



Kostenloses eBook

LERNEN

winforms

Free unaffiliated eBook created from
Stack Overflow contributors.

#winforms

Inhaltsverzeichnis

Über.....	1
Kapitel 1: Erste Schritte mit winforms.....	2
Bemerkungen.....	2
Siehe auch:.....	2
Examples.....	2
Erstellen einer einfachen WinForms-Anwendung mit Visual Studio.....	2
Erstellen Sie ein Windows Forms-Projekt.....	3
Fügen Sie dem Formular Steuerelemente hinzu.....	3
Code schreiben.....	4
Laufen und testen.....	5
Erstellen einer einfachen C # WinForms-Anwendung mithilfe eines Texteditors.....	5
Erstellen einer einfachen VB.NET WinForms-Anwendung mit einem Texteditor.....	6
Kapitel 2: Datenbindung.....	9
Parameter.....	9
Bemerkungen.....	9
Examples.....	9
Binden von Steuerelementen an Datenobjekte.....	9
Kapitel 3: Ein Formular anzeigen.....	11
Einführung.....	11
Examples.....	11
Zeigen Sie ein Modell oder ein modales Formular.....	11
Schließen eines modelllosen Formulars.....	11
Ein modales Formular schließen.....	12
Kapitel 4: Grundlegende Bedienelemente.....	14
Examples.....	14
Taste.....	14
Textfeld.....	14
Kombinationsfeld.....	15
CheckBox.....	17
ListBox.....	17

NumericUpDown.....	21
Nützliche Ereignisse.....	23
Kapitel 5: Hilfe bei der Integration.....	25
Bemerkungen.....	25
HelpProvider-Komponente.....	25
Hilfe Klasse.....	25
HelpRequested-Ereignis.....	25
Hilfeschaltfläche des Formulars.....	25
Hilfeschaltfläche von MessageBox und CommonDialogs.....	25
QuickInfo-Komponente.....	26
Examples.....	26
Hilfdatei anzeigen.....	26
Hilfe für MessageBox anzeigen.....	26
Zeigen Sie eine CHM-Datei an und navigieren Sie zu einem Stichwort (Index).....	27
Zeigen Sie eine CHM-Datei an und navigieren Sie zu einem Thema.....	27
Zeigen Sie eine CHM-Datei an und navigieren Sie zur ersten Hilfeseite im Inhaltsverzeichnis.....	27
Öffnen Sie den Standardbrowser und navigieren Sie zu einer URL.....	27
Führen Sie eine benutzerdefinierte Aktion aus, wenn Sie die Hilfetaste oder die Taste F1 d.....	27
Hilfe für CommonDialogs anzeigen.....	28
Behandeln des HelpRequested-Ereignisses von Steuerelementen und Formular.....	29
Hilfe mit Hilfe der Klasse anzeigen.....	30
Hilfe-Popup-Fenster anzeigen.....	30
CHM-Hilfdatei anzeigen.....	30
Hilfeinhaltsverzeichnis anzeigen.....	30
Hilfe für ein bestimmtes Keyword anzeigen (Index).....	31
Hilfe zu einem bestimmten Thema anzeigen.....	31
URL anzeigen.....	31
Hilfeschaltfläche in der Titelleiste des Formulars anzeigen.....	31
Erstellen Sie eine benutzerdefinierte Hilfeschaltfläche, die sich wie eine Standard-Formul.....	32
Behandeln des HelpButtonClicked-Ereignisses von Form.....	32
Kapitel 6: Steuerelemente übernehmen.....	33
Bemerkungen.....	33

Examples.....	33
Anwendungsweite Einstellungen.....	33
NumberBox.....	36
Kapitel 7: Textfeld.....	45
Examples.....	45
Automatische Vervollständigung aus einer Sammlung von Zeichenfolgen.....	45
Erlaube nur Ziffern im Text.....	45
So blättern Sie bis zum Ende.....	45
Platzhalter in ein Textfeld einfügen.....	46
Credits.....	47



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.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

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

Kapitel 1: Erste Schritte mit winforms

Bemerkungen

Windows Forms (kurz "WinForms") ist eine GUI-Klassenbibliothek, die in .NET Framework enthalten ist. Es ist ein komplexer objektorientierter Wrapper für die [Win32-API](#), der die Entwicklung von Windows-Desktop- und mobilen Anwendungen für [.NET Framework ermöglicht](#).

WinForms ist hauptsächlich [ereignisgesteuert](#). Eine Anwendung besteht aus mehreren *Formularen* (als Fenster auf dem Bildschirm angezeigt), die *Steuerelemente* (Beschriftungen, Schaltflächen, Textfelder, Listen usw.) enthalten, mit denen der Benutzer direkt interagiert. Als Reaktion auf Benutzerinteraktion lösen diese Steuerelemente Ereignisse aus, die vom Programm zur Ausführung von Aufgaben verarbeitet werden können.

Wie in Windows ist alles in WinForms ein Steuerelement, das selbst eine Art Fenster ist. Die Basissteuerungsklasse bietet grundlegende Funktionen, einschließlich Eigenschaften für das Festlegen von Text, Ort, Größe und Farbe sowie allgemeine Ereignisse, die behandelt werden können. Alle Steuerelemente werden von der Steuerelementklasse abgeleitet und fügen zusätzliche Funktionen hinzu. Einige Steuerelemente können andere Steuerelemente enthalten, entweder zur Wiederverwendbarkeit (`Form`, `UserControl`) oder zum Layout (`TableLayoutPanel`, `FlowLayoutPanel`).

WinForms wird seit der ursprünglichen Version von .NET Framework (Version 1.0) unterstützt und ist in modernen Versionen (Version 4.5) weiterhin verfügbar. Es wird jedoch nicht mehr aktiv entwickelt und es werden keine neuen Funktionen hinzugefügt. [Laut 9 Microsoft-Entwicklern auf der Build 2014-Konferenz](#) :

Windows Forms wird weiterhin unterstützt, befindet sich jedoch im Wartungsmodus. Sie beheben Fehler, sobald sie entdeckt werden, aber neue Funktionen sind nicht in der Tabelle.

Die plattformübergreifende, Open-Source- [Mono-Bibliothek](#) stellt eine grundlegende Implementierung von Windows Forms bereit und unterstützt alle Features, die die Microsoft-Implementierung ab .NET 2.0 verwendet hat. WinForms wird jedoch nicht aktiv auf Mono entwickelt, und eine vollständige Implementierung wird als unmöglich angesehen, da das Framework unlösbar mit der nativen Windows-API verbunden ist (die auf anderen Plattformen nicht verfügbar ist).

Siehe auch:

- [Windows Forms- Dokumentation auf MSDN](#)

Examples

Erstellen einer einfachen WinForms-Anwendung mit Visual Studio

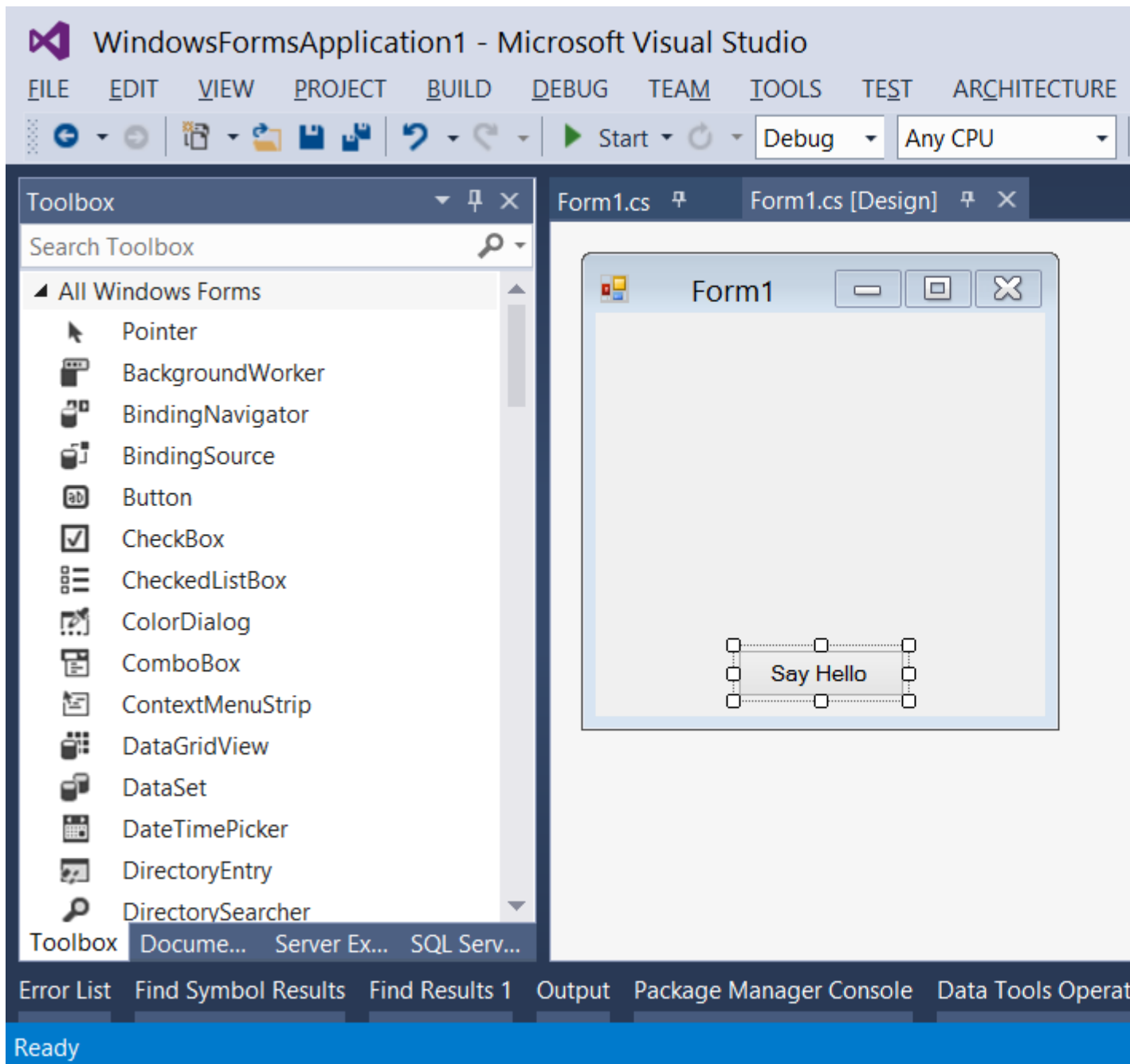
In diesem Beispiel wird gezeigt, wie Sie ein Windows Forms-Anwendungsprojekt in Visual Studio erstellen.

Erstellen Sie ein Windows Forms-Projekt

1. Starten Sie Visual Studio.
2. **Zeigen Sie im Menü Datei auf Neu** , und wählen Sie dann **Projekt aus** . Das Dialogfeld **Neues Projekt** wird angezeigt.
3. Wählen Sie im Bereich "**Installierte Vorlagen**" die Option "Visual C #" oder "Visual Basic" aus.
4. Oberhalb des mittleren Bereichs können Sie das Zielframework aus der Dropdown-Liste auswählen.
5. Wählen Sie im mittleren Bereich die **Windows Forms-Anwendungsvorlage aus** .
6. **Geben Sie im Textfeld Name** einen Namen für das Projekt ein.
7. Im Textfeld, wählen Sie einen Ordner , um das Projekt zu speichern.
8. Klicken Sie auf **OK** .
9. Der Windows Forms-Designer wird geöffnet und zeigt **Form1** des Projekts an.

Fügen Sie dem Formular Steuerelemente hinzu

1. Ziehen Sie aus der **Toolbox**- Palette ein **Button**- Steuerelement auf das Formular.
2. Klicken Sie auf die Schaltfläche, um sie auszuwählen. Legen Sie im Eigenschaftfenster die Eigenschaft `Text` auf **Hallo sagen fest** .



Code schreiben

1. Doppelklicken Sie auf die Schaltfläche, um einen Ereignishandler für das `click` Ereignis hinzuzufügen. Der Code-Editor wird mit der Einfügemarke innerhalb der Event-Handler-Funktion geöffnet.
2. Geben Sie den folgenden Code ein:

C

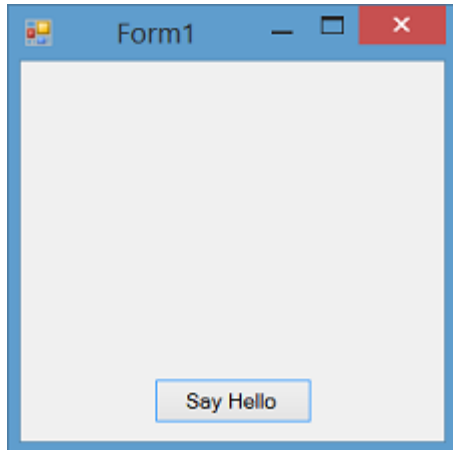
```
MessageBox.Show("Hello, World!");
```

VB.NET

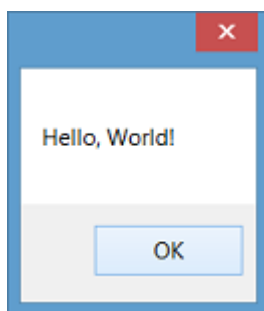

```
MessageBox.Show("Hello, World!");
```

Laufen und testen

1. Drücken Sie `F5`, um die Anwendung auszuführen.



2. Wenn Ihre Anwendung ausgeführt wird, klicken Sie auf die Schaltfläche, um "Hallo, Welt!" Botschaft.



3. Schließen Sie das Formular, um zu Visual Studio zurückzukehren.

Erstellen einer einfachen C # WinForms-Anwendung mithilfe eines Texteditors

1. Öffnen Sie einen Texteditor (z. B. Notepad) und geben Sie den folgenden Code ein:

```
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());
    }
}

```

- Speichern Sie die Datei unter einem Pfad, auf den Sie Lese- / Schreibzugriff haben. Es ist üblich, die Datei nach der enthaltenen Klasse zu benennen, beispielsweise `X:\MainForm.cs`.
- Führen Sie den C#-Compiler von der Befehlszeile aus, und übergeben Sie den Pfad als Argument an die Codedatei:

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

Hinweis: Um eine Version des C#-Compilers für andere .NET Framework-Versionen zu verwenden, schauen Sie in den Pfad `%WINDIR%\Microsoft.NET` und ändern das obige Beispiel entsprechend. Weitere Informationen zum Kompilieren von C#-Anwendungen finden Sie unter [Kompilieren und Ausführen Ihres ersten C#-Programms](#).

- Nach Abschluss der Kompilierung wird eine Anwendung namens `MainForm.exe` im selben Verzeichnis wie Ihre `MainForm.exe` erstellt. Sie können diese Anwendung entweder über die Befehlszeile oder durch Doppelklick im Explorer ausführen.

Erstellen einer einfachen VB.NET WinForms-Anwendung mit einem Texteditor

- Öffnen Sie einen Texteditor (z. B. Notepad) und geben Sie den folgenden Code ein:

```

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.
        <SThread> _
        Public Shared Sub Main()
            Application.EnableVisualStyles()
            Application.Run(New MainForm())
        End Sub
    End Class
End Namespace

```

2. Speichern Sie die Datei unter einem Pfad, auf den Sie Lese- / Schreibzugriff haben. Es ist üblich, die Datei nach der enthaltenen Klasse zu benennen, beispielsweise `X:\MainForm.vb`.
3. Führen Sie den VB.NET-Compiler über die Befehlszeile aus, und übergeben Sie den Pfad als Argument an die Codedatei:

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

Hinweis: Um eine Version des VB.NET-Compilers für andere .NET Framework-Versionen zu verwenden, schauen Sie in den Pfad `%WINDIR%\Microsoft.NET` und ändern das obige Beispiel entsprechend. Weitere Informationen zum Kompilieren von VB.NET-Anwendungen finden Sie unter [Hello World](#).

4. Nach Abschluss der Kompilierung wird eine Anwendung namens `MainForm.exe` im selben Verzeichnis wie Ihre `MainForm.exe` erstellt. Sie können diese Anwendung entweder über die Befehlszeile oder durch Doppelklick im Explorer ausführen.

Erste Schritte mit winforms online lesen: <https://riptutorial.com/de/winforms/topic/1018/erste-schritte-mit-winforms>

Kapitel 2: Datenbindung

Parameter

Streit	Beschreibung
Name des Anwesens	Der Name der zu bindenden Steuerelementeigenschaft.
Datenquelle	Ein Objekt, das die Datenquelle darstellt.
dataMember	Die Eigenschaft oder Liste, an die gebunden werden soll.
formattingEnabled	Legt fest, ob die angezeigten Daten formatiert werden sollen.
Aktualisierungsmodus	Die Datenquelle wird aktualisiert, wenn die Steuerelementeigenschaft überprüft wird (Standardeinstellung) oder sofort, wenn die Eigenschaft geändert wurde
nullValue	Wenn die Datenquelle über diesen Wert verfügt, wird die gebundene Eigenschaft auf DBNull gesetzt.
formatString	Ein oder mehrere Formatbezeichner, die angeben, wie ein Wert angezeigt werden soll
formatInfo	Eine Implementierung von IFormatProvider zum Überschreiben des Standardformatierungsverhaltens.

Bemerkungen

Siehe <https://msdn.microsoft.com/de-de/library/ef2xyb33.aspx> Datenbinding funktioniert nur mit Eigenschaften, niemals mit Feldern!

Examples

Binden von Steuerelementen an Datenobjekte

Jedes Steuerelement verfügt über eine Eigenschaft `DataBindings`, eine Liste von `System.Windows.Forms.Binding` Objekten. Die `Add()` - Methode weist einige Überladungen auf, mit denen Sie problemlos an die Eigenschaft eines Objekts binden können:

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

Beachten Sie, dass das Binden im Grunde bedeutet, dass Sie sich gegenseitig `changeevent` abonnieren. Der obige Code abonniert das `changeevent` von `dataObj.MyProperty` und passt

textBox.Text an, wenn es geändert wird. Umgekehrt abonniert es textBox.TextChanged und passt dataObj.MyProperty an, wenn sich Änderungen ergeben.

Datenbindung online lesen: <https://riptutorial.com/de/winforms/topic/7362/datenbindung>

Kapitel 3: Ein Formular anzeigen

Einführung

In diesem Thema wird erläutert, wie die WinForms-Engine zum Anzeigen von Formularen funktioniert und wie Sie deren Lebensdauer steuern.

Examples

Zeigen Sie ein Modell oder ein modales Formular

Nachdem Sie die Struktur Ihres Formulars mit dem WinForms-Designer definiert haben, können Sie Ihre Formulare mit zwei verschiedenen Methoden im Code anzeigen.

- Methode - Ein Modeless-Formular

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

- Methode - Ein modaler Dialog

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

Die beiden Methoden unterscheiden sich sehr. Die erste Methode (die modelllose Methode) zeigt Ihr Formular und kehrt sofort zurück, ohne das Schließen des gerade geöffneten Formulars abzuwarten. Ihr Code fährt also mit dem fort, was auf den Show-Aufruf folgt. Die zweite Methode (die modale Methode) öffnet stattdessen das Formular und blockiert alle Aktivitäten in der gesamten Anwendung, bis Sie das Formular über die Schaltfläche "Schließen" oder mit den entsprechenden Schaltflächen zum Schließen des Formulars schließen

Schließen eines modelllosen Formulars

Ein modellloses Formular wird (in der Regel) verwendet, wenn Sie etwas permanent neben Ihrem Anwendungshauptbildschirm anzeigen müssen (denken Sie an eine Legende oder eine Ansicht in einem Datenstrom, der asynchron von einem Gerät oder einem MDI-Unterfenster kommt).

Eine modelllose Form stellt jedoch eine einzigartige Herausforderung dar, wenn Sie sie schließen möchten. Wie kann ich die Instanz abrufen und in dieser Instanz die Close-Methode aufrufen?

Sie können eine globale Variable beibehalten, die auf die Instanz verweist, die Sie schließen möchten.

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

Wir können jedoch auch die `Application.OpenForms`-Auflistung verwenden, in der die Formular-Engine alle erstellten und noch offenen Formularinstanzen speichert.

Sie können diese bestimmte Instanz aus dieser Auflistung abrufen und die `Close`-Methode aufrufen

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

Ein modales Formular schließen

Wenn ein Formular mit der `ShowDialog` Methode `ShowDialog` wird, müssen Sie die `DialogResult` Eigenschaft des Formulars so `DialogResult`, dass es sich dem Formular `DialogResult`. Diese Eigenschaft kann mithilfe der [Enumeration](#) festgelegt werden, die auch als `DialogResult` bezeichnet wird.

Um ein Formular zu schließen, müssen Sie lediglich die `DialogResult` Eigenschaft des `DialogResult` (in einem beliebigen Wert von `DialogResult.None`) in einem Ereignishandler `DialogResult.None`. Wenn Ihr Code den Ereignishandler verlässt, blendet das WinForm-Modul das Formular aus, und der Code, der auf den ersten Aufruf der `ShowDialog` Methode folgt, setzt die Ausführung fort.

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

Der aufrufende Code kann den Rückgabewert von `ShowDialog` erfassen, um festzustellen, auf welche Schaltfläche der Benutzer in das Formular geklickt hat. Bei der Anzeige mit `ShowDialog()` wird das Formular nicht automatisch `ShowDialog()` da es einfach ausgeblendet und nicht geschlossen wurde). `ShowDialog()` ist es wichtig, einen `using` Block zu verwenden, um sicherzustellen, dass das Formular entsorgt wird.

Im Folgenden finden Sie ein Beispiel für die Überprüfung des Ergebnisses der Verwendung des integrierten `OpenFileDialog`, das Überprüfen des Ergebnisses und den Zugriff auf eine Eigenschaft über das Dialogfeld, bevor Sie es bereitstellen.

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

Sie können die `DialogResult` Eigenschaft auch für eine Schaltfläche `DialogResult`. Durch Klicken auf diese Schaltfläche wird die `DialogResult` Eigenschaft im Formular auf den mit der Schaltfläche

verknüpften Wert festgelegt. Dadurch können Sie das Formular schließen, ohne einen Ereignishandler hinzuzufügen, um `DialogResult` im Code `DialogResult` .

Wenn Sie beispielsweise Ihrem Formular einen OK-Button hinzufügen und dessen Eigenschaft auf `DialogResult.OK` wird das Formular automatisch geschlossen, wenn Sie auf diese Schaltfläche `DialogResult.OK` , und der aufrufende Code erhält ein `DialogResult.OK` vom `ShowDialog()` .

Ein Formular anzeigen online lesen: <https://riptutorial.com/de/winforms/topic/8768/ein-formular-anzeigen>

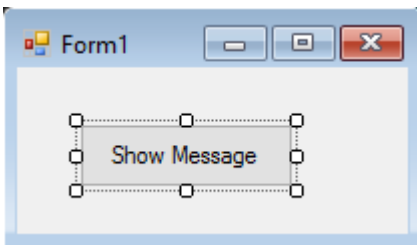
Kapitel 4: Grundlegende Bedienelemente

Examples

Taste

Schaltflächen sind eines der einfachsten Steuerelemente und werden hauptsächlich zum Ausführen von Code verwendet, wenn der Benutzer dies wünscht.

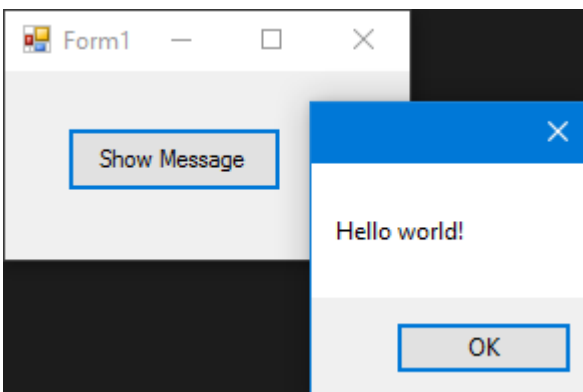
Hier haben wir einen sehr einfachen Fall, zeigen Sie ein Meldungsfeld an, wenn Sie auf eine Schaltfläche klicken. Wir fügen einem Formular eine Schaltfläche hinzu, nennen es `cmdShowMessage` wie es im Code verwendet wird, um das Objekt zu identifizieren, und setzen den Text der Schaltfläche auf "Nachricht `cmdShowMessage` .



Wir müssen nur die Schaltfläche im visuellen Designer doppelklicken, und Visual Studio generiert den Code für das Klickereignis. Jetzt müssen wir nur noch den Code für die MessageBox dort hinzufügen:

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

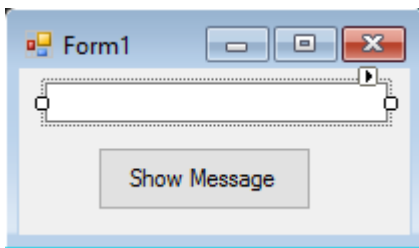
Wenn wir das Programm jetzt ausführen und auf die Schaltfläche klicken, wird die folgende Meldung angezeigt:



Textfeld

TextBoxen erlauben dem Benutzer, Daten in das Programm einzugeben.

Wir werden das Formular ändern und ein Textfeld hinzufügen, sodass das Meldungsfeld uns die vom Benutzer gewünschte Nachricht anzeigt. Nun sieht unser Formular so aus:

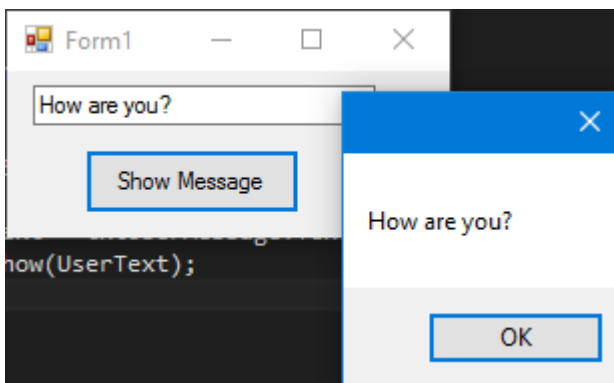


Ändern Sie dann das Schaltflächenklickereignis, um den Text des Textfelds zu verwenden:

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

Wie Sie sehen, verwenden wir die `.Text` Eigenschaft des Textfelds, das den in der Textbox enthaltenen Text darstellt.

Wenn wir das Programm ausführen, können wir in das Textfeld schreiben. Wenn wir auf die Schaltfläche klicken, zeigt die `MessageBox` den von uns geschriebenen Text an:

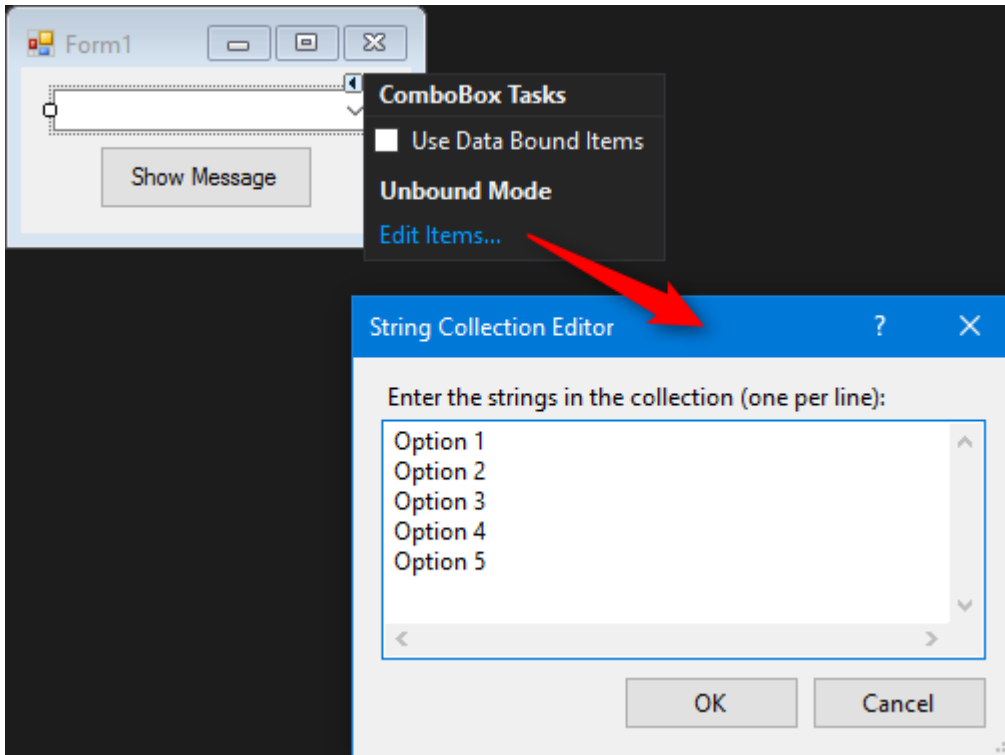


Kombinationsfeld

ComboBoxes ermöglichen dem Benutzer die Auswahl einer der verschiedenen vom Entwickler bereitgestellten Optionen.

Wir werden das Formular ändern und eine Combobox hinzufügen, so dass uns in der Messagebox die Nachricht angezeigt wird, die der Benutzer aus einer von uns gewünschten Liste möchte.

Nachdem Sie die Kombination zum Formular hinzugefügt haben, fügen Sie der Liste nun eine Liste mit Optionen hinzu. Dazu müssen wir die `Items` Eigenschaft ändern:

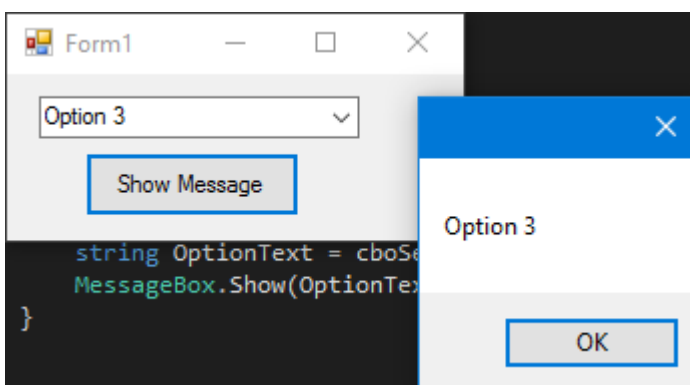


Nun müssen wir den Code des Click-Events ändern:

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

Wie Sie sehen, verwenden wir die `SelectedItem` Eigenschaft. Sie enthält das Objekt der ausgewählten Option. Da wir einen String zum `ToString()` benötigen und der Compiler nicht weiß, ob das Objekt ein String ist oder nicht, müssen wir die `ToString()` Methode verwenden.

Wenn wir das Programm ausführen, können wir die Option auswählen, die wir bevorzugen. Wenn wir auf die Schaltfläche klicken, wird das Meldungsfeld angezeigt:



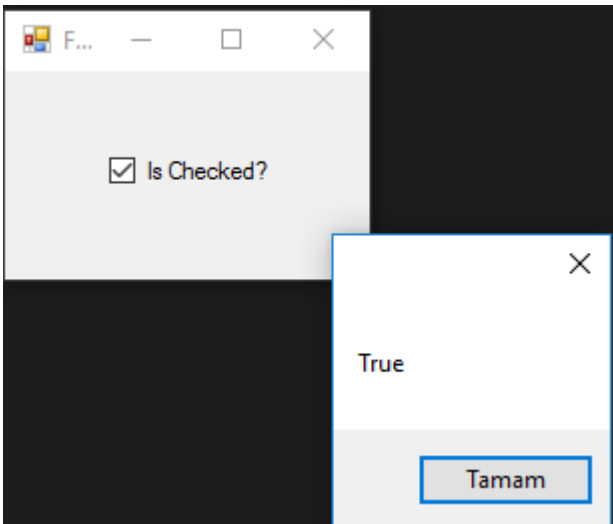
Verwenden Sie das `SelectionChangeCommitted` Ereignis, um benachrichtigt zu werden, wenn ein Benutzer ein Element aus der Combobox auswählt. Wir könnten das `SelectedIndexChanged` - Ereignis verwenden. Dies wird jedoch auch ausgelöst, wenn wir das select-Element in der Combobox programmgesteuert ändern.

CheckBox

CheckBox ist ein Steuerelement, mit dem Benutzer `boolean` Werte vom Benutzer für eine spezielle Frage wie " Geht es **dir gut?**" .

Hat ein Ereignis namens `CheckedChanged` , das immer dann auftritt, wenn die `check` Eigenschaft geändert wird.

Hier ist eine CheckBox mit der Frage "**Wird geprüft?**" .



Wir haben diese `MessageBox` vom `CheckedChanged` Ereignis erhalten.

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

Wenn **CheckBox** markiert ist, ist die Variable `isChecked` `true` .

Wenn **CheckBox** nicht `isChecked` ist, wird die Variable `isChecked` auf `false` .

ListBox

`Listbox` ist ein Steuerelement, das eine Sammlung von Objekten enthalten kann. `Listbox` ähnelt der `Combobox` jedoch in der `Combobox` . Nur ausgewählte Elemente sind für den Benutzer sichtbar. In der `Listbox` ; Alle Elemente sind für den Benutzer sichtbar.

Wie füge ich Elemente zu `Listbox` hinzu?

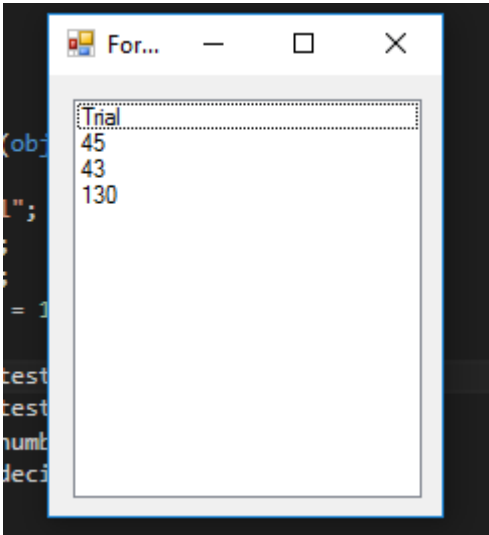
```
private void Form3_Load(object sender, EventArgs e)
{
    string test = "Trial";
    string test2 = "45";
    int numberTest = 43;
    decimal decimalTest = 130;
```

```

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

```

Ausgabe ;



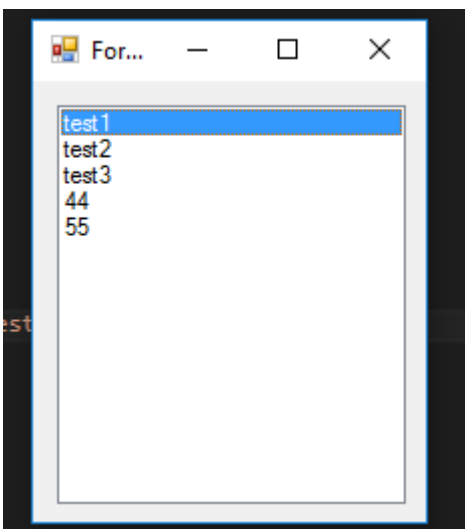
Oder datasources können gegeben werden,

```

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

```

Ausgabe;



```

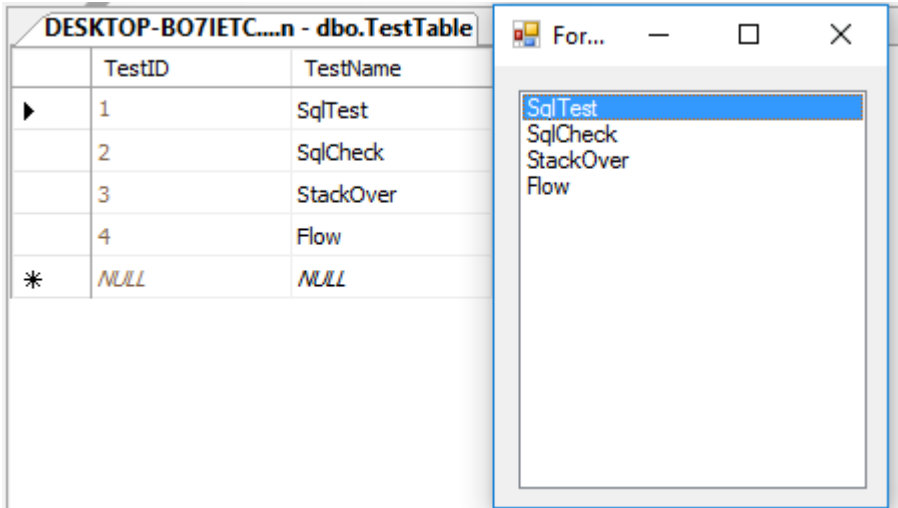
private void Form3_Load(object sender, EventArgs e)
{
    SqlConnection Connection = new
    SqlConnection("Server=serverName;Database=db;Trusted_Connection=True;"); //Connetion to MS-

```

SQL (RDBMS)

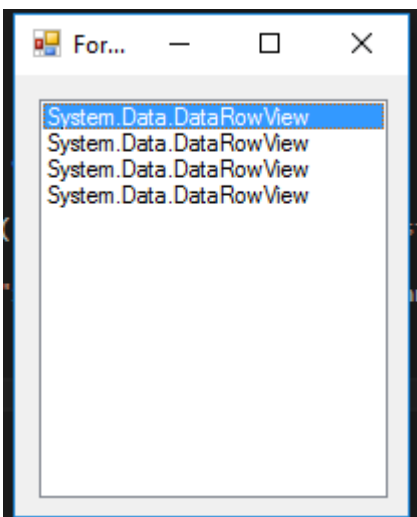
```
Connection.Open(); //Connection open
SqlDataAdapter Adapter = new SqlDataAdapter("Select * From TestTable",
Connection); // Get all records from TestTable.
DataTable DT = new DataTable();
Adapter.Fill(DT); // Fill records to DataTable.
listBox1.DataSource = DT; // DataTable is the datasource.
listBox1.ValueMember = "TestID";
listBox1.DisplayMember= "TestName";
}
```

Die richtige Ausgabe ;



Das Übergeben einer externen SQL-Datenquelle an das Listenfeld erfordert `ValueMember` und `DisplayMember`

Wenn **nicht** , dass es wie folgt aussieht,



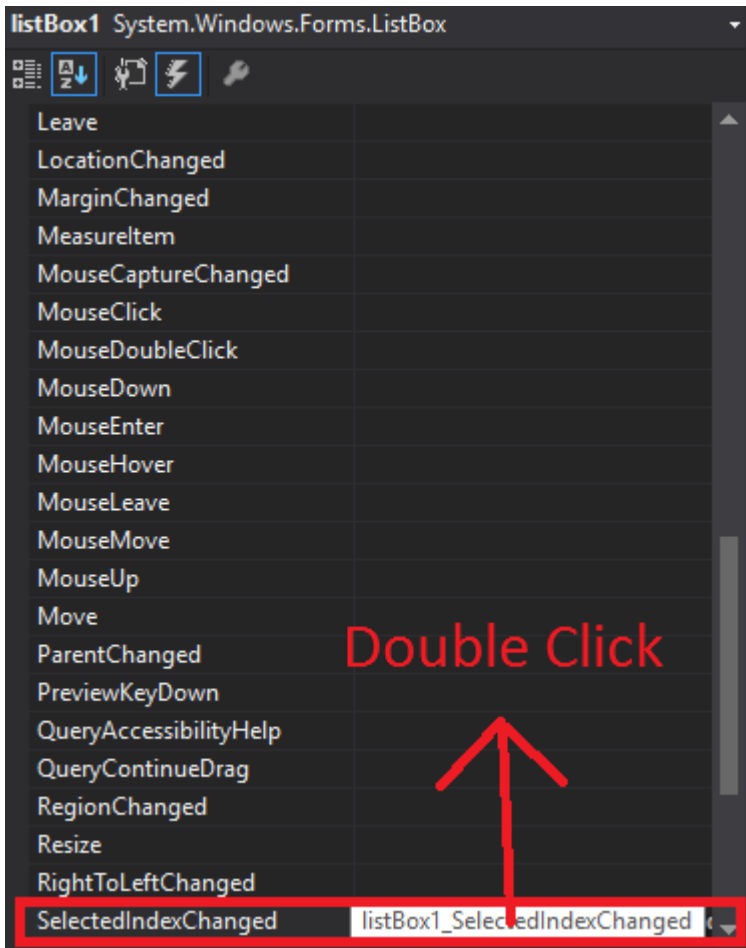
Nützliche Ereignisse;

SelectedIndex_Changed;

Definieren Sie eine Liste für die Datenquelle

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

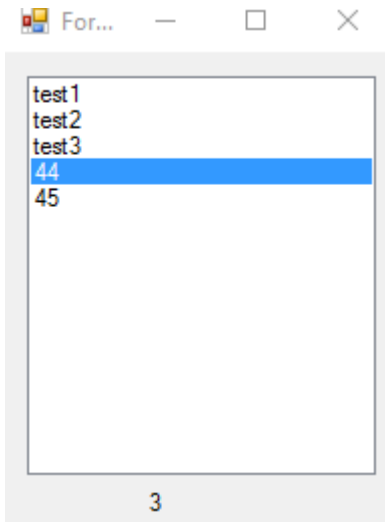
Bei der Gestaltung des Formulars wählen Sie `Listbox` und drücken Sie F4 oder klicken Sie rechts auf das Beleuchtungssymbol.



Visual Studio generiert `listBox1_SelectedIndexChanged` für codebehind.

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

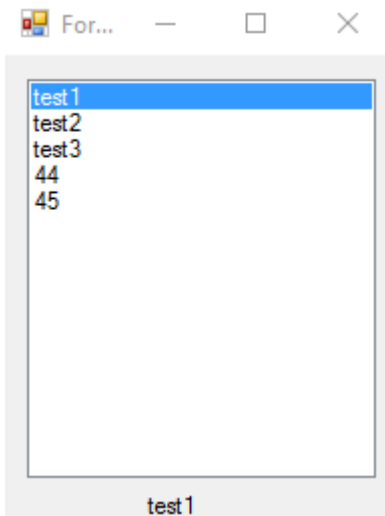
Ergebnis von `SelectedIndex_Changed` ; (Etikett unten zeigt den Index jedes ausgewählten Elements)



SelectedValue_Changed; (Die Datenquelle ist dieselbe wie oben und Sie können dieses Ereignis wie SelectedIndex_Changed generieren.)

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

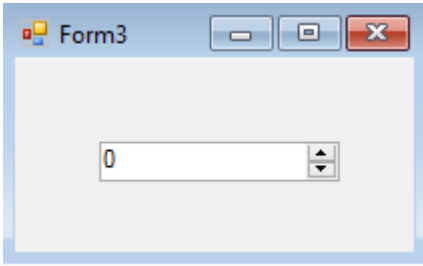
Ausgabe ;



NumericUpDown

NumericUpDown ist ein Steuerelement, das wie TextBox aussieht. Mit diesem Steuerelement kann der Benutzer eine Nummer aus einem Bereich anzeigen / auswählen. Aufwärts- und Abwärtspfeile aktualisieren den Textfeldwert.

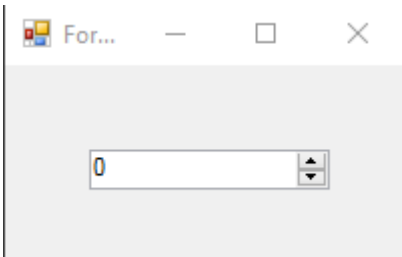
Kontrolle aussehen wie;



In `Form_Load` kann ein Bereich festgelegt werden.

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

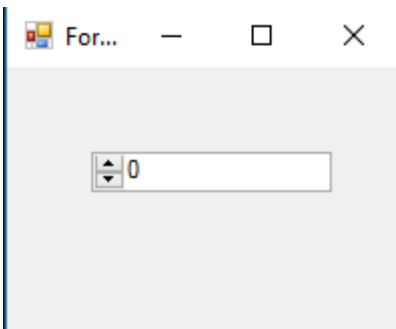
Ausgabe;



UpDownAlign legt die Position der Pfeile fest.

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

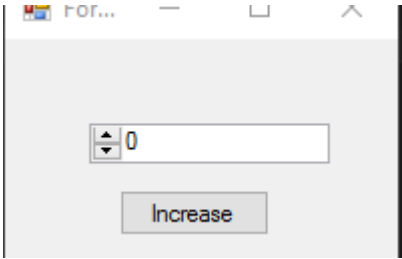
Ausgabe;



`UpButton()` Methode erhöht die Anzahl der Steuerelemente. (kann von überall aus angerufen werden. Ich habe ihn mit einer `button` angerufen.)

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

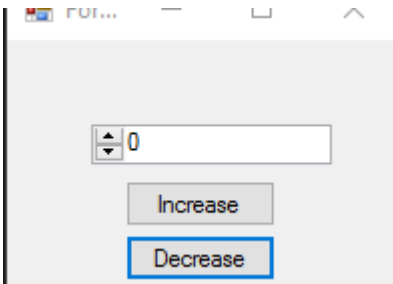
**Ausgabe



`DownButton()` verringert die Nummer des Steuerelements. (kann von überall aus angerufen werden. Ich habe es mit einer `button` erneut aufgerufen.)

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

Ausgabe;



Nützliche Ereignisse

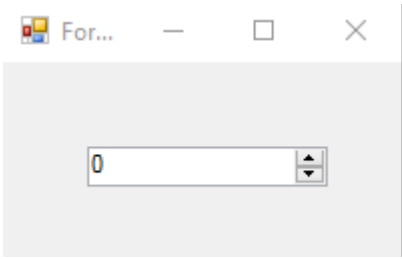
ValueChanged;

Dieses Ereignis funktioniert, wenn Sie auf den Aufwärts- oder Abwärtspfeil klicken.

```
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.
    }
}
```

```
}  
}
```

Ausgabe ;



Grundlegende Bedienelemente online lesen:

<https://riptutorial.com/de/winforms/topic/5816/grundlegende-bedienelemente>

Kapitel 5: Hilfe bei der Integration

Bemerkungen

Sie können auf verschiedene Arten Hilfe für Formulare und Steuerelemente in Windows Forms-Anwendungen bereitstellen. Sie können eine Popup-Hilfe anzeigen, eine CHM-Datei oder eine URL öffnen. Sie können eine kontextsensitive Hilfe für Formulare, Steuerelemente und Dialoge anzeigen.

HelpProvider-Komponente

Sie können eine `HelpProvider` Komponente `HelpProvider`, um kontextsensitive Hilfe für die Komponente bereitzustellen. Auf diese Weise können Sie, wenn der Benutzer die Taste `F1` oder die Hilfetaste des Formulars drückt, automatisch Folgendes tun:

- Zeigt ein kontextsensitives Hilfe-Popup für Steuerelemente an
- CHM-Datei basierend auf Kontext öffnen (Inhaltsverzeichnis anzeigen, Stichwort oder Index anzeigen, Thema anzeigen)
- Navigieren Sie mit einem Standardbrowser zu einer URL

Hilfe Klasse

Sie können die `Help` Klasse im Code verwenden, um diese Art von Hilfe bereitzustellen:

- Ein Hilfe-Popup für ein Steuerelement anzeigen
- CHM-Datei basierend auf Kontext öffnen (Inhaltsverzeichnis anzeigen, Stichwort oder Index anzeigen, Thema anzeigen)
- Navigieren Sie mit einem Standardbrowser zu einer URL

HelpRequested-Ereignis

Sie können das `HelpRequested` Ereignis von `Control` oder `Form`, um benutzerdefinierte Aktionen auszuführen, wenn der Benutzer `F1` drückt oder auf die Schaltfläche Hilfe des Formulars klickt.

Hilfeschatfläche des Formulars

Sie können das `Form`, dass die Hilfe-Schaltfläche in der Titelleiste angezeigt wird. Wenn der Benutzer auf die Schaltfläche "Hilfe" klickt, ändert sich der Cursor zu einem `?` Cursor und nach einem Klick auf einen beliebigen Punkt wird jede kontextabhängige Hilfe `HelpProvider`, die dem Steuerelement mithilfe von `HelpProvider` ist.

Hilfeschatfläche von MessageBox und CommonDialogs

Sie können Hilfe für `MessageBox` , `OpenFileDialog` , `SaveDialog` und `ColorDialog` über die Schaltfläche Hilfe der Komponenten `ColorDialog` .

QuickInfo-Komponente

Sie können die `ToolTip` Komponente verwenden, um `ToolTip` anzuzeigen, wenn der Benutzer auf Steuerelemente zeigt. Eine `ToolTip` kann mit jedem Steuerelement `ToolTip` werden.

Hinweis

`HelpProvider` und `Help` Klasse verwenden Sie können kompilierte Hilfedateien (.chm) oder HTML-Dateien im HTML-Hilfeformat anzeigen. Kompilierte Hilfedateien enthalten ein Inhaltsverzeichnis, einen Index, Suchfunktionen und Stichwortverknüpfungen auf Seiten. Verknüpfungen funktionieren nur in kompilierten Hilfedateien. Sie können HTML Help 1.x-Dateien mithilfe von HTML Help Workshop generieren. Weitere Informationen zur HTML-Hilfe finden Sie unter "HTML Help Workshop" und anderen HTML-Hilfethemen in der [Microsoft HTML-Hilfe](#) .

Examples

Hilfedatei anzeigen

Die `Help Class` kapselt die HTML Help 1.0-Engine. Sie können das Hilfeobjekt verwenden, um kompilierte Hilfedateien (.chm) oder HTML-Dateien im HTML-Hilfeformat anzuzeigen. Kompilierte Hilfedateien enthalten Inhaltsverzeichnisse, Index-, Such- und Stichwortlinks auf Seiten. Verknüpfungen funktionieren nur in kompilierten Hilfedateien. Sie können HTML Help 1.x-Dateien mit einem kostenlosen Tool von Microsoft namens `HTML Help Workshop` generieren.

Eine einfache Möglichkeit, eine kompilierte Hilfedatei in einem zweiten Fenster anzuzeigen:

C

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

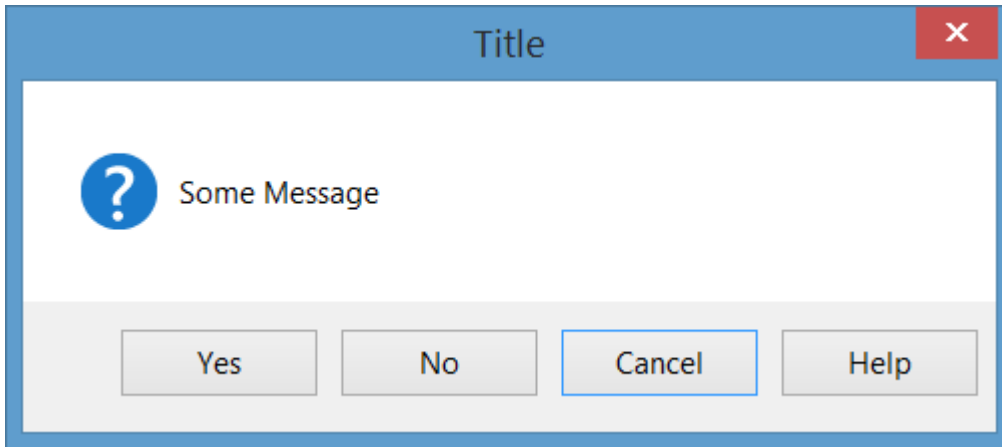
VB.NET

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

Hilfe für `MessageBox` anzeigen

Sie können auf verschiedene Arten Hilfe für das Meldungsfeld bereitstellen. Sie können eine `MessageBox` so konfigurieren, dass eine `Help` Schaltfläche `MessageBox` oder nicht. Sie können `MessageBox` auch so konfigurieren, dass der Benutzer, wenn er Hilfe anfordert, durch Klicken auf die Schaltfläche Hilfe oder durch Drücken von `F1` eine CHM-Datei anzeigt oder zu einer URL navigiert oder eine benutzerdefinierte Aktion ausführt. Hier sind einige Beispiele in diesem Thema.

In allen folgenden Beispielen würde die `MessageBox` folgendermaßen aussehen:



Zeigen Sie eine CHM-Datei an und navigieren Sie zu einem Stichwort (Index).

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

Zeigen Sie eine CHM-Datei an und navigieren Sie zu einem Thema

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

Zeigen Sie eine CHM-Datei an und navigieren Sie zur ersten Hilfeseite im Inhaltsverzeichnis

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

Öffnen Sie den Standardbrowser und navigieren Sie zu einer URL

```
MessageBox.Show("Some Message", "Title", MessageBoxButtons.YesNoCancel,  
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button3, 0,  
    "http://example.com");
```

Führen Sie eine benutzerdefinierte Aktion aus, wenn Sie die Hilfetaste oder die Taste F1 drücken

In diesem Fall sollten Sie das `HelpRequested` Ereignis des übergeordneten `HelpRequested` von `MessageBox` und eine benutzerdefinierte Operation ausführen:

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

Dann können Sie die `MessageBox` mit Hilfe der Schaltfläche `MessageBox` :

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

Oder zeigen Sie es ohne Hilfe-Button:

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

Hilfe für CommonDialogs anzeigen

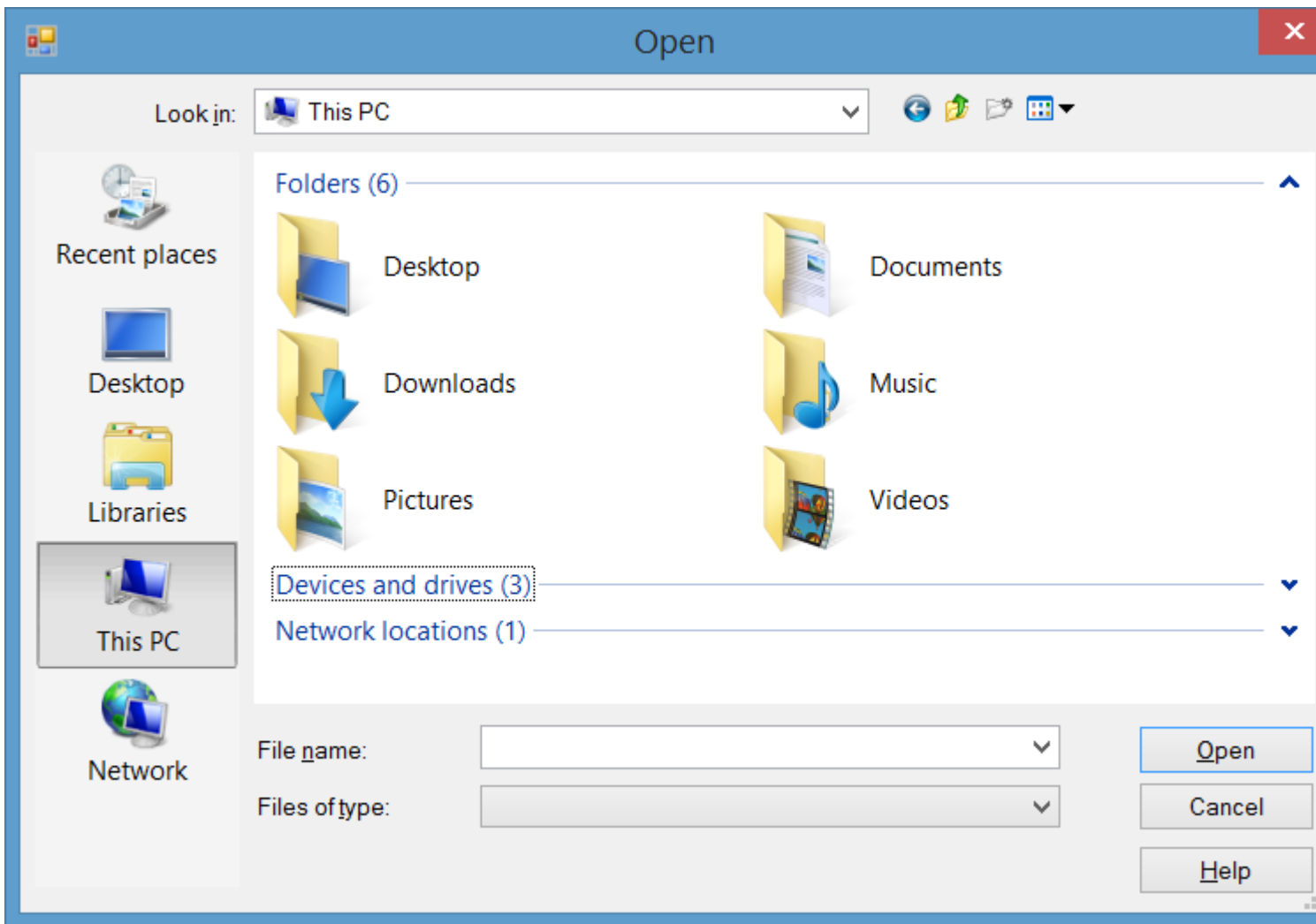
Sie können Hilfe für `OpenFileDialog` , `SaveFileDialog` und `ColorDialog` . Setzen Sie dazu die `ShowHelp` Eigenschaft des Dialogfelds auf `true` und behandeln Sie das `HelpRequest` Ereignis für das Dialogfeld:

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

Hinweis

- Das Ereignis wird nur `ShowHelp` , wenn Sie `ShowHelp` auf `true` .
- Das Ereignis wird nur durch Klicken auf die Schaltfläche `Help` ausgelöst und nicht mit der Taste F1 erhöht.

Im Bild unten sehen Sie einen `OpenFileDialog` mit einer Hilfe-Schaltfläche:



Behandeln des HelpRequested-Ereignisses von Steuerelementen und Formular

Wenn ein Benutzer **F1** an einem Steuerelement drückt oder auf die Schaltfläche Hilfe des Formulars (?) `HelpRequested` und dann auf ein Steuerelement `HelpRequested` , wird das `HelpRequested` Ereignis `HelpRequested` .

Sie können dieses Ereignis behandeln, um eine benutzerdefinierte Aktion bereitzustellen, wenn der Benutzer Hilfe zu Steuerelementen oder Formular anfordert.

Der `HelpRequested` unterstützt den Bubble-Up-Mechanismus. Es wird für Ihr aktives Steuerelement `Handled` Wenn Sie das Ereignis nicht behandeln und die Eigenschaft `Handled` des Ereignisses `arg` nicht auf `true` , wird die übergeordnete Steuerelementhierarchie bis zum Formular `Handled` .

Wenn Sie beispielsweise das `HelpRequested` Ereignis des Formulars wie `HelpRequested` behandeln, wird beim Drücken von **F1** ein Meldungsfeld mit dem Namen des aktiven Steuerelements `textBox1` Bei `textBox1` wird jedoch eine andere Meldung `textBox1` :

```
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");
    }

```

Sie können jede andere benutzerdefinierte Aktion ausführen, z. B. das Navigieren zu einer URL oder das Anzeigen einer CHM-Datei mithilfe der [Help](#) .

Hilfe mit Hilfe der Klasse anzeigen

Sie können die [Help](#) Klasse im Code verwenden, um diese Art von Hilfe bereitzustellen:

- Ein Hilfe-Popup für ein Steuerelement anzeigen
- CHM-Datei basierend auf Kontext öffnen (Inhaltsverzeichnis anzeigen, Stichwort oder Index anzeigen, Thema anzeigen)
- Navigieren Sie mit einem Standardbrowser zu einer URL

Hilfe-Popup-Fenster anzeigen

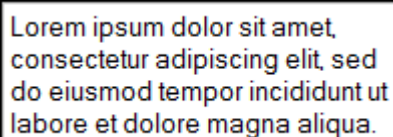
Sie können [Help.ShowPopup](#) , um ein Hilfe-Popup-Fenster anzuzeigen:

```

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));
}

```

Es wird ein solches Hilfefenster an Ihrer Mauszeigerposition angezeigt:



Lorem ipsum dolor sit amet,
 consectetur adipiscing elit, sed
 do eiusmod tempor incididunt ut
 labore et dolore magna aliqua.

CHM-Hilfedatei anzeigen

Sie können verschiedene Überladungen der [Help.ShowHelp](#) Methode verwenden, um eine CHM-Datei [Help.ShowHelp](#) und zu einem Stichwort, einem Thema, einem Index oder einem Inhaltsverzeichnis zu navigieren:

Hilfeinhaltsverzeichnis anzeigen

```

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

```

Hilfe für ein bestimmtes Keyword anzeigen (Index)

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

Hilfe zu einem bestimmten Thema anzeigen

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

URL anzeigen

Sie können jede URL im Standardbrowser mithilfe der `ShowHelp` Methode `ShowHelp` :

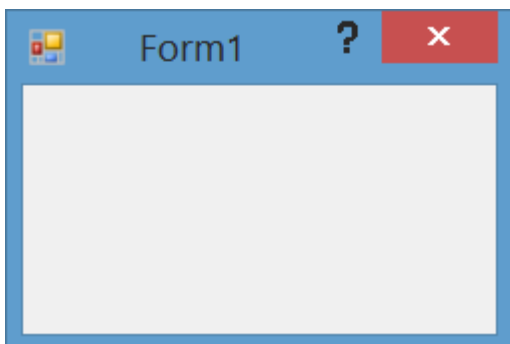
```
Help.ShowHelp(this, "Http://example.com");
```

Hilfeschaltfläche in der Titelleiste des Formulars anzeigen

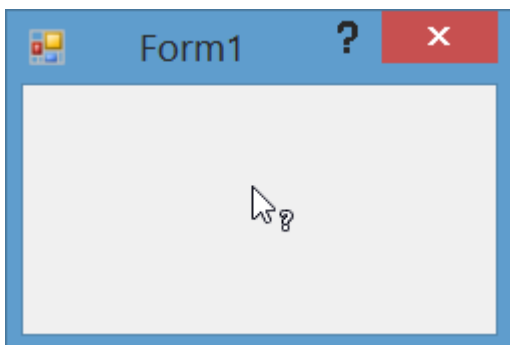
Sie können eine Hilfeschaltfläche in der Titelleiste eines `Form` . Um dies zu tun, sollten Sie:

1. Setzen Sie die `HelpButton` Eigenschaft des Formulars auf `true` .
2. Setzen Sie `MinimizeBox` und `MaximizeBox` auf `false` .

Dann erscheint eine Hilfeschaltfläche in der Titelleiste von `Form` :



Wenn Sie auf die Schaltfläche Hilfe klicken, wird der Cursor in ein ? Mauszeiger:



Dann , wenn Sie auf einer klicken `Control` oder `Form` , das `HelpRequested` wird Ereignis ausgelöst werden , und auch , wenn Sie Setup eine `HelpProvider` , die Hilfe für die Steuerung wird mit angezeigt `HelpProvider` .

Erstellen Sie eine benutzerdefinierte Hilfeschnittfläche, die sich wie eine Standard-Formularhilfeschnittfläche verhält

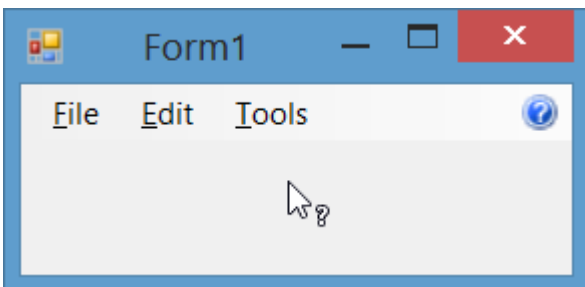
Wenn Sie ein `Form` mit `MinimizeBox` und `MaximizeBox` auf `true`, können Sie die Hilfe-Schnittfläche nicht in der Titelleiste des `Form` anzeigen. Dadurch wird die Funktion des Klickens auf die Hilfe-Schnittfläche verloren, um den Cursor zu konvertieren, damit der Cursor auf die Steuerelemente klicken kann Zeig Hilfe.

Sie können einen Menüeintrag in `MenuStrip` wie eine Standard- `MenuStrip` . `MenuStrip` dazu dem Formular ein `ToolStripMenuItem` hinzu, fügen Sie ein `MenuStrip` und `ToolStripMenuItem` Sie dann das `Click` Ereignis des Elements:

```
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);
}
```

Hinweis: Wenn Sie einen `Button` möchten, müssen Sie auch `button1.Capture = false;` vor dem Senden der Nachricht. Für ein `ToolStripMenuItem` ist dies jedoch nicht erforderlich.

Wenn Sie dann auf das Hilfemenü klicken, ändert sich der Cursor in `? Cursor` und verhält sich wie beim Klicken auf die Standard-Hilfe-Schnittfläche:



Behandeln des HelpButtonClicked-Ereignisses von Form

Sie können erkennen, wann ein Benutzer auf einen `HelpButton` in der Titelleiste des Formulars `HelpButtonClicked` indem Sie `HelpButtonClicked` . Sie können das Ereignis fortsetzen oder abbrechen, indem Sie die `Cancel` Eigenschaft der Ereignisargumente auf `true` .

```
private void Form1_HelpButtonClicked(object sender, CancelEventArgs e)
{
    e.Cancel = true;
    //Perform some custom action
    MessageBox.Show("Some Custom Help");
}
```

Hilfe bei der Integration online lesen: <https://riptutorial.com/de/winforms/topic/3285/hilfe-bei-der-integration>

Kapitel 6: Steuerelemente übernehmen

Bemerkungen

Steuerelemente werden genauso abgeleitet wie andere Klassen. Das einzige, worauf Sie achten sollten, ist das Überschreiben von Ereignissen: Es ist normalerweise ratsam, dass Sie den Basisereignishandler nach Ihrem eigenen aufrufen. Meine Faustregel: Rufen Sie im Zweifelsfall das Basisereignis an.

Examples

Anwendungsweite Einstellungen

Ein kurzer Blick auf die meisten Entwicklerseiten zeigt, dass WinForms der WPF unterlegen ist. Einer der am häufigsten genannten Gründe ist die vermeintliche Schwierigkeit, anwendungsweite Änderungen am "Look & Feel" einer gesamten Anwendung vorzunehmen.

Es ist tatsächlich überraschend einfach, eine Anwendung in WinForms zu erstellen, die sowohl zur Entwurfszeit als auch zur Laufzeit leicht konfigurierbar ist, wenn Sie einfach die Verwendung der Standardsteuerelemente meiden und von ihnen ableiten.

Nehmen Sie die `TextBox` als Beispiel. Es ist schwer vorstellbar, dass eine Windows-Anwendung zu irgendeinem Zeitpunkt nicht die Verwendung einer `TextBox` erfordert. Daher ist es immer sinnvoll, eine eigene `TextBox` zu haben. Nehmen Sie das folgende Beispiel:

```
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);
        }
    }
}
```

```
}
```

Benutzer, die in einem Dateneingabeformular mit vielen Eingabefeldern am hilfreichsten sind, sind die Hintergrundfarbe des Felds mit Fokusänderung. Sichtbar ist es leichter zu sehen als ein normaler blinkender vertikaler Cursor. Der obige Code enthält eine TextBox, die genau das tut.

Dabei nutzt er die statischen Eigenschaften einer statischen Klasse. Ich gebe unten einen Auszug aus meinem:

```
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 formBackColor;
        private static int formForeColor;
        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();
                    formForeColor = int.Parse(kSOPrefs.GetValue("FormForeColor", "-2147483630").ToString());
                    formBackColor = 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\\";
                    formForeColor = -2147483630;
                    formBackColor = -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);
        }
    }
}
}
}

```

Diese Klasse verwendet die Windows-Registrierung, um die Eigenschaften beizubehalten. Sie können jedoch auch eine Datenbank oder eine Einstellungsdatei verwenden. Der Vorteil der Verwendung einer statischen Klasse auf diese Weise ist, dass anwendungsweite Änderungen nicht nur zur Entwurfszeit, sondern auch vom Benutzer zur Laufzeit vorgenommen werden können. Ich füge in meinen Anwendungen immer ein Formular hinzu, mit dem der Benutzer die bevorzugten Werte ändern kann. Die Sicherungsfunktion speichert nicht nur in der Registry (oder in der Datenbank usw.), sondern sie ändert auch zur Laufzeit die Eigenschaften der statischen Klasse. Beachten Sie, dass statische Eigenschaften einer statischen Klasse nicht konstant sind. In diesem Sinne können sie als anwendungsweite Variablen betrachtet werden. Dies bedeutet, dass jedes nach den gespeicherten Änderungen geöffnete Formular sofort von den gespeicherten Änderungen beeinflusst wird.

Sie können sich leicht andere anwendungsweite Eigenschaften vorstellen, die Sie auf die gleiche Weise konfigurieren möchten. Schriftarten sind ein weiteres sehr gutes Beispiel.

NumberBox

Oft möchten Sie ein Eingabefeld, das nur Zahlen enthält. Durch Ableitung von den Standardsteuerelementen kann dies leicht erreicht werden, zum Beispiel:

```

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) != 1)
        {
            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;
}
}

```

Diese Klasse beschränkt sich nicht nur auf Zahlen, sondern bietet auch einige Besonderheiten. Es macht eine Eigenschaft Value verfügbar, die den doppelten Wert der Zahl darstellt, formatiert den Text, optional mit Tausendertrennzeichen, und ermöglicht die Eingabe großer Zahlen in kurzen Zügen: 10 Millionen erweitert sich auf 1000000,00 (die Anzahl der Dezimalstellen ist eine

Eigenschaft). Der Kürze halber wurden die Dezimal- und Tausendertrennzeichen hartcodiert. In einem Produktionssystem sind dies auch Benutzervorlieben.

Steuerelemente übernehmen online lesen:

<https://riptutorial.com/de/winforms/topic/6476/steuerelemente-ubernehmen>

Kapitel 7: Textfeld

Examples

Automatische Vervollständigung aus einer Sammlung von Zeichenfolgen

```
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);
```

Dies wird **automatisch abgeschlossen**, wenn der Benutzer versucht, **G** oder **L einzugeben** .

`AutoCompleteMode.SuggestAppend` wird sowohl eine Liste der vorgeschlagenen Werte angezeigt werden, und es wird das erste Spiel automatisch eingeben, `Append` nur und `Suggest` nur verfügbar sind, auch.

Erlaube nur Ziffern im Text

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

Dies erlaubt nur die Verwendung von Ziffern und Steuerzeichen in der `TextBox` . Andere Kombinationen sind möglich, `TextBox` die Eigenschaft `Handle` auf `true` setzen, um den Text zu blockieren.

Der Benutzer kann weiterhin unerwünschte Zeichen kopieren / einfügen. `TextChanged` sollte der `TextChanged` zusätzlich überprüft werden, um die Eingabe zu `TextChanged` :

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

In diesem Beispiel wird ein **regulärer Ausdruck** verwendet, um den Text zu filtern.

`NumericUpDown` sollte möglichst für Zahlen bevorzugt werden.

So blättern Sie bis zum Ende

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

Nach demselben Prinzip kann `SelectionStart` auf 0 gesetzt werden, um nach oben zu scrollen, oder zu einer bestimmten Zahl, um zu einem bestimmten Zeichen zu gelangen.

Platzhalter in ein Textfeld einfügen

Dieser Code setzt den *Hinweistext* beim Laden des Formulars ein und ändert ihn wie folgt:

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
```

Textfeld online lesen: <https://riptutorial.com/de/winforms/topic/4674/textfeld>

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit winforms	4444 , Bjørn-Roger Kringsjå , Chris Shao , Cody Gray , Community , Reza Aghaei
2	Datenbindung	Kai Thoma
3	Ein Formular anzeigen	Cody Gray , Jeff Bridgman , Steve
4	Grundlegende Bedienelemente	Aimnox , Berkay , help-info.de , Jeff Bridgman
5	Hilfe bei der Integration	help-info.de , Reza Aghaei
6	Steuerelemente übernehmen	Balagurunathan Marimuthu
7	Textfeld	gplumb , Jones Joseph , Stefano d'Antonio