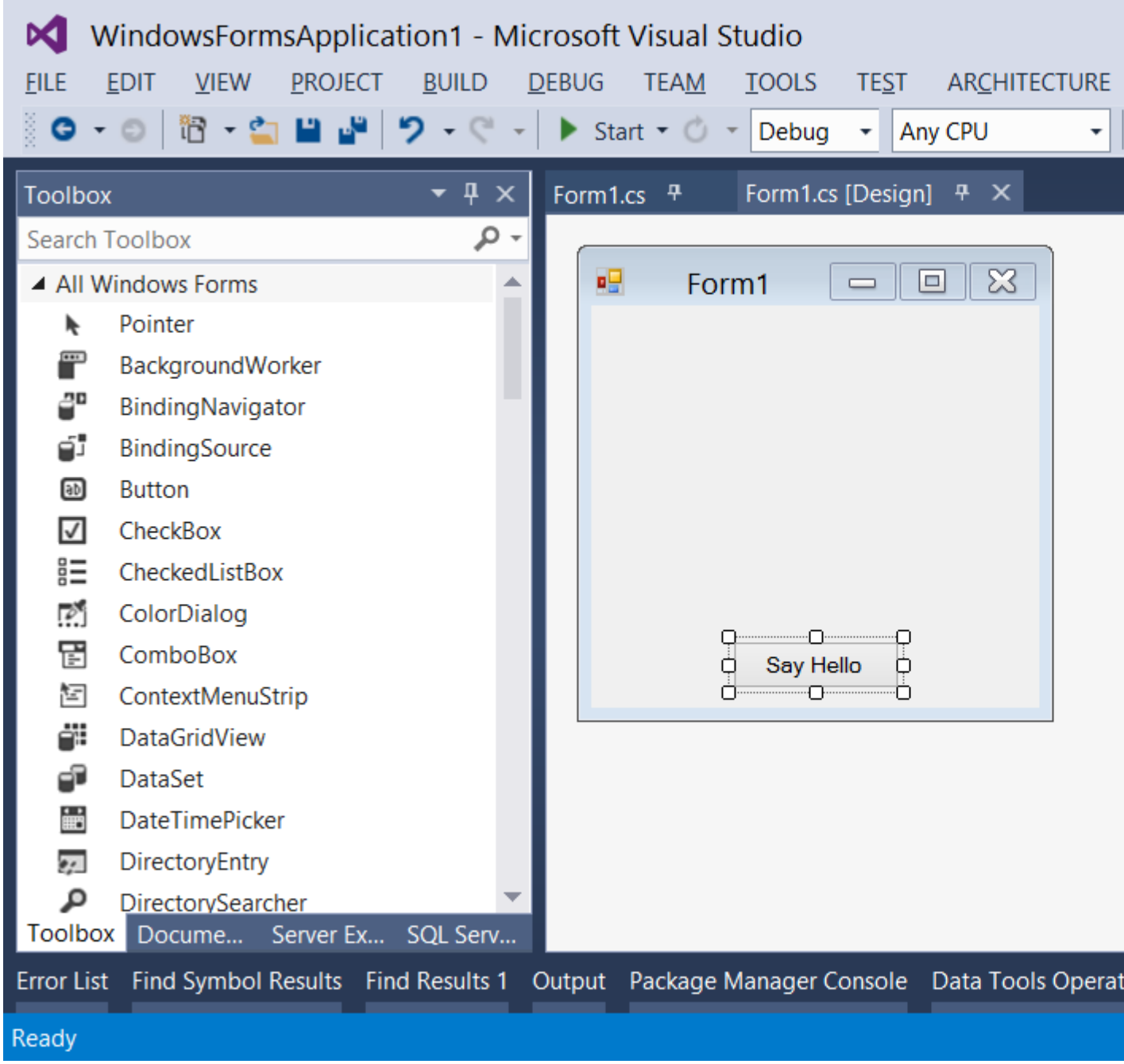
В этом примере показано, как создать проект приложения Windows Forms в Visual Studio.

Создание проекта Windows Forms

1. Запустите Visual Studio.
2. В меню « **Файл»** выберите « **Создать»** , а затем « **Проект»** . Откроется диалоговое окно « **Новый проект** ».
3. На панели « **Установленные шаблоны»** выберите «Visual C #» или «Visual Basic».
4. Над средней областью вы можете выбрать целевую структуру из раскрывающегося списка.
5. В средней области выберите шаблон **приложения Windows Forms** .
6. В текстовом поле « **Имя»** введите имя для проекта.
7. В текстовом поле « **Расположение»** выберите папку для сохранения проекта.
8. Нажмите « **OK»** .
9. Разработчик Windows Forms Designer открывает и отображает **Form1** проекта.

Добавить элементы управления в форму

1. В палитре **Toolbox** перетащите элемент управления **Button** в форму.
2. Нажмите кнопку, чтобы выбрать его. В окне «Свойства» установите для свойства « `Text` **Приветствие»** .



Введите код

1. Дважды нажмите кнопку, чтобы добавить обработчик событий для события `Click` . Редактор кода откроется с точкой вставки, помещенной в функцию обработчика события.

2. Введите следующий код:

   C #

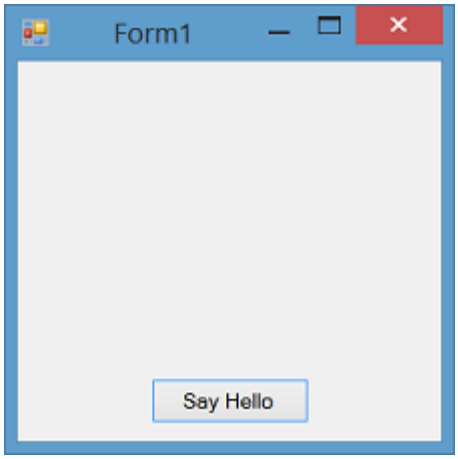
   ```
   MessageBox.Show("Hello, World!");
   ```

   VB.NET

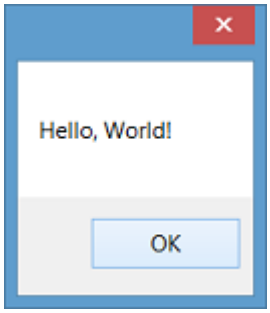
```
MessageBox.Show("Hello, World!")
```

Запуск и тестирование

1. Нажмите `F5`, чтобы запустить приложение.



2. Когда ваше приложение запущено, нажмите кнопку, чтобы увидеть «Hello, World!». сообщение.



3. Закройте форму, чтобы вернуться в Visual Studio.

Создание простого приложения C # WinForms с использованием текстового редактора

1. Откройте текстовый редактор (например, Блокнот) и введите код ниже:

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace SampleApp
{
    public class MainForm : Form
    {
        private Button btnHello;

        // The form's constructor: this initializes the form and its controls.
        public MainForm()
```

```
        {
            // Set the form's caption, which will appear in the title bar.
            this.Text = "MainForm";

            // Create a button control and set its properties.
            btnHello = new Button();
            btnHello.Location = new Point(89, 12);
            btnHello.Name = "btnHello";
            btnHello.Size = new Size(105, 30);
            btnHello.Text = "Say Hello";

            // Wire up an event handler to the button's "Click" event
            // (see the code in the btnHello_Click function below).
            btnHello.Click += new EventHandler(btnHello_Click);

            // Add the button to the form's control collection,
            // so that it will appear on the form.
            this.Controls.Add(btnHello);
        }

        // When the button is clicked, display a message.
        private void btnHello_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Hello, World!");
        }

        // This is the main entry point for the application.
        // All C# applications have one and only one of these methods.
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new MainForm());
        }
    }
}
```

2. Сохраните файл на пути, к которому вы читали / записывали доступ. `X:\MainForm.cs` является имя файла после класса, который он содержит, например, `X:\MainForm.cs` .

3. Запустите компилятор C # из командной строки, передав путь к файлу кода в качестве аргумента:

```
%WINDIR%\Microsoft.NET\Framework64\v4.0.30319\csc.exe /target:winexe "X:\MainForm.cs"
```

   Примечание. Чтобы использовать версию компилятора C # для других версий платформы .NET, посмотрите на путь, `%WINDIR%\Microsoft.NET` и соответствующим образом измените приведенный выше пример. Дополнительные сведения о компиляции приложений C # см. В разделе Компиляция и запуск вашей первой программы на C # .

4. После завершения компиляции приложение `MainForm.exe` будет создано в том же каталоге, что и ваш файл кода. Вы можете запустить это приложение либо из командной строки, либо путем двойного щелчка по нему в проводнике.

Создание простого приложения WinForms VB.NET с использованием текстового редактора

1. Откройте текстовый редактор (например, Блокнот) и введите код ниже:

```
Imports System.ComponentModel
Imports System.Drawing
Imports System.Windows.Forms

Namespace SampleApp
    Public Class MainForm : Inherits Form
        Private btnHello As Button

        ' The form's constructor: this initializes the form and its controls.
        Public Sub New()
            ' Set the form's caption, which will appear in the title bar.
            Me.Text = "MainForm"

            ' Create a button control and set its properties.
            btnHello = New Button()
            btnHello.Location = New Point(89, 12)
            btnHello.Name = "btnHello"
            btnHello.Size = New Size(105, 30)
            btnHello.Text = "Say Hello"

            ' Wire up an event handler to the button's "Click" event
            ' (see the code in the btnHello_Click function below).
            AddHandler btnHello.Click, New EventHandler(AddressOf btnHello_Click)

            ' Add the button to the form's control collection,
            ' so that it will appear on the form.
            Me.Controls.Add(btnHello)
        End Sub

        ' When the button is clicked, display a message.
        Private Sub btnHello_Click(sender As Object, e As EventArgs)
            MessageBox.Show("Hello, World!")
        End Sub

        ' This is the main entry point for the application.
        ' All VB.NET applications have one and only one of these methods.
        <STAThread> _
        Public Shared Sub Main()
            Application.EnableVisualStyles()
            Application.Run(New MainForm())
        End Sub
    End Class
End Namespace
```

2. Сохраните файл на пути, к которому вы читали / записывали доступ. `X:\MainForm.vb` является имя файла после класса, который он содержит, например, `X:\MainForm.vb` .

3. Запустите компилятор VB.NET из командной строки, передав путь к файлу кода в качестве аргумента:

```
%WINDIR%\Microsoft.NET\Framework64\v4.0.30319\vbc.exe /target:winexe "X:\MainForm.vb"
```

Примечание. Чтобы использовать версию компилятора VB.NET для других версий платформы .NET, просмотрите путь `%WINDIR%\Microsoft.NET` и соответствующим образом измените приведенный выше пример. Дополнительные сведения о компиляции приложений VB.NET см. В разделе Hello World .

4. После завершения компиляции приложение `MainForm.exe` будет создано в том же каталоге, что и ваш файл кода. Вы можете запустить это приложение либо из командной строки, либо путем двойного щелчка по нему в проводнике.

Прочитайте Начало работы с winforms онлайн: https://riptutorial.com/ru/winforms/topic/1018/начало-работы-с-winforms

глава 2: Databinding

параметры

аргументация	Описание
Имя свойства	Имя свойства управления для привязки.
источник данных	Объект, представляющий источник данных.
DataMember	Свойство или список для привязки.
formattingEnabled	Определяет, должны ли отображаемые данные форматироваться.
UpdateMode	Источник данных обновляется, когда свойство управления проверяется (по умолчанию) или сразу же после изменения свойства
nullValue	Когда источник данных имеет это значение, свойство bound имеет значение DBNull.
FormatString	Один или несколько символов спецификатора формата, которые указывают, как должно отображаться значение
formatInfo	Реализация IFormatProvider для переопределения поведения форматирования по умолчанию.

замечания

См. Https://msdn.microsoft.com/en-us/library/ef2xyb33.aspx. Связывание данных работает только со свойствами, а не с полями!

Examples

Управление привязкой к объектам данных

Каждый элемент управления имеет свойство `DataBindings` которое представляет собой список объектов `System.Windows.Forms.Binding` . Метод Add () - имеет некоторые перегрузки, которые позволяют легко привязываться к свойству объекта:

```
textBox.DataBindings.Add( "Text", dataObj, "MyProperty" );
```

Обратите внимание, что привязка в основном означает подписку на другие переменные. Приведенный выше код подписывается на changeevent of dataObj.MyProperty и адаптирует textBox.Text, когда он изменяется. И наоборот, он подписывается на textBox.TextChanged и адаптирует dataObj.MyPropery, когда он изменяется.

Прочитайте Databinding онлайн: https://riptutorial.com/ru/winforms/topic/7362/databinding

глава 3: Наследование элементов управления

замечания

Элементы управления производятся точно так же, как и другие классы. Единственное, на что нужно обратить внимание, это переопределить события: обычно рекомендуется убедиться, что вы вызываете обработчик базового события после своего собственного. Мое собственное правило: если есть сомнения, вызовите базовое событие.

Examples

Широкие настройки приложения

Быстрое чтение большинства сайтов разработчиков покажет, что WinForms считается ниже WPF. Одной из наиболее часто упоминаемых причин является предполагаемая трудность внесения широкого применения приложений в «внешний вид» всего приложения.

На самом деле удивительно просто создать приложение в WinForms, которое легко настраивается как во время разработки, так и во время выполнения, если вы просто избегаете использования стандартных элементов управления и извлекаете из них свои собственные.

Возьмите TextBox в качестве примера. Трудно представить себе приложение Windows, которое не требует использования TextBox на той или иной стадии. Поэтому наличие вашего собственного TextBox всегда будет иметь смысл. Возьмем следующий пример:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace StackOverflowDocumentation
{
    public class SOTextBox : TextBox
    {
        public SOTextBox() : base()
        {
            base.BackColor = SOUserPreferences.BackColor;
            base.ForeColor = SOUserPreferences.ForeColor;
        }
        protected override void OnEnter(EventArgs e)
        {
            base.BackColor = SOUserPreferences.FocusColor;
```

```
            base.OnEnter(e);
        }
        protected override void OnLeave(EventArgs e)
        {
            base.BackColor = SOUserPreferences.BackColor;
            base.OnLeave(e);
        }
    }
}
```

Одна из вещей, которые пользователи находят наиболее полезными в форме ввода данных, со многими полями ввода, заключается в том, чтобы иметь фоновый цвет окна с изменением фокуса. Видимо, это легче увидеть, чем стандартный мигающий вертикальный курсор. Вышеприведенный код предоставляет TextBox, который делает именно это.

В этом процессе он использует статические свойства статического класса. Ниже приводится выдержка из моего:

```
using System;
using System.Threading;
using Microsoft.Win32;
using System.Globalization;
using System.Data;
using System.Drawing;

namespace StackOverflowDocumentation
{
    public class SOUserPreferences
    {
        private static string language;
        private static string logPath;
        private static int formBackCol;
        private static int formForeCol;
        private static int backCol;
        private static int foreCol;
        private static int focusCol;

        static SOUserPreferences()
        {
            try
            {
                RegistryKey HKCU = Registry.CurrentUser;
                RegistryKey kSOPrefs = HKCU.OpenSubKey("SOPrefs");
                if (kSOPrefs != null)
                {
                    language = kSOPrefs.GetValue("Language", "EN").ToString();
                    logPath = kSOPrefs.GetValue("LogPath", "c:\\windows\\logs\\").ToString();
                    formForeCol = int.Parse(kSOPrefs.GetValue("FormForeColor", "-
2147483630").ToString());
                    formBackCol = int.Parse(kSOPrefs.GetValue("FormBackColor", "-
2147483633").ToString());
                    foreCol = int.Parse(kSOPrefs.GetValue("ForeColor", "-
2147483640").ToString());
                    backCol = int.Parse(kSOPrefs.GetValue("BackColor", "-
2147483643").ToString());
                    focusCol = int.Parse(kSOPrefs.GetValue("FocusColor", "-
2147483643").ToString());
                }
```

```
            else
            {
                language = "EN";
                logPath = "c:\\windows\\logs\\";
                formForeCol = -2147483630;
                formBackCol = -2147483633;
                foreCol = -2147483640;
                backCol = -2147483643;
                focusCol = -2147483643;
            }
        }
        catch (Exception ex)
        {
            //handle exception here;
        }
    }

    public static string Language
    {
        get
        {
            return language;
        }
        set
        {
            language = value;
        }
    }

    public static string LogPath
    {
        get
        {
            return logPath;
        }
        set
        {
            logPath = value;
        }
    }

    public static Color FormBackColor
    {
        get
        {
            return ColorTranslator.FromOle(formBackCol);
        }
        set
        {
            formBackCol = ColorTranslator.ToOle(value);
        }
    }

    public static Color FormForeColor
    {
        get
        {
            return ColorTranslator.FromOle(formForeCol);
        }
        set
        {
```

```
                formForeCol = ColorTranslator.ToOle(value);
            }
        }

        public static Color BackColor
        {
            get
            {
                return ColorTranslator.FromOle(backCol);
            }
            set
            {
                backCol = ColorTranslator.ToOle(value);
            }
        }

        public static Color ForeColor
        {
            get
            {
                return ColorTranslator.FromOle(foreCol);
            }
            set
            {
                foreCol = ColorTranslator.ToOle(value);
            }
        }

        public static Color FocusColor
        {
            get
            {
                return ColorTranslator.FromOle(focusCol);
            }
            set
            {
                focusCol = ColorTranslator.ToOle(value);
            }
        }
    }
}
```

Этот класс использует реестр Windows для сохранения свойств, но вы можете использовать базу данных или файл настроек, если хотите. Преимущество использования статического класса таким образом заключается в том, что широкомасштабные изменения приложения могут производиться не только во время разработки, но и во время выполнения пользователем. Я всегда включаю форму в свои приложения, позволяющую пользователю изменять предпочтительные значения. Функция сохранения не только сохраняет реестр (или базу данных и т. Д.), Но также во время выполнения изменяет пропозиции в статическом классе. Обратите внимание, что статические свойства статического класса не являются постоянными; в этом смысле их можно рассматривать как прикладные широкие переменные. Это означает, что любая форма, открытая после сохранения изменений, сразу же будет зависеть от любых сохраненных изменений.

Вы легко сможете подумать о других свойствах приложения, которые вы хотели бы

настроить таким же образом. Шрифты - еще один очень хороший пример.

NumberBox

Часто вам нужно иметь поле ввода, которое принимает только числа. Опять же, исходя из стандартных элементов управления, это легко достигается, например:

```
using System;
using System.Windows.Forms;
using System.Globalization;

namespace StackOverflowDocumentation
{
    public class SONumberBox : SOTextBox
    {
        private int decPlaces;
        private int extraDecPlaces;
        private bool perCent;
        private bool useThouSep = true;
        private string decSep = ".";
        private string thouSep = ",";
        private double numVal;

        public SONumberBox() : base()

    {
    }

    public bool PerCent
    {
        get
        {
            return perCent;
        }
        set
        {
            perCent = value;
        }
    }

    public double Value
    {
        get
        {
            return numVal;
        }
        set
        {
            numVal = value;
            if (perCent)
            {
                double test = numVal * 100.0;
                this.Text = FormatNumber(test) + "%";
            }
            else
            {
                this.Text = FormatNumber(value);
            }
        }
```

```
    }
    public bool UseThousandSeparator
    {
        get
        {
            return useThouSep;
        }
        set
        {
            useThouSep = value;
        }
    }
    public int DecimalPlaces
    {
        get
        {
            return decPlaces;
        }
        set
        {
            decPlaces = value;
        }
    }
    public int ExtraDecimalPlaces
    {
        get
        {
            return extraDecPlaces;
        }
        set
        {
            extraDecPlaces = value;
        }
    }
    protected override void OnTextChanged(EventArgs e)
    {
        string newVal = this.Text;
        int len = newVal.Length;
        if (len == 0)
        {
            return;
        }
        bool neg = false;
        if (len > 1)
        {
            if (newVal.Substring(0, 1) == "-")
            {
                newVal = newVal.Substring(1, len - 1);
                len = newVal.Length;
                neg = true;
            }
        }
        double val = 1.0;
        string endChar = newVal.Substring(newVal.Length - 1);
        switch (endChar)
        {
            case "M":
            case "m":
                if (len > 1)
                {
                    val = double.Parse(newVal.Substring(0, len - 1)) * 1000000.0;
```

```
            }
            else
            {
                val *= 1000000.0;
            }
            if (neg)
            {
                val = -val;
            }
            this.Text = FormatNumber(val);
            break;
        case "T":
        case "t":
            if (len > 1)
            {
                val = double.Parse(newVal.Substring(0, len - 1)) * 1000.0;
            }
            else
            {
                val *= 1000.0;
            }
            if (neg)
            {
                val = -val;
            }
            this.Text = FormatNumber(val);
            break;
    }

    base.OnTextChanged(e);
}
protected override void OnKeyPress(KeyPressEventArgs e)
{
    bool handled = false;
    switch (e.KeyChar)
    {
        case '-':
            if (this.Text.Length == 0)
            {
                break;
            }
            else if (this.SelectionStart == 0)
            {
                //negative being inserted first
                break;
            }
            else
            {
                handled = true;
                break;
            }
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
        case '8':
        case '9':
        case '0':
```

```
            case (char)Keys.Back:
                break;
            case 'M':
            case 'm':
            case 'T':
            case 't':
            case '%':
                //check last pos
                int l = this.Text.Length;
                int sT = this.SelectionStart;
                int sL = this.SelectionLength;
                if ((sT + sL) != l)
                {
                    handled = true;
                }
                break;
            default:
                string thisChar = e.KeyChar.ToString();
                if (thisChar == decSep)
                {
                    char[] txt = this.Text.ToCharArray();
                    for (int i = 0; i < txt.Length; i++)
                    {
                        if (txt[i].ToString() == decSep)
                        {
                            handled = true;
                            break;
                        }
                    }
                    break;
                }
                else if (thisChar != thouSep)
                {
                    handled = true;
                }
                break;
        }

        if (!handled)
        {
            base.OnKeyPress(e);
        }
        else
        {
            e.Handled = true;
        }

    }
    protected override void OnLeave(EventArgs e)
    {
        string tmp = this.Text;
        if (tmp == "")
        {
            tmp = "0";
            numVal = NumberLostFocus(ref tmp);
            this.Text = tmp;
        }
        if (tmp.Substring(tmp.Length - 1) == "%")
        {
            tmp = tmp.Substring(0, tmp.Length - 1);
            numVal = 0.0;
```

```
            numVal = NumberLostFocus(ref tmp) / 100.0;
            double test = numVal * 100.0;
            this.Text = FormatNumber(test) + "%";
        }
        else if (perCent)
        {
            numVal = NumberLostFocus(ref tmp);
            double test = numVal * 100.0;
            this.Text = FormatNumber(test) + "%";
        }
        else
        {
            numVal = NumberLostFocus(ref tmp);
            this.Text = tmp;
        }
        base.OnLeave(e);
    }
    private string FormatNumber(double amount)
    {
        NumberFormatInfo nF = new NumberFormatInfo();
        nF.NumberDecimalSeparator = decSep;
        nF.NumberGroupSeparator = thouSep;

        string decFormat;
        if (useThouSep)
        {
            decFormat = "#,##0";
        }
        else
        {
            decFormat = "#0";
        }
        if (decPlaces > 0)
        {
            decFormat += ".";
            for (int i = 0; i < decPlaces; i++)
            {
                decFormat += "0";
            }
            if (extraDecPlaces > 0)
            {
                for (int i = 0; i < extraDecPlaces; i++)
                {
                    decFormat += "#";
                }
            }
        }
        else if (extraDecPlaces > 0)
        {
            decFormat += ".";
            for (int i = 0; i < extraDecPlaces; i++)
            {
                decFormat += "#";
            }
        }
        return (amount.ToString(decFormat, nF));
    }
    private double NumberLostFocus(ref string amountBox)
    {
        if (amountBox.Substring(0, 1) == decSep)
            amountBox = "0" + amountBox;
```

```
NumberFormatInfo nF = new NumberFormatInfo();
nF.NumberDecimalSeparator = decSep;
nF.NumberGroupSeparator = thouSep;

try
{
    double d = 0.0;
    int l = amountBox.Length;
    if (l > 0)
    {

        char[] c = amountBox.ToCharArray();
        char endChar = c[l - 1];

        switch (endChar)
        {
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                {
                    stripNonNumerics(ref amountBox);
                    d = Double.Parse(amountBox, nF);
                    break;
                }
            case 'm':
            case 'M':
                {
                    if (amountBox.Length == 1)
                        d = 1000000.0;
                    else
                    {
                        string s = amountBox.Substring(0, l - 1);
                        stripNonNumerics(ref s);
                        d = Double.Parse(s, nF) * 1000000.0;
                    }
                    break;

                }
            case 't':
            case 'T':
                {
                    if (amountBox.Length == 1)
                        d = 1000.0;
                    else
                    {
                        string s = amountBox.Substring(0, l - 1);
                        stripNonNumerics(ref s);
                        d = Double.Parse(s, nF) * 1000.0;
                    }
                    break;
                }
            default:
                {
                    //remove offending char
```

```
                        string s = amountBox.Substring(0, l - 1);
                        if (s.Length > 0)
                        {
                            stripNonNumerics(ref s);
                            d = Double.Parse(s, nF);
                        }
                        else
                            d = 0.0;
                        break;
                    }
            }
        }
        amountBox = FormatNumber(d);
        return (d);
    }
    catch (Exception e)
    {
        //handle exception here;
        return 0.0;
    }
}
private void stripNonNumerics(ref string amountBox)
{
    bool dSFound = false;
    char[] tmp = decSep.ToCharArray();
    char dS = tmp[0];
    string cleanNum = "";
    int l = amountBox.Length;
    if (l > 0)
    {
        char[] c = amountBox.ToCharArray();
        for (int i = 0; i < l; i++)
        {
            char b = c[i];
            switch (b)
            {
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                case '8':
                case '9':
                    cleanNum += b;
                    break;
                case '-':
                    if (i == 0)
                        cleanNum += b;
                    break;
                default:
                    if ((b == dS) && (!dSFound))
                    {
                        dSFound = true;
                        cleanNum += b;
                    }
                    break;
            }
        }
```

```
            }
            amountBox = cleanNum;
        }
    }
```

Помимо ограничения ввода чисел, этот класс имеет несколько специальных функций. Он предоставляет свойство Value для представления двойного значения числа, оно форматирует текст, необязательно с тысячами разделителей, и обеспечивает короткий доступ к большим числам: 10M расширяется в отпуск до 10 000 000,00 (количество десятичных знаков, являющихся собственностью). Для краткости десятичный и тысячный разделители были жестко закодированы. В производственной системе это также пользовательские настройки.

Прочитайте Наследование элементов управления онлайн:
https://riptutorial.com/ru/winforms/topic/6476/наследование-элементов-управления

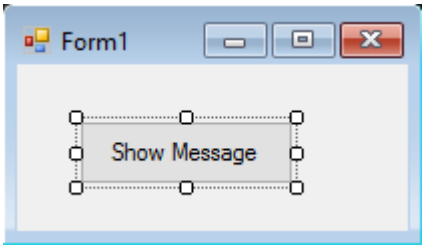
глава 4: Основные элементы управления

Examples

кнопка

Кнопки являются одним из самых простых элементов управления и в основном используются для выполнения некоторого кода, когда пользователь хочет.

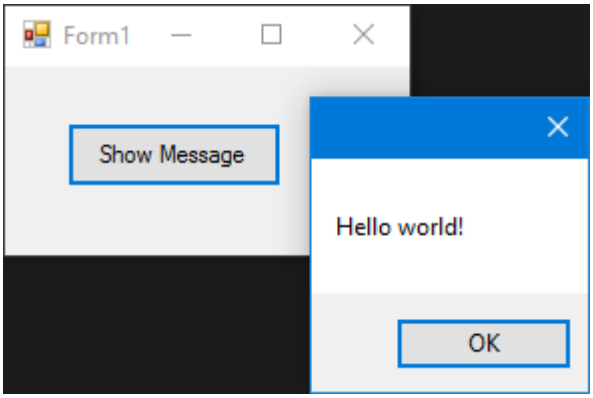
Здесь у нас действительно простой случай, показывается кнопка «Сообщение» при нажатии кнопки. Мы добавляем кнопку в форму, назовите ее `cmdShowMessage` используется в коде, чтобы идентифицировать объект и установить текст кнопок для отображения сообщения.



Нам просто нужно дважды щелкнуть кнопку на визуальном дизайнере, а Visual Studio создаст код для события click. Теперь нам просто нужно добавить код для MessageBox:

```
private void cmdShowMessage_Click(object sender, EventArgs e)
{
  MessageBox.Show("Hello world!");
}
```

Если мы запустим программу сейчас и нажмите кнопку, мы увидим сообщение:

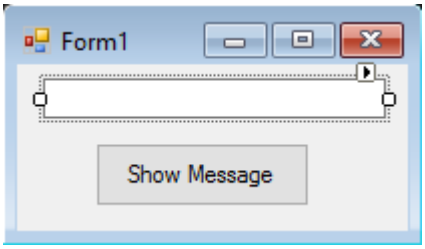


Текстовое окно

Текстовые поля позволяют пользователю вводить данные в программу.

Мы собираемся изменить форму и добавить текстовое поле, чтобы окно сообщения

отображало нам сообщение, которое хочет пользователь. Теперь наша форма выглядит так:

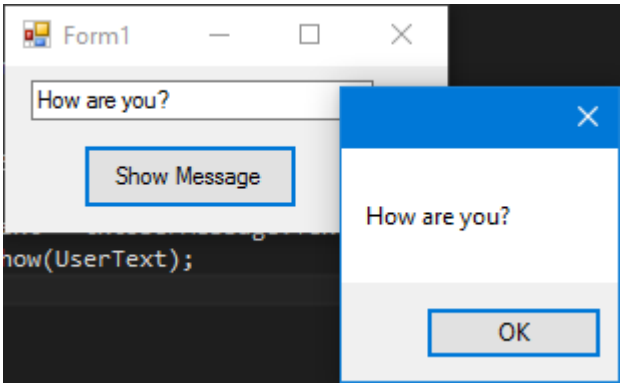


Затем измените событие нажатия кнопки, чтобы использовать текст текстового поля:

```
private void cmdShowMessage_Click(object sender, EventArgs e)
{
  string UserText = txtUserMessage.Text;
  MessageBox.Show(UserText);
}
```

Как вы можете видеть , что мы используем `.Text` свойство Textbox , что является текст , содержащийся в texbox.

Если мы запустим программу, мы сможем писать в текстовом поле. Когда мы нажимаем кнопку, MessageBox покажет текст, который мы написали:

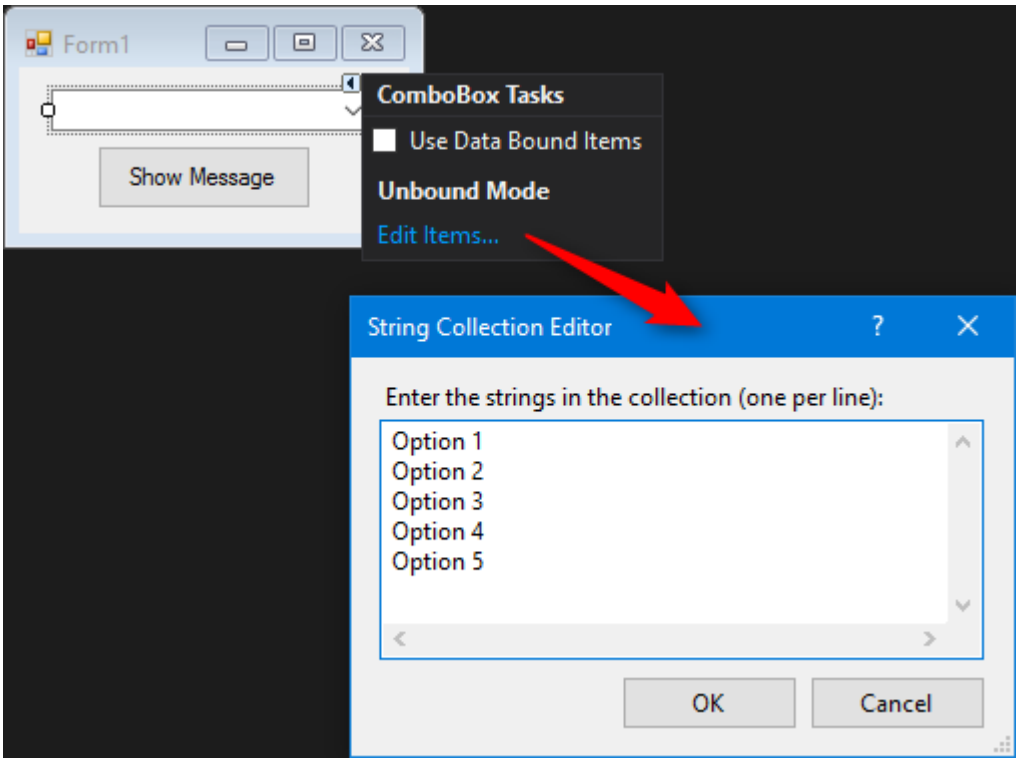


Поле со списком

Комбобокс позволяет пользователю выбирать один из вариантов, предоставляемых разработчиком.

Мы собираемся изменить форму и добавить combobox, чтобы в окне сообщений отображалось сообщение, которое пользователь хочет из списка, который мы предоставим.

После добавления комбо в форму теперь добавим список опций в комбо. Для этого нам нужно изменить свойство `Items` :

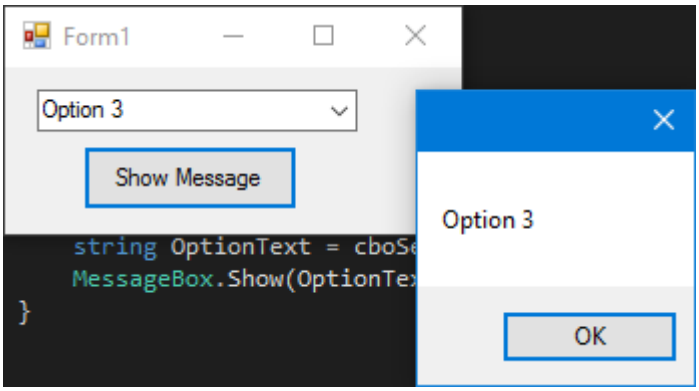


Теперь нам нужно изменить код события click:

```
private void cmdShowMessage_Click(object sender, EventArgs e)
{
  string OptionText = cboSelectOption.SelectedItem.ToString();
  MessageBox.Show(OptionText);
}
```

Как вы видите, мы используем свойство `SelectedItem` , оно содержит объект выбранной опции. Поскольку нам нужна строка для показа, и компилятор не знает, является ли объект строкой или нет, нам нужно использовать метод `ToString()` .

Если мы запустим программу, мы сможем выбрать тот вариант, который мы предпочитаем, и когда мы нажмем кнопку, появится окно сообщения:



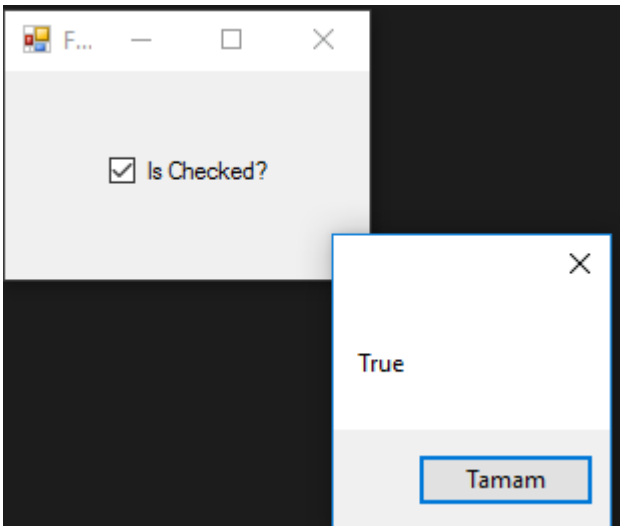
Чтобы получать уведомление, когда пользователь выбирает элемент из поля со списком, используйте событие `SelectionChangeCommitted` . Мы могли бы использовать событие `SelectedIndexChanged` , но это также возникает, когда мы программным образом меняем элемент select в combobox.

CheckBox

Флажок - это элемент управления, который позволяет пользователю получать `boolean` значения от пользователя по специальному вопросу типа **«Все в порядке?».** ,

Имеет событие под названием `CheckedChanged` , которое происходит всякий раз, когда свойство `check` изменяется.

Вот CheckBox, на который есть вопрос **«Проверено?».** ,



Мы получили этот `MessageBox` из события `CheckedChanged` ,

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
        {
            bool IsChecked = checkBox1.Checked;
            MessageBox.Show(IsChecked.ToString());
        }
```

Если **CheckBox** `IsChecked` переменная `IsChecked` будет `true` .

Если **CheckBox** не установлен - переменная `IsChecked` будет `false` .

ListBox

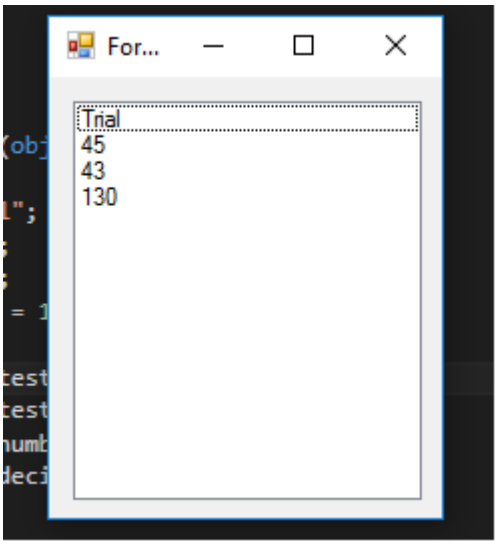
`Listbox` - это элемент управления, который может содержать коллекцию объектов. `Listbox` похож на `Combobox` но в `Combobox` ; Только избранные элементы видны пользователю. В `Listbox` ; все элементы видны пользователю.

Как добавить элементы в ListBox?

```
private void Form3_Load(object sender, EventArgs e)
        {
            string test = "Trial";
            string test2 = "45";
            int numberTest = 43;
            decimal decimalTest = 130;
```

```
            listBox1.Items.Add(test);
            listBox1.Items.Add(test2);
            listBox1.Items.Add(numberTest);
            listBox1.Items.Add(decimalTest);
        }
```

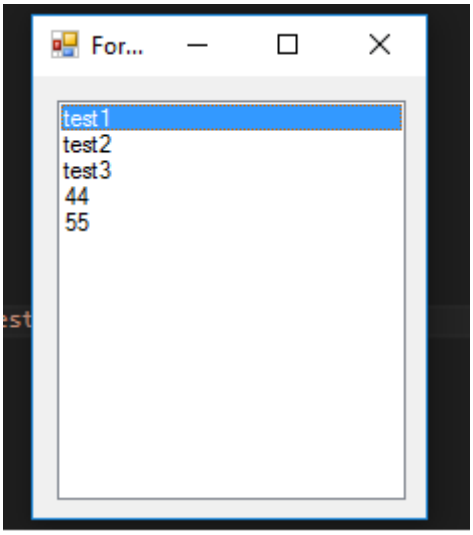
Выход ;



Или могут быть предоставлены `datasources` данных,

```
private void Form3_Load(object sender, EventArgs e)
        {
            List<string> TestList = new List<string> { "test1", "test2", "test3", "44", "55"
};
            listBox1.DataSource = TestList;
        }
```

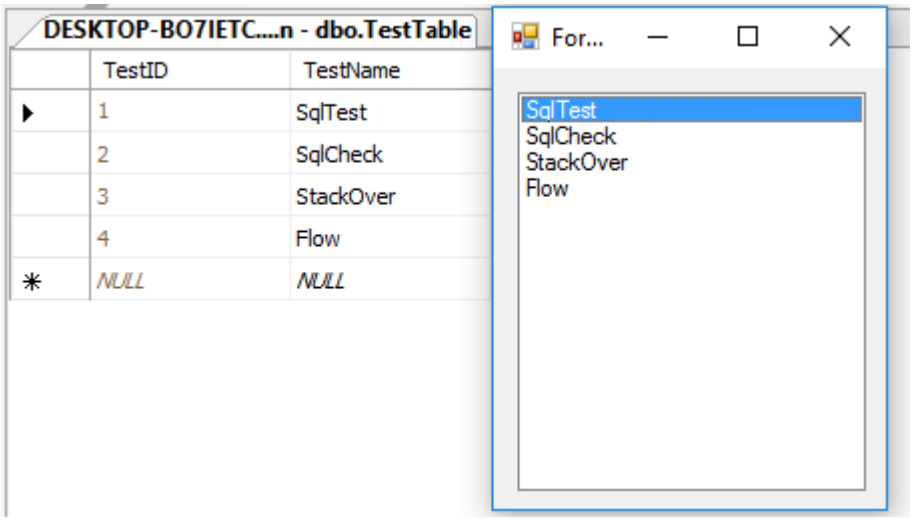
Выход;




```
private void Form3_Load(object sender, EventArgs e)
        {
            SqlConnection Connection = new
```

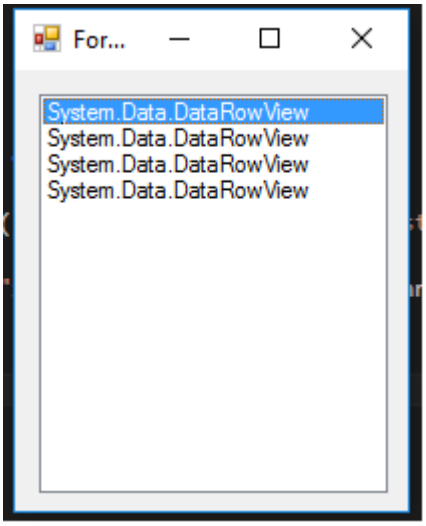
```
SqlConnection("Server=serverName;Database=db;Trusted_Connection=True;"); //Connetion to MS-
SQL(RDBMS)
            Connection.Open(); //Connection open
            SqlDataAdapter Adapter = new SqlDataAdapter("Select * From TestTable",
Connection); // Get all records from TestTable.
            DataTable DT = new DataTable();
            Adapter.Fill(DT); // Fill records to DataTable.
            listBox1.DataSource = DT; // DataTable is the datasource.
            listBox1.ValueMember = "TestID";
            listBox1.DisplayMember= "TestName";
        }
```

Правильный вывод ;



Предоставление внешнего источника данных sql для listbox требуется, `ValueMember` и `DisplayMember`

Если **НЕ** будет выглядеть так,



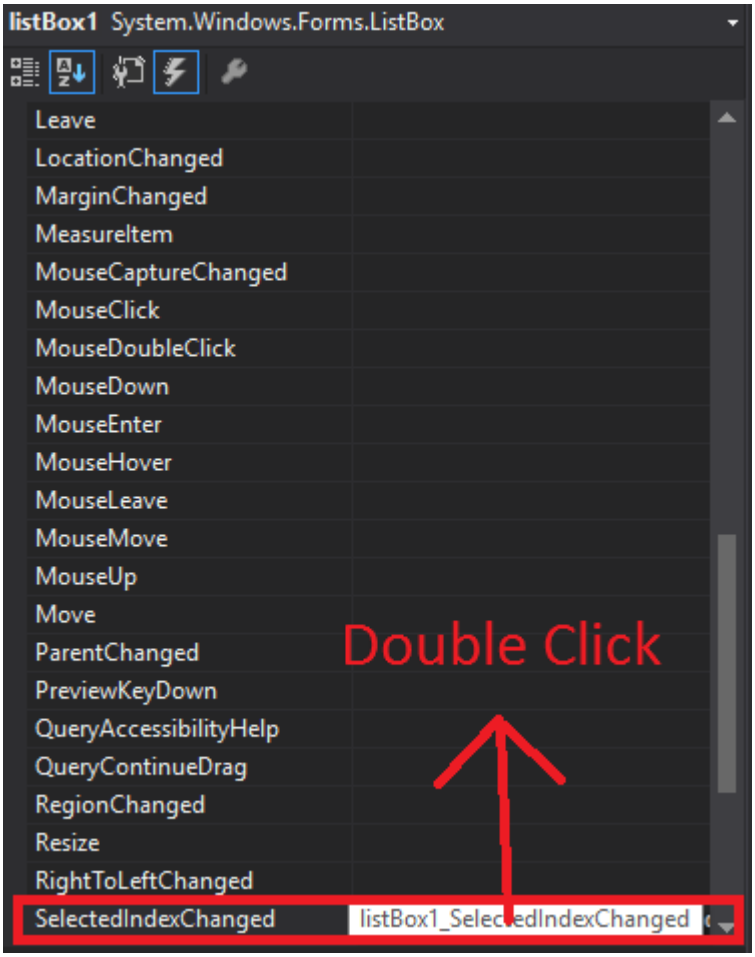
Полезные события;

SelectedIndex_Changed;

Определить список для предоставления источника данных

```
private void Form3_Load(object sender, EventArgs e)
        {
            List<string> DataList = new List<string> {"test1" , "test2" , "test3" , "44" ,
"45" };
            listBox1.DataSource = TestList;
        }
```

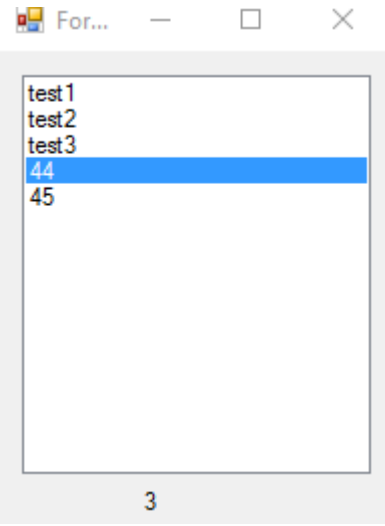
В дизайне формы выберите « `Listbox` и нажмите F4, или с правой стороны щелкните значок подсветки.



Visual studio будет генерировать `listBox1_SelectedIndexChanged` для codebehind.

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            int Index = listBox1.SelectedIndex;
            label1.Text = Index.ToString();
        }
```

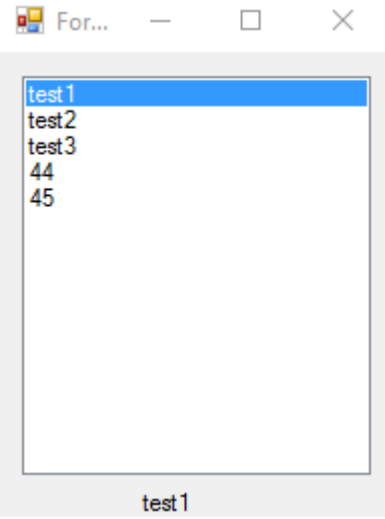
Результат `SelectedIndex_Changed` ; (метка внизу показывает индекс каждого выбранного элемента)



SelectedValue_Changed; (Источник данных такой же, как и сверху, и вы можете сгенерировать это событие, например SelectedIndex_Changed)

```
private void listBox1_SelectedValueChanged(object sender, EventArgs e)
        {
            label1.Text = listBox1.SelectedValue.ToString();
        }
```

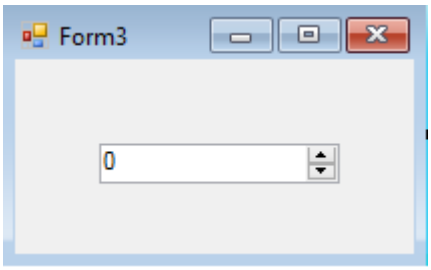
Выход ;



NumericUpDown

NumericUpDown - это элемент управления, который выглядит как TextBox. Этот элемент управления позволяет пользователю отображать / выбирать номер из диапазона. Стрелки вверх и вниз обновляют значение текстового поля.

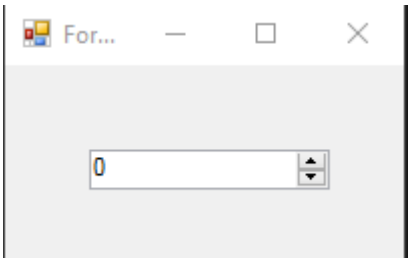
Контроль выглядит;



В диапазоне `Form_Load` можно установить.

```
private void Form3_Load(object sender, EventArgs e)
    {
        numericUpDown1.Maximum = 10;
        numericUpDown1.Minimum = -10;
    }
```

Выход;

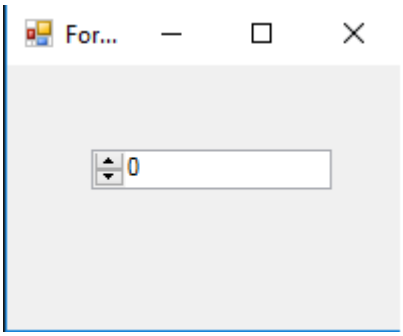


UpDownAlign будет устанавливать положение стрелок;

```
 private void Form3_Load(object sender, EventArgs e)
        {
            numericUpDown1.UpDownAlign = LeftRightAlignment.Left;

        }
```

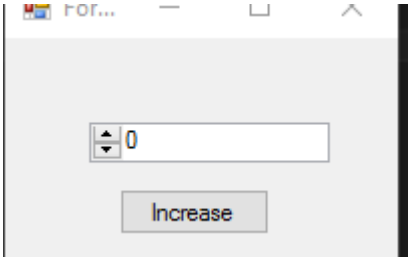
Выход;



`UpButton()` увеличивает количество элементов управления. (можно вызывать из любого места. Я использовал `button` чтобы позвонить).

```
private void button1_Click(object sender, EventArgs e)
        {
            numericUpDown1.UpButton();
        }
```

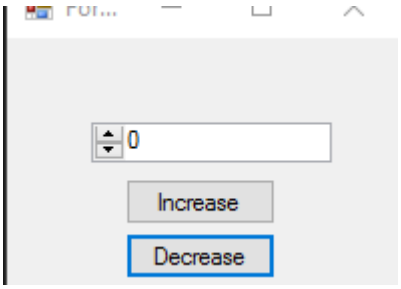
**Выход



`DownButton()` уменьшает количество элементов управления. (можно вызывать из любого места. Я использовал `button` чтобы вызвать ее снова.)

```
 private void button2_Click(object sender, EventArgs e)
        {
            numericUpDown1.DownButton();
        }
```

Выход;



Полезные события

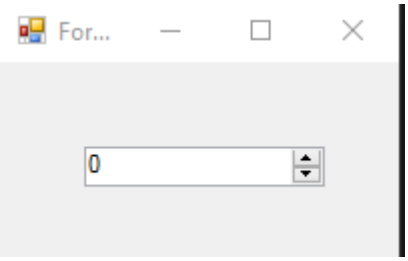
ValueChanged;

Это событие будет работать при нажатии стрелки вверх или вниз.

```
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
        {
            decimal result = numericUpDown1.Value; // it will get the current value
            if (result == numericUpDown1.Maximum) // if value equals Maximum value that we set
in Form_Load.
            {
                label1.Text = result.ToString() + " MAX!"; // it will add "MAX" at the end of
the label
            }
            else if (result == numericUpDown1.Minimum) //  if value equals Minimum value that
we set in Form_Load.
            {
                label1.Text = result.ToString() + " MIN!"; // it will add "MIN" at the end of
the label
            }
            else
            {
                label1.Text = result.ToString(); // If Not Max or Min, it will show only the
```

```
number.
            }
        }
```

Выход ;



Прочитайте Основные элементы управления онлайн:
https://riptutorial.com/ru/winforms/topic/5816/основные-элементы-управления

глава 5: Отображение формы

Вступление

В этом разделе объясняется, как работает механизм WinForms для отображения форм и того, как вы контролируете их жизнь.

Examples

Показать немодальную или модальную форму

После определения структуры вашей формы с помощью дизайнера WinForms вы можете отображать свои формы в коде двумя разными способами.

- Метод - модельная форма

```
Form1 aForm1Instance = new Form1();
aForm1Instance.Show();
```

- Метод - Модальный диалог

```
Form2 aForm2Instance = new Form2();
aForm2Instance.ShowDialog();
```

Эти два метода имеют очень важное различие. Первый метод (немодальный) показывает вашу форму и затем немедленно возвращается, не дожидаясь закрытия только что открытой формы. Таким образом, ваш код продолжает все, что следует за вызовом Show. Второй метод вместо этого (модальный) открывает форму и блокирует любую активность во всем приложении до тех пор, пока вы не закроете форму с помощью кнопки закрытия или с некоторыми кнопками, соответствующим образом настроенными для закрытия формы

Закрытие немодальной формы

Используется немодальная форма (обычно), когда вам нужно показать что-то постоянное рядом с основным экраном приложения (подумайте о легенде или представлении о потоке данных, поступающих асинхронно с устройства или в дочернее окно MDI).
Но немодальная форма представляет собой уникальную задачу, когда вы хотите ее закрыть. Как получить экземпляр и вызвать метод Close в этом экземпляре?

Вы можете сохранить глобальную переменную, ссылающуюся на экземпляр, который вы хотите закрыть.

```
theGlobalInstance.Close();
```

```
theGlobalInstance.Dispose();
theGlobalInstance = null;
```

Но мы также можем использовать коллекцию Application.OpenForms, где механизм формы хранит все экземпляры форм, созданные и открытые.

Вы можете извлечь этот конкретный экземпляр из этой коллекции и вызвать метод Close

```
Form2 toClose = Application.OpenForms.OfType<Form2>().FirstOrDefault();
if(toClose != null)
{
    toClose.Close();
    toClose.Dispose();
}
```

Закрытие модальной формы

Когда форма показана с помощью метода `ShowDialog` , необходимо установить свойство `DialogResult` формы, чтобы закрыть форму. Это свойство может быть установлено с использованием перечисления, которое также называется DialogResult .

Чтобы закрыть форму, вам просто нужно установить свойство `DialogResult` формы (любому значению с помощью `DialogResult.None`) в каком-либо обработчике событий. Когда ваш код выйдет из обработчика событий, механизм WinForm скроет форму, а код, следующий за начальным вызовом метода `ShowDialog` , продолжит выполнение.

```
private cmdClose_Click(object sender, EventArgs e)
{
    this.DialogResult = DialogResult.Cancel;
}
```

Вызывающий код может отображать возвращаемое значение из ShowDialog, чтобы определить, какую кнопку пользователь нажал в форме. При отображении с использованием `ShowDialog()` , форма не удаляться автоматически (так как она была просто скрыта и не закрыта), поэтому очень важно , чтобы использовать , `using` блок , чтобы обеспечить форма расположена.

Ниже приведен пример проверки результата использования встроенного `OpenFileDialog` , проверки результата и доступа к свойствам из диалогового окна перед его удалением.

```
using (var form = new OpenFileDialog())
{
    DialogResult result = form.ShowDialog();
    if (result == DialogResult.OK)
    {
        MessageBox.Show("Selected file is: " + form.FileName);
    }
}
```

Вы также можете установить свойство `DialogResult` на кнопку. Нажатие этой кнопки устанавливает для свойства `DialogResult` в форме значение, связанное с кнопкой. Это позволяет закрыть форму без добавления обработчика событий для установки `DialogResult` в коде.

Например, если вы добавите кнопку OK в свою форму и установите ее свойство в `DialogResult.OK` форма автоматически закрывается, когда вы нажимаете эту кнопку, а вызывающий код получает `DialogResult.OK` в ответ на `ShowDialog()` метода `ShowDialog()` .

Прочитайте Отображение формы онлайн: https://riptutorial.com/ru/winforms/topic/8768/отображение-формы

глава 6: Помощь интеграции

замечания

Вы можете предоставлять помощь для форм и элементов управления в приложениях Windows Forms по-разному. Вы можете открыть всплывающее окно, открыть CHM-файл или URL-адрес. Вы можете показать контекстно-зависимую справку для форм, элементов управления и диалогов.

Компонент HelpProvider

Вы можете настроить компонент `HelpProvider` для предоставления контекстно-зависимой справки для компонента. Таким образом, когда пользователь нажимает клавишу `F1` или кнопку справки формы, вы можете автоматически:

- Показать контекстно-зависимую подсказку для элементов управления
- Откройте CHM-файл на основе контекста (Показать таблицу содержимого, Показать ключевое слово или индекс, показать тему)
- Перейдите к URL-адресу, используя браузер по умолчанию.

Класс справки

Вы можете использовать класс `Help` в коде, чтобы обеспечить такую помощь:

- Показывать всплывающее окно справки для элемента управления
- Откройте CHM-файл на основе контекста (Показать таблицу содержимого, Показать ключевое слово или индекс, показать тему)
- Перейдите к URL-адресу, используя браузер по умолчанию.

Событие HelpRequested

Вы можете обрабатывать событие `HelpRequested` объектов `Control` или `Form` для выполнения пользовательских действий при нажатии пользователем `F1` или нажатии кнопки справки формы.

Кнопка помощи формы

Вы можете настроить `Form` для отображения кнопки справки в заголовке. Таким образом, если пользователь нажмет кнопку «Справка», курсор изменится на а `?` курсор и после щелчка по любой точке будет отображаться любая контекстно-зависимая справка, связанная с `HelpProvider` управления с помощью `HelpProvider` .

Кнопка справки MessgeBox и CommonDialogs

Вы можете предоставить помощь для `MessageBox` , `OpenFileDialog` , `SaveDialog` и `ColorDialog` с помощью кнопки справки компонентов.

Компонент ToolTip

Вы можете использовать компонент `ToolTip` для отображения некоторого текста справки, когда пользователь указывает на элементы управления. `ToolTip` может быть связана с любым `ToolTip` управления.

Заметка

Использование `HelpProvider` и `Help` Вы можете отображать скомпилированные файлы справки (.chm) или HTML-файлы в формате справки HTML. Скомпилированные файлы справки содержат оглавление, индекс, возможности поиска и ссылки на ключевые слова на страницах. Ярлыки работают только в скомпилированных файлах справки. Вы можете создавать файлы справки HTML 1.x с помощью справочной системы справки HTML. Дополнительные сведения о справке HTML см. В «Справочном руководстве по HTML» и других разделах справки HTML в Справке Microsoft HTML .

Examples

Показать файл справки

Класс `Help Class` инкапсулирует HTML-версию 1.0. Вы можете использовать объект справки для отображения скомпилированных файлов справки (.chm) или HTML-файлов в формате справки HTML. Скомпилированные файлы справки содержат оглавления, индексы, поиск и ссылки на ключевые слова на страницах. Ярлыки работают только в скомпилированных файлах справки. Вы можете создавать файлы справки HTML 1.x с помощью бесплатного инструмента из Microsft, называемого `HTML Help Workshop` службой `HTML Help Workshop` .

Простой способ показать скомпилированный файл справки во втором окне:

C #

```
Help.ShowHelp(this, helpProviderMain.HelpNamespace);
```

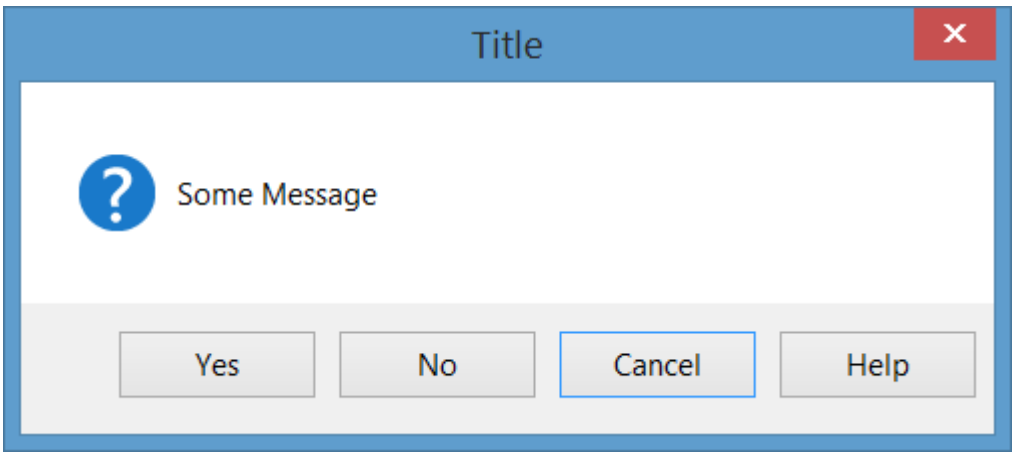
VB.NET

```
Help.ShowHelp(Me, hlpProviderMain.HelpNamespace)
```

Показать справку для MessageBox

Вы можете предоставить помощь для окна сообщений по-разному. Вы можете настроить `MessageBox` для отображения кнопки `Help` или нет. Также вы можете настроить `MessageBox` таким образом, чтобы при запросе пользователя на помощь, нажав кнопку «Справка» или нажав клавишу `F1` , он отображает CHM-файл или перемещается по URL-адресу или выполняет собственное действие. Вот несколько примеров в этом разделе.

В приведенных ниже примерах `MessageBox` будет выглядеть так:



Показать CHM-файл и перейти к ключевому слову (index)

```
MessageBox.Show("Some Message", "Title", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button3, 0,
    "help.chm", HelpNavigator.KeywordIndex, "SomeKeyword");
```

Показать CHM-файл и перейти к теме

```
MessageBox.Show("Some Message", "Title", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button3, 0,
    "help.chm", HelpNavigator.Topic, "/SomePath/SomePage.html");
```

Показать CHM-файл и перейти к первой странице справки в оглавлении

```
MessageBox.Show("Some Message", "Title", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button3, 0,
    "help.chm");
```

Откройте браузер по умолчанию и перейдите к URL-адресу

```
MessageBox.Show("Some Message", "Title", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button3, 0,
```

```
    "http://example.com");
```

Выполняйте индивидуальное действие при нажатии кнопки «Справка» или клавиши F1

В этом случае вы должны обработать событие `HelpRequested` родителя `MessageBox` и выполнить пользовательскую операцию:

```
private void Form1_HelpRequested(object sender, HelpEventArgs hlpevent)
{
    // Perform custom action, for example show a custom help form
    var f = new Form();
    f.ShowDialog();
}
```

Затем вы можете показать `MessageBox` кнопкой справки:

```
MessageBox.Show("Some Message", "Title", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button3, 0, true);
```

Или покажите его без кнопки справки:

```
MessageBox.Show("Some Message", "Title", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button3, 0, false);
```

Показать справку для CommonDialogs

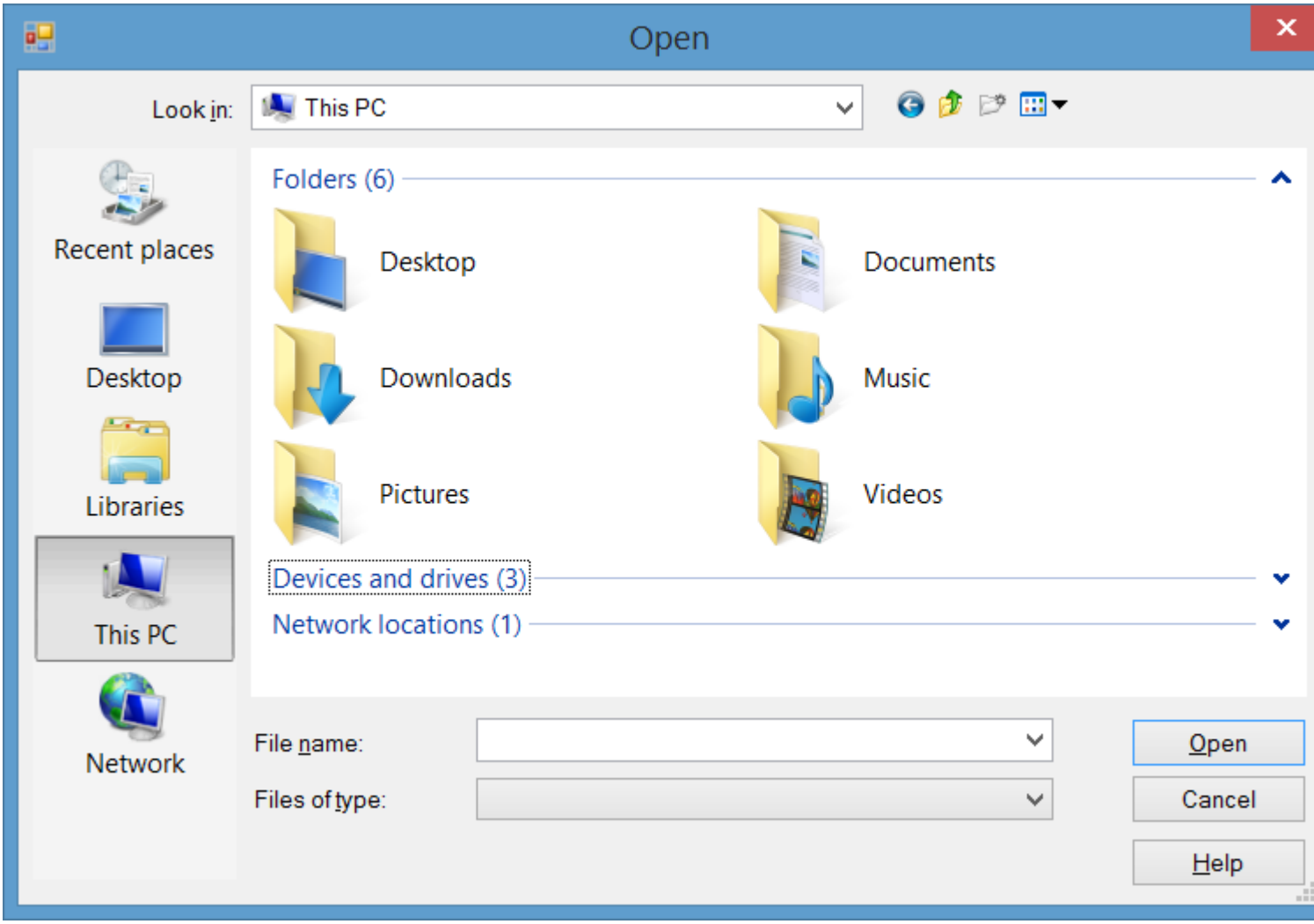
Вы можете предоставить помощь для `OpenFileDialog` , `SaveFileDialog` и `ColorDialog` . Для этого установите `ShowHelp` свойства `ShowHelp` диалогового окна значение `true` и обработайте событие `HelpRequest` для диалога:

```
void openFileDialog1_HelpRequest(object sender, EventArgs e)
{
    //Perform custom action
    Help.ShowHelp(this, "Http://example.com");
}
```

Заметка

- Событие будет поднято только в том случае, если для параметра `ShowHelp` установлено `ShowHelp` `true` .
- Событие будет поднято только нажатием кнопки « `Help` и не будет подниматься с помощью клавиши F1.

На изображении ниже вы можете увидеть `OpenFileDialog` с кнопкой справки:



Обработка справки Проверенное событие элементов управления и формы

Когда пользователь нажимает `F1` на элементе управления или нажимает кнопку «Справка» формы (`?`), А затем нажимает на элемент управления, событие `HelpRequested` будет поднято.

Вы можете обработать это событие, чтобы предоставить настраиваемое действие, когда пользователь запрашивает помощь для элементов управления или формы.

`HelpRequested` поддерживает механизм пузырьков. Он запускает ваш активный элемент управления, и если вы не обрабатываете событие и не `Handled` свойство `Handled` своего события arg `true` , тогда он пузырится до иерархии родительских элементов управления до формы.

Например, если вы обрабатываете событие `HelpRequested` формы, как `HelpRequested` ниже, то, когда вы нажмете `F1`, появится окно с сообщением и отобразит имя активного элемента управления, но для `textBox1` появится другое сообщение:

```
private void Form1_HelpRequested(object sender, HelpEventArgs hlpevent)
```

```
{
    var c = this.ActiveControl;
    if(c!=null)
        MessageBox.Show(c.Name);
}
private void textBox1_HelpRequested(object sender, HelpEventArgs hlpevent)
{
    hlpevent.Handled = true;
    MessageBox.Show("Help request handled and will not bubble up");
}
```

Вы можете выполнять любые другие пользовательские действия, например, используя навигацию к URL-адресу или отображение CHM-файла с `Help` класса `Help` .

Показать справку, используя класс справки

Вы можете использовать класс `Help` в коде, чтобы обеспечить такую помощь:

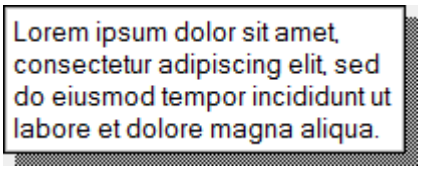
- Показывать всплывающее окно справки для элемента управления
- Откройте CHM-файл на основе контекста (Показать таблицу содержимого, Показать ключевое слово или индекс, показать тему)
- Перейдите к URL-адресу, используя браузер по умолчанию.

Показать всплывающее окно справки

Вы можете использовать `Help.ShowPopup` для отображения всплывающего окна справки:

```
private void control_MouseClick(object sender, MouseEventArgs e)
{
    var c = (Control)sender;
    var help = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, " +
               "sed do eiusmod tempor incididunt ut labore et dolore magna aliqua."
    if (c != null)
        Help.ShowPopup(c, "Lorem ipsum dolor sit amet.", c.PointToScreen(e.Location));
}
```

Он отобразит такую подсказку в месте расположения указателя мыши:



Показать файл справки CHM

Вы можете использовать различные перегрузки метода `Help.ShowHelp` , чтобы показать CHM-файл и перейти к ключевому слову, теме, индексу или таблице содержимого:

Показать таблицу справки

```
Help.ShowHelp(this, "Help.chm");
```

Показать справку по конкретному ключевому слову (указателю)

```
Help.ShowHelp(this, "Help.chm", HelpNavigator.Index, "SomeKeyword");
```

Показать справку по конкретной теме

```
Help.ShowHelp(this, "Help.chm", HelpNavigator.Topic, "/SomePath/SomePage.html");
```

Показать URL

Вы можете показать любой URL-адрес в браузере по умолчанию, используя метод `ShowHelp` :

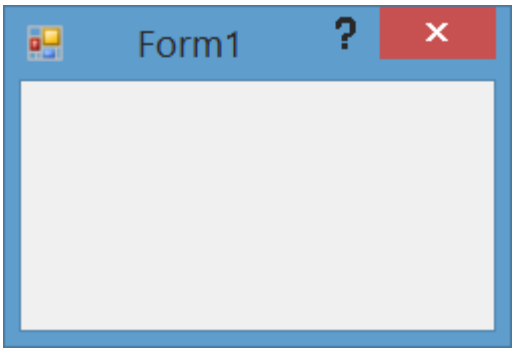
```
Help.ShowHelp(this, "Http://example.com");
```

Показать кнопку справки на панели заголовка формы

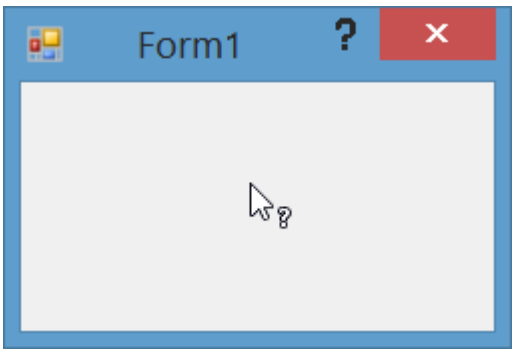
Вы можете показать кнопку справки в строке заголовка `Form` . Для этого вам необходимо:

1. Установите `HelpButton` свойства `HelpButton` формы значение `true` .
2. Установите `MinimizeBox` и `MaximizeBox` на `false` .

Затем в строке заголовка `Form` появится кнопка помощи:



Также, когда вы нажимаете кнопку «Справка», курсор будет изменен на `?` курсор:



Затем, если вы нажмете на элемент `Control` или `Form` , событие `HelpRequested` будет поднято, а также если вы настроите `HelpProvider` , то помощь для `HelpProvider` управления будет показана с помощью `HelpProvider` .

Создайте собственную кнопку справки, которая действует как стандартная форма HelpButton

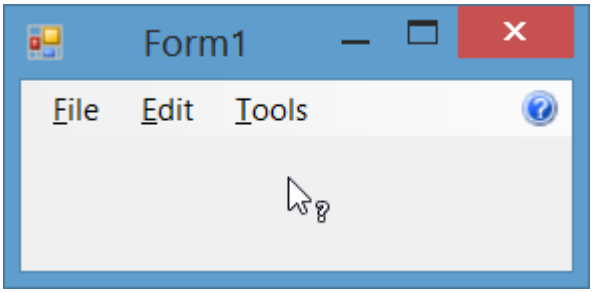
Если у вас есть `Form` с `MinimizeBox` и `MaximizeBox` установленная в `true` , то вы не можете показать кнопку справки на панели заголовка `Form` и потеряете функцию щелчка по кнопке справки, чтобы преобразовать ее, чтобы помочь курсору, чтобы можно было щелкнуть элементы управления, чтобы показать справку.

Вы можете сделать пункт меню в `MenuStrip` как стандартная кнопка справки. Для этого добавьте `MenuStrip` в форму и добавьте в нее `ToolStripMenuItem` , а затем обработайте событие `Click` элемента:

```
private const int WM_SYSCOMMAND = 0x0112;
private const int SC_CONTEXTHELP = 0xF180;
[System.Runtime.InteropServices.DllImport("user32.dll")]
static extern IntPtr SendMessage(IntPtr hWnd, int Msg, int wParam, int lParam);
private void helpToolStripMenuItem_Click(object sender, EventArgs e)
{
    SendMessage(this.Handle, WM_SYSCOMMAND, SC_CONTEXTHELP, 0);
}
```

Примечание. Если вы хотите сделать это с помощью `Button` , вам также необходимо установить `button1.Capture = false;` перед отправкой сообщения. Но это не обязательно для `ToolStripMenuItem` .

Затем, когда вы нажимаете на меню справки, курсор будет изменен на `?` курсор и будет действовать, когда вы нажимаете кнопку стандартной справки:



Обработка события HelpButtonClicked формы

Вы можете обнаружить, когда пользователь `HelpButton` на `HelpButton` на панели заголовка формы, `HelpButtonClicked` к `HelpButtonClicked` . Вы можете позволить событию продолжить или отменить его, установив для параметра `Cancel` свойство своего события значение `true` .

```
private void Form1_HelpButtonClicked(object sender, CancelEventArgs e)
{
```

```
    e.Cancel = true;
    //Perform some custom action
    MessageBox.Show("Some Custom Help");
}
```

Прочитайте Помощь интеграции онлайн: https://riptutorial.com/ru/winforms/topic/3285/помощь-интеграции

глава 7: Текстовое окно

Examples

Автозаполнение из набора строк

```
var source = new AutoCompleteStringCollection();

// Add your collection of strings.
source.AddRange(new[] { "Guybrush Threepwood", "LeChuck" });

var textBox = new TextBox
{
    AutoCompleteCustomSource = source,
    AutoCompleteMode = AutoCompleteMode.SuggestAppend,
    AutoCompleteSource = AutoCompleteSource.CustomSource
};

form.Controls.Add(textBox);
```

Это будет автоматически **заполнено,** когда пользователь попытается ввести **G** или **L.**

`AutoCompleteMode.SuggestAppend` отобразит список предлагаемых значений, и он будет автоматически печатать первое совпадение, только `Append` и `Suggest` .

Разрешить только цифры в тексте

```
textBox.KeyPress += (sender, e) => e.Handled = !char.IsControl(e.KeyChar) &&
!char.IsDigit(e.KeyChar);
```

Это позволит использовать только цифры и управляющие символы в `TextBox` , возможны другие комбинации, используя тот же подход, что и свойство `Handle` для true для блокировки текста.

Пользователь все равно может копировать / вставлять нежелательные символы, поэтому дополнительная проверка должна выполняться в `TextChanged` для очистки ввода:

```
textBox.TextChanged += (sender, e) => textBox.Text = Regex.Match(textBox.Text, @"\d+").Value
```

В этом примере для фильтрации текста используется **регулярное выражение** .

NumericUpDown должно быть предпочтительным для чисел, когда это возможно.

Как прокрутить до конца

```
textBox.SelectionStart = textBox.TextLength;
textBox.ScrollToCaret();
```

Применяя тот же принцип, `SelectionStart` может быть установлен в `0` для прокрутки вверх или до определенного номера, чтобы перейти к определенному символу.

Добавление заполнитель в текстовое поле

Этот код помещает текст *подсказки* при загрузке формы и манипулирует им следующим образом:

C #

```
private void Form_load(object sender, EventArgs e)
{
    textBox.Text = "Place Holder text...";
}

private void textBox_Enter(object sender, EventArgs e)
{
    if(textBox.Text == "Place Holder text...")
    {
        textBox.Text = "";
    }
}

private void textBox_Leave(object sender, EventArgs e)
{
    if(textBox.Text.Trim() == "")
    {
        textBox.Text = "Place Holder text...";
    }
}
```

VB.NET

```
Private Sub Form_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    textBox.Text = "Place Holder text..."
End Sub

Private Sub textBox_GotFocus(sender as Object,e as EventArgs) Handles textBox.GotFocus
    if Trim(textBox.Text) = "Place Holder text..." Then
        textBox.Text = ""
    End If
End Sub

Private Sub textBox_LostFocus(sender as Object,e as EventArgs) Handles textBox.LostFocus
    if Trim(textBox.Text) = "" Then
        textBox.Text = "Place Holder text..."
    End If
End Sub
```

Прочитайте Текстовое окно онлайн: https://riptutorial.com/ru/winforms/topic/4674/текстовое-окно

кредиты

S. No	Главы	Contributors
1	Начало работы с winforms	4444, Bjørn-Roger Kringsjå, Chris Shao, Cody Gray, Community, Reza Aghaei
2	Databinding	Kai Thoma
3	Наследование элементов управления	Balagurunathan Marimuthu
4	Основные элементы управления	Aimnox, Berkay, help-info.de, Jeff Bridgman
5	Отображение формы	Cody Gray, Jeff Bridgman, Steve
6	Помощь интеграции	help-info.de, Reza Aghaei
7	Текстовое окно	gplumb, Jones Joseph, Stefano d'Antonio