eBook Gratuit

APPRENEZ winforms

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#winforms

Table des matières

À propos

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

Chapitre 1: Démarrer avec winforms

Remarques

Windows Forms ("WinForms" en abrégé) est une bibliothèque de classes d'interface graphique intégrée au .NET Framework. Il s'agit d'un wrapper orienté objet sophistiqué autour de l' API Win32 , permettant le développement d'applications de bureau et d'applications mobiles Windows ciblant le .NET Framework .

WinForms est principalement axé sur les événements . Une application se compose de plusieurs *formulaires* (affichés sous forme de fenêtres à l'écran), qui contiennent des *commandes* (étiquettes, boutons, zones de texte, listes, etc.) avec lesquelles l'utilisateur interagit directement. En réponse à l'interaction de l'utilisateur, ces contrôles déclenchent des événements pouvant être traités par le programme pour effectuer des tâches.

Comme dans Windows, tout dans WinForms est un contrôle, qui est lui-même un type de fenêtre. La classe Control de base fournit des fonctionnalités de base, notamment des propriétés pour la définition du texte, l'emplacement, la taille et la couleur, ainsi qu'un ensemble commun d'événements pouvant être gérés. Tous les contrôles dérivent de la classe Control, ajoutant des fonctionnalités supplémentaires. Certains contrôles peuvent héberger d'autres contrôles, que ce soit pour la réutilisabilité (`Form` , `UserControl`) ou la disposition (`TableLayoutPanel` , `FlowLayoutPanel`).

WinForms a été pris en charge depuis la version originale de .NET Framework (v1.0) et est toujours disponible dans les versions modernes (v4.5). Cependant, il n'est plus en développement actif et aucune nouvelle fonctionnalité n'est ajoutée. Selon 9 développeurs Microsoft lors de la conférence Build 2014 :

> Windows Forms continue à être pris en charge, mais en mode maintenance. Ils corrigeront les bogues au fur et à mesure qu'ils seront découverts, mais les nouvelles fonctionnalités ne sont plus à l'ordre du jour.

La bibliothèque Mono Open Source multiplate-forme fournit une implémentation de base de Windows Forms, prenant en charge toutes les fonctionnalités de Microsoft .NET 2.0. Cependant, WinForms n'est pas activement développé sur Mono et une implémentation complète est considérée comme impossible, compte tenu du lien inextricable entre le framework et l'API Windows native (qui n'est pas disponible sur les autres plates-formes).

Voir également:

- Documentation Windows Forms sur MSDN

Examples

Création d'une application WinForms simple à l'aide de Visual Studio

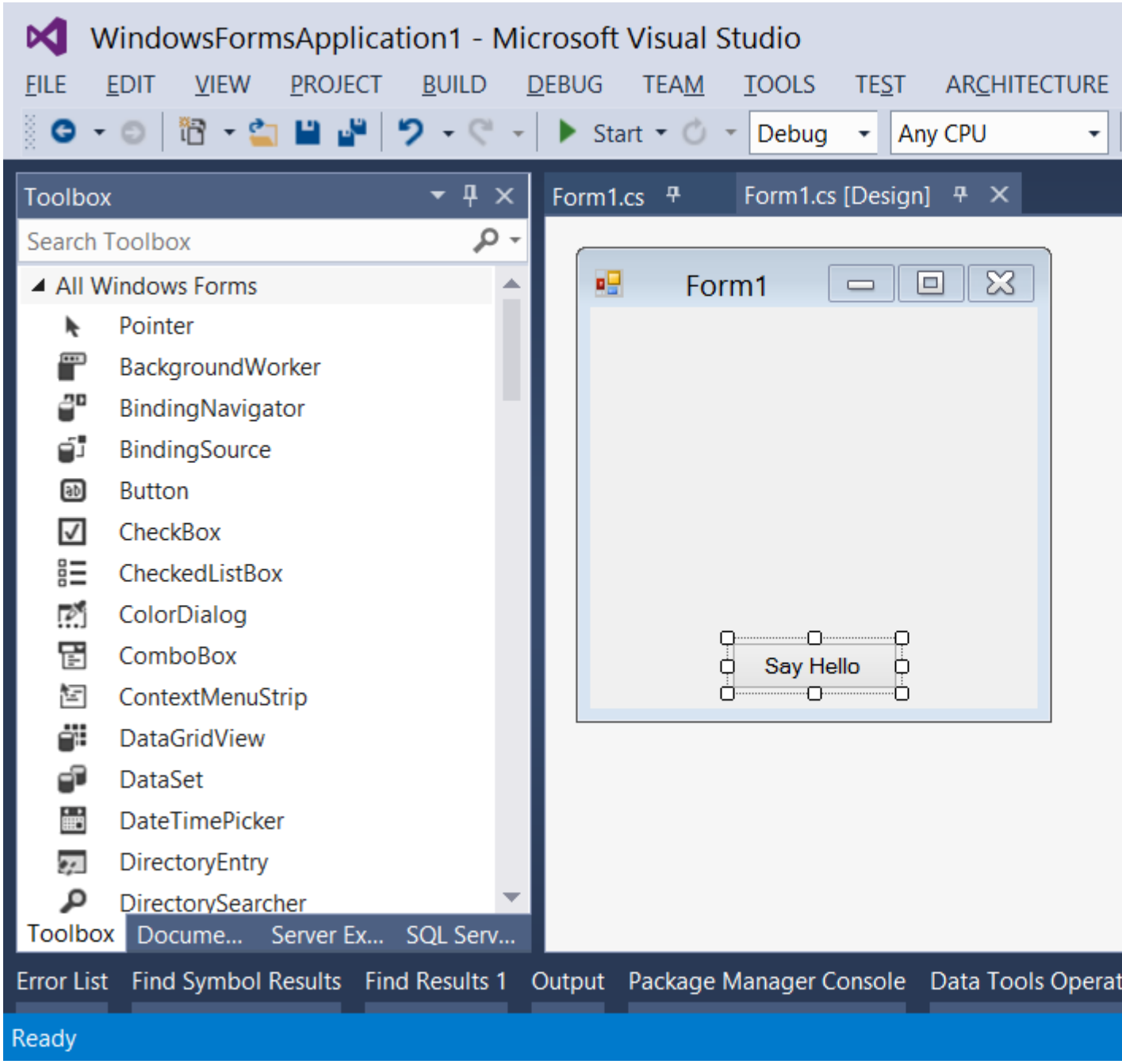
Cet exemple vous montrera comment créer un projet d'application Windows Forms dans Visual Studio.

Créer un projet Windows Forms

1. Démarrez Visual Studio.
2. Dans le menu **Fichier** , pointez sur **Nouveau** , puis sélectionnez **Projet** . La boîte de dialogue **Nouveau projet** apparaît.
3. Dans le volet **Modèles installés** , sélectionnez "Visual C #" ou "Visual Basic".
4. Au-dessus du volet du milieu, vous pouvez sélectionner l’infrastructure cible dans la liste déroulante.
5. Dans le volet du milieu, sélectionnez le modèle **Application Windows Forms** .
6. Dans la zone de texte **Nom** , tapez un nom pour le projet.
7. Dans la zone de texte **Emplacement** , choisissez un dossier pour enregistrer le projet.
8. Cliquez sur **OK** .
9. Le Concepteur Windows Forms s'ouvre et affiche **Form1** du projet.

Ajouter des contrôles au formulaire

1. Dans la palette **Boîte à outils** , faites glisser un contrôle **Button** sur le formulaire.
2. Cliquez sur le bouton pour le sélectionner. Dans la fenêtre Propriétés, définissez la propriété `Text` sur **Say Hello** .



Ecrire un code

1. Double-cliquez sur le bouton pour ajouter un gestionnaire d'événements pour l'événement `Click` . L'éditeur de code s'ouvrira avec le point d'insertion placé dans la fonction de gestionnaire d'événement.

2. Tapez le code suivant:

   C #

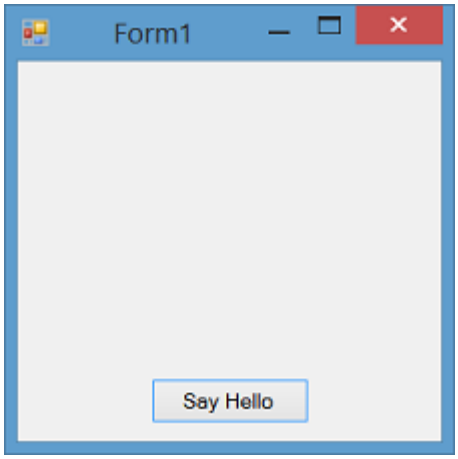
   ```
   MessageBox.Show("Hello, World!");
   ```

   VB.NET

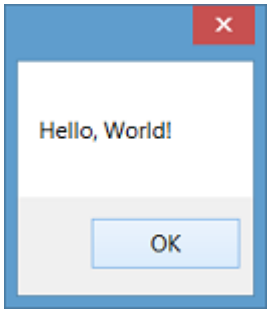
```
MessageBox.Show("Hello, World!")
```

Exécuter et tester

1. Appuyez sur `F5` pour exécuter l'application.



2. Lorsque votre application est en cours d'exécution, cliquez sur le bouton pour afficher "Hello, World!" message.



3. Fermez le formulaire pour revenir à Visual Studio.

Création d'une application C # WinForms simple à l'aide d'un éditeur de texte

1. Ouvrez un éditeur de texte (tel que le Bloc-notes) et tapez le code ci-dessous:

```
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. Enregistrez le fichier sur un chemin d'accès en lecture / écriture. Il est classique de nommer le fichier après la classe qu'il contient, par exemple `X:\MainForm.cs` .

3. Exécutez le compilateur C # à partir de la ligne de commande, en passant le chemin d'accès au fichier de code en tant qu'argument:

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

   Remarque: Pour utiliser une version du compilateur C # pour d'autres versions de .NET Framework, consultez le chemin d'accès, `%WINDIR%\Microsoft.NET` et modifiez l'exemple ci-dessus en conséquence. Pour plus d'informations sur la compilation des applications C #, voir Compiler et exécuter votre premier programme C # .

4. Une fois la compilation terminée, une application appelée `MainForm.exe` sera créée dans le même répertoire que votre fichier de code. Vous pouvez exécuter cette application à partir de la ligne de commande ou en double-cliquant dessus dans l'Explorateur.

Création d'une application simple VB.NET WinForms à l'aide d'un éditeur de texte

1. Ouvrez un éditeur de texte (tel que le Bloc-notes) et tapez le code ci-dessous:

```
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. Enregistrez le fichier sur un chemin d'accès en lecture / écriture. Il est classique de nommer le fichier après la classe qu'il contient, par exemple `X:\MainForm.vb` .

3. Exécutez le compilateur VB.NET à partir de la ligne de commande en transmettant le chemin d'accès au fichier de code en tant qu'argument:

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

   Remarque: Pour utiliser une version du compilateur VB.NET pour d'autres versions de .NET Framework, consultez le chemin d'accès `%WINDIR%\Microsoft.NET` et modifiez l'exemple ci-dessus en conséquence. Pour plus d'informations sur la compilation d'applications VB.NET,

voir Hello World .

4. Une fois la compilation terminée, une application appelée `MainForm.exe` sera créée dans le même répertoire que votre fichier de code. Vous pouvez exécuter cette application à partir de la ligne de commande ou en double-cliquant dessus dans l'Explorateur.

Lire Démarrer avec winforms en ligne: https://riptutorial.com/fr/winforms/topic/1018/demarrer-avec-winforms

Chapitre 2: Afficher un formulaire

Introduction

Cette rubrique explique comment le moteur WinForms fonctionne pour afficher les formulaires et contrôler leur durée de vie.

Examples

Montrer une forme modale ou modale

Après avoir défini la structure de votre formulaire avec le concepteur WinForms, vous pouvez afficher vos formulaires en code avec deux méthodes différentes.

- Method - Un formulaire sans modalité

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

- Méthode - Un dialogue modal

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

Les deux méthodes ont une distinction très importante. La première méthode (la non modale) affiche votre formulaire et retourne immédiatement sans attendre la fermeture du formulaire ouvert. Donc, votre code continue avec ce qui suit l'appel Show. La deuxième méthode à la place (la modale) ouvre le formulaire et bloque toute activité sur l'ensemble de l'application jusqu'à ce que vous fermiez le formulaire via le bouton Fermer ou avec certains boutons correctement configurés pour fermer le formulaire.

Fermeture d'un formulaire modeless

Un formulaire non modal est utilisé (généralement) lorsque vous devez afficher quelque chose de permanent à côté de l'écran principal de votre application (pensez à une légende ou à une vue d'un flux de données provenant de manière asynchrone d'un périphérique ou d'une fenêtre enfant MDI).
Mais une forme sans modèle pose un défi unique lorsque vous voulez la fermer. Comment récupérer l'instance et appeler la méthode Close dans cette instance?

Vous pouvez conserver une variable globale faisant référence à l'instance que vous souhaitez fermer.

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

Mais nous pouvons également choisir d'utiliser la collection Application.OpenForms où le moteur de formulaire stocke toutes les instances de formulaire créées et toujours ouvertes.

Vous pouvez extraire cette instance particulière de cette collection et appeler la méthode Close

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

Fermer une fiche modale

Lorsqu'un formulaire est affiché à l'aide de la méthode `ShowDialog` , il est nécessaire de définir la propriété `DialogResult` du `DialogResult` pour fermer le formulaire. Cette propriété peut être définie à l'aide de l' énum appelé DialogResult .

Pour fermer un formulaire, il vous suffit de définir la propriété `DialogResult` du `DialogResult` (à une valeur quelconque par `DialogResult.None`) dans un gestionnaire d'événement. Lorsque votre code quitte le gestionnaire d'événements, le moteur WinForm masque le formulaire et le code qui suit l'appel de la méthode `ShowDialog` initiale continue son exécution.

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

Le code appelant peut capturer la valeur de retour de ShowDialog pour déterminer quel bouton l'utilisateur a cliqué dans le formulaire. Lorsqu'il est affiché à l'aide de `ShowDialog()` , le formulaire n'est pas éliminé automatiquement (puisqu'il était simplement masqué et non fermé), il est donc important d'utiliser un bloc `using` pour s'assurer que le formulaire est supprimé.

Vous trouverez ci-dessous un exemple de vérification du résultat de l'utilisation d' `OpenFileDialog` , de la vérification du résultat et de l'accès à une propriété à partir de la boîte de dialogue avant sa suppression.

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

Vous pouvez également définir la propriété `DialogResult` sur un bouton. En cliquant sur ce bouton, la propriété `DialogResult` du formulaire sera définie sur la valeur associée au bouton. Cela vous permet de fermer le formulaire sans ajouter de gestionnaire d'événements pour définir le `DialogResult` dans le code.

Par exemple, si vous ajoutez un bouton OK à votre formulaire et définissez sa propriété sur `DialogResult.OK` le formulaire se ferme automatiquement lorsque vous appuyez sur ce bouton et le code appelant reçoit un `DialogResult.OK` en retour de l'appel de méthode `ShowDialog()` .

Lire Afficher un formulaire en ligne: https://riptutorial.com/fr/winforms/topic/8768/afficher-un-formulaire

Chapitre 3: Aide à l'intégration

Remarques

Vous pouvez fournir une aide pour les formulaires et les contrôles dans une application Windows Forms de différentes manières. Vous pouvez afficher une aide contextuelle, ouvrir un fichier CHM ou une URL. Vous pouvez afficher une aide contextuelle pour les formulaires, les contrôles et les boîtes de dialogue.

Composant HelpProvider

Vous pouvez configurer un composant `HelpProvider` pour fournir une aide contextuelle pour le composant. De cette façon, lorsque l'utilisateur appuie sur la touche `F1` ou sur le bouton Aide du formulaire, vous pouvez automatiquement:

- Afficher une fenêtre contextuelle d'aide contextuelle pour les contrôles
- Ouvrez un fichier CHM en fonction du contexte (Afficher la table des matières, Afficher un mot-clé ou un index, afficher un sujet)
- Accédez à une URL à l'aide du navigateur par défaut

Classe d'aide

Vous pouvez utiliser la classe d' `Help` dans le code pour fournir ce type d'aide:

- Afficher une fenêtre d'aide pour un contrôle
- Ouvrez un fichier CHM en fonction du contexte (Afficher la table des matières, Afficher un mot-clé ou un index, afficher un sujet)
- Accédez à une URL à l'aide du navigateur par défaut

Evénement HelpRequested

Vous pouvez gérer l'événement `HelpRequested` des objets `Control` ou `Form` pour effectuer des actions personnalisées lorsque l'utilisateur appuie sur `F1` ou sur le bouton Aide du formulaire.

Bouton d'aide du formulaire

Vous pouvez configurer le `Form` pour afficher le bouton Aide sur la barre de titre. De cette façon, si l'utilisateur clique sur le bouton Aide, le curseur se transforme en un `?` Après avoir cliqué sur un point, toute aide contextuelle associée au contrôle utilisant `HelpProvider` sera `HelpProvider` .

Bouton d'aide de MessgeBox et CommonDialogs

Vous pouvez fournir une aide pour `MessageBox` , `OpenFileDialog` , `SaveDialog` et `ColorDialog` utilisant

le bouton Aide des composants.

Composant ToolTip

Vous pouvez utiliser le composant `ToolTip` pour afficher du texte d'aide lorsque l'utilisateur pointe des contrôles. Une `ToolTip` peut être associée à n'importe quel contrôle.

Remarque

Utilisation de `HelpProvider` et de la classe `Help` Vous pouvez afficher les fichiers d'aide compilés (.chm) ou les fichiers HTML au format HTML Help. Les fichiers d'aide compilés fournissent une table des matières, un index, une fonction de recherche et des liens de mots clés dans les pages. Les raccourcis ne fonctionnent que dans les fichiers d'aide compilés. Vous pouvez générer des fichiers HTML Help 1.x à l'aide de HTML Help Workshop. Pour plus d'informations sur l'aide HTML, voir "HTML Help Workshop" et d'autres rubriques d'aide HTML dans Microsoft HTML Help .

Examples

Afficher le fichier d'aide

La `Help Class` encapsule le moteur HTML Help 1.0. Vous pouvez utiliser l'objet d'aide pour afficher les fichiers d'aide compilés (.chm) ou les fichiers HTML au format HTML Help. Les fichiers d'aide compilés fournissent des tables des matières, des index, des recherches et des liens de mots clés dans les pages. Les raccourcis ne fonctionnent que dans les fichiers d'aide compilés. Vous pouvez générer des fichiers HTML Help 1.x avec un outil gratuit de Microsft appelé `HTML Help Workshop` .

Un moyen facile d'afficher un fichier d'aide compilé dans une seconde fenêtre:

C #

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

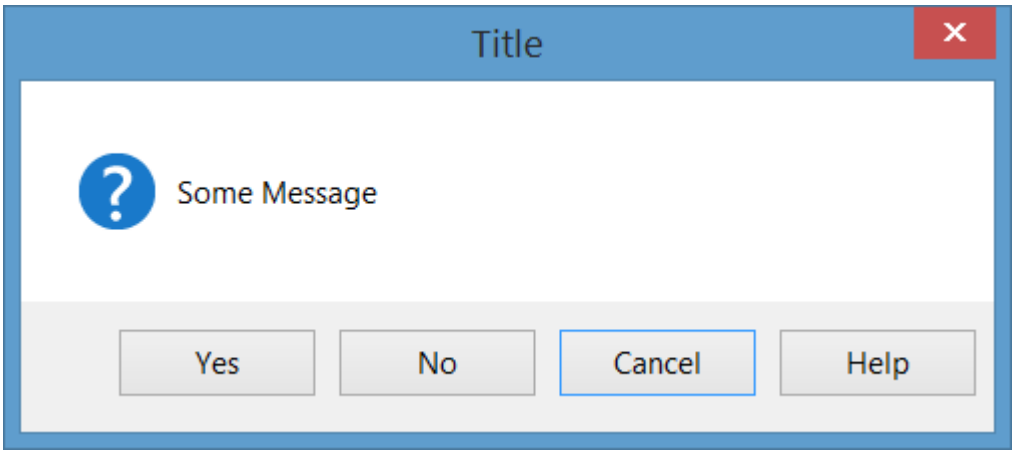
VB.NET

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

Afficher l'aide de MessageBox

Vous pouvez fournir de l'aide pour la boîte de message de différentes manières. Vous pouvez configurer un `MessageBox` pour afficher un bouton `Help` ou non. Vous pouvez également configurer `MessageBox` de manière à ce que lorsque l'utilisateur demande de l'aide en cliquant sur le bouton Aide ou en appuyant sur `F1` , il affiche un fichier CHM ou navigue vers une URL ou effectue une action personnalisée. Voici quelques exemples dans ce sujet.

Dans tous les exemples ci-dessous, le `MessageBox` serait comme ceci:



Afficher un fichier CHM et accéder à un mot clé (index)

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

Afficher un fichier CHM et accéder à un sujet

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

Afficher un fichier CHM et parcourir la première page d'aide dans la table des matières

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

Ouvrez le navigateur par défaut et accédez à une URL

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

Effectuez une action personnalisée lorsque vous appuyez sur le bouton Aide ou la touche F1

Dans ce cas, vous devez gérer l'événement `HelpRequested` du parent de `MessageBox` et effectuer une opération personnalisée:

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

```
{
    // Perform custom action, for example show a custom help form
    var f = new Form();
    f.ShowDialog();
}
```

Ensuite, vous pouvez afficher le bouton `MessageBox` avec l'aide:

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

Ou affichez-le sans bouton d'aide:

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

Afficher l'aide pour CommonDialogs

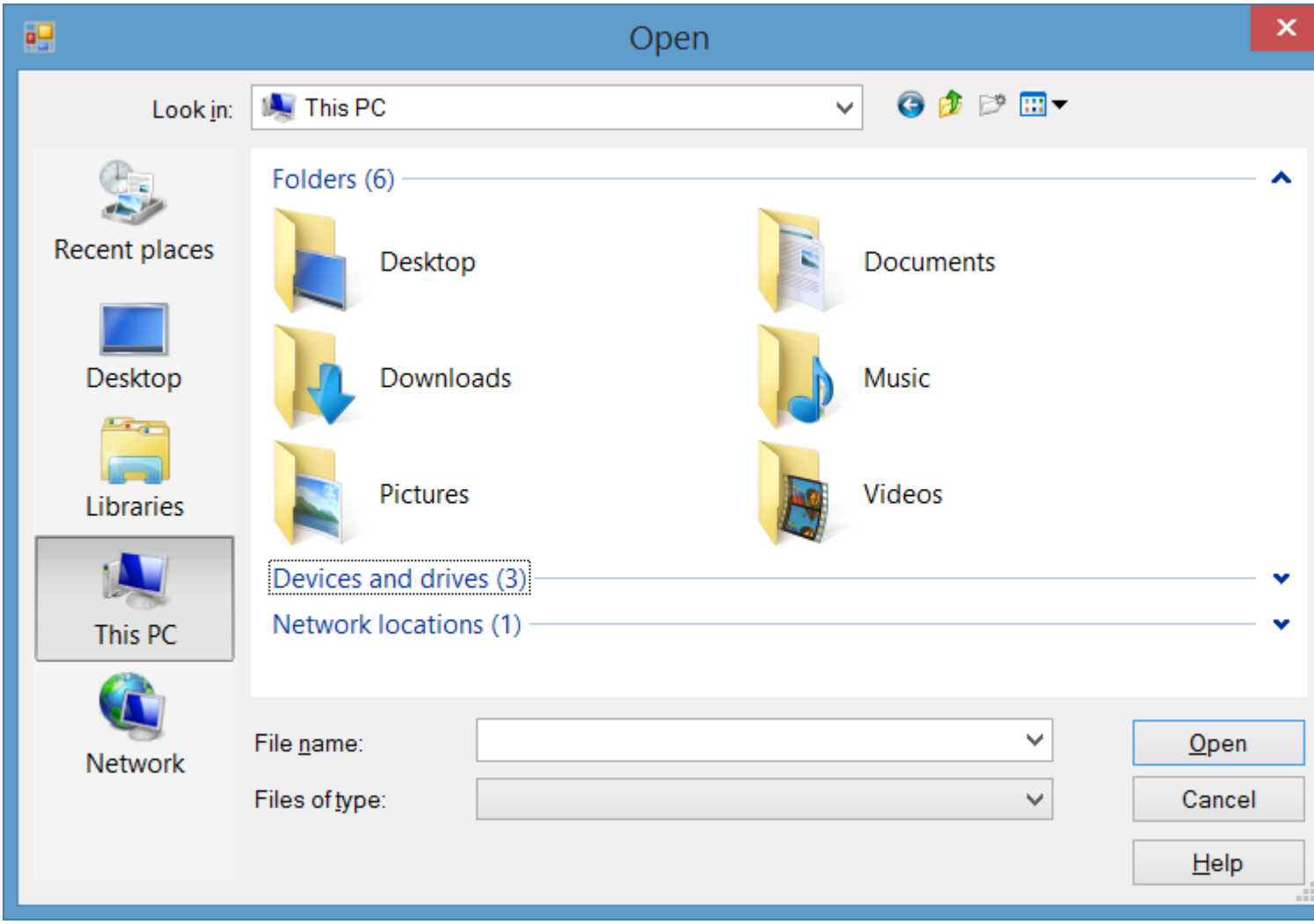
Vous pouvez fournir une aide pour `OpenFileDialog` , `SaveFileDialog` et `ColorDialog` . Pour ce faire, définissez la propriété `ShowHelp` de dialog sur `true` et gérez l'événement `HelpRequest` pour la boîte de dialogue:

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

Remarque

- L'événement sera déclenché uniquement si vous définissez `ShowHelp` sur `true` .
- L'événement ne sera déclenché que par un clic sur le bouton `Help` et ne sera pas déclenché avec la touche F1.

Dans l'image ci-dessous, vous pouvez voir un `OpenFileDialog` avec un bouton d'aide:



Gestion de l'événement EventRequested des contrôles et du formulaire

Lorsqu'un utilisateur appuie sur `F1` sur un contrôle ou clique sur le bouton Aide de formulaire (`?`) Puis clique sur un contrôle, l'événement `HelpRequested` .

Vous pouvez gérer cet événement pour fournir une action personnalisée lorsque l'utilisateur demande de l'aide pour des contrôles ou un formulaire.

Le `HelpRequested` prend en charge le mécanisme de création de bulles. Il se déclenche pour votre contrôle actif et si vous ne gérez pas l'événement et ne définissez pas la propriété `Handled` de son argument d'événement sur `true` , il se transforme en bulle jusqu'à la hiérarchie de contrôle parent.

Par exemple, si vous gérez l'événement `HelpRequested` du formulaire comme ci-dessous, lorsque vous appuyez sur `F1`, une boîte de message apparaîtra et affichera le nom du contrôle actif, mais pour `textBox1` il affichera un message différent:

```
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");
}
```

Vous pouvez effectuer toute autre action personnalisée, telle que la navigation vers une URL ou l'affichage d'un fichier CHM à l' `Help` classe d' `Help` .

Afficher l'aide à l'aide de la classe d'aide

Vous pouvez utiliser la classe d' `Help` dans le code pour fournir ce type d'aide:

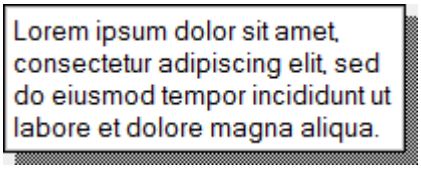
- Afficher une fenêtre d'aide pour un contrôle
- Ouvrez un fichier CHM en fonction du contexte (Afficher la table des matières, Afficher un mot-clé ou un index, afficher un sujet)
- Accédez à une URL à l'aide du navigateur par défaut

Afficher la fenêtre contextuelle Aide

Vous pouvez utiliser `Help.ShowPopup` pour afficher une fenêtre contextuelle d'aide:

```
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));
}
```

Il affichera cette aide contextuelle à l'emplacement du pointeur de la souris:



Afficher le fichier d'aide CHM

Vous pouvez utiliser différentes surcharges de la méthode `Help.ShowHelp` pour afficher un fichier CHM et accéder à un mot-clé, un sujet, un index ou une table de contenu:

Afficher la table des matières de l'aide

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

Afficher l'aide pour un mot clé spécifique (index)

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

Afficher l'aide pour un sujet spécifique

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

Afficher l'URL

Vous pouvez afficher n'importe quelle URL dans le navigateur par défaut à l'aide de la méthode `ShowHelp` :

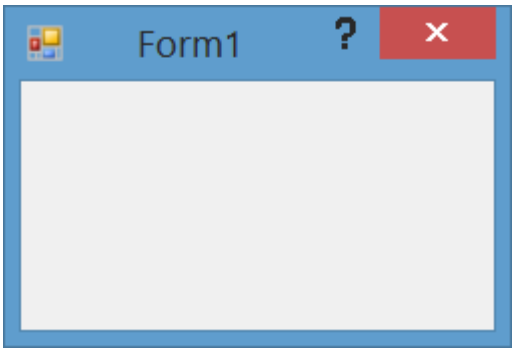
```
Help.ShowHelp(this, "Http://example.com");
```

Afficher le bouton d'aide sur la barre de titre du formulaire

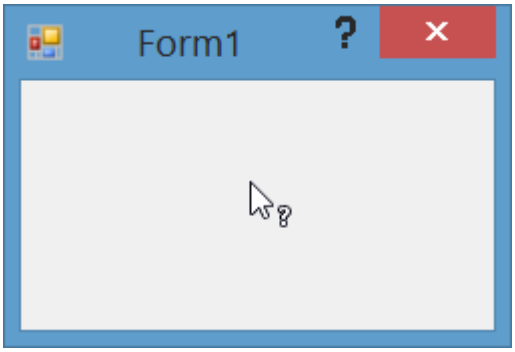
Vous pouvez afficher un bouton d'aide sur la barre de titre d'un `Form` . Pour ce faire, vous devriez:

1. Définissez la propriété `HelpButton` du formulaire sur `true` .
2. Définissez `MinimizeBox` et `MaximizeBox` sur `false` .

Ensuite, un bouton d'aide apparaîtra sur la barre de titre du `Form` :



De plus, lorsque vous cliquez sur le bouton Aide, le curseur devient un `?` le curseur:



Si vous cliquez ensuite sur un `Control` ou un `Form` , l'événement `HelpRequested` sera `HelpRequested` et si vous avez configuré un `HelpProvider` , l'aide du contrôle sera affichée à l'aide de `HelpProvider` .

Créer un bouton d'aide personnalisé qui agit comme un bouton d'aide

standard

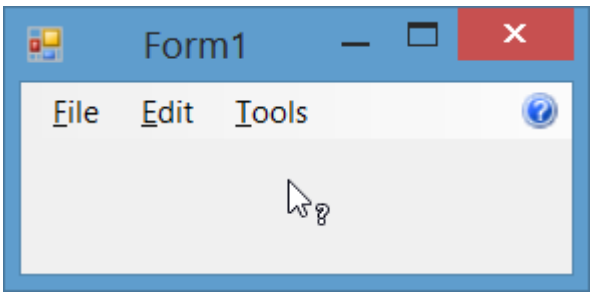
Si vous avez un `Form` avec `MinimizeBox` et `MaximizeBox` défini sur `true` , vous ne pouvez pas afficher le bouton Aide sur la barre de titre de `Form` et vous perdrez la fonction de cliquer sur le bouton d'aide pour le convertir afin de pouvoir cliquer sur les contrôles. montrer de l'aide.

Vous pouvez faire un élément de menu sur `MenuStrip` agir comme bouton d'aide standard. Pour ce faire, ajoutez un `MenuStrip` au formulaire et ajoutez-y un `ToolStripMenuItem` , puis gérez l'événement `Click` de l'élément:

```
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);
}
```

Remarque: Si vous voulez le faire en utilisant un `Button` , vous devez également définir `button1.Capture = false;` avant d'envoyer le message. Mais ce n'est pas nécessaire pour un `ToolStripMenuItem` .

Ensuite, lorsque vous cliquez sur le menu d'aide, le curseur sera remplacé par `?` curseur et agira comme lorsque vous cliquez sur le bouton d'aide standard:



Gestion de l'événement HelpButtonClicked de formulaire

Vous pouvez détecter lorsqu'un utilisateur a cliqué sur un `HelpButton` d' `HelpButton` sur la barre de titre du formulaire en gérant `HelpButtonClicked` . Vous pouvez laisser l'événement continuer ou l'annuler en définissant la propriété `Cancel` de ses arguments d'événement sur `true` .

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

Lire Aide à l'intégration en ligne: https://riptutorial.com/fr/winforms/topic/3285/aide-a-l-integration

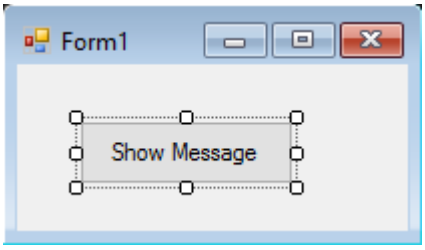
Chapitre 4: Contrôles de base

Examples

Bouton

Les boutons sont l'un des contrôles les plus simples et surtout utilisés pour exécuter du code lorsque l'utilisateur le souhaite.

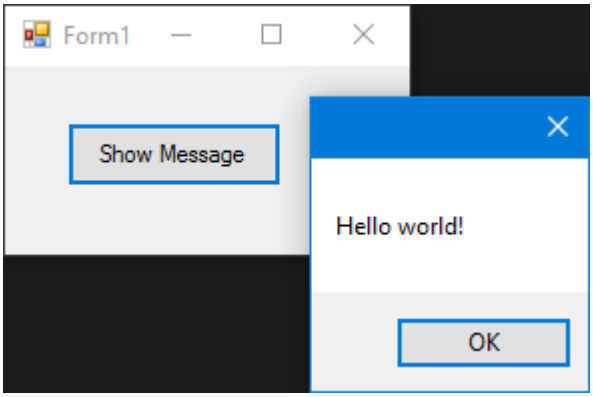
Ici, nous avons un cas très simple, afficher une boîte de message quand un bouton est cliqué. Nous ajoutons un bouton à un formulaire, nommez-le `cmdShowMessage` tel qu'utilisé dans le code pour identifier l'objet et définissez le texte des boutons sur Afficher le message.



Il suffit de double-cliquer sur le bouton du concepteur visuel et Visual Studio générera le code du clic d'événement. Il ne reste plus qu'à ajouter le code de la MessageBox:

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

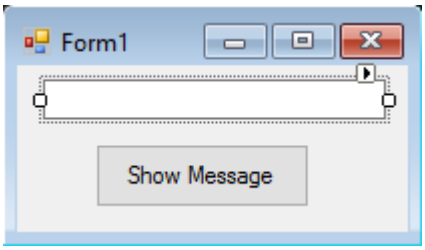
Si nous exécutons le programme maintenant et cliquez sur le bouton, le message apparaîtra:



Zone de texte

Les TextBox permettent à l'utilisateur d'entrer des données dans le programme.

Nous allons modifier le formulaire et ajouter une zone de texte afin que le messagebox affiche le message souhaité par l'utilisateur. Maintenant, notre formulaire ressemble à:

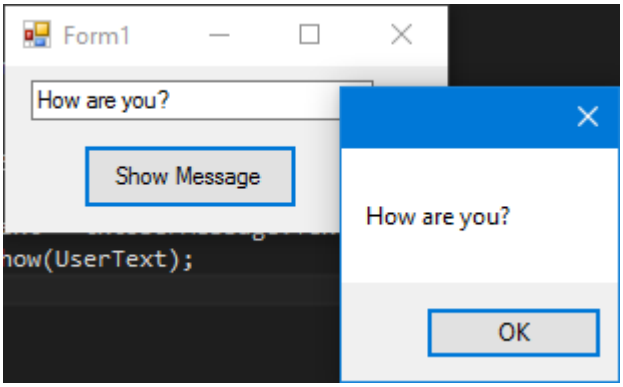


Et puis modifiez l'événement clic de bouton pour utiliser le texte de la zone de texte:

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

Comme vous pouvez le voir, nous utilisons la propriété `.Text` de la zone de texte qui est le texte contenu dans la texbox.

Si nous exécutons le programme, nous pourrons écrire dans la zone de texte. Lorsque nous cliquons sur le bouton, la MessageBox affiche le texte que nous avons écrit:

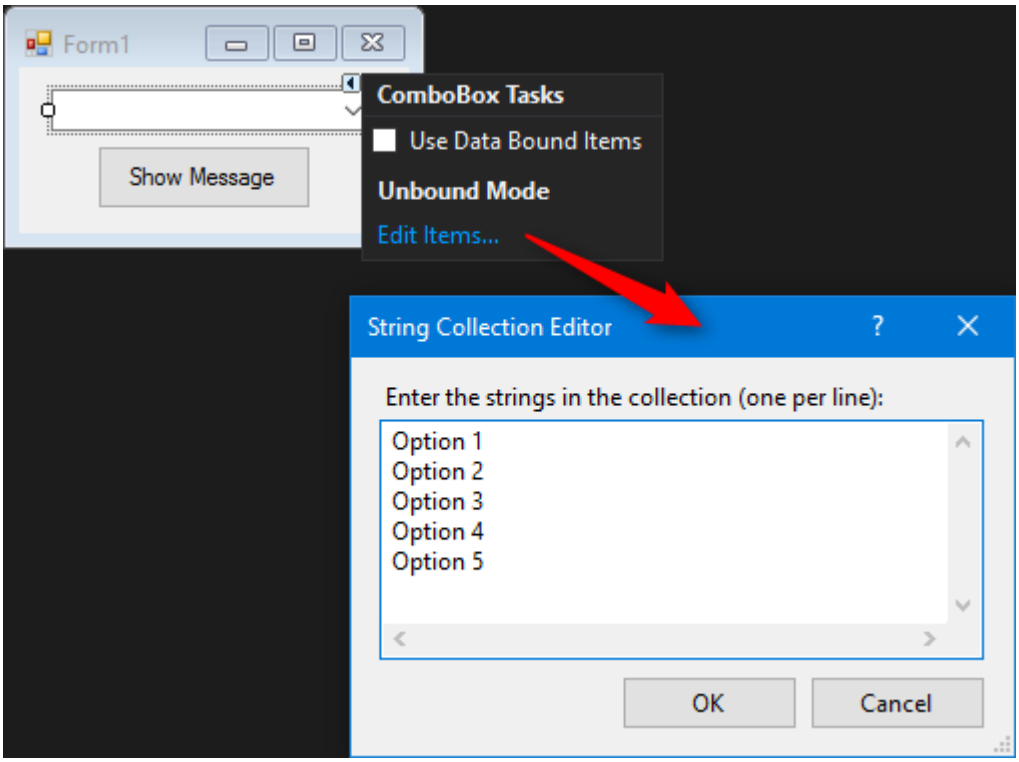


Boîte combo

ComboBoxes permettent à l'utilisateur de choisir l'une des différentes options fournies par le développeur.

Nous allons modifier le formulaire et ajouter une liste déroulante afin que le messagebox affiche le message que l'utilisateur souhaite recevoir dans une liste que nous allons fournir.

Après avoir ajouté le combo au formulaire, nous ajoutons maintenant une liste d'options au combo. Pour ce faire, nous devons modifier la propriété `Items` :

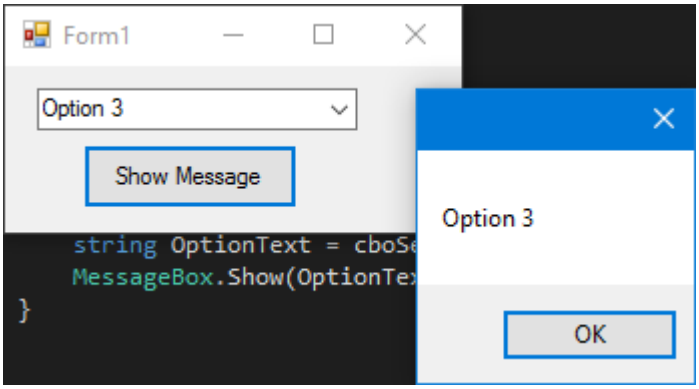


Maintenant, nous devons modifier le code de l'événement click:

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

Comme vous pouvez le voir, nous utilisons la propriété `SelectedItem` , elle contient l'objet de l'option sélectionnée. Comme nous avons besoin d'une chaîne à afficher et que le compilateur ne sait pas si l'objet est ou non une chaîne, nous devons utiliser la méthode `ToString()` .

Si nous exécutons le programme, nous serons en mesure de choisir l'option que nous préférons et lorsque nous cliquons sur le bouton, la boîte de message l'affichera:



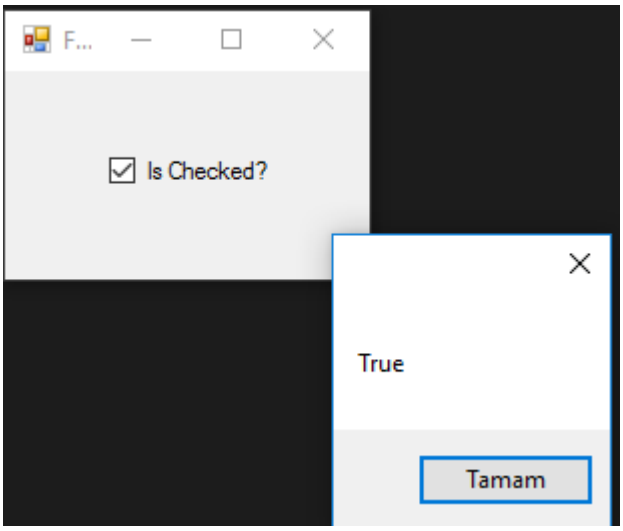
Pour être averti lorsqu'un utilisateur sélectionne un élément de la liste déroulante, utilisez l'événement `SelectionChangeCommitted` . Nous pourrions utiliser l'événement `SelectedIndexChanged` , mais cela se produit également lorsque nous modifions par programme l'élément de sélection dans la liste déroulante.

CheckBox

Checkbox est un contrôle qui permet à l'utilisateur d'obtenir `boolean` valeurs `boolean` de l'utilisateur pour une question spécifique telle que **"Êtes-vous d'accord?"** .

A un événement appelé `CheckedChanged` , qui se produit chaque fois que la propriété `check` est modifiée.

Voici un CheckBox qui a une question **"Est-ce vérifié?"** .



Nous avons eu cet événement `MessageBox` from `CheckedChanged` ,

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

Si **CheckBox** est coché -> la variable `IsChecked` sera `true` .

Si **CheckBox** n'est pas coché -> la variable `IsChecked` sera `false` .

ListBox

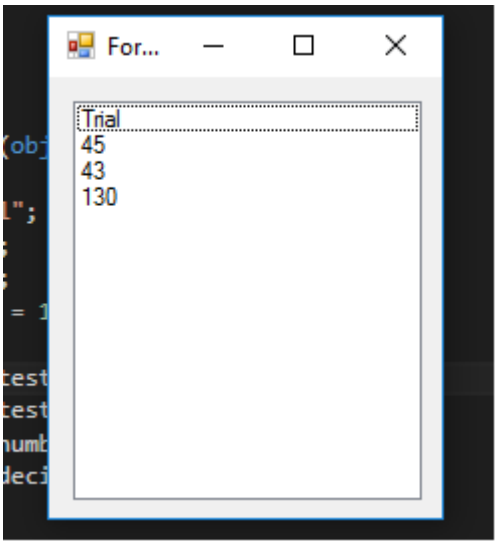
`Listbox` est un contrôle pouvant contenir une collection d'objets. `Listbox` est similaire à `Combobox` mais dans `Combobox` ; Seuls les éléments sélectionnés sont visibles pour l'utilisateur. Dans la `Listbox` tous les éléments sont visibles pour l'utilisateur.

Comment ajouter des éléments à 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);
        }
```

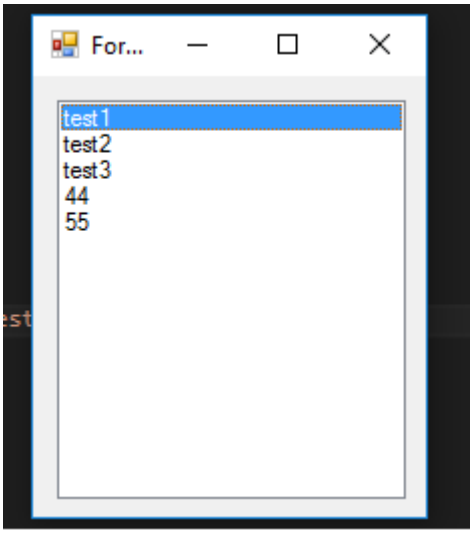
Sortie ;



Ou des `datasources` peuvent être données,

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

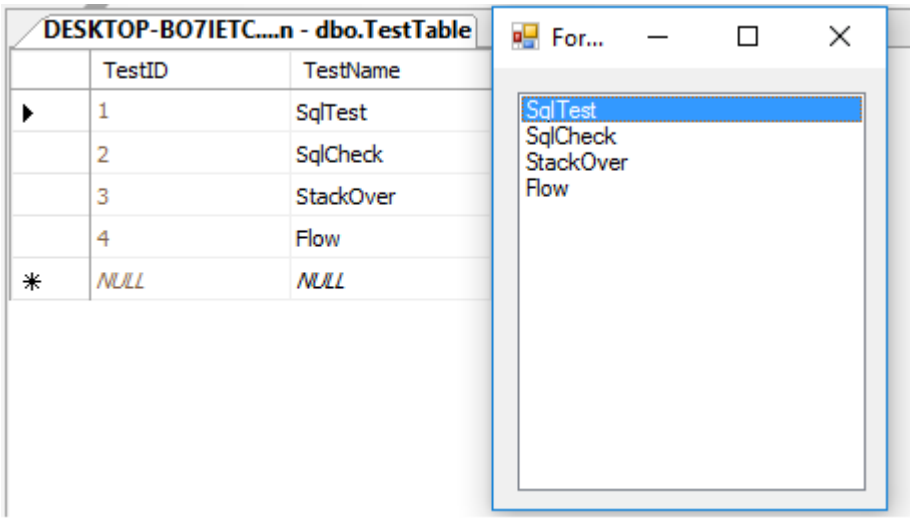
Sortie;




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

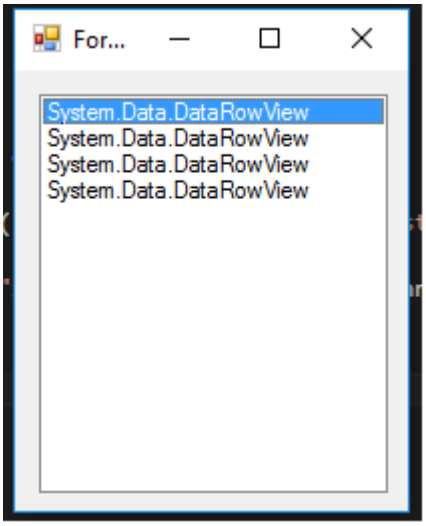
```
SQL(RDBMS)
            Connection.Open(); //Connection open
            SqlDataAdapter Adapter = new SqlDataAdapter("Select * From TestTable",
Connection); // Get all records from TestTable.
            DataTable DT = new DataTable();
            Adapter.Fill(DT); // Fill records to DataTable.
 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

La **sortie correcte** ;



Donner une source de données sql externe à la listbox requise, `ValueMember` et `DisplayMember`

Si ce n'est **pas le** cas,



Événements utiles

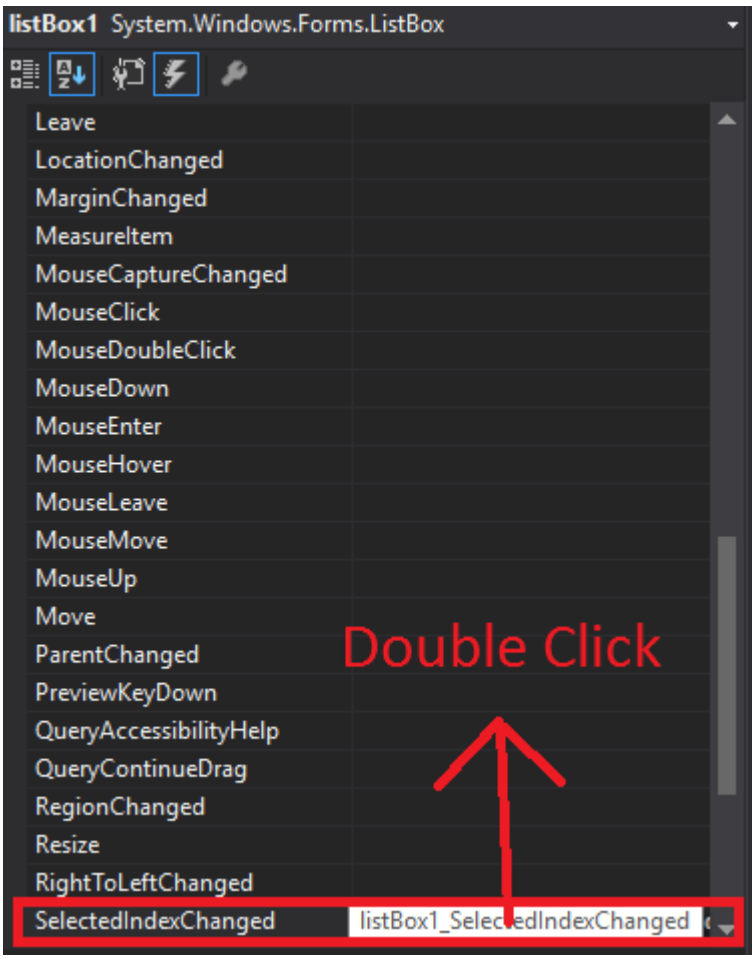
SelectedIndex_Changed;

Définir une liste pour donner une source de données

```
private void Form3_Load(object sender, EventArgs e)
```

```
        {
            List<string> DataList = new List<string> {"test1" , "test2" , "test3" , "44" ,
"45" };
            listBox1.DataSource = TestList;
        }
```

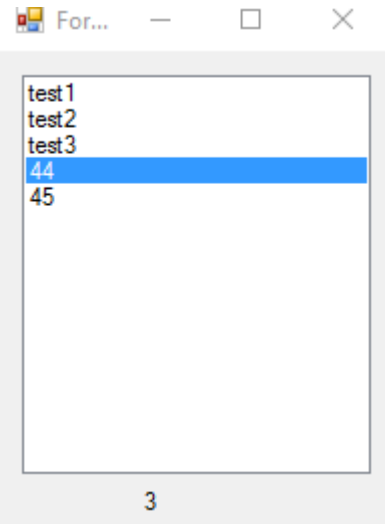
À la conception du formulaire, sélectionnez `Listbox` et appuyez sur F4 ou à droite, cliquez sur l'icône lightining.



Visual Studio va générer `listBox1_SelectedIndexChanged` à codebehind.

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

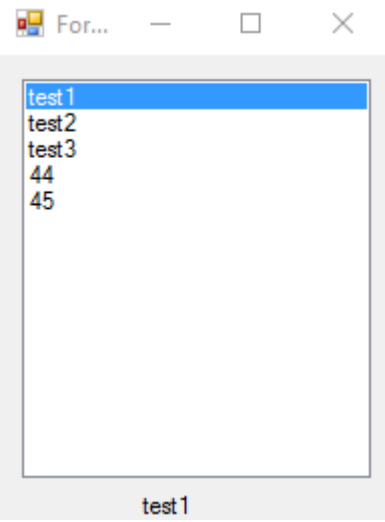
Résultat de `SelectedIndex_Changed` ; (l'étiquette en bas indiquera l'index de chaque article sélectionné)



SelectedValue_Changed; (La source de données est identique à celle du haut et vous pouvez générer cet événement comme SelectedIndex_Changed)

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

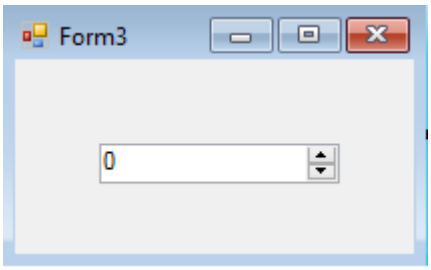
Sortie ;



NumericUpDown

NumericUpDown est un contrôle qui ressemble à TextBox. Ce contrôle permet à l'utilisateur d'afficher / sélectionner un numéro dans une plage. Les flèches vers le haut et vers le bas mettent à jour la valeur de la zone de texte.

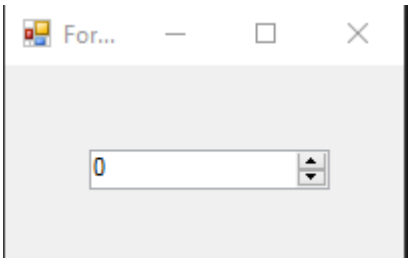
Le contrôle ressemble à;



Dans `Form_Load` plage peut être définie.

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

Sortie;

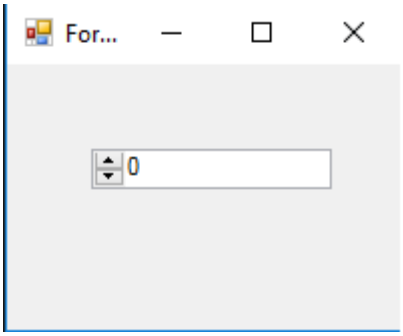


UpDownAlign définira la position des flèches;

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

        }
```

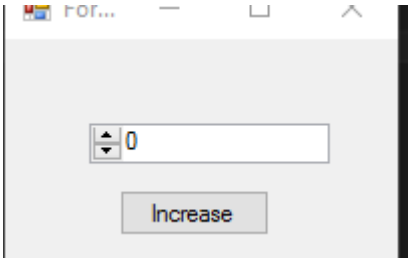
Sortie;



`UpButton()` méthode `UpButton()` augmente le nombre du contrôle. (peut être appelé de n'importe où. J'ai utilisé un `button` pour l'appeler.)

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

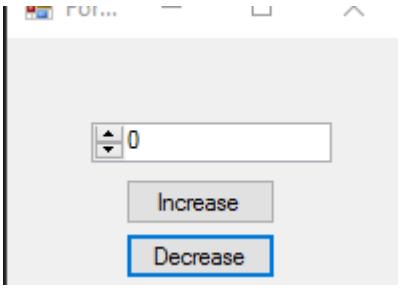
**Sortie



`DownButton()` méthode diminue le nombre du contrôle. (peut être appelé de n'importe où. J'ai utilisé un `button` pour l'appeler à nouveau.)

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

Sortie;



Événements utiles

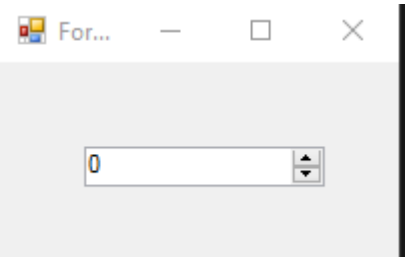
ValueChanged;

Cet événement fonctionnera lorsque la flèche haut ou bas sera cliquée.

```
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.
```

```
            }
        }
```

Sortie ;



Lire Contrôles de base en ligne: https://riptutorial.com/fr/winforms/topic/5816/controles-de-base

Chapitre 5: Contrôles hérités

Remarques

Les contrôles sont dérivés exactement de la même manière que les autres classes. La seule chose à prendre en compte est la neutralisation des événements: il est généralement conseillé de s'assurer d'appeler le gestionnaire d'événements de base après les vôtres. Ma propre règle de base: en cas de doute, appelez l'événement de base.

Examples

Paramètres d'application étendus

Une lecture rapide de la plupart des sites de développeurs révélera que WinForms est considéré comme inférieur à WPF. L'une des raisons les plus souvent citées est la difficulté supposée de modifier l'application d'une application dans son ensemble.

En fait, il est étonnamment facile de créer une application dans WinForms facilement configurable à la conception et à l'exécution, si vous évitez tout simplement d'utiliser les contrôles standard et en dérivez les vôtres.

Prenez la TextBox comme exemple. Il est difficile d'imaginer une application Windows qui n'appelle pas l'utilisation d'une TextBox à un moment ou à un autre. Par conséquent, avoir votre propre TextBox aura toujours un sens. Prenons l'exemple suivant:

```
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);
        }
    }
```

```
}
```

L'une des choses que les utilisateurs trouvent le plus utile dans un formulaire de saisie de données, avec de nombreuses zones de saisie, consiste à modifier la couleur d'arrière-plan de la boîte. Visiblement, il est plus facile à voir qu'un curseur vertical clignotant standard. Le code ci-dessus fournit une zone de texte qui fait exactement cela.

Dans le processus, il utilise les propriétés statiques d'une classe statique. Je donne ci-dessous un extrait du mien:

```
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);
            }
        }
    }
}
```

Cette classe utilise le registre Windows pour conserver les propriétés, mais vous pouvez utiliser une base de données ou un fichier de paramètres si vous préférez. L'avantage d'utiliser une classe statique de cette manière est que les modifications à l'échelle de l'application peuvent être effectuées non seulement au moment de la conception, mais également par l'utilisateur au moment de l'exécution. J'inclus toujours un formulaire dans mes applications permettant à l'utilisateur de modifier les valeurs préférées. La fonction de sauvegarde enregistre non seulement dans le registre (ou la base de données, etc.), mais lors de l'exécution, elle modifie également les propriétés de la classe statique. Notez que les propriétés statiques d'une classe statique ne sont pas constantes. en ce sens, ils peuvent être considérés comme des variables d'application. Cela signifie que tous les formulaires ouverts après les modifications enregistrées seront immédiatement affectés par les modifications enregistrées.

Vous pourrez facilement penser à d'autres propriétés d'application que vous souhaitez pouvoir configurer de la même manière. Les polices sont un autre très bon exemple.

NumberBox

Souvent, vous voudrez avoir une zone de saisie qui prend uniquement des chiffres. Encore une fois, en dérivant des contrôles standard, ceci est facilement réalisé, par exemple:

```
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;
        }
    }
```

En plus de limiter les entrées aux nombres, cette classe comporte quelques fonctionnalités spéciales. Il expose une propriété Value pour représenter la valeur double du nombre, formate le texte, éventuellement avec des séparateurs de milliers, et fournit une saisie rapide de grands

nombres: 10M se développe en congé à 10.000.000.00 (le nombre de décimales étant une propriété). Par souci de brièveté, les séparateurs décimaux et les séparateurs de milliers ont été codés en dur. Dans un système de production, il s'agit également de préférences de l'utilisateur.

Lire Contrôles hérités en ligne: https://riptutorial.com/fr/winforms/topic/6476/controles-herites

Chapitre 6: Liaison de données

Paramètres

Argument	La description
nom de la propriété	Le nom de la propriété de contrôle à lier.
la source de données	Un objet représentant la source de données.
dataMember	La propriété ou la liste à lier.
formatageEnabled	Détermine si les données affichées doivent être formatées.
updateMode	La source de données est mise à jour lorsque la propriété de contrôle est validée (par défaut) ou immédiatement lorsque la propriété a changé
nullValue	Lorsque la source de données a cette valeur, la propriété liée est définie sur DBNull.
formatChaîne	Un ou plusieurs caractères spécificateurs de format indiquant comment une valeur doit être affichée
formatInfo	Une implémentation de IFormatProvider pour remplacer le comportement de mise en forme par défaut.

Remarques

Voir https://msdn.microsoft.com/en-us/library/ef2xyb33.aspx La liaison de données fonctionne uniquement avec des propriétés, jamais avec des champs!

Examples

Liaison des contrôles aux objets de données

Chaque contrôle a une propriété `DataBindings` qui est une liste d'objets `System.Windows.Forms.Binding` . La méthode Add () contient des surcharges qui vous permettent de lier facilement la propriété d'un objet:

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

Notez que cette liaison signifie essentiellement que l'abonnement à l'autre change. Le code ci-

dessus souscrit à la modification de dataObj.MyProperty et adapte textBox.Text lorsqu'il change. Et vice-versa, il s'abonne à textBox.TextChanged et adapte dataObj.MyPropery lorsqu'il change.

Lire Liaison de données en ligne: https://riptutorial.com/fr/winforms/topic/7362/liaison-de-donnees

Chapitre 7: Zone de texte

Examples

Complétion automatique à partir d'une collection de chaînes

```
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);
```

Cela va se **compléter automatiquement** lorsque l'utilisateur essaie de taper **G** ou **L.**

`AutoCompleteMode.SuggestAppend` affichera tous les deux une liste de valeurs suggérées et il saisira automatiquement la première correspondance, `Append` uniquement et `Suggest` uniquement sont également disponibles.

Autoriser uniquement les chiffres dans le texte

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

Cela permettra uniquement l'utilisation de chiffres et de caractères de contrôle dans la zone de `TextBox` , d'autres combinaisons sont possibles en utilisant la même approche pour définir la propriété `Handle` sur true pour bloquer le texte.

L'utilisateur peut toujours copier / coller des caractères indésirables afin qu'une vérification supplémentaire soit effectuée sur le `TextChanged` pour nettoyer l'entrée:

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

Dans cet exemple, une **expression régulière** est utilisée pour filtrer le texte.

NumericUpDown devrait être préféré pour les nombres si possible.

Comment faire défiler jusqu'à la fin

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

En appliquant le même principe, `SelectionStart` peut être défini sur `0` pour faire défiler vers le haut ou vers un numéro spécifique pour accéder à un caractère spécifique.

Ajout d'un espace réservé à la zone de texte

Ce code place le texte de *conseil* au chargement du formulaire et le manipule comme suit:

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
```

Lire Zone de texte en ligne: https://riptutorial.com/fr/winforms/topic/4674/zone-de-texte

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec winforms	4444, Bjørn-Roger Kringsjå, Chris Shao, Cody Gray, Community, Reza Aghaei
2	Afficher un formulaire	Cody Gray, Jeff Bridgman, Steve
3	Aide à l'intégration	help-info.de, Reza Aghaei
4	Contrôles de base	Aimnox, Berkay, help-info.de, Jeff Bridgman
5	Contrôles hérités	Balagurunathan Marimuthu
6	Liaison de données	Kai Thoma
7	Zone de texte	gplumb, Jones Joseph, Stefano d'Antonio