

Contents

Windows Forms

- Mise en route des Windows Forms

- Vue d'ensemble des Windows Forms

- Création d'un nouveau Windows Form

 - Comment : créer une application Windows Forms à partir de la ligne de commande

 - Coordonnées Windows Forms

- Création de gestionnaires d'événements dans les Windows Forms

 - Vue d'ensemble des événements

 - Vue d'ensemble des gestionnaires d'événements

 - Comment : créer des gestionnaires d'événements pour les Windows Forms au moment de l'exécution

 - Comment : connecter plusieurs événements à un même gestionnaire d'événements dans les Windows Forms

 - Ordre des événements dans les Windows Forms

- Réglage de la taille et de l'échelle des Windows Forms

 - Comment : redimensionner des Windows Forms

 - Mise à l'échelle automatique dans les Windows Forms

 - Comment : répondre aux modifications de jeu de polices dans une application Windows Forms

 - Prise en charge de la haute résolution dans Windows Forms

- Modification de l'apparence des Windows Forms

 - Comment : modifier les bordures des Windows Forms

- contrôles Windows Forms

- Entrées d'utilisateur dans les Windows Forms

 - Entrée d'utilisateur dans une application Windows Forms

 - Entrée au clavier dans une application Windows Forms

 - Fonctionnement de l'entrée au clavier

 - Utilisation des événements du clavier

 - Comment : modifier l'entrée au clavier pour un contrôle standard

Comment : identifier la touche de modification activée

Comment : gérer l'entrée au clavier au niveau du formulaire

Entrée de la souris dans une application Windows Forms

Fonctionnement des entrées de la souris dans les Windows Forms

Événements liés à la souris dans les Windows Forms

Comment : distinguer les clics des double-clics

Pointeurs de souris dans les Windows Forms

Capture de la souris dans les Windows Forms

Fonctionnalité de glisser-déplacer dans les Windows Forms

Comment : simuler des événements de la souris et du clavier dans le code

Comment : gérer des événements d'entrée d'utilisateur dans les contrôles Windows Forms

Validation des entrées d'utilisateur dans les Windows Forms

Boîtes de dialogue dans les Windows Forms

Comment : afficher des boîtes de dialogue pour les Windows Forms

Liaison de données Windows Forms

Liaison de données et Windows Forms

Sources de données prises en charge par les Windows Forms

Interfaces participant à la liaison de données

Notification de modifications dans la liaison de données Windows Forms

Comment : appliquer le modèle PropertyChanged

Comment : créer un contrôle dépendant et mettre en forme les données affichées

Comment : créer un contrôle à liaison simple dans un Windows Form

Comment : s'assurer que plusieurs contrôles liés à la même source de données restent synchronisés

Comment : s'assurer que la ligne sélectionnée dans une table enfant reste au bon emplacement

Comment : implémenter l'interface IListSource

Comment : implémenter l'interface INotifyPropertyChanged

Comment : implémenter l'interface IList

Comment : naviguer au sein des données dans les Windows Forms

Sécurité des Windows Forms

Vue d'ensemble de la sécurité dans les Windows Forms

[Accès plus sécurisé aux fichiers et aux données dans les Windows Forms](#)

[Impression plus sécurisée dans les Windows Forms](#)

[Considérations supplémentaires sur la sécurité des Windows Forms](#)

[Déploiement ClickOnce pour les Windows Forms](#)

[Comment : accéder à des collections à clé dans les Windows Forms](#)

[Amélioration des applications Windows Forms](#)

Windows Forms

3 minutes to read • [Edit Online](#)

Les formulaires étant l'unité de base de votre application, il est essentiel de réfléchir à leur fonction et à leur conception. Si l'on y réfléchit bien, un formulaire est une feuille blanche que vous, en tant que développeur, améliorez avec des contrôles pour créer une interface utilisateur et avec du code pour manipuler des données. À cette fin, Visual Studio vous fournit un environnement de développement intégré (IDE) pour vous aider à écrire du code, ainsi qu'un ensemble de contrôles enrichis écrit avec le .NET Framework. En associant les fonctionnalités de ces contrôles à votre code, vous pouvez facilement et rapidement développer les solutions dont vous avez besoin.

Dans cette section

[Bien démarrer avec Windows Forms](#)

Fournit des liens vers des rubriques expliquant comment exploiter toute la puissance des Windows Forms pour afficher des données, gérer l'entrée d'utilisateur et déployer vos applications facilement et avec une sécurité plus robuste.

[Amélioration des applications Windows Forms](#)

Fournit des liens vers des rubriques expliquant comment améliorer vos Windows Forms avec un large éventail de fonctionnalités.

Sections connexes

[Contrôles Windows Forms](#)

Contient des liens vers des rubriques décrivant les contrôles Windows Forms et expliquant comment les implémenter.

[Liaison de données Windows Forms](#)

Contient des liens vers des rubriques décrivant l'architecture de liaison de données Windows Forms.

[Vue d'ensemble des graphiques](#)

Explique comment créer des graphismes, dessiner du texte et manipuler des images graphiques comme objets à l'aide de l'implémentation avancée de l'interface de conception graphique Windows.

[Sécurité et déploiement ClickOnce](#)

Présente les principes du déploiement ClickOnce.

[Différences de programmation entre Windows Forms et MFC](#)

Décrit les différences entre les applications MFC et les Windows Forms.

[Accès aux données dans Visual Studio](#)

Décrit comment incorporer les fonctionnalités d'accès aux données à vos applications.

[Applications Windows Forms](#)

Présente le processus de débogage des applications créées avec le modèle de projet Application Windows et comment modifier les configurations Debug et Release.

[Premier aperçu du déploiement dans Visual Studio](#)

Décrit le processus par lequel vous distribuez une application ou un composant terminé pour l'installer sur d'autres ordinateurs.

[Génération d'applications console](#)

Décrit les principes fondamentaux de la création d'une application console à l'aide de la classe [Console](#).

Mise en route des Windows Forms

3 minutes to read • [Edit Online](#)

Avec Windows Forms, vous pouvez créer de puissantes applications basées sur Windows. Les rubriques suivantes décrivent en détail comment exploiter la puissance de Windows Forms pour afficher des données, gérer l'entrée d'utilisateur et déployer vos applications facilement et avec une sécurité renforcée.

Dans cette section

[Vue d'ensemble des Windows Forms](#)

Contient une vue d'ensemble des Windows Forms et des applications clientes intelligentes.

[Création d'un nouveau Windows Form](#)

Contient des liens vers des rubriques qui décrivent les concepts de base pour la création d'applications Windows Forms.

[Création de gestionnaires d'événements dans Windows Forms](#)

Contient des liens vers des rubriques qui décrivent comment créer des gestionnaires d'événements Windows Forms.

[Réglage de la taille et de l'échelle des Windows Forms](#)

Contient des liens vers des rubriques qui montrent comment ajuster la taille et l'échelle des Windows Forms.

[Modification de l'apparence des Windows Forms](#)

Contient des liens vers des rubriques qui montrent comment modifier l'apparence des applications Windows Forms.

[Contrôles Windows Forms](#)

Contient des liens vers des rubriques qui décrivent et expliquent comment utiliser Windows Forms des contrôles et des composants.

[Entrée d'utilisateur dans Windows Forms](#)

Contient des liens vers des rubriques qui décrivent et expliquent comment gérer les entrées de l'utilisateur dans les applications de Windows Forms.

[Boîtes de dialogue dans les Windows Forms](#)

Contient des liens vers des rubriques qui décrivent les différentes boîtes de dialogue à utiliser dans Windows Forms.

[Liaison de données Windows Forms](#)

Contient des liens vers des rubriques qui décrivent l'architecture de liaison de données Windows Forms et comment l'utiliser dans les applications Windows Forms.

[Sécurité de Windows Forms](#)

Contient des liens vers des rubriques qui décrivent comment créer des Windows Forms des applications qui ont une sécurité renforcée.

[Déploiement ClickOnce pour les Windows Forms](#)

Contient des liens vers des rubriques qui décrivent comment déployer facilement des applications Windows Forms.

[Guide pratique pour accéder à des collections à clé dans les Windows Forms](#)

Montre comment accéder aux collections avec des clés plutôt que des index.

Sections connexes

[Amélioration des applications Windows Forms](#)

Contient des liens vers des rubriques qui décrivent des concepts plus avancés pour la création d'applications Windows Forms.

Présentation de Windows Forms

15 minutes to read • [Edit Online](#)

La présentation suivante décrit les avantages offerts par les applications Smart Client, les principales fonctionnalités de programmation Windows Forms et comment utiliser Windows Forms pour générer des clients intelligents qui répondent aux besoins des utilisateurs finaux et des entreprises d'aujourd'hui.

Applications clientes Windows Forms et intelligentes

Avec Windows Forms, vous pouvez développer des clients intelligents (Smart Clients). Les *clients intelligents* sont des applications enrichies du point de vue graphique, faciles à déployer et à mettre à jour, pouvant fonctionner même si elles ne sont pas connectées à Internet et pouvant accéder aux ressources sur l'ordinateur local de façon plus sécurisée que les applications Windows traditionnelles.

Créez des interfaces utilisateur riches et interactives

Windows Forms est une technologie Smart client pour les .NET Framework, un ensemble de bibliothèques managées qui simplifient les tâches d'application courantes telles que la lecture et l'écriture dans le système de fichiers. Lorsque vous utilisez un environnement de développement tel que Visual Studio, vous pouvez créer Windows Forms applications clientes intelligentes qui affichent des informations, demandent une entrée d'utilisateurs et communiquent avec des ordinateurs distants sur un réseau.

Dans Windows Forms, un *formulaire* est une surface visuelle sur laquelle vous présentez des informations à l'utilisateur. Habituellement, vous générez des applications Windows Forms en ajoutant des contrôles à des formulaires et en développant des réponses aux actions de l'utilisateur, telles que des clics de souris ou des pressions sur des touches du clavier. Un *contrôle* est un élément d'interface utilisateur discret qui affiche des données ou accepte l'entrée de données.

Quand un utilisateur modifie un formulaire ou l'un de ses contrôles, l'action génère un événement. Votre application réagit à ces événements en exécutant du code et traite les événements quand ils se produisent. Pour plus d'informations, consultez [Création de gestionnaires d'événements dans les Windows Forms](#).

Windows Forms contient divers contrôles que vous pouvez ajouter aux formulaires : des contrôles qui affichent des zones de texte, des boutons, des listes déroulantes, des cases d'option et même des pages web. Pour obtenir la liste de tous les contrôles que vous pouvez utiliser sur un formulaire, consultez [Contrôles à utiliser dans les Windows Forms](#). Si un contrôle existant ne répond pas à vos besoins, Windows Forms prend également en charge la création de vos propres contrôles personnalisés à l'aide de la classe [UserControl](#).

Windows Forms offre des contrôles d'interface utilisateur enrichis qui émulent les fonctionnalités des applications haut de gamme telles que Microsoft Office. Quand vous utilisez les contrôles [ToolStrip](#) et [MenuStrip](#), vous pouvez créer des barres d'outils et des menus qui contiennent du texte et des images, qui affichent des sous-menus et qui hébergent d'autres contrôles tels que des zones de texte et des zones de liste déroulante.

Le **Concepteur Windows Forms** de glisser-déplacer dans Visual Studio vous permet de créer facilement des applications Windows Forms. Il vous suffit de sélectionner les contrôles avec votre curseur et de les ajouter à l'emplacement souhaité sur le formulaire. Le concepteur fournit des outils tels que des quadrillages et des lignes d'alignement pour simplifier l'alignement des contrôles. Et si vous utilisez Visual Studio ou compilez sur la ligne de commande, vous pouvez utiliser les contrôles [FlowLayoutPanel](#), [TableLayoutPanel](#) et [SplitContainer](#) pour créer des dispositions de formulaire avancées en moins de temps.

Pour finir, si vous devez créer vos propres éléments d'interface utilisateur personnalisés, l'espace de noms [System.Drawing](#) contient une large sélection de classes pour afficher des lignes, des cercles et autres formes

directement sur un formulaire.

NOTE

Les contrôles Windows Forms ne sont pas conçus pour être marshalés entre des domaines d'application. Pour cette raison, Microsoft ne prend pas en charge le passage d'un contrôle Windows Forms à travers une frontière [AppDomain](#), même si le type de base [Control](#) de [MarshalByRefObject](#) semble indiquer que cela est possible. Les applications Windows Forms qui ont plusieurs domaines d'application sont prises en charge tant qu'aucun contrôle Windows Forms ne franchit les frontières des domaines d'applications.

Créer des formulaires et des contrôles

Pour obtenir des informations détaillées sur l'utilisation de ces fonctionnalités, consultez les rubriques d'aide suivantes.

DESCRIPTION	RUBRIQUE D'AIDE
Utilisation de contrôles sur des formulaires	Comment : ajouter des contrôles à des Windows Forms
Utilisation du contrôle ToolStrip	Comment : créer un ToolStrip de base avec des éléments standard à l'aide du concepteur
Création de graphismes avec System.Drawing	Mise en route de la programmation graphique
Création de contrôles personnalisés	Comment : hériter de la classe UserControl

Afficher et manipuler des données

De nombreuses applications doivent afficher des données provenant d'une base de données, d'un fichier XML, d'un service web XML ou autre source de données. Windows Forms fournit un contrôle flexible, nommé [DataGridView](#), pour afficher ces données tabulaires dans un format classique avec des lignes et des colonnes où chaque élément de données occupe sa propre cellule. Quand vous utilisez [DataGridView](#), vous pouvez personnaliser l'apparence de chaque cellule, verrouiller des lignes et des colonnes arbitraires et afficher des contrôles complexes dans des cellules, entre autres fonctionnalités.

La connexion à des sources de données sur un réseau est une tâche simple avec les clients intelligents Windows Forms. Le composant [BindingSource](#) représente une connexion à une source de données et expose des méthodes pour lier des données aux contrôles, naviguer vers les enregistrements précédents et suivants, modifier les enregistrements et enregistrer les modifications dans la source d'origine. Le contrôle [BindingNavigator](#) fournit une interface simple sur le composant [BindingSource](#) permettant aux utilisateurs de naviguer parmi les enregistrements.

Vous pouvez facilement créer des contrôles liés aux données à l'aide de la fenêtre Sources de données. Cette fenêtre affiche des sources de données telles que les bases de données, les services web et les objets dans votre projet. Pour créer des contrôles liés aux données, vous pouvez faire glisser des éléments depuis cette fenêtre vers des formulaires dans votre projet. Vous pouvez également lier des contrôles existants à des données en faisant glisser des objets depuis la fenêtre Sources de données vers des contrôles existants.

Les *paramètres* sont un autre type de liaison de données que vous pouvez gérer dans Windows Forms. La plupart des applications Smart Client doivent conserver certaines informations sur leur état d'exécution, telles que la dernière taille connue des formulaires, et conserver des données de préférence utilisateur, telles que les emplacements par défaut des fichiers enregistrés. La fonctionnalité Paramètres d'application répond à ces exigences en offrant un mécanisme simple de stockage des deux types de paramètres sur l'ordinateur client. Une fois que vous avez défini ces paramètres à l'aide de Visual Studio ou d'un éditeur de code, les paramètres sont conservés au format XML et lus automatiquement en mémoire au moment de l'exécution.

Afficher et manipuler des données

Pour obtenir des informations détaillées sur l'utilisation de ces fonctionnalités, consultez les rubriques d'aide suivantes.

DESCRIPTION	RUBRIQUE D'AIDE
Utilisation du composant BindingSource	Comment : lier des contrôles Windows Forms au composant BindingSource à l'aide du concepteur
Utilisation des sources de données ADO.NET	Comment : trier et filtrer des données ADO.NET avec le composant BindingSource Windows Forms
Utilisation de la fenêtre Sources de données	Lier des contrôles Windows Forms à des données dans Visual Studio
Utilisation des paramètres d'application	Comment : créer des paramètres d'application

Déployer des applications sur des ordinateurs clients

Une fois que vous avez écrit votre application, vous devez l'envoyer à vos utilisateurs pour qu'ils puissent l'installer et l'exécuter sur leurs propres ordinateurs clients. Lorsque vous utilisez la technologie ClickOnce, vous pouvez déployer vos applications à partir de Visual Studio à l'aide de quelques clics et fournir aux utilisateurs une URL pointant vers votre application sur le Web. ClickOnce gère tous les éléments et dépendances dans votre application et s'assure que l'application est correctement installée sur l'ordinateur client.

Les applications ClickOnce peuvent être configurées pour s'exécuter uniquement lorsque l'utilisateur est connecté au réseau ou pour s'exécuter à la fois en ligne et hors connexion. Lorsque vous spécifiez qu'une application doit prendre en charge l'opération hors connexion, ClickOnce ajoute un lien à votre application dans le menu

Démarrer de l'utilisateur. L'utilisateur peut alors ouvrir l'application sans utiliser l'URL.

Quand vous mettez à jour votre application, vous publiez un nouveau manifeste de déploiement et une nouvelle copie de votre application sur votre serveur web. ClickOnce détectera qu'une mise à jour est disponible et mettra à niveau l'installation de l'utilisateur. aucune programmation personnalisée n'est requise pour mettre à jour les anciens assemblés.

Déployer des applications ClickOnce

Pour une présentation complète de ClickOnce, consultez [sécurité et déploiement ClickOnce](#). Pour obtenir des informations détaillées sur l'utilisation de ces fonctionnalités, consultez les rubriques d'aide suivantes.

DESCRIPTION	RUBRIQUE D'AIDE
Déploiement d'une application à l'aide de ClickOnce	Guide pratique pour publier une application ClickOnce à l'aide de l'Assistant Publication Procédure pas à pas : déploiement manuel d'une application ClickOnce
Mise à jour d'un déploiement ClickOnce	Guide pratique pour gérer les mises à jour pour une application ClickOnce
Gestion de la sécurité avec ClickOnce	Guide pratique pour activer les paramètres de sécurité ClickOnce

Autres contrôles et fonctionnalités

Il existe de nombreuses autres fonctionnalités dans Windows Forms qui simplifient et accélèrent l'implémentation des tâches courantes, par exemple la prise en charge de la création de boîtes de dialogue, de l'impression, de l'ajout d'aide et de documentation et de la localisation de votre application en plusieurs langues. En outre, Windows

Forms s'appuie sur le système de sécurité fiable du .NET Framework. Ce système vous permet de distribuer des applications plus sécurisées à vos clients.

Implémenter d'autres contrôles et fonctionnalités

Pour obtenir des informations détaillées sur l'utilisation de ces fonctionnalités, consultez les rubriques d'aide suivantes.

DESCRIPTION	RUBRIQUE D'AIDE
Impression du contenu d'un formulaire	Comment : imprimer des graphiques dans Windows Forms Comment : imprimer un fichier texte composé de plusieurs pages dans les Windows Forms
En savoir plus sur la sécurité Windows Forms	Vue d'ensemble de la sécurité dans les Windows Forms

Voir aussi

- [Bien démarrer avec Windows Forms](#)
- [Création d'un nouveau Windows Form](#)
- [Vue d'ensemble du contrôle ToolStrip](#)
- [Vue d'ensemble du contrôle DataGridView](#)
- [Vue d'ensemble du composant BindingSource](#)
- [Vue d'ensemble des paramètres d'application](#)
- [Sécurité et déploiement ClickOnce](#)

Création d'un nouveau Windows Form

2 minutes to read • [Edit Online](#)

Cette rubrique contient des liens vers des rubriques qui décrivent comment créer votre première application Windows Forms. Les rubriques de cette section présentent également la terminologie de base que vous devez comprendre et les recommandations à suivre quand vous commencez à créer une application Windows Forms. Pour en savoir plus sur les applications Windows Forms, les contrôles que vous pouvez utiliser sur les événements et gestion des événements et comment gérer l'entrée de l'utilisateur, consultez la liste des rubriques connexes.

Dans cette section

[Coordonnées de Windows Forms.](#)

Décrit les coordonnées clientes et d'écran.

[Guide pratique pour Créer une Application de formulaires Windows à partir de la ligne de commande](#)

Décrit comment créer un Windows Form de base et le compiler depuis la ligne de commande.

Référence

[Form](#)

Décrit cette classe et propose des liens vers tous ses membres.

[Control](#)

Décrit cette classe et propose des liens vers tous ses membres.

Rubriques connexes

[Gestion des entrées utilisateur](#)

Contient des liens vers des rubriques qui traitent de l'entrée d'utilisateur et expliquent comment la gérer dans les applications Windows Forms.

[Création de gestionnaires d'événements dans les Windows Forms](#)

Contient des liens vers des rubriques qui décrivent comment gérer les événements dans les applications Windows Forms.

[Modification de l'apparence des Windows Forms](#)

Contient des liens vers des rubriques qui montrent comment modifier l'apparence des applications Windows Forms.

[Classement par fonction des contrôles Windows Forms](#)

Contient des liens vers des rubriques qui décrivent les contrôles que vous pouvez utiliser dans les applications Windows Forms.

Comment : Créer une application Windows Forms à partir de la ligne de commande

5 minutes to read • [Edit Online](#)

Les procédures suivantes décrivent les étapes de base que vous devez effectuer pour créer et exécuter une application Windows Forms à partir de la ligne de commande. Ces procédures sont très bien prises en charge dans Visual Studio. Voir aussi [Pas à pas : Héberger un contrôle des formulaires Windows dans WPF](#).

Procédure

Pour créer le formulaire

1. Dans un fichier de code `Imports` `using` vide, tapez les éléments suivants ou les relevés :

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

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

2. Déclarez une `Form1` classe nommée qui hérite de la classe de formulaire :

```
public class Form1 : Form
```

```
Public Class Form1
    Inherits Form
```

3. Créez un constructeur sans `Form1` paramètres pour .

Vous ajouterez du code au constructeur lors d'une procédure ultérieure.

```
public Form1() {}
```

```
Public Sub New()

End Sub
```

4. Ajoutez une méthode `Main` à la classe.
 - a. Appliquez [STAThreadAttribute](#) la méthode `Main` C pour spécifier votre application Windows Forms est un appartement à threads unique. (L'attribut n'est pas nécessaire dans Visual Basic, puisque Windows forme des applications développées avec Visual Basic utiliser un modèle d'appartement à threads unique par défaut.)
 - b. Appelez [EnableVisualStyles](#) pour appliquer des styles de système d'exploitation à votre application.

c. Créez une instance du formulaire et exécutez-la.

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

```
Public Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())

End Sub
End Class
```

Pour compiler et exécuter l'application

1. À l'invite de commandes .NET Framework, accédez au répertoire où vous avez créé la classe `Form1`.
2. Compilez le formulaire.
 - Si vous utilisez le C, tapez : `csc form1.cs`
 - or-
 - Si vous utilisez Visual Basic, tapez : `vbc form1.vb`
3. À l'invite de commandes, tapez : `Form1.exe`

Ajout d'un contrôle et gestion d'un événement

Les étapes de la procédure précédente illustrent simplement comment créer un Windows Form de base qui se compile et s'exécute. La procédure suivante illustre comment créer et ajouter un contrôle au formulaire et comment gérer un événement pour le contrôle. Pour plus d'informations sur les contrôles que vous pouvez ajouter aux formulaires Windows, voir [Windows Forms Controls](#).

Outre la création d'applications Windows Forms, vous devez comprendre la programmation basée sur les événements et comment gérer l'entrée d'utilisateur. Pour plus d'informations, voir [Créer des gestionnaires d'événements dans les formulaires Windows](#) et traiter [l'entrée des utilisateurs](#)

Pour déclarer un contrôle bouton et gérer son événement click

1. Déclarez un contrôle bouton nommé `button1`.
2. Dans le constructeur, créez le bouton et définissez ses propriétés [Size](#), [Location](#) et [Text](#).
3. Ajoutez le bouton au formulaire.

L'exemple de code suivant montre comment déclarer le contrôle du bouton :

```

public Button button1;
public Form1()
{
    button1 = new Button();
    button1.Size = new Size(40, 40);
    button1.Location = new Point(30, 30);
    button1.Text = "Click me";
    this.Controls.Add(button1);
    button1.Click += new EventHandler(button1_Click);
}

```

```

Public WithEvents button1 As Button

Public Sub New()
    button1 = New Button()
    button1.Size = New Size(40, 40)
    button1.Location = New Point(30, 30)
    button1.Text = "Click me"
    Me.Controls.Add(button1)
End Sub

```

4. Créez une méthode pour gérer l'événement [Click](#) pour le bouton.
5. Dans le gestionnaire d'événements click, affichez un [MessageBox](#) avec le message « Hello World ».

L'exemple de code suivant montre comment gérer l'événement de clic du contrôle des boutons :

```

private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello World");
}

```

```

Private Sub button1_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles button1.Click
    MessageBox.Show("Hello World")
End Sub

```

6. Associez l'événement [Click](#) à la méthode que vous avez créée.

L'exemple de code suivant montre comment associer l'événement à la méthode.

```

button1.Click += new EventHandler(button1_Click);

```

```

Private Sub button1_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles button1.Click

```

7. Compilez et exécutez l'application comme décrit dans la procédure précédente.

Exemple

L'exemple de code suivant est l'exemple complet des procédures précédentes :

```

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

namespace FormWithButton
{
    public class Form1 : Form
    {
        public Button button1;
        public Form1()
        {
            button1 = new Button();
            button1.Size = new Size(40, 40);
            button1.Location = new Point(30, 30);
            button1.Text = "Click me";
            this.Controls.Add(button1);
            button1.Click += new EventHandler(button1_Click);
        }
        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Hello World");
        }
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new Form1());
        }
    }
}

```

```

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

Public Class Form1
    Inherits Form
    Public WithEvents button1 As Button

    Public Sub New()
        button1 = New Button()
        button1.Size = New Size(40, 40)
        button1.Location = New Point(30, 30)
        button1.Text = "Click me"
        Me.Controls.Add(button1)
    End Sub

    Private Sub button1_Click(ByVal sender As Object, _
        ByVal e As EventArgs) Handles button1.Click
        MessageBox.Show("Hello World")
    End Sub

    <STAThread()> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())
    End Sub
End Class

```

Voir aussi

- [Form](#)
- [Control](#)
- [Modification de l'apparence des Windows Forms](#)
- [Amélioration des applications Windows Forms](#)
- [Bien démarrer avec Windows Forms](#)

Coordonnées Windows Forms

3 minutes to read • [Edit Online](#)

Le système de coordonnées d'un Windows Form est basé sur les coordonnées de l'appareil, et l'unité de mesure de base lors du dessin dans Windows Forms est l'unité de l'appareil (en général, le pixel). Les points sur l'écran sont décrits par des paires de coordonnées x et y, avec les coordonnées x qui s'accroissent à droite et les coordonnées y qui s'incrémentent de haut en bas. L'emplacement de l'origine, par rapport à l'écran, varie selon que vous spécifiez des coordonnées d'écran ou de client.

Coordonnées d'écran

Une application Windows Forms spécifie la position d'une fenêtre sur l'écran en coordonnées d'écran. Pour les coordonnées d'écran, l'origine est l'angle supérieur gauche de l'écran. La position complète d'une fenêtre est souvent décrite par une structure [Rectangle](#) contenant les coordonnées d'écran de deux points qui définissent les angles supérieur gauche et inférieur droit de la fenêtre.

Coordonnées clientes

Une application Windows Forms spécifie la position des points dans un formulaire ou un contrôle à l'aide de coordonnées clientes. L'origine des coordonnées clientes est l'angle supérieur gauche de la zone cliente du contrôle ou du formulaire. Les coordonnées clientes garantissent qu'une application peut utiliser des valeurs de coordonnées cohérentes lors du dessin dans un formulaire ou un contrôle, indépendamment de la position du formulaire ou du contrôle sur l'écran.

Les dimensions de la zone cliente sont également décrites par une structure [Rectangle](#) qui contient les coordonnées clientes pour la zone. Dans tous les cas, la coordonnée supérieure gauche du rectangle est incluse dans la zone cliente, tandis que la coordonnée inférieure droite est exclue. Les opérations graphiques n'incluent pas les bords droits et inférieurs d'une zone cliente. Par exemple, la méthode [FillRectangle](#) se remplit à droite et à la limite inférieure du rectangle spécifié, mais n'inclut pas ces bords.

Mappage d'un type de coordonnée à un autre

Parfois, vous devrez peut-être mapper les coordonnées d'écran aux coordonnées clientes. Vous pouvez facilement le faire à l'aide des méthodes [PointToClient](#) et [PointToScreen](#) disponibles dans la classe [Control](#). Par exemple, la propriété [MousePosition](#) de [Control](#) est signalée dans les coordonnées d'écran, mais vous pouvez les convertir en coordonnées clientes.

Voir aussi

- [PointToClient](#)
- [PointToScreen](#)

Création de gestionnaires d'événements dans les Windows Forms

2 minutes to read • [Edit Online](#)

Un gestionnaire d'événements est une procédure de votre code qui détermine les actions à effectuer quand un événement se produit, par exemple, quand un utilisateur clique sur un bouton ou une file d'attente reçoit un message. Au déclenchement de l'événement, le ou les gestionnaires d'événements qui reçoivent l'événement sont exécutés. Des événements peuvent être affectés à plusieurs gestionnaires et les méthodes qui gèrent des événements particuliers peuvent être modifiées de façon dynamique. Vous pouvez également utiliser la Conception Windows Forms dans Visual Studio pour créer des gestionnaires d'événements.

Dans cette section

[Vue d'ensemble des événements](#)

Présente le modèle d'événement et explique le rôle des délégués.

[Vue d'ensemble des gestionnaires d'événements](#)

Explique comment gérer les événements.

[Comment : créer des gestionnaires d'événements au moment de l'exécution pour Windows Forms](#)

Explique comment répondre dynamiquement aux événements système ou utilisateur.

[Comment : connecter plusieurs événements à un même gestionnaire d'événements dans Windows Forms](#)

Explique comment assigner la même fonctionnalité à plusieurs contrôles par le biais d'événements.

[Ordre des événements dans Windows Forms](#)

Décrit l'ordre dans lequel les événements sont déclenchés dans les contrôles Windows Forms.

[Comment : créer des gestionnaires d'événements à l'aide du concepteur](#) Décrit comment utiliser l'Concepteur Windows Forms pour créer des gestionnaires d'événements.

Sections connexes

[Événements](#)

Fournit des liens vers des rubriques sur la gestion et le déclenchement d'événements à l'aide de l'.NET Framework.

[Dépannage des gestionnaires d'événements hérités en Visual Basic](#)

Répertorie les problèmes courants qui surviennent avec les gestionnaires d'événements dans les composants hérités.

Vue d'ensemble des événements (Windows Forms)

6 minutes to read • [Edit Online](#)

Un événement est une action à laquelle vous pouvez répondre ou que vous pouvez « gérer » dans le code. Les événements peuvent être déclenchés par une action de l'utilisateur (quand il clique sur un bouton de la souris ou appuie sur une touche), par le code d'un programme ou par le système.

Les applications pilotées par événements exécutent du code en réponse à un événement. Chaque formulaire et chaque contrôle exposent un ensemble prédéfini d'événements que vous pouvez programmer. Si l'un de ces événements se produit et que le gestionnaire d'événements associé contient du code, celui-ci est appelé.

Les types d'événements déclenchés par un objet sont divers, mais bon nombre d'entre eux sont communs à la plupart des contrôles. Par exemple, la plupart des objets peuvent gérer un événement [Click](#). Si un utilisateur clique sur un formulaire, le code du gestionnaire de l'événement [Click](#) de ce dernier est exécuté.

NOTE

De nombreux événements se produisent conjointement à d'autres événements. Par exemple, les événements [DoubleClick](#), [MouseDown](#) et [MouseUp](#) se produisent en même temps que l'événement [Click](#).

Pour plus d'informations sur la façon de déclencher et de consommer un événement, consultez [événements](#).

Rôle des délégués

Les délégués sont des classes couramment utilisées dans le .NET Framework pour créer des mécanismes de gestion des événements. Les délégués sont à peu près équivalents aux pointeurs de fonction C++ , couramment utilisés dans des langages visuels et d'autres langages orientés objet. Toutefois à la différence des pointeurs de fonction, les délégués sont orientés objet, de type sécurisé et sûrs. De plus, alors qu'un pointeur de fonction ne contient qu'une référence à une fonction spécifique, un délégué consiste en une référence à un objet et des références à une ou plusieurs méthodes de cet objet.

Ce modèle d'événement utilise des *délégués* pour lier des événements aux méthodes utilisées pour les gérer. Le délégué permet aux autres classes de s'inscrire pour la notification d'événement en spécifiant une méthode de gestionnaire. Quand l'événement se produit, le délégué appelle la méthode liée. Pour plus d'informations sur la façon de définir des délégués, consultez [événements](#).

Les délégués peuvent être liés à une ou plusieurs méthodes (ou multidiffusion). Lorsque vous créez un délégué pour un événement, vous (ou les fenêtres) créez généralement un événement de multidiffusion. Il existe cependant une exception à cette règle : quand un événement déclenche une procédure particulière (par exemple, l'affichage d'une boîte de dialogue) qui logiquement ne se répéterait pas plusieurs fois par événement. Pour plus d'informations sur la création d'un délégué multicast, consultez [Comment combiner des délégués \(délégués multicast\)](#).

Un délégué multidiffusion tient à jour une liste d'appel des méthodes auxquelles il est lié. Celui-ci prend en charge une méthode [Combine](#) pour ajouter une méthode à la liste d'appel et une méthode [Remove](#) pour la supprimer.

Quand un événement est enregistré par l'application, le contrôle le déclenche en appelant son délégué. Le délégué appelle à son tour la méthode liée. Dans le cas le plus courant (délégué multidiffusion), le délégué appelle à tour de rôle chaque méthode liée de la liste d'appel, ce qui produit une notification un à plusieurs. Cette stratégie signifie que le contrôle n'a pas besoin de tenir à jour une liste des objets cible pour la notification

d'événement car le délégué gère totalement l'inscription et la notification.

Les délégués permettent également de lier plusieurs événements à la même méthode et par conséquent de produire une notification plusieurs-à-un. Par exemple, un événement button-click et un événement menu-command-click peuvent tous deux appeler le même délégué, lequel appelle à son tour une méthode unique pour qu'elle prenne en charge ces deux événements distincts de la même façon.

Le mécanisme de liaison utilisé avec les délégués est dynamique : un délégué peut être lié au moment de l'exécution à n'importe quelle méthode dont la signature correspond à celle du gestionnaire d'événements. Cette fonctionnalité vous permet de configurer ou de modifier la méthode de liaison en fonction d'une condition et d'attacher dynamiquement un gestionnaire d'événements à un contrôle.

Voir aussi

- [Création de gestionnaires d'événements dans Windows Forms](#)
- [Vue d'ensemble des gestionnaires d'événements](#)

Vue d'ensemble des gestionnaires d'événements (Windows Forms)

2 minutes to read • [Edit Online](#)

Un gestionnaire d'événements est une méthode qui est liée à un événement. Lorsque l'événement est déclenché, le code dans le gestionnaire de l'événement est exécuté. Chaque gestionnaire d'événements fournit deux paramètres qui vous permettent de gérer l'événement correctement. L'exemple suivant montre un [Button](#) gestionnaire [Click](#) d'événements pour l'événement d'un contrôle.

```
Private Sub button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles button1.Click  
  
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)  
{  
  
}
```

```
private:  
void button1_Click(System::Object ^ sender,  
    System::EventArgs ^ e)  
{  
  
}
```

Le premier `sender` paramètre, fournit une référence à l'objet qui a soulevé l'événement. Le deuxième `e` paramètre, dans l'exemple ci-dessus, passe un objet spécifique à l'événement qui est manipulé. En faisant référence aux propriétés de l'objet (et, parfois, à ses méthodes), vous pouvez obtenir des informations telles que l'emplacement de la souris pour les événements de souris ou les données transférées lors d'événements de drag-and-drop.

En règle générale, chaque événement produit un gestionnaire d'événements avec un type d'objet d'événement différent pour le deuxième paramètre. Certains gestionnaires d'événements, comme [MouseDown](#) [MouseUp](#) ceux pour les événements et les événements, ont le même type d'objet pour leur deuxième paramètre. Pour ce type d'événements, vous pouvez utiliser le même gestionnaire d'événements pour gérer les deux événements.

Vous pouvez également utiliser le même gestionnaire d'événements pour gérer le même événement pour différents contrôles. Par exemple, si vous [RadioButton](#) avez un groupe de contrôles sur un [Click](#) formulaire, vous pouvez [Click](#) créer un gestionnaire d'événement unique pour l'événement et avoir l'événement de chaque contrôle lié au gestionnaire d'événement unique. Pour plus d'informations, voir [Comment : Connectez plusieurs événements à un gestionnaire d'événements unique dans les formulaires Windows](#).

Voir aussi

- [Création de gestionnaires d'événements dans Windows Forms](#)
- [Vue d'ensemble des événements](#)

Comment : créer des gestionnaires d'événements pour les Windows Forms au moment de l'exécution

2 minutes to read • [Edit Online](#)

En plus de créer des événements à l'aide de l'Concepteur Windows Forms dans Visual Studio, vous pouvez également créer un gestionnaire d'événements au moment de l'exécution. Cette action vous permet de connecter des gestionnaires d'événements en fonction de conditions spécifiées dans le code lors de l'exécution au lieu de les connecter au premier démarrage du programme.

Créer un gestionnaire d'événements au moment de l'exécution

1. Ouvrez le formulaire auquel vous souhaitez ajouter un gestionnaire d'événements.
2. Ajoutez une méthode à votre formulaire avec la signature de méthode pour l'événement que vous souhaitez gérer.

Par exemple, si vous gérez l'événement [Click](#) d'un contrôle [Button](#), vous devez créer une méthode telle que la suivante :

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
    ' Add event handler code here.
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)
{
    // Add event handler code here.
}
```

```
private:
    void button1_Click(System::Object ^ sender,
        System::EventArgs ^ e)
    {
        // Add event handler code here.
    }
```

3. Ajoutez le code au gestionnaire d'événements en fonction de votre application.
4. Déterminez le formulaire ou le contrôle pour lequel vous souhaitez créer un gestionnaire d'événements.
5. Dans une méthode de classe de votre formulaire, ajoutez le code qui spécifie au gestionnaire d'événements de gérer l'événement. Par exemple, le code suivant spécifie que le gestionnaire d'événements `button1_Click` gère l'événement [Click](#) d'un contrôle [Button](#) :

```
AddHandler Button1.Click, AddressOf Button1_Click
```

```
button1.Click += new EventHandler(button1_Click);
```

```
button1->Click += gnew System::EventHandler(this, &Form1::button1_Click);
```

La méthode [AddHandler](#) présentée dans le code Visual Basic ci-dessus établit un gestionnaire d'événements Click pour le bouton.

Voir aussi

- [Création de gestionnaires d'événements dans Windows Forms](#)
- [Vue d'ensemble des gestionnaires d'événements](#)
- [Dépannage des gestionnaires d'événements hérités en Visual Basic](#)

Comment : connecter plusieurs événements à un même gestionnaire d'événements dans les Windows Forms

3 minutes to read • [Edit Online](#)

Dans la conception de votre application, il peut s'avérer nécessaire d'utiliser un seul gestionnaire d'événements pour plusieurs événements ou de faire en sorte que plusieurs événements effectuent la même procédure. Par exemple, il est souvent plus efficace de faire en sorte qu'une commande de menu génère le même événement qu'un bouton de votre formulaire si elle expose la même fonctionnalité. Pour ce faire, vous pouvez utiliser la vue événements de la Fenêtre Propriétés C# dans ou à l'aide du mot clé `Handles` et des zones de liste déroulante nom de la **classe** et nom de la **méthode** dans l'éditeur de code Visual Basic.

Pour connecter plusieurs événements à un même gestionnaire d'événements dans Visual Basic

1. Cliquez avec le bouton droit sur le formulaire, puis choisissez **afficher le code**.
2. Dans la zone de liste déroulante nom de la **classe**, sélectionnez l'un des contrôles dont vous souhaitez obtenir le handle de gestionnaire d'événements.
3. Dans la zone de liste déroulante nom de la **méthode**, sélectionnez l'un des événements que vous souhaitez que le gestionnaire d'événements gère.
4. L'éditeur de code insère le gestionnaire d'événements approprié et positionne le point d'insertion dans la méthode. Dans l'exemple ci-dessous, il s'agit de l'événement `Click` pour le contrôle `Button`.


```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles  
    Button1.Click  
    ' Add event-handler code here.  
End Sub
```

5. Ajoutez les autres événements que vous souhaitez gérer à la clause `Handles`.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles  
    Button1.Click, Button2.Click  
    ' Add event-handler code here.  
End Sub
```

6. Ajoutez le code approprié au gestionnaire d'événements.

Pour connecter plusieurs événements à un même gestionnaire d'événements en C#

1. Sélectionnez le contrôle auquel vous souhaitez connecter un gestionnaire d'événements.
2. Dans la Fenêtre Propriétés, cliquez sur le bouton **événements** (.
3. Cliquez sur le nom de l'événement que vous souhaitez gérer.
4. Dans la section valeur en regard du nom de l'événement, cliquez sur le bouton déroulant pour afficher la liste des gestionnaires d'événements existants qui correspondent à la signature de la méthode de l'événement que vous souhaitez gérer.
5. Sélectionnez le gestionnaire d'événements approprié dans la liste.

Du code sera ajouté au formulaire pour lier l'événement au gestionnaire d'événements existant.

Voir aussi

- [Création de gestionnaires d'événements dans Windows Forms](#)
- [Vue d'ensemble des gestionnaires d'événements](#)

Ordre des événements dans les Windows Forms

2 minutes to read • [Edit Online](#)

L'ordre dans lequel les événements sont déclenchés dans les applications Windows Forms est particulièrement intéressant pour les développeurs soucieux de gérer chacun de ces événements tour à tour. Quand une situation nécessite une gestion méticuleuse des événements, par exemple quand vous redessinez certaines parties du formulaire, une connaissance de l'ordre exact dans lequel les événements sont déclenchés au moment de l'exécution est nécessaire. Cette rubrique fournit des détails sur l'ordre des événements durant plusieurs phases importantes de la durée de vie des applications et des contrôles. Pour plus d'informations sur l'ordre des événements d'entrée de souris, consultez [événements de souris dans Windows Forms](#). Pour obtenir une vue d'ensemble des événements dans Windows Forms, consultez [vue d'ensemble des événements](#). Pour plus d'informations sur la composition de gestionnaires d'événements, consultez [vue d'ensemble des gestionnaires d'événements](#).

Événements de démarrage et d'arrêt des applications

Les classes [Form](#) et [Control](#) exposent un ensemble d'événements liés au démarrage et à l'arrêt de l'application. Lors du démarrage d'une application Windows Forms, les événements de démarrage du formulaire principal sont déclenchés dans l'ordre suivant :

- [Control.HandleCreated](#)
- [Control.BindingContextChanged](#)
- [Form.Load](#)
- [Control.VisibleChanged](#)
- [Form.Activated](#)
- [Form.Shown](#)

Lors de la fermeture d'une application, les événements d'arrêt du formulaire principal sont déclenchés dans l'ordre suivant :

- [Form.Closing](#)
- [Form.FormClosing](#)
- [Form.Closed](#)
- [Form.FormClosed](#)
- [Form.Deactivate](#)

L'événement [ApplicationExit](#) de la classe [Application](#) est déclenché après les événements d'arrêt du formulaire principal.

NOTE

Visual Basic 2005 fournit des événements d'application supplémentaires, tels que [WindowsFormsApplicationBase.Startup](#) et [WindowsFormsApplicationBase.Shutdown](#).

Événements de focus et de validation

Quand vous modifiez le focus à l'aide du clavier (Tab, Maj+Tab, etc.) en appelant la méthode [Select](#) ou [SelectNextControl](#) ou en affectant le formulaire actuel comme valeur de la propriété [ActiveControl](#), les événements de focus de la classe [Control](#) se produisent dans l'ordre suivant :

- [Enter](#)
- [GotFocus](#)
- [Leave](#)
- [Validating](#)
- [Validated](#)
- [LostFocus](#)

Quand vous modifiez le focus à l'aide de la souris ou en appelant la méthode [Focus](#), les événements de focus de la classe [Control](#) se produisent dans l'ordre suivant :

- [Enter](#)
- [GotFocus](#)
- [LostFocus](#)
- [Leave](#)
- [Validating](#)
- [Validated](#)

Voir aussi

- [Création de gestionnaires d'événements dans Windows Forms](#)

Ajustement de la taille et de l'échelle des formulaires Windows

2 minutes to read • [Edit Online](#)

Cette rubrique fournit des liens vers des informations sur le redimensionnement des Windows Forms.

Dans cette section

[Comment : redimensionner des Windows Forms](#)

Fournit des instructions pour spécifier la taille des Windows Forms.

[Mise à l'échelle automatique dans les formulaires Windows](#)

Explique comment la mise à l'échelle automatique permet à un formulaire et à ses contrôles d'être affichés correctement sur tous les ordinateurs.

[Support DPI élevé dans les formulaires Windows](#) Discute du support de Windows Forms pour l'IDP élevé et de la mise à l'échelle dynamique.

Informations de référence

[Size](#)

Décrit cette classe et propose des liens vers tous ses membres.

[TableLayoutPanel](#)

Décrit cette classe et propose des liens vers tous ses membres.

[FlowLayoutPanel](#)

Décrit cette classe et propose des liens vers tous ses membres.

Sections connexes

[Modification de l'apparence des formulaires Windows](#)

Fournit des liens vers des rubriques décrivant d'autres moyens de modifier l'apparence des Windows Forms.

Comment : redimensionner des Windows Forms

3 minutes to read • [Edit Online](#)

Vous pouvez spécifier la taille de votre Windows Form de plusieurs façons. Vous pouvez modifier à la fois la hauteur et la largeur du formulaire par programmation en affectant une nouvelle valeur à la propriété [Size](#), ou ajuster les propriétés [Height](#) et [Width](#) individuellement. Si vous utilisez Visual Studio, vous pouvez modifier la taille à l'aide de la Conception Windows Forms. Consultez également [Comment : redimensionner des Windows Forms à l'aide du concepteur](#).

Redimensionner un formulaire par programmation

Définissez la taille d'un formulaire au moment de l'exécution en définissant la propriété [Size](#) du formulaire.

L'exemple de code suivant montre la taille de formulaire définie sur 100 × 100 pixels.

```
Form1.Size = New System.Drawing.Size(100, 100)
```

```
Form1.Size = new System.Drawing.Size(100, 100);
```

```
Form1->Size = System::Drawing::Size(100, 100);
```

Modifier la largeur et la hauteur d'un formulaire par programmation

Une fois le [Size](#) défini, modifiez la hauteur ou la largeur du formulaire à l'aide de la propriété [Width](#) ou [Height](#).

L'exemple de code suivant montre la largeur du formulaire définie sur 300 pixels à partir du bord gauche du formulaire, alors que la hauteur reste constante.

```
Form1.Width = 300
```

```
Form1.Width = 300;
```

```
Form1->Width = 300;
```

-ou-

Modifiez [Width](#) ou [Height](#) en définissant la propriété [Size](#).

Toutefois, comme le montre l'exemple de code suivant, cette approche est plus laborieuse que la simple définition de la propriété [Width](#) ou [Height](#).

```
Form1.Size = New Size(300, Form1.Size.Height)
```

```
Form1.Size = new Size(300, Form1.Size.Height);
```

```
Form1->Size = System::Drawing::Size(300, Form1->Size.Height);
```

Modifier la taille de formulaire par incréments par programmation

Pour incrémenter la taille du formulaire, définissez les propriétés [Width](#) et [Height](#).

L'exemple de code suivant montre la largeur du formulaire définie sur 200 pixels de plus que la valeur actuelle.

```
Form1.Width += 200
```

```
Form1.Width += 200;
```

```
Form1->Width += 200;
```

Caution

Utilisez toujours la propriété [Height](#) ou [Width](#) pour modifier une dimension d'un formulaire, sauf si vous définissez les dimensions de hauteur et de largeur en même temps en affectant une nouvelle structure [Size](#) à la propriété [Size](#). La propriété [Size](#) retourne une structure [Size](#), qui est un type valeur. Vous ne pouvez pas attribuer une nouvelle valeur à la propriété d'un type valeur. Le code suivant, par exemple, ne sera pas compilé :

```
' NOTE: CODE WILL NOT COMPILE
Dim f As New Form()
f.Size.Width += 100
```

```
// NOTE: CODE WILL NOT COMPILE
Form f = new Form();
f.Size.Width += 100;
```

```
// NOTE: CODE WILL NOT COMPILE
Form^ f = gcnew Form();
f->Size->X += 100;
```

Voir aussi

- [Bien démarrer avec Windows Forms](#)
- [Amélioration des applications Windows Forms](#)

Mise à l'échelle automatique dans Windows Forms

14 minutes to read • [Edit Online](#)

La mise à l'échelle automatique permet d'afficher correctement un formulaire et ses contrôles conçus sur un ordinateur avec une certaine police système et une certaine résolution d'affichage sur un autre ordinateur avec une police système ou une résolution d'affichage différente. Elle garantit que le formulaire et ses contrôles seront redimensionnés intelligemment et de manière cohérente avec les fenêtres natives et d'autres applications sur les ordinateurs des utilisateurs et d'autres développeurs. La prise en charge des .NET Framework pour la mise à l'échelle automatique et les styles visuels permet .NET Framework applications de conserver une apparence cohérente par rapport aux applications Windows natives sur l'ordinateur de chaque utilisateur.

Dans la plupart des cas, la mise à l'échelle automatique fonctionne comme prévu dans .NET Framework version 2,0 et versions ultérieures. Toutefois, les modifications de schéma de police peuvent être problématiques. Pour obtenir un exemple de résolution de ce type, consultez [Comment : répondre aux modifications de jeu de polices dans une Application Windows Forms](#).

Nécessité d'une mise à l'échelle automatique

Sans mise à l'échelle automatique, une application conçue pour une résolution d'écran ou une police particulière apparaîtra trop petite ou trop grande en cas de modification de cette résolution ou police. Par exemple, si l'application est conçue à l'aide de la police Tahoma 9 points comme ligne de base, sans réglage elle apparaîtra trop petite si vous l'exécutez sur un ordinateur où la police système est Tahoma 12 points. Les éléments de texte (tels que les titres, les menus, le contenu des zones de texte et ainsi de suite) apparaîtront plus petits que dans les autres applications. En outre, la taille des éléments d'interface utilisateur qui contiennent du texte (comme la barre de titre, les menus et de nombreux contrôles) dépend de la police utilisée. Dans cet exemple, ces éléments apparaîtront aussi relativement plus petits.

Une situation analogue se produit quand une application est conçue pour une certaine résolution d'écran. La résolution d'affichage la plus courante est 96 points par pouce (DPI), ce qui équivaut à une mise à l'échelle d'affichage de 100%, mais les affichages de plus haute résolution prennent en charge 125%, 150%, 200% (respectivement les 120, 144 et 192 PPP) et les versions ultérieures sont devenues plus courantes. Sans réglage, une application (en particulier basée sur des graphismes) conçue pour une résolution spécifique apparaîtra trop grande ou trop petite en cas d'exécution dans une autre résolution.

La mise à l'échelle automatique vise à améliorer ces problèmes en redimensionnant automatiquement le formulaire et ses contrôles enfants en fonction de la taille de police ou résolution d'affichage relative. Le système d'exploitation Windows prend en charge la mise à l'échelle automatique des boîtes de dialogue à l'aide d'une unité de mesure relative appelée unité de boîte de dialogue. Une unité de boîte de dialogue est basée sur la police système et sa relation aux pixels peut être déterminée à l'aide de la fonction du SDK Win32 `GetDialogBaseUnits`. Quand un utilisateur change le thème utilisé par Windows, toutes les boîtes de dialogue sont automatiquement ajustés en conséquence. En outre, le .NET Framework prend en charge la mise à l'échelle automatique en fonction de la police système par défaut ou de la résolution d'affichage. Le cas échéant, la mise à l'échelle automatique peut être désactivée dans une application.

Prise en charge d'origine de la mise à l'échelle automatique

Les versions 1,0 et 1,1 du .NET Framework pris en charge la mise à l'échelle automatique d'une manière simple qui dépendait de la police Windows par défaut utilisée pour l'interface utilisateur, représentée par la valeur **DEFAULT_GUI_FONT** du kit de développement logiciel (SDK) Win32. Cette police est généralement modifiée uniquement quand la résolution d'écran change. Le mécanisme suivant était utilisé pour implémenter la mise à

l'échelle automatique :

1. Au moment du design, la propriété [AutoSizeBaseSize](#) (qui est maintenant déconseillée) prenait comme valeur la hauteur et la largeur de la police système par défaut sur l'ordinateur du développeur.
2. Au moment de l'exécution, la police système par défaut de l'ordinateur de l'utilisateur était utilisée pour initialiser la propriété [Font](#) de la classe [Form](#).
3. Avant d'afficher le formulaire, la méthode [ApplyAutoScaling](#) était appelée pour mettre à l'échelle le formulaire. Cette méthode calculait les tailles d'échelle relatives à partir des propriétés [AutoSizeBaseSize](#) et [Font](#), puis appelait la méthode [Scale](#) pour mettre à l'échelle le formulaire et ses enfants.
4. La valeur de [AutoSizeBaseSize](#) était mise à jour pour que les appels ultérieurs à [ApplyAutoScaling](#) ne redimensionnent pas progressivement le formulaire.

Même si ce mécanisme était suffisant dans la plupart des cas, il souffrait des limitations suivantes :

- Étant donné que la propriété [AutoSizeBaseSize](#) représente la taille de police de base sous forme de valeurs entières, des erreurs d'arrondi apparaissent clairement quand un formulaire passe par plusieurs résolutions.
- La mise à l'échelle automatique était implémentée uniquement dans la classe [Form](#), et non dans la classe [ContainerControl](#). Ainsi, les contrôles utilisateur étaient mis à l'échelle correctement uniquement quand le contrôle utilisateur était conçu à la même résolution que le formulaire et quand il était placé dans le formulaire au moment du design.
- Les formulaires et leurs contrôles enfants pouvaient uniquement être conçus simultanément par plusieurs développeurs si la résolution de leurs ordinateurs était identique. De même, cela rendait l'héritage d'un formulaire dépendant de la résolution associée au formulaire parent.
- Il n'est pas compatible avec les gestionnaires de mise en page plus récents introduits avec la version 2,0 de .NET Framework, tels que [FlowLayoutPanel](#) et [TableLayoutPanel](#).
- Il ne prenait pas en charge la mise à l'échelle basée directement sur la résolution d'affichage requise pour la compatibilité avec le .NET Compact Framework.

Bien que ce mécanisme soit conservé dans la version 2,0 de .NET Framework pour assurer la compatibilité descendante, il a été remplacé par le mécanisme de mise à l'échelle plus fiable décrit ci-après. Par conséquent, [AutoScale](#), [ApplyAutoScaling](#), [AutoSizeBaseSize](#) et certaines surcharges [Scale](#) sont marquées comme obsolètes.

NOTE

Vous pouvez supprimer sans risque les références à ces membres quand vous mettez à niveau votre code hérité vers le .NET Framework version 2,0.

Prise en charge actuelle de la mise à l'échelle automatique

La version 2,0 de .NET Framework surmonte les limitations précédentes en introduisant les modifications suivantes de la mise à l'échelle automatique des Windows Forms :

- La prise en charge de base de la mise à l'échelle a été déplacée vers la classe [ContainerControl](#) pour que les formulaires, les contrôles composites natifs et les contrôles utilisateur bénéficient tous d'une prise en charge uniforme de la mise à l'échelle. Les nouveaux membres [AutoScaleFactor](#), [AutoScaleDimensions](#), [AutoScaleMode](#) et [PerformAutoScale](#) ont été ajoutés.
- La classe [Control](#) comporte également plusieurs nouveaux membres qui lui permettent de participer à la mise à l'échelle et de prendre en charge la mise à l'échelle mixte sur le même formulaire. En particulier les membres [Scale](#), [ScaleChildren](#), et [GetScaledBounds](#) prennent en charge la mise à l'échelle.

- La prise en charge de la mise à l'échelle en fonction de la résolution d'écran a été ajoutée pour compléter la prise en charge de la police système, comme défini par l'énumération [AutoScaleMode](#). Ce mode est compatible avec la mise à l'échelle automatique prise en charge par le .NET Compact Framework facilitant la migration des applications.
- La compatibilité avec les gestionnaires de présentation tels que [FlowLayoutPanel](#) et [TableLayoutPanel](#) a été ajoutée à l'implémentation de la mise à l'échelle automatique.
- Les facteurs d'échelle sont désormais représentés sous forme de valeurs à virgule flottante, généralement avec la structure [SizeF](#). Les erreurs d'arrondi sont donc pratiquement inexistantes.

Caution

Les combinaisons arbitraires de modes de mise à l'échelle de police et PPP ne sont pas prises en charge. Bien que vous puissiez mettre à l'échelle un contrôle utilisateur à l'aide d'un mode (par exemple PPP) et le placer sur un formulaire à l'aide d'un autre mode (Police) sans problème, le fait de combiner un formulaire de base dans un mode et un formulaire dérivé dans un autre peut produire des résultats inattendus.

Mise à l'échelle automatique en action

Windows Forms utilise désormais la logique suivante pour mettre automatiquement à l'échelle les formulaires et leur contenu :

1. Au moment du design, chaque [ContainerControl](#) enregistre le mode de mise à l'échelle et sa résolution actuelle dans les propriétés [AutoScaleMode](#) et [AutoScaleDimensions](#), respectivement.
2. Au moment de l'exécution, la résolution réelle est stockée dans la propriété [CurrentAutoScaleDimensions](#). La propriété [AutoScaleFactor](#) calcule dynamiquement le rapport entre la résolution de mise à l'échelle au moment de l'exécution et au moment du design.
3. Quand le formulaire se charge, si les valeurs de [CurrentAutoScaleDimensions](#) et [AutoScaleDimensions](#) sont différentes, la méthode [PerformAutoScale](#) est appelée pour mettre à l'échelle le contrôle et ses enfants. Cette méthode suspend la disposition et appelle la méthode [Scale](#) pour effectuer la mise à l'échelle. Par la suite, la valeur de [AutoScaleDimensions](#) est mise à jour pour éviter la mise à l'échelle progressive.
4. [PerformAutoScale](#) est également appelée automatiquement dans les situations suivantes :
 - En réponse à l'événement [OnFontChanged](#) si le mode de mise à l'échelle est [Font](#).
 - Quand la disposition du contrôle conteneur reprend et qu'une modification est détectée dans la propriété [AutoScaleDimensions](#) ou [AutoScaleMode](#).
 - Comme expliqué ci-dessus, quand un [ContainerControl](#) parent est mis à l'échelle. Chaque contrôle conteneur est responsable de la mise à l'échelle de ses enfants à l'aide de ses propres facteurs d'échelle, et non de celui de son conteneur parent.
5. Les contrôles enfants peuvent modifier leur comportement de mise à l'échelle de plusieurs manières :
 - La propriété [ScaleChildren](#) peut être substituée pour déterminer si leurs contrôles enfants doivent être mis à l'échelle ou non.
 - La méthode [GetScaledBounds](#) peut être substituée pour ajuster les limites de mise à l'échelle du contrôle, mais pas la logique de mise à l'échelle.
 - La méthode [ScaleControl](#) peut être substituée pour modifier la logique de mise à l'échelle pour le contrôle actuel.

Voir aussi

- [AutoScaleMode](#)
- [Scale](#)

- [PerformAutoScale](#)
- [AutoScaleDimensions](#)
- [Rendu des contrôles avec les styles visuels](#)
- [Guide pratique pour améliorer les performances en évitant la mise à l'échelle automatique](#)

Comment : répondre aux modifications de jeu de polices dans une application Windows Forms

5 minutes to read • [Edit Online](#)

Dans les systèmes d'exploitation Windows, un utilisateur peut modifier les paramètres de police à l'échelle du système pour que la police par défaut apparaisse plus grande ou plus petite. La modification de ces paramètres de police est essentielle pour les utilisateurs qui sont affectés visuellement et qui nécessitent un plus grand type de lecture du texte sur leurs écrans. Vous pouvez ajuster votre application Windows Forms pour réagir à ces modifications en accroissant ou en diminuant la taille du formulaire et de tout le texte contenu chaque fois que le jeu de polices change. Si vous souhaitez que votre formulaire accepte les modifications de taille de police de manière dynamique, vous pouvez ajouter du code à votre formulaire.

En règle générale, la police par défaut utilisée par Windows Forms est la police retournée par l'appel de l'espace de noms [Microsoft.Win32](#) à `GetStockObject(DEFAULT_GUI_FONT)`. La police retournée par cet appel change uniquement lorsque la résolution d'écran change. Comme indiqué dans la procédure suivante, votre code doit modifier la police par défaut en [IconTitleFont](#) pour répondre aux modifications de la taille de police.

Pour utiliser la police du bureau et répondre aux modifications du jeu de polices

1. Créez votre formulaire et ajoutez-y les contrôles qui vous intéressent. Pour plus d'informations, consultez [Comment : créer un Windows Forms application à partir de la ligne de commande](#) et des [contrôles à utiliser sur Windows Forms](#).
2. Ajoutez une référence à l'espace de noms [Microsoft.Win32](#) dans votre code.

```
using Microsoft.Win32;
```

```
Imports Microsoft.Win32
```

3. Ajoutez le code suivant au constructeur de votre formulaire pour raccorder les gestionnaires d'événements requis et modifier la police par défaut en cours d'utilisation pour le formulaire.

```
this.Font = SystemFonts.IconTitleFont;  
SystemEvents.UserPreferenceChanged += new  
UserPreferenceChangedEventHandler(SystemEvents_UserPreferenceChanged);  
this.FormClosing += new FormClosingEventHandler(Form1_FormClosing);
```

```
Public Sub New()  
    ' This call is required by the Windows Form Designer.  
    InitializeComponent()  
  
    ' Add any initialization after the InitializeComponent() call.  
    AddHandler SystemEvents.UserPreferenceChanged, New UserPreferenceChangedEventHandler(AddressOf  
SystemEvents_UserPreferenceChangesEventHandler)  
End Sub
```

4. Implémentez un gestionnaire pour l'événement [UserPreferenceChanged](#) qui provoque la mise à l'échelle automatique du formulaire lorsque la catégorie [Window](#) change.

```
void SystemEvents_UserPreferenceChanged(object sender, UserPreferenceChangedEventArgs e)
{
    if (e.Category == UserPreferenceCategory.Window)
    {
        this.Font = SystemFonts.IconTitleFont;
    }
}
```

```
Private Sub SystemEvents_UserPreferenceChangesEventHandler(ByVal sender As Object, ByVal e As
UserPreferenceChangedEventArgs)
    If (e.Category = UserPreferenceCategory.Window) Then
        Me.Font = SystemFonts.IconTitleFont
    End If
End Sub
```

5. Enfin, implémentez un gestionnaire pour l'événement [FormClosing](#) qui détache le gestionnaire d'événements [UserPreferenceChanged](#).

IMPORTANT

Si vous n'incluez pas ce code, votre application risque de provoquer une fuite de mémoire.

```
void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    SystemEvents.UserPreferenceChanged -= new
UserPreferenceChangedEventArgs(SystemEvents_UserPreferenceChanged);
}
```

```
Private Sub Form1_FormClosing(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
    RemoveHandler SystemEvents.UserPreferenceChanged, New UserPreferenceChangedEventArgs(AddressOf
SystemEvents_UserPreferenceChangesEventHandler)
End Sub
```

6. Compilez, puis exécutez le code.

Pour modifier manuellement le jeu de polices dans Windows XP

1. Pendant que votre application Windows Forms est en cours d'exécution, cliquez avec le bouton droit sur le bureau Windows, puis choisissez **Propriétés** dans le menu contextuel.
2. Dans la boîte de dialogue **Propriétés de Affichage**, cliquez sur l'onglet **Apparence**.
3. Dans la zone de liste déroulante taille de la **police**, sélectionnez une nouvelle taille de police.

Vous remarquerez que le formulaire réagit maintenant aux modifications au moment de l'exécution dans le schéma de police du bureau. Lorsque l'utilisateur change de police **normale**, de **grandes polices** et de très **grandes polices**, le formulaire change la police et s'adapte correctement.

Exemple

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Microsoft.Win32;

namespace WinFormsAutoScaling
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            this.Font = SystemFonts.IconTitleFont;
            SystemEvents.UserPreferenceChanged += new
UserPreferenceChangedEventHandler(SystemEvents_UserPreferenceChanged);
            this.FormClosing += new FormClosingEventHandler(Form1_FormClosing);
        }

        void SystemEvents_UserPreferenceChanged(object sender, UserPreferenceChangedEventArgs e)
        {
            if (e.Category == UserPreferenceCategory.Window)
            {
                this.Font = SystemFonts.IconTitleFont;
            }
        }

        void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            SystemEvents.UserPreferenceChanged -= new
UserPreferenceChangedEventHandler(SystemEvents_UserPreferenceChanged);
        }
    }
}

```

```

Imports Microsoft.Win32

Public Class Form1
    Public Sub New()
        ' This call is required by the Windows Form Designer.
        InitializeComponent()

        ' Add any initialization after the InitializeComponent() call.
        AddHandler SystemEvents.UserPreferenceChanged, New UserPreferenceChangedEventHandler(AddressOf
SystemEvents_UserPreferenceChangesEventHandler)
    End Sub

    Private Sub SystemEvents_UserPreferenceChangesEventHandler(ByVal sender As Object, ByVal e As
UserPreferenceChangedEventArgs)
        If (e.Category = UserPreferenceCategory.Window) Then
            Me.Font = SystemFonts.IconTitleFont
        End If
    End Sub

    Private Sub Form1_FormClosing(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
        RemoveHandler SystemEvents.UserPreferenceChanged, New UserPreferenceChangedEventHandler(AddressOf
SystemEvents_UserPreferenceChangesEventHandler)
    End Sub
End Class

```

Le constructeur de cet exemple de code contient un appel à `InitializeComponent`, qui est défini lorsque vous créez un projet de Windows Forms dans Visual Studio. Supprimez cette ligne de code si vous générez votre application sur la ligne de commande.

Voir aussi

- [PerformAutoScale](#)
- [Mise à l'échelle automatique dans les Windows Forms](#)

Prise en charge des résolutions élevées en Windows Forms

7 minutes to read • [Edit Online](#)

À partir de la .NET Framework 4,7, Windows Forms comprend des améliorations pour les scénarios haute résolution et PPP dynamique courants. Elles incluent notamment les suivantes :

- Améliorations de la mise à l'échelle et de la disposition d'un certain nombre de contrôles Windows Forms, tels que le contrôle [MonthCalendar](#) et le contrôle [CheckedListBox](#).
- Mise à l'échelle à passage unique. Dans le .NET Framework 4,6 et les versions antérieures, la mise à l'échelle a été effectuée à travers plusieurs passes, ce qui a entraîné une mise à l'échelle de certains contrôles plus que nécessaire.
- Prise en charge des scénarios PPP dynamiques dans lesquels l'utilisateur modifie la résolution ou le facteur d'échelle après le lancement d'une application Windows Forms.

Dans les versions de la .NET Framework à partir de la .NET Framework 4,7, la prise en charge de la haute résolution améliorée est une fonctionnalité d'abonnement. Vous devez configurer votre application pour en tirer parti.

Configuration de votre application Windows Forms pour la prise en charge de la haute résolution

Les nouvelles fonctionnalités de Windows Forms qui prennent en charge la reconnaissance haute résolution sont uniquement disponibles dans les applications qui ciblent le .NET Framework 4,7 et qui s'exécutent sur des systèmes d'exploitation Windows à partir de Windows 10 Creators Update.

En outre, pour configurer la prise en charge des résolutions élevées dans votre application Windows Forms, vous devez effectuer les opérations suivantes :

- Déclarer la compatibilité avec Windows 10.

Pour ce faire, ajoutez le code suivant à votre fichier manifeste :

```
<compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
  <application>
    <!-- Windows 10 compatibility -->
    <supportedOS Id="{8e0f7a12-bfb3-4fe8-b9a5-48fd50a15a9a}" />
  </application>
</compatibility>
```

- Activez la reconnaissance de la résolution par moniteur dans le fichier *app.config* .

Windows Forms introduit un nouvel élément `<System.Windows.Forms.ApplicationConfigurationSection>` pour prendre en charge les nouvelles fonctionnalités et personnalisations ajoutées à partir du .NET Framework 4,7. Pour tirer parti des nouvelles fonctionnalités qui prennent en charge la haute résolution, ajoutez ce qui suit à votre fichier de configuration d'application.


```
<System.Windows.Forms.ApplicationConfigurationSection>
  <add key="DpiAwareness" value="PerMonitorV2" />
</System.Windows.Forms.ApplicationConfigurationSection>
```

IMPORTANT

Dans les versions précédentes de la .NET Framework, vous utilisiez le manifeste pour ajouter la prise en charge des résolutions élevées. Cette approche n'est plus recommandée, car elle remplace les paramètres définis dans le fichier app. config.

- Appelez la méthode statique [EnableVisualStyles](#).

Il doit s'agir du premier appel de méthode dans le point d'entrée de votre application. Par exemple :

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form2());
}
```

Désactivation des fonctionnalités haute résolution individuelles

La définition de la valeur `DpiAwareness` sur `PerMonitorV2` active toutes les fonctionnalités de reconnaissance haute résolution prises en charge par les versions de .NET Framework à partir de la .NET Framework 4,7. En règle générale, cela convient à la plupart des applications Windows Forms. Toutefois, vous souhaitez peut-être refuser une ou plusieurs fonctionnalités individuelles. La raison la plus importante pour cela est que votre code d'application existant gère déjà cette fonctionnalité. Par exemple, si votre application gère la mise à l'échelle automatique, vous souhaitez peut-être désactiver la fonctionnalité de redimensionnement automatique comme suit :

```
<System.Windows.Forms.ApplicationConfigurationSection>
  <add key="DpiAwareness" value="PerMonitorV2" />
  <add key="EnableWindowsFormsHighDpiAutoResizing" value="false" />
</System.Windows.Forms.ApplicationConfigurationSection>
```

Pour obtenir la liste des clés individuelles et leurs valeurs, consultez [Windows Forms ajouter un élément de configuration](#).

Nouveaux événements de modification DPI

À partir de la .NET Framework 4,7, trois nouveaux événements vous permettent de gérer par programmation les modifications PPP dynamiques :

- [DpiChangedAfterParent](#), qui est déclenché lorsque le paramètre DPI d'un contrôle est modifié par programme après qu'un événement de modification de PPP s'est produit pour son contrôle ou formulaire parent.
- [DpiChangedBeforeParent](#), qui est déclenché lorsque le paramètre DPI d'un contrôle est modifié par programme avant qu'un événement de modification DPI pour son contrôle ou formulaire parent s'est produit.
- [DpiChanged](#), qui est déclenché lorsque le paramètre DPI change sur le périphérique d'affichage où le formulaire est actuellement affiché.

Nouvelles méthodes et propriétés d'assistance

La .NET Framework 4,7 ajoute également un certain nombre de nouvelles méthodes et propriétés d'assistance qui fournissent des informations sur la mise à l'échelle DPI et vous permettent d'effectuer une mise à l'échelle PPP. Elles incluent notamment les suivantes :

- [LogicalToDeviceUnits](#), qui convertit une valeur logique en pixels de périphérique.
- [ScaleBitmapLogicalToDevice](#), qui met à l'échelle une image bitmap sur les DPI logiques pour un appareil.
- [DeviceDpi](#), qui retourne la valeur PPP de l'appareil actuel.

Considérations relatives au contrôle de version

En plus de s'exécuter sur .NET Framework 4,7 et Windows 10 Creators Update, votre application peut également s'exécuter dans un environnement dans lequel elle n'est pas compatible avec les améliorations à haute résolution. Dans ce cas, vous devez développer une solution de secours pour votre application. Vous pouvez effectuer cette opération pour effectuer un [dessin personnalisé](#) afin de gérer la mise à l'échelle.

Pour ce faire, vous devez également déterminer le système d'exploitation sur lequel votre application s'exécute. Vous pouvez le faire avec du code semblable au suivant :

```
// Create a reference to the OS version of Windows 10 Creators Update.
Version OsMinVersion = new Version(10, 0, 15063, 0);

// Access the platform/version of the current OS.
Console.WriteLine(Environment.OSVersion.Platform.ToString());
Console.WriteLine(Environment.OSVersion.VersionString);

// Compare the current version to the minimum required version.
Console.WriteLine(Environment.OSVersion.Version.CompareTo(OsMinVersion));
```

Notez que votre application ne détectera pas correctement Windows 10 si elle n'était pas listée en tant que système d'exploitation pris en charge dans le manifeste de l'application.

Vous pouvez également vérifier la version de la .NET Framework à laquelle l'application a été générée :

```
Console.WriteLine(AppDomain.CurrentDomain.SetupInformation.TargetFrameworkName);
```

Voir aussi

- [Windows Forms ajouter un élément de configuration](#)
- [Réglage de la taille et de l'échelle des Windows Forms](#)

Modification de l'apparence des Windows Forms

2 minutes to read • [Edit Online](#)

Vous pouvez personnaliser l'apparence de vos applications Windows Forms de nombreuses manières, par exemple en modifiant la bordure, l'opacité, la forme ou le style ou en définissant une image d'arrière-plan.

Dans cette section

[Guide pratique pour modifier les bordures des Windows Forms](#)

Montre comment modifier le style de bordure d'un formulaire.

Référence

[Form](#)

Décrit cette classe et propose des liens vers tous ses membres.

[FormBorderStyle](#)

Décrit cette énumération et contient des descriptions de tous ses membres.

[VisualStyleRenderer](#)

Décrit cette classe et propose des liens vers tous ses membres.

[Image](#)

Décrit cette classe et propose des liens vers tous ses membres.

[Region](#)

Décrit cette classe et propose des liens vers tous ses membres.

[Color](#)

Décrit cette classe et propose des liens vers tous ses membres.

Sections connexes

[Réglage de la taille et de l'échelle des Windows Forms](#)

Contient des liens vers des rubriques qui montrent comment modifier la taille et l'échelle d'un formulaire.

[Graphiques et dessins dans Windows Forms](#)

Contient des liens vers des rubriques qui décrivent comment effectuer un dessin personnalisé dans des Windows Forms.

[Contrôles avec prise en charge intégrée du dessin owner-drawn](#)

Indique la prise en charge de la fonctionnalité Owner-Draw dans les contrôles Windows Forms.

Comment : modifier les bordures des Windows Forms

2 minutes to read • [Edit Online](#)

Vous pouvez choisir parmi plusieurs styles de bordure quand vous déterminez l'apparence et le comportement de vos Windows Forms. En modifiant la propriété [FormBorderStyle](#), vous pouvez contrôler le comportement de redimensionnement du formulaire. De plus, la définition de [FormBorderStyle](#) affecte l'affichage de la barre de légende et de ses boutons. Pour plus d'informations, consultez [FormBorderStyle](#).

Cette tâche est très bien prise en charge dans Visual Studio.

Voir aussi [Comment : modifier les bordures d'Windows Forms à l'aide du concepteur](#).

Pour définir le style de bordure des Windows Forms par programmation

- Affectez à la propriété [FormBorderStyle](#) le style souhaité. L'exemple de code suivant définit le style de bordure des `DlgBx1` de formulaire sur [FixedDialog](#).

```
DlgBx1.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog
```

```
DlgBx1.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;
```

```
DlgBx1->FormBorderStyle =  
    System::Windows::Forms::FormBorderStyle::FixedDialog;
```

Consultez également [Comment : créer des boîtes de dialogue au moment du design](#).

En outre, si vous avez choisi un style de bordure pour le formulaire qui fournit des boutons de **réduction** et d' **agrandissement** facultatifs, vous pouvez spécifier si vous souhaitez que l'un ou l'autre ou les deux boutons soient fonctionnels. Ils sont utiles quand vous souhaitez contrôler étroitement l'expérience utilisateur. Les boutons **réduire** et **agrandir** sont activés par défaut et leurs fonctionnalités sont manipulées par le biais de la fenêtre **Propriétés**.

Voir aussi

- [FormBorderStyle](#)
- [FixedDialog](#)
- [Bien démarrer avec Windows Forms](#)

Contrôles Windows Forms

2 minutes to read • [Edit Online](#)

À mesure que vous concevez et modifiez l'interface utilisateur de vos applications Windows Forms, vous devrez ajouter, aligner et positionner des contrôles. Les contrôles sont des objets contenus dans des objets de formulaire. Chaque type de contrôle possède son propre ensemble de propriétés, méthodes et événements qui le rendent adapté à un usage particulier. Vous pouvez manipuler les contrôles dans le concepteur et écrire du code pour ajouter dynamiquement des contrôles au moment de l'exécution.

Contenu de cette section

[Placement de contrôles dans les Windows Forms](#)

Fournit des liens relatifs au placement de contrôles dans les formulaires.

[Réorganisation des contrôles sur Windows Forms](#)

Articles relatifs à la réorganisation des contrôles sur les formulaires.

[Étiquetage des contrôles de Windows Forms individuels et leur fournissant des raccourcis](#)

Décrit les utilisations des raccourcis clavier, des étiquettes de texte sur les contrôles et des touches de modification.

[Contrôles à utiliser sur Windows Forms](#)

Répertorie les contrôles qui fonctionnent avec Windows Forms et les tâches de base que vous pouvez effectuer avec chaque contrôle.

[Développement de contrôles Windows Forms personnalisés avec le .NET Framework](#)

Fournit des informations générales et des exemples pour aider les utilisateurs à développer des contrôles Windows Forms personnalisés.

[Développement de contrôles Windows Forms au moment du design](#)

Décrit les techniques de création de contrôles personnalisés par conception et héritage.

Rubriques connexes

[Applications clientes](#)

Fournit une vue d'ensemble du développement d'applications Windows.

Entrées d'utilisateur dans les Windows Forms

2 minutes to read • [Edit Online](#)

Windows Forms comprend un modèle d'entrée d'utilisateur basé sur les événements qui sont déclenchés lors du traitement des messages Windows connexes. Les rubriques de cette section fournissent des informations sur l'entrée d'utilisateur avec la souris et le clavier, y compris des exemples de code qui montrent comment effectuer des tâches spécifiques.

Dans cette section

[Entrée d'utilisateur dans une application Windows Forms](#)

Fournit une vue d'ensemble des événements d'entrée d'utilisateur et des méthodes qui traitent les messages Windows.

[Entrée au clavier dans une application Windows Forms](#)

Fournit des informations sur la gestion des messages du clavier, les différents types de touches et les événements de clavier.

[Entrée de la souris dans une application Windows Forms](#)

Fournit des informations sur les événements de souris et d'autres fonctionnalités liées à la souris, y compris les curseurs de souris et la capture de souris.

[Guide pratique pour simuler des événements de la souris et du clavier dans le code](#)

Illustre différentes manières de simuler l'entrée de souris et de clavier par programmation.

[Guide pratique pour gérer des événements d'entrée d'utilisateur dans les contrôles Windows Forms](#)

Présente un exemple de code qui gère la plupart des événements d'entrée d'utilisateur et fournit des informations sur chaque événement.

[Validation des entrées d'utilisateur dans les Windows Forms](#)

Décrit les méthodes permettant de valider l'entrée d'utilisateur dans les applications Windows Forms.

Sections connexes

Consultez également [création de gestionnaires d'événements dans Windows Forms](#).

Entrée d'utilisateur dans une application Windows Forms

4 minutes to read • [Edit Online](#)

Dans Windows Forms, l'entrée utilisateur est envoyée aux applications sous la forme de messages Windows. Une série de méthodes substituables traite ces messages au niveau de l'application, du formulaire et du contrôle. Lorsque ces méthodes reçoivent des messages de souris et de clavier, elles déclenchent des événements qui peuvent être gérés pour obtenir des informations sur la souris ou l'entrée au clavier. Dans de nombreux cas, Windows Forms applications peuvent traiter toutes les entrées utilisateur simplement en gérant ces événements. Dans d'autres cas, une application peut avoir besoin de substituer l'une des méthodes qui traitent les messages pour intercepter un message particulier avant qu'il ne soit reçu par l'application, le formulaire ou le contrôle.

Événements de souris et de clavier

Tous les contrôles Windows Forms héritent d'un ensemble d'événements liés à l'entrée de la souris et du clavier. Par exemple, un contrôle peut gérer l'événement [KeyPress](#) pour déterminer le code de caractère d'une touche qui a été enfoncée, ou un contrôle peut gérer l'événement [MouseClick](#) pour déterminer l'emplacement d'un clic de souris. Pour plus d'informations sur les événements de souris et de clavier, consultez [utilisation des événements de clavier](#) et des [événements de souris dans Windows Forms](#).

Méthodes qui traitent les messages d'entrée utilisateur

Les formulaires et les contrôles ont accès à l'interface [IMessageFilter](#) et à un ensemble de méthodes substituables qui traitent les messages Windows à différents moments de la file d'attente de messages. Ces méthodes ont toutes un paramètre [Message](#), qui encapsule les détails de bas niveau des messages Windows. Vous pouvez implémenter ou substituer ces méthodes pour examiner le message, puis utiliser le message ou le passer au consommateur suivant dans la file d'attente de messages. Le tableau suivant présente les méthodes qui traitent tous les messages Windows dans Windows Forms.

MÉTHODE	REMARQUES
PreFilterMessage	Cette méthode intercepte les messages Windows mis en file d'attente (également appelés) au niveau de l'application.
PreProcessMessage	Cette méthode intercepte les messages Windows au niveau du formulaire et du contrôle avant qu'ils aient été traités.
WndProc	Cette méthode traite les messages Windows au niveau du formulaire et du contrôle.
DefWndProc	Cette méthode effectue le traitement par défaut des messages Windows au niveau du formulaire et du contrôle. Cela fournit les fonctionnalités minimales d'une fenêtre.
OnNotifyMessage	Cette méthode intercepte les messages au niveau du formulaire et du contrôle, une fois qu'ils ont été traités. Le bit de style de EnableNotifyMessage doit être défini pour que cette méthode soit appelée.

Les messages du clavier et de la souris sont également traités par un ensemble supplémentaire de méthodes

substituables qui sont spécifiques à ces types de messages. Pour plus d'informations, consultez [fonctionnement de l'entrée au clavier](#) et [fonctionnement de l'entrée de la souris dans Windows Forms](#).

Voir aussi

- [Entrée d'utilisateur dans Windows Forms](#)
- [Entrée au clavier dans une application Windows Forms](#)
- [Entrée de la souris dans une application Windows Forms](#)

Entrée au clavier dans une application Windows Forms

2 minutes to read • [Edit Online](#)

Windows Forms inclut des événements de clavier standard qui vous permettent de répondre à des enfoncements de touches spécifiques, et vous permet également d'intercepter, de modifier et d'utiliser des enfoncés de touches au niveau de l'application, du formulaire et du contrôle.

Dans cette section

[Fonctionnement de l'entrée au clavier](#)

Décrit comment les messages de clavier sont traités et transformés en événements de clavier.

[Utilisation des événements de clavier](#)

Fournit des informations sur les types d'événements de clavier et les informations reçues par les gestionnaires d'événements de clavier.

[Guide pratique pour modifier l'entrée au clavier pour un contrôle standard](#)

Présente un exemple de code qui montre comment modifier des valeurs de clés avant qu'elles atteignent un contrôle.

[Guide pratique pour identifier la touche de modification activée](#)

Montre comment déterminer si l'utilisateur a appuyé sur la touche Maj, ALT ou CTRL en plus d'une autre clé.

[Guide pratique pour gérer l'entrée au clavier au niveau du formulaire](#)

Présente un exemple de code qui montre comment intercepter des clés avant qu'elles atteignent un contrôle.

Fonctionnement de l'entrée au clavier

11 minutes to read • [Edit Online](#)

Windows Forms traite l'entrée au clavier en déclenchant des événements de clavier en réponse aux messages de Windows. La plupart des applications Windows Forms traitent l'entrée au clavier exclusivement en gérant les événements de clavier. Toutefois, vous devez comprendre le fonctionnement des messages de clavier pour pouvoir implémenter des scénarios d'entrée au clavier plus avancés, comme l'interception des touches avant qu'elles n'atteignent un contrôle. Cette rubrique décrit les types de données de touches que Windows Forms reconnaît et fournit une vue d'ensemble du routage des messages de clavier. Pour plus d'informations sur les événements de clavier, consultez [Utilisation des événements de clavier](#).

Types de touches

Windows Forms identifie l'entrée au clavier en tant que codes de clé virtuelle représentés par [Keys](#) l'énumération de bits. Avec l' [Keys](#) énumération, vous pouvez combiner une série de touches enfoncées pour générer une valeur unique. Ces valeurs correspondent aux valeurs qui accompagnent les messages Windows WM_KEYDOWN et WM_SYSKEYDOWN. Vous pouvez détecter la plupart des presse-clés physiques en [KeyDown](#) gérant [KeyUp](#) les événements ou. Les touches de caractère sont un sous [Keys](#) -ensemble de l'énumération et correspondent aux valeurs qui accompagnent les messages Windows WM_CHAR et WM_SYSCHAR. Si la combinaison de touches enfoncées produit un caractère, vous pouvez détecter le caractère en gérant [KeyPress](#) l'événement. Vous pouvez également utiliser [Keyboard](#), exposé par Visual Basic interface de programmation, pour découvrir les touches qui ont été enfoncées et les touches d'envoi. Pour plus d'informations, consultez [Accès au clavier](#).

Ordre des événements de clavier

Comme indiqué précédemment, il existe 3 événements liés au clavier qui peuvent se produire sur un contrôle. La séquence suivante montre l'ordre général des événements :

1. L'utilisateur pousse la touche «a», la clé est prétraitée, distribuée et un [KeyDown](#) événement se produit.
2. L'utilisateur maintient la touche «a», la clé est prétraitée, distribuée et un [KeyPress](#) événement se produit.

Cet événement se produit plusieurs fois lorsque l'utilisateur maintient une touche enfoncée.

3. L'utilisateur relâche la touche «a», la clé est prétraitée, distribuée et un [KeyUp](#) événement se produit.

Prétraitement des touches

Comme les autres messages, les messages de clavier sont [WndProc](#) traités dans la méthode d'un formulaire ou d'un contrôle. Toutefois, avant le traitement des messages de clavier [PreProcessMessage](#), la méthode appelle une ou plusieurs méthodes qui peuvent être substituées pour gérer des clés de caractères spéciaux et des clés physiques. Vous pouvez substituer ces méthodes pour détecter et filtrer certaines touches avant que les messages ne soient traités par le contrôle. Le tableau suivant présente l'action exécutée et la méthode associée qui se produit, dans l'ordre où la méthode se produit.

Prétraitement d'un événement KeyDown

ACTION	MÉTHODE ASSOCIÉE	NOTES
Rechercher une touche de commande comme un accélérateur ou un raccourci du menu.	ProcessCmdKey	Cette méthode traite une touche de commande, qui est prioritaire par rapport aux touches ordinaires. Si cette méthode renvoie <code>true</code> , le message de la touche n'est pas distribué et aucun événement de touche ne se produit. Si elle retourne <code>false</code> , IsInputKey est appelée.
Recherchez une touche spéciale qui nécessite un prétraitement ou une touche de caractère normale qui doit déclencher un KeyDown événement et être distribuée à un contrôle.	IsInputKey	Si la méthode retourne <code>true</code> , cela signifie que le contrôle est un caractère normal et KeyDown qu'un événement est déclenché. Si <code>false</code> la méthode est appelée. ProcessDialogKey Remarque : Pour garantir qu'un contrôle obtient une clé ou une combinaison de touches, vous pouvez PreviewKeyDown gérer l'événement IsInputKey et l' PreviewKeyDownEventArgs ensemble <code>true</code> du à pour la ou les clés que vous souhaitez.
Rechercher une touche de navigation (touches ÉCHAP, de tabulation, de retour ou de direction).	ProcessDialogKey	Cette méthode traite une touche physique qui emploie une fonctionnalité spéciale dans le contrôle, comme basculer l'objectif entre le contrôle et son parent. Si le contrôle immédiat ne gère pas la clé, ProcessDialogKey est appelé sur le contrôle parent, et ainsi de suite au contrôle le plus haut dans la hiérarchie. Si cette méthode renvoie <code>true</code> , le prétraitement est terminé et un événement de touche n'est pas généré. Si elle retourne <code>false</code> , un KeyDown événement se produit.

Prétraitement d'un événement KeyPress

ACTION	MÉTHODE ASSOCIÉE	NOTES
Vérifier si la touche est un caractère normal qui doit être traité par le contrôle	IsInputChar	Si le caractère est un caractère normal, cette méthode retourne <code>true</code> , l' KeyPress événement est déclenché et aucun prétraitement supplémentaire ne se produit. Sinon ProcessDialogChar , sera appelé.
Vérifier si le caractère est un caractère mnémonique (comme &OK sur un bouton)	ProcessDialogChar	Cette méthode, similaire à ProcessDialogKey , sera appelée dans la hiérarchie des contrôles. Si le contrôle est un contrôle conteneur, il recherche les mnémoniques en appelant ProcessMnemonic lui-même et ses contrôles enfants. Si ProcessDialogChar retourne <code>true</code> , un KeyPress événement ne se produit pas.

Traitement des messages de clavier

Une fois que les messages [WndProc](#) de clavier atteignent la méthode d'un formulaire ou d'un contrôle, ils sont traités par un ensemble de méthodes qui peuvent être substituées. Chacune de ces méthodes retourne une [Boolean](#) valeur qui spécifie si le message du clavier a été traité et consommé par le contrôle. Si l'une des méthodes renvoie `true`, alors le message est considéré comme traité et n'est pas transmis à la base ou au parent du contrôle pour un traitement ultérieur. Dans le cas contraire, le message reste dans la file d'attente des messages et peut être traité dans une autre méthode de la base ou du parent du contrôle. Le tableau suivant présente les méthodes qui traitent les messages de clavier.

MÉTHODE	NOTES
ProcessKeyMessage	Cette méthode traite tous les messages de clavier reçus par la WndProc méthode du contrôle.
ProcessKeyPreview	Cette méthode envoie le message de clavier au parent du contrôle. Si ProcessKeyPreview ProcessKeyEventArgs retourne <code>true</code> , aucun événement de touche n'est généré; sinon, est appelé.
ProcessKeyEventArgs	Cette méthode déclenche les KeyDown événements KeyPress , et KeyUp , le cas échéant.

Substitution des méthodes de clavier

Il existe de nombreuses méthodes pouvant être substituées pendant le prétraitement et le traitement d'un message de clavier. Toutefois, certaines méthodes sont mieux adaptées que d'autres. Le tableau suivant présente des tâches que vous voulez peut-être accomplir et la meilleure façon de substituer les méthodes de clavier. Pour plus d'informations sur la substitution de méthodes, consultez [substitution de propriétés et de méthodes dans les classes dérivées](#).

TÂCHE	MÉTHODE
Intercepter une touche de navigation et KeyDown déclencher un événement. Par exemple, vous souhaitez que la touche de tabulation et la touche de retour soient gérées dans une zone de texte.	Substituez IsInputKey Remarque : Vous pouvez également gérer PreviewKeyDown l'événement et l'ensemble IsInputKey du PreviewKeyDownEventArgs à <code>true</code> pour la ou les clés de votre choix.
Effectuer une entrée spéciale ou une gestion de navigation sur un contrôle. Par exemple, vous souhaitez utiliser les touches de direction dans votre contrôle de liste pour modifier l'élément sélectionné.	Substituer la méthode ProcessDialogKey
Intercepter une touche de navigation et KeyPress déclencher un événement. Par exemple, dans un contrôle de zone de sélection, vous voulez que des activations multiples de la touche de direction accélèrent la progression au sein des éléments.	Substituez IsInputChar
Effectuer une entrée spéciale ou une gestion de KeyPress navigation pendant un événement. Par exemple, dans une liste de contrôle, le fait de maintenir la touche « r » enfoncée permet d'ignorer les éléments qui commencent par la lettre r.	Substituer la méthode ProcessDialogChar

TÂCHE	MÉTHODE
Effectuer une gestion personnalisée des caractères mnémoniques ; par exemple, vous souhaitez gérer les caractères mnémoniques sur des boutons owner-drawn contenus dans une barre d'outils.	Substituez ProcessMnemonic

Voir aussi

- [Keys](#)
- [WndProc](#)
- [PreProcessMessage](#)
- [My.Computer.Keyboard](#) (objet)
- [Accès au clavier](#)
- [Utilisation des événements de clavier](#)

Utilisation des événements du clavier

4 minutes to read • [Edit Online](#)

La plupart des programmes Windows Forms traitent l'entrée au clavier en gérant les événements de clavier. Cette rubrique fournit une vue d'ensemble des événements de clavier, explique quand utiliser chaque événement et détaille les données fournies pour chaque événement. Consultez également [vue d'ensemble des gestionnaires d'événements \(Windows Forms\)](#) et [vue d'ensemble des événements \(Windows Forms\)](#).

Événements de clavier

Windows Forms fournit deux événements qui se produisent quand l'utilisateur appuie sur une touche du clavier et un événement qui se produit quand l'utilisateur relâche une touche du clavier :

- L'événement [KeyDown](#) se produit une fois.
- L'événement [KeyPress](#), qui peut se produire plusieurs fois quand l'utilisateur maintient la même touche enfoncée.
- L'événement [KeyUp](#) se produit une fois quand l'utilisateur relâche une touche.

Quand l'utilisateur appuie sur une touche, Windows Forms détermine quel événement déclencher selon que le message de clavier spécifie une touche de caractère ou une touche physique. Pour plus d'informations sur les caractères et les touches physiques, consultez [fonctionnement des entrées au clavier](#).

Le tableau suivant décrit les trois événements de clavier.

ÉVÉNEMENTS DE CLAVIER	DESCRIPTION	RÉSULTATS
KeyDown	Cet événement est déclenché quand l'utilisateur appuie sur une touche physique.	<p>Le gestionnaire de KeyDown reçoit :</p> <ul style="list-style-type: none">• un paramètre KeyEventArgs, qui fournit la propriété KeyCode (qui spécifie un bouton de clavier physique) ;• la propriété Modifiers (Maj, Ctrl ou Alt) ;• la propriété KeyData (qui combine le code de touche et le modificateur). Le paramètre KeyEventArgs fournit également :<ul style="list-style-type: none">◦ la propriété Handled, qui peut être définie pour empêcher le contrôle sous-jacent de recevoir la touche ;◦ la propriété SuppressKeyPress, qui peut être utilisée pour supprimer les événements KeyPress et KeyUp pour cette séquence de touches.

ÉVÉNEMENTS DE CLAVIER	DESCRIPTION	RÉSULTATS
KeyPress	Cet événement est déclenché quand la ou les touches enfoncées génèrent un caractère. Par exemple, l'utilisateur appuie sur les touches Maj et « a », ce qui produit un caractère « A » majuscule.	<p>KeyPress est déclenché après KeyDown.</p> <ul style="list-style-type: none"> Le gestionnaire de KeyPress reçoit : un paramètre KeyPressEventArgs, qui contient le code de caractère de la touche qui a été enfoncée. Ce code de caractère est unique à chaque combinaison touche de caractère/touche de modification. <p>Par exemple, la touche « A » génère :</p> <ul style="list-style-type: none"> le code de caractère 65, si elle est enfoncée avec la touche Maj ; ou la touche Verr. Maj, 97, si elle est activée seule, Et 1, si elle est enfoncée avec la touche Ctrl.
KeyUp	Cet événement est déclenché quand l'utilisateur relâche une touche physique.	<p>Le gestionnaire de KeyUp reçoit :</p> <ul style="list-style-type: none"> un paramètre KeyEventArgs : <ul style="list-style-type: none"> qui fournit la propriété KeyCode (qui spécifie un bouton de clavier physique) ; la propriété Modifiers (Maj, Ctrl ou Alt) ; la propriété KeyData (qui combine le code de touche et le modificateur).

Voir aussi

- [Entrée au clavier dans une application Windows Forms](#)
- [Fonctionnement de l'entrée au clavier](#)
- [Entrée de la souris dans une application Windows Forms](#)

Procédure : changer l'entrée de clavier en contrôle standard

9 minutes to read • [Edit Online](#)

Windows Forms offre la possibilité de consommer et de modifier l'entrée au clavier. Consommer une touche signifie gérer une touche dans une méthode ou un gestionnaire d'événements pour que d'autres méthodes et événements plus loin dans la file d'attente de messages ne reçoivent pas la valeur de la touche. Modifier une touche signifie modifier sa valeur pour que les méthodes et les gestionnaires d'événements plus loin dans la file d'attente de messages reçoivent une valeur de touche différente. Cette rubrique montre comment accomplir ces tâches.

Pour consommer une touche

- Dans un gestionnaire d'événements [KeyPress](#), affectez la valeur `true` à la propriété [Handled](#) de la classe [KeyPressEventArgs](#).

ou

Dans un gestionnaire d'événements [KeyDown](#), affectez la valeur `true` à la propriété [Handled](#) de la classe [KeyEventArgs](#).

NOTE

La définition de la propriété [Handled](#) dans le gestionnaire d'événements [KeyDown](#) n'empêche pas les événements [KeyPress](#) et [KeyUp](#) d'être déclenchés pour la séquence de touches actuelle. Vous devez pour cela utiliser la propriété [SuppressKeyPress](#).

L'exemple suivant est un extrait d'instruction `switch` qui examine la propriété [KeyChar](#) du [KeyPressEventArgs](#) reçu par un gestionnaire d'événements [KeyPress](#). Ce code consomme les touches de caractère « A » et « a ».

```
// Consume 'A' and 'a'.
case (char)65:
case (char)97:
    MessageBox.Show("Control.KeyPress: '" +
        e.KeyChar.ToString() + "' consumed.");
    e.Handled = true;
    break;
```

```
' Consume 'A' and 'a'.
Case ChrW(65), ChrW(97)
    MessageBox.Show(("Control.KeyPress: '" + _
        e.KeyChar.ToString() + "' consumed."))
    e.Handled = True
```

Pour modifier une touche de caractère standard

- Dans un gestionnaire d'événements [KeyPress](#), affectez à la propriété [KeyChar](#) de la classe [KeyPressEventArgs](#) la valeur de la nouvelle touche de caractère.

L'exemple suivant est un extrait d'instruction `switch` qui remplace « B » par « A » et « b » par « a ». Notez que la propriété [Handled](#) du paramètre [KeyPressEventArgs](#) a la valeur `false`, pour que la nouvelle valeur

de touche soit propagée aux autres méthodes et événements dans la file d'attente de messages.

```
// Detect 'B', modify it to 'A', and forward the key.
case (char)66:
    MessageBox.Show("Control.KeyPress: '" +
        e.KeyChar.ToString() + "' replaced by 'A'.");
    e.KeyChar = (char)65;
    e.Handled = false;
    break;

// Detect 'b', modify it to 'a', and forward the key.
case (char)98:
    MessageBox.Show("Control.KeyPress: '" +
        e.KeyChar.ToString() + "' replaced by 'a'.");
    e.KeyChar = (char)97;
    e.Handled = false;
    break;
```

```
' Modify 'B' to 'A' and forward the key.
Case ChrW(66)
    MessageBox.Show(("Control.KeyPress: '" + _
        e.KeyChar.ToString() + "' replaced by 'A'."))
    e.KeyChar = ChrW(65)
    e.Handled = False

' Modify 'b' to 'a' and forward the key.
Case ChrW(98)
    MessageBox.Show(("Control.KeyPress: '" + _
        e.KeyChar.ToString() + "' replaced by 'a'."))
    e.KeyChar = ChrW(97)
    e.Handled = False
```

Pour modifier une touche non-caractère

- Substituez une méthode [Control](#) qui traite les messages Windows, détectez le message WM_KEYDOWN ou WM_SYSKEYDOWN et affectez à la propriété [WParam](#) du paramètre [Message](#) la valeur [Keys](#) qui représente la nouvelle touche non-caractère.

L'exemple de code suivant montre comment substituer la méthode [PreProcessMessage](#) d'un contrôle pour détecter les touches F1 à F9 et modifier toute activation de la touche F3 en F1. Pour plus d'informations [Control](#) sur les méthodes que vous pouvez substituer pour intercepter des messages de clavier, consultez [entrée d'utilisateur dans une application Windows Forms](#) et fonctionnement de [l'entrée au clavier](#).

```

// Detect F1 through F9 during preprocessing and modify F3.
public override bool PreProcessMessage(ref Message m)
{
    if (m.Msg == WM_KEYDOWN)
    {
        Keys keyCode = (Keys)m.WParam & Keys.KeyCode;

        // Detect F1 through F9.
        switch (keyCode)
        {
            case Keys.F1:
            case Keys.F2:
            case Keys.F3:
            case Keys.F4:
            case Keys.F5:
            case Keys.F6:
            case Keys.F7:
            case Keys.F8:
            case Keys.F9:

                MessageBox.Show("Control.PreProcessMessage: '" +
                    keyCode.ToString() + "' pressed.");

                // Replace F3 with F1, so that ProcessKeyMessage will
                // receive F1 instead of F3.
                if (keyCode == Keys.F3)
                {
                    m.WParam = (IntPtr)Keys.F1;
                    MessageBox.Show("Control.PreProcessMessage: '" +
                        keyCode.ToString() + "' replaced by F1.");
                }
                break;
        }
    }

    // Send all other messages to the base method.
    return base.PreProcessMessage(ref m);
}

```

```

' Detect F1 through F9 during preprocessing and modify F3.
Public Overrides Function PreProcessMessage(ByRef m As Message) _
    As Boolean

    If m.Msg = WM_KEYDOWN Then
        Dim keyCode As Keys = CType(m.WParam, Keys) And Keys.KeyCode

        ' Detect F1 through F9.
        Select Case keyCode
            Case Keys.F1, Keys.F2, Keys.F3, Keys.F4, Keys.F5, _
                Keys.F6, Keys.F7, Keys.F8, Keys.F9

                MessageBox.Show(("Control.PreProcessMessage: '" + _
                    keyCode.ToString() + "' pressed."))

                ' Replace F3 with F1, so that ProcessKeyMessage will
                ' receive F1 instead of F3.
                If keyCode = Keys.F3 Then
                    m.WParam = CType(Keys.F1, IntPtr)
                    MessageBox.Show(("Control.PreProcessMessage: '" + _
                        keyCode.ToString() + "' replaced by F1."))
                End If
            End Select
        End If

        ' Send all other messages to the base method.
        Return MyBase.PreProcessMessage(m)
    End Function

```

Exemple

L'exemple de code suivant est l'application complète pour les exemples de code des sections précédentes. L'application utilise un contrôle personnalisé dérivé de la classe [TextBox](#) pour consommer et modifier l'entrée au clavier.

```

using System;
using System.Drawing;
using System.Windows.Forms;

namespace KeyboardInput
{
    [System.Security.Permissions.PermissionSet(System.Security.Permissions.SecurityAction.Demand,
        Name="FullTrust")]
    class Form1 : Form
    {
        // The following Windows message value is defined in Winuser.h.
        private int WM_KEYDOWN = 0x100;
        CustomTextBox CustomTextBox1 = new CustomTextBox();

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

        public Form1()
        {
            this.AutoSize = true;
            this.Controls.Add(CustomTextBox1);
            CustomTextBox1.KeyPress +=
                new KeyPressEventHandler(CustomTextBox1_KeyPress);
        }
    }
}

```

```

// Consume and modify several character keys.
void CustomTextBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    switch (e.KeyChar)
    {
        // Consume 'A' and 'a'.
        case (char)65:
        case (char)97:
            MessageBox.Show("Control.KeyPress: '" +
                e.KeyChar.ToString() + "' consumed.");
            e.Handled = true;
            break;

        // Detect 'B', modify it to 'A', and forward the key.
        case (char)66:
            MessageBox.Show("Control.KeyPress: '" +
                e.KeyChar.ToString() + "' replaced by 'A'.");
            e.KeyChar = (char)65;
            e.Handled = false;
            break;

        // Detect 'b', modify it to 'a', and forward the key.
        case (char)98:
            MessageBox.Show("Control.KeyPress: '" +
                e.KeyChar.ToString() + "' replaced by 'a'.");
            e.KeyChar = (char)97;
            e.Handled = false;
            break;
    }
}

[System.Security.Permissions.PermissionSet(System.Security.Permissions.SecurityAction.Demand,
Name="FullTrust")]
public class CustomTextBox : TextBox
{
    // The following Windows message value is defined in Winuser.h.
    private int WM_KEYDOWN = 0x100;

    public CustomTextBox()
    {
        this.Size = new Size(100, 100);
        this.AutoSize = false;
    }

    // Detect F1 through F9 during preprocessing and modify F3.
    public override bool PreProcessMessage(ref Message m)
    {
        if (m.Msg == WM_KEYDOWN)
        {
            Keys keyCode = (Keys)m.WParam & Keys.KeyCode;

            // Detect F1 through F9.
            switch (keyCode)
            {
                case Keys.F1:
                case Keys.F2:
                case Keys.F3:
                case Keys.F4:
                case Keys.F5:
                case Keys.F6:
                case Keys.F7:
                case Keys.F8:
                case Keys.F9:

                    MessageBox.Show("Control.PreProcessMessage: '" +
                        keyCode.ToString() + "' pressed.");

                    // Replace F3 with F1, so that ProcessKeyMessage will
                    // receive F1 instead of F3.

```

```

        if (keyCode == Keys.F3)
        {
            m.WParam = (IntPtr)Keys.F1;
            MessageBox.Show("Control.PreProcessMessage: '" +
                keyCode.ToString() + "' replaced by F1.");
        }
        break;
    }
}

// Send all other messages to the base method.
return base.PreProcessMessage(ref m);
}

// Detect F1 through F9 during processing.
protected override bool ProcessKeyMessage(ref Message m)
{
    if (m.Msg == WM_KEYDOWN)
    {
        Keys keyCode = (Keys)m.WParam & Keys.KeyCode;

        // Detect F1 through F9.
        switch (keyCode)
        {
            case Keys.F1:
            case Keys.F2:
            case Keys.F3:
            case Keys.F4:
            case Keys.F5:
            case Keys.F6:
            case Keys.F7:
            case Keys.F8:
            case Keys.F9:

                MessageBox.Show("Control.ProcessKeyMessage: '" +
                    keyCode.ToString() + "' pressed.");
                break;
        }
    }

    // Send all messages to the base method.
    return base.ProcessKeyMessage(ref m);
}
}

```

```

Imports System.Drawing
Imports System.Security
Imports System.Security.Permissions
Imports System.Windows.Forms

Namespace KeyboardInput
<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.Demand,
Name:="FullTrust")> _
    Class Form1
        Inherits Form

        ' The following Windows message value is defined in Winuser.h.
        Private WM_KEYDOWN As Integer = &H100
        Private WithEvents CustomTextBox1 As New CustomTextBox()

        <STAThread()> _
        Public Shared Sub Main()
            Application.EnableVisualStyles()
            Application.Run(New Form1())
        End Sub
    End Class

```

```

Public Sub New()
    Me.AutoSize = True
    Me.Controls.Add(CustomTextBox1)
End Sub

' Consume and modify several character keys.
Sub CustomTextBox1_KeyPress(ByVal sender As Object, _
    ByVal e As KeyPressEventArgs) Handles CustomTextBox1.KeyPress

    Select Case e.KeyChar

        ' Consume 'A' and 'a'.
        Case ChrW(65), ChrW(97)
            MessageBox.Show(("Control.KeyPress: '" + _
                e.KeyChar.ToString() + "' consumed."))
            e.Handled = True

        ' Modify 'B' to 'A' and forward the key.
        Case ChrW(66)
            MessageBox.Show(("Control.KeyPress: '" + _
                e.KeyChar.ToString() + "' replaced by 'A'."))
            e.KeyChar = ChrW(65)
            e.Handled = False

        ' Modify 'b' to 'a' and forward the key.
        Case ChrW(98)
            MessageBox.Show(("Control.KeyPress: '" + _
                e.KeyChar.ToString() + "' replaced by 'a'."))
            e.KeyChar = ChrW(97)
            e.Handled = False

    End Select
End Sub
End Class

```

```

<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.Demand,
Name:="FullTrust")> _

```

```

Public Class CustomTextBox
    Inherits TextBox

    ' The following Windows message value is defined in Winuser.h.
    Private WM_KEYDOWN As Integer = &H100

    Public Sub New()
        Me.Size = New Size(100, 100)
        Me.AutoSize = False
    End Sub

    ' Detect F1 through F9 during preprocessing and modify F3.
    Public Overrides Function PreProcessMessage(ByRef m As Message) _
        As Boolean

        If m.Msg = WM_KEYDOWN Then
            Dim keyCode As Keys = CType(m.WParam, Keys) And Keys.KeyCode

            ' Detect F1 through F9.
            Select Case keyCode
                Case Keys.F1, Keys.F2, Keys.F3, Keys.F4, Keys.F5, _
                    Keys.F6, Keys.F7, Keys.F8, Keys.F9

                    MessageBox.Show(("Control.PreProcessMessage: '" + _
                        keyCode.ToString() + "' pressed."))

                    ' Replace F3 with F1, so that ProcessKeyMessage will
                    ' receive F1 instead of F3.
                    If keyCode = Keys.F3 Then
                        m.WParam = CType(Keys.F1, IntPtr)
                        MessageBox.Show(("Control.PreProcessMessage: '" + _
                            keyCode.ToString() + "' replaced by F1."))
                    End If
                End Select
            End If
        End If
    End Function
End Class

```

```

        End If
    End Select
End If

' Send all other messages to the base method.
Return MyBase.PreProcessMessage(m)
End Function

' Detect F1 through F9 during processing.
Protected Overrides Function ProcessKeyMessage(ByRef m As Message) _
    As Boolean

    If m.Msg = WM_KEYDOWN Then
        Dim keyCode As Keys = CType(m.WParam, Keys) And Keys.KeyCode

        ' Detect F1 through F9.
        Select Case keyCode
            Case Keys.F1, Keys.F2, Keys.F3, Keys.F4, Keys.F5, _
                Keys.F6, Keys.F7, Keys.F8, Keys.F9

                MessageBox.Show(("Control.ProcessKeyMessage: '" + _
                    keyCode.ToString() + "' pressed."))
        End Select
    End If

    ' Send all messages to the base method.
    Return MyBase.ProcessKeyMessage(m)
End Function

End Class
End Namespace

```

Compilation du code

Cet exemple nécessite :

- des références aux assemblés System, System.Drawing et System.Windows.Forms.

Voir aussi

- [Entrée au clavier dans une application Windows Forms](#)
- [Entrée d'utilisateur dans une application Windows Forms](#)
- [Fonctionnement de l'entrée au clavier](#)

Procédure : déterminer la touche de modification activée

3 minutes to read • [Edit Online](#)

Lorsque vous créez une application qui accepte les séquences de touches de l'utilisateur, vous pouvez également surveiller les touches de modification telles que les touches Maj, ALT et CTRL. Quand une touche de modification est enfoncée en combinaison avec d'autres touches, ou avec des clics de souris, votre application peut réagir de manière appropriée. Par exemple, si la lettre S est enfoncée, cela peut simplement entraîner l'affichage d'un «s» à l'écran, mais si vous appuyez sur les touches CTRL + S, le document actif peut être enregistré. Si vous gérez l' [KeyDown](#) événement, la [Modifiers](#) propriété du [KeyEventArgs](#) reçu par le gestionnaire d'événements spécifie quelles touches de modification sont enfoncées. La [KeyData](#) propriété de spécifie également [KeyEventArgs](#) le caractère qui a été activé, ainsi que les touches de modification associées à une opération or au niveau du bit. Toutefois, si vous gérez l' [KeyPress](#) événement ou un événement de souris, le gestionnaire d'événements ne reçoit pas ces informations. Dans ce cas, vous devez utiliser la [ModifierKeys](#) propriété de la [Control](#) classe. Dans les deux cas, vous devez effectuer une opération de bits and [Keys](#) de la valeur appropriée et de la valeur que vous testez. L' [Keys](#) énumération offre des variations de chaque touche de modification. il est donc important que vous exécutiez l'opération de bits and avec la valeur correcte. Par exemple, la touche Maj est représentée par [Shift RShiftKey](#) , [ShiftKey](#) et [LShiftKey](#) la valeur correcte pour tester Shift comme touche de modification est [Shift](#). De même, pour tester les touches Ctrl et Alt en tant que modificateurs, [Control](#) vous [Alt](#) devez utiliser respectivement les valeurs et.

NOTE

Visual Basic programmeurs peuvent également accéder aux informations de clé [Keyboard](#) par le biais de la propriété

Pour déterminer la touche de modification qui a été enfoncée

- Utilisez l'opérateur `AND` au niveau du [ModifierKeys](#) bit avec la propriété et une [Keys](#) valeur de l'énumération pour déterminer si une touche de modification particulière est enfoncée. L'exemple de code suivant montre comment déterminer si la touche Maj est enfoncée [KeyPress](#) dans un gestionnaire d'événements.

```
private:
    void textBox1_KeyPress(Object^ sender, KeyPressEventArgs e)
    {
        if ((Control::ModifierKeys & Keys::Shift) == Keys::Shift)
        {
            MessageBox::Show("Pressed " + Keys::Shift.ToString());
        }
    }
}
```

```
public void TextBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if ((Control.ModifierKeys & Keys.Shift) == Keys.Shift)
    {
        MessageBox.Show("Pressed " + Keys.Shift);
    }
}
```



```
Public Sub TextBox1_KeyPress(ByVal sender As Object, _  
    ByVal e As KeyPressEventArgs) Handles TextBox1.KeyPress  
  
    If ((Control.ModifierKeys And Keys.Shift) = Keys.Shift) Then  
        MsgBox("Pressed " + Keys.Shift.ToString())  
    End If  
End Sub
```

Voir aussi

- [Keys](#)
- [ModifierKeys](#)
- [Entrée au clavier dans une application Windows Forms](#)
- [Guide pratique pour Déterminer si CapsLock est activé dans Visual Basic](#)

Procédure : gérer l'entrée de clavier au niveau du formulaire

7 minutes to read • [Edit Online](#)

Windows Forms offre la possibilité de gérer les messages de clavier au niveau du formulaire, avant que les messages n'atteignent un contrôle. Cette rubrique montre comment procéder.

Pour gérer un message de clavier au niveau du formulaire

- Gérez l'événement `KeyPress` ou `KeyDown` du formulaire de démarrage et affectez la valeur `true` à la propriété `KeyPreview` du formulaire pour que les messages de clavier soient reçus par le formulaire avant qu'ils atteignent les contrôles sur le formulaire. L'exemple de code suivant gère l'événement `KeyPress` en détectant toutes les touches numériques et en consommant « 1 », « 4 » et « 7 ».

```
// Detect all numeric characters at the form level and consume 1,
// 4, and 7. Note that Form.KeyPreview must be set to true for this
// event handler to be called.
private:
void Form1_KeyPress(Object^ sender, KeyPressEventArgs^ e)
{
    if ((e->KeyChar >= '0') && (e->KeyChar <= '9'))
    {
        MessageBox::Show("Form.KeyPress: '" +
            e->KeyChar.ToString() + "' pressed.");

        switch (e->KeyChar)
        {
            case '1':
            case '4':
            case '7':
                MessageBox::Show("Form.KeyPress: '" +
                    e->KeyChar.ToString() + "' consumed.");
                e->Handled = true;
                break;
        }
    }
}
```

```
// Detect all numeric characters at the form level and consume 1,
// 4, and 7. Note that Form.KeyPreview must be set to true for this
// event handler to be called.
void Form1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar >= 48 && e.KeyChar <= 57)
    {
        MessageBox.Show("Form.KeyPress: '" +
            e.KeyChar.ToString() + "' pressed.");

        switch (e.KeyChar)
        {
            case (char)49:
            case (char)52:
            case (char)55:
                MessageBox.Show("Form.KeyPress: '" +
                    e.KeyChar.ToString() + "' consumed.");
                e.Handled = true;
                break;
        }
    }
}
```

```
' Detect all numeric characters at the form level and consume 1,
' 4, and 7. Note that Form.KeyPreview must be set to true for this
' event handler to be called.
Sub Form1_KeyPress(ByVal sender As Object, _
    ByVal e As KeyPressEventArgs) Handles Me.KeyPress

    If e.KeyChar >= ChrW(48) And e.KeyChar <= ChrW(57) Then
        MessageBox.Show(("Form.KeyPress: '" + _
            e.KeyChar.ToString() + "' pressed.))

        Select Case e.KeyChar
            Case ChrW(49), ChrW(52), ChrW(55)
                MessageBox.Show(("Form.KeyPress: '" + _
                    e.KeyChar.ToString() + "' consumed.))
                e.Handled = True
        End Select
    End If
End Sub
```

Exemple

L'exemple de code suivant est l'application complète pour l'exemple ci-dessus. L'application comprend un [TextBox](#) avec plusieurs autres contrôles qui vous permettent de déplacer le focus hors du [TextBox](#). L'événement [KeyPress](#) du [Form](#) principal consomme « 1 », « 4 » et « 7 » et l'événement [KeyPress](#) du [TextBox](#) consomme « 2 », « 5 » et « 8 » tout en affichant les touches restantes. Comparez la sortie de [MessageBox](#) quand vous appuyez sur une touche numérique pendant que le [TextBox](#) a le focus avec la sortie de [MessageBox](#) quand vous appuyez sur une touche numérique pendant que le focus est sur l'un des autres contrôles.

```
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>
#using <System.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::Windows::Forms;
using namespace System::Security::Permissions;

namespace KeyboardInputForm
{
```

```

public ref class Form1 sealed: public Form, public IMessageFilter
{
    // The following Windows message value is defined in Winuser.h.
private:
    static const int WM_KEYDOWN = 0x100;
private:
    TextBox^ inputTextBox;
public:
    Form1()
    {
        inputTextBox = gcnew TextBox();
        this->AutoSize = true;
        Application::AddMessageFilter(this);
        FlowLayoutPanel^ panel = gcnew FlowLayoutPanel();
        panel->AutoSize = true;
        panel->FlowDirection = FlowDirection::TopDown;
        panel->Controls->Add(gcnew Button());
        panel->Controls->Add(gcnew RadioButton());
        panel->Controls->Add(inputTextBox);
        this->Controls->Add(panel);
        this->KeyPreview = true;
        this->KeyPress +=
            gcnew KeyPressEventHandler(this, &Form1::Form1_KeyPress);
        inputTextBox->KeyPress +=
            gcnew KeyPressEventHandler(this,
                &Form1::inputTextBox_KeyPress);
    }

    // Detect all numeric characters at the
    // application level and consume 0.
    [SecurityPermission(SecurityAction::LinkDemand,
        Flags=SecurityPermissionFlag::UnmanagedCode)]
    virtual bool PreFilterMessage(Message% m)
    {
        // Detect key down messages.
        if (m.Msg == WM_KEYDOWN)
        {
            Keys keyCode = (Keys)((int)m.WParam) & Keys::KeyCode;
            // Determine whether the keystroke is a number from the top of
            // the keyboard, or a number from the keypad.
            if (((keyCode >= Keys::D0) && (keyCode <= Keys::D9))
                || ((keyCode >= Keys::NumPad0)
                    && (keyCode <= Keys::NumPad9)))
            {
                MessageBox::Show(
                    "IMessageFilter.PreFilterMessage: '" +
                    keyCode.ToString() + "' pressed.");

                if ((keyCode == Keys::D0) || (keyCode == Keys::NumPad0))
                {
                    MessageBox::Show(
                        "IMessageFilter.PreFilterMessage: '" +
                        keyCode.ToString() + "' consumed.");
                    return true;
                }
            }
        }

        // Forward all other messages.
        return false;
    }

    // Detect all numeric characters at the form level and consume 1,
    // 4, and 7. Note that Form.KeyPreview must be set to true for this
    // event handler to be called.
private:
    void Form1_KeyPress(Object^ sender, KeyPressEventArgs^ e)
    {
        if ((e->KeyCode >= Keys::D0) && (e->KeyCode <= Keys::D9))
    }

```

```

        if ((e->KeyChar >= '0') && (e->KeyChar <= '9'))
        {
            MessageBox::Show("Form.KeyPress: '" +
                e->KeyChar.ToString() + "' pressed.");

            switch (e->KeyChar)
            {
                case '1':
                case '4':
                case '7':
                    MessageBox::Show("Form.KeyPress: '" +
                        e->KeyChar.ToString() + "' consumed.");
                    e->Handled = true;
                    break;
            }
        }
    }

    // Detect all numeric characters at the TextBox level and consume
    // 2, 5, and 8.
private:
    void inputTextBox_KeyPress(Object^ sender, KeyPressEventArgs^ e)
    {
        if ((e->KeyChar >= '0') && (e->KeyChar <= '9'))
        {
            MessageBox::Show("Control.KeyPress: '" +
                e->KeyChar.ToString() + "' pressed.");

            switch (e->KeyChar)
            {
                case '2':
                case '5':
                case '8':
                    MessageBox::Show("Control.KeyPress: '" +
                        e->KeyChar.ToString() + "' consumed.");
                    e->Handled = true;
                    break;
            }
        }
    }
};

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run(gcnew KeyboardInputForm::Form1());
}

```

```

using System;
using System.Drawing;
using System.Windows.Forms;

namespace KeyboardInputForm
{
    class Form1 : Form
    {
        TextBox TextBox1 = new TextBox();

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

        public Form1()

```

```

public Form1()
{
    this.AutoSize = true;

    FlowLayoutPanel panel = new FlowLayoutPanel();
    panel.AutoSize = true;
    panel.FlowDirection = FlowDirection.TopDown;
    panel.Controls.Add(TextBox1);
    this.Controls.Add(panel);

    this.KeyPreview = true;
    this.KeyPress +=
        new KeyPressEventHandler(Form1_KeyPress);
    TextBox1.KeyPress +=
        new KeyPressEventHandler(TextBox1_KeyPress);
}

// Detect all numeric characters at the form level and consume 1,
// 4, and 7. Note that Form.KeyPreview must be set to true for this
// event handler to be called.
void Form1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar >= 48 && e.KeyChar <= 57)
    {
        MessageBox.Show("Form.KeyPress: '" +
            e.KeyChar.ToString() + "' pressed.");

        switch (e.KeyChar)
        {
            case (char)49:
            case (char)52:
            case (char)55:
                MessageBox.Show("Form.KeyPress: '" +
                    e.KeyChar.ToString() + "' consumed.");
                e.Handled = true;
                break;
        }
    }
}

// Detect all numeric characters at the TextBox level and consume
// 2, 5, and 8.
void TextBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar >= 48 && e.KeyChar <= 57)
    {
        MessageBox.Show("Control.KeyPress: '" +
            e.KeyChar.ToString() + "' pressed.");

        switch (e.KeyChar)
        {
            case (char)50:
            case (char)53:
            case (char)56:
                MessageBox.Show("Control.KeyPress: '" +
                    e.KeyChar.ToString() + "' consumed.");
                e.Handled = true;
                break;
        }
    }
}
}
}

```

```

Imports System.Drawing
Imports System.Windows.Forms

Namespace KeyboardInputForm

```

Namespace Registry namespace

```
Class Form1
Inherits Form

    Private WithEvents TextBox1 As New TextBox()

    <STAThread> _
    Public Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())
    End Sub

    Public Sub New()
        Me.AutoSize = True

        Dim panel As New FlowLayoutPanel()
        panel.AutoSize = True
        panel.FlowDirection = FlowDirection.TopDown
        panel.Controls.Add(TextBox1)
        Me.Controls.Add(panel)

        Me.KeyPreview = True
    End Sub

    ' Detect all numeric characters at the form level and consume 1,
    ' 4, and 7. Note that Form.KeyPreview must be set to true for this
    ' event handler to be called.
    Sub Form1_KeyPress(ByVal sender As Object, _
        ByVal e As KeyPressEventArgs) Handles Me.KeyPress

        If e.KeyChar >= ChrW(48) And e.KeyChar <= ChrW(57) Then
            MessageBox.Show(("Form.KeyPress: '" + _
                e.KeyChar.ToString() + "' pressed."))

            Select Case e.KeyChar
                Case ChrW(49), ChrW(52), ChrW(55)
                    MessageBox.Show(("Form.KeyPress: '" + _
                        e.KeyChar.ToString() + "' consumed."))
                    e.Handled = True
            End Select
        End If
    End Sub

    ' Detect all numeric characters at the TextBox level and consume
    ' 2, 5, and 8.
    Sub TextBox1_KeyPress(ByVal sender As Object, _
        ByVal e As KeyPressEventArgs) Handles TextBox1.KeyPress

        If e.KeyChar >= ChrW(48) And e.KeyChar <= ChrW(57) Then
            MessageBox.Show(("Control.KeyPress: '" + _
                e.KeyChar.ToString() + "' pressed."))

            Select Case e.KeyChar
                Case ChrW(50), ChrW(53), ChrW(56)
                    MessageBox.Show(("Control.KeyPress: '" + _
                        e.KeyChar.ToString() + "' consumed."))
                    e.Handled = True
            End Select
        End If
    End Sub

End Class
End Namespace
```

Compilation du code

Cet exemple nécessite :

- des références aux assemblés System, System.Drawing et System.Windows.Forms.

Voir aussi

- [Entrée au clavier dans une application Windows Forms](#)

Entrée de la souris dans une application Windows Forms

2 minutes to read • [Edit Online](#)

Windows Forms inclut divers événements de souris et une prise en charge supplémentaire pour les curseurs de souris, la capture de souris et le comportement de glisser-déplacer personnalisés.

Dans cette section

[Fonctionnement des entrées de la souris dans les Windows Forms](#)

Fournit des informations sur les événements de souris et sur l'obtention des informations et des paramètres système actuels pour la souris.

[Événements liés à la souris dans les Windows Forms](#)

Fournit des informations sur l'ordre dans lequel les événements de souris se produisent et sur la manière dont les événements de souris sont déclenchés dans des contrôles spécifiques.

[Comment : distinguer les clics des double-clics](#)

Illustre l'utilisation des clics et des double-clics pour initier des actions incompatibles.

[Pointeurs de souris dans les Windows Forms](#)

Décrit comment modifier le curseur de souris.

[Capture de la souris dans les Windows Forms](#)

Décrit comment un contrôle peut capturer la souris.

[Fonctionnalité de glisser-déposer dans les Windows Forms](#)

Décrit comment implémenter le comportement de glisser-déplacer.

Sections connexes

[Accès à la souris](#)

Répertorie les rubriques permettant d'accéder à la souris à l'aide de Visual Basic.

Fonctionnement des entrées de la souris dans les Windows Forms

9 minutes to read • [Edit Online](#)

La réception et le traitement de l'entrée de souris constitue une partie importante de chaque application Windows. Vous pouvez gérer les événements de souris pour exécuter une action dans votre application, ou utiliser les informations d'emplacement de la souris pour effectuer un test de positionnement ou d'autres actions. En outre, vous pouvez modifier la façon dont les contrôles de votre application gèrent l'entrée de la souris. Cette rubrique décrit ces événements de souris en détail et explique comment obtenir et modifier les paramètres système de la souris. Pour plus d'informations sur les données fournies avec les événements de souris et l'ordre dans lequel les événements de clic de souris sont déclenchés, consultez [événements de souris dans Windows Forms](#).

Emplacement de la souris et test de positionnement

Lorsque l'utilisateur déplace la souris, le système d'exploitation déplace le pointeur de la souris. Le pointeur de la souris contient un pixel unique, appelé zone réactive, que le système d'exploitation suit et reconnaît comme position du pointeur. Quand l'utilisateur déplace la souris ou appuie sur un bouton de la souris, le [Control](#) qui contient le [HotSpot](#) déclenche l'événement de souris approprié. Vous pouvez obtenir la position actuelle de la souris avec la propriété [Location](#) du [MouseEventArgs](#) lors de la gestion d'un événement de souris ou à l'aide de la propriété [Position](#) de la classe [Cursor](#). Vous pouvez ensuite utiliser les informations d'emplacement de la souris pour effectuer des tests de positionnement, puis effectuer une action en fonction de l'emplacement de la souris. La fonctionnalité de test d'atteinte est intégrée à plusieurs contrôles dans Windows Forms tels que les contrôles [ListView](#), [TreeView](#), [MonthCalendar](#) et [DataGridView](#). Utilisé avec l'événement de souris approprié, [MouseHover](#) par exemple, le test d'atteinte est très utile pour déterminer quand votre application doit effectuer une action spécifique.

Événements de souris

Le principal moyen de répondre à l'entrée de la souris consiste à gérer les événements de souris. Le tableau suivant montre les événements de souris et décrit quand ils sont déclenchés.

ÉVÉNEMENT DE SOURIS	DESCRIPTION
Click	Cet événement se produit lorsque le bouton de la souris est relâché, en général avant l'événement MouseUp . Le gestionnaire pour cet événement reçoit un argument de type EventArgs . Gérez cet événement lorsque vous devez uniquement déterminer quand un clic se produit.
MouseClicked	Cet événement se produit lorsque l'utilisateur clique sur le contrôle avec la souris. Le gestionnaire pour cet événement reçoit un argument de type MouseEventArgs . Gérez cet événement lorsque vous avez besoin d'obtenir des informations sur la souris lorsqu'un clic se produit.
DoubleClick	Cet événement se produit lors d'un double-clic sur le contrôle. Le gestionnaire pour cet événement reçoit un argument de type EventArgs . Gérez cet événement lorsque vous devez uniquement déterminer quand un double-clic se produit.

ÉVÉNEMENT DE SOURIS	DESCRIPTION
MouseDown	Cet événement se produit lorsque le pointeur de la souris se trouve sur le contrôle et que l'utilisateur appuie sur un bouton de la souris. Le gestionnaire pour cet événement reçoit un argument de type MouseEventArgs .
MouseEnter	Cet événement se produit lorsque le pointeur de la souris entre dans la bordure ou la zone cliente du contrôle, en fonction du type de contrôle. Le gestionnaire pour cet événement reçoit un argument de type EventArgs .
MouseHover	Cet événement se produit lorsque le pointeur de la souris s'arrête et se positionne sur le contrôle. Le gestionnaire pour cet événement reçoit un argument de type EventArgs .
MouseLeave	Cet événement se produit lorsque le pointeur de la souris quitte la bordure ou la zone cliente du contrôle, en fonction du type du contrôle. Le gestionnaire pour cet événement reçoit un argument de type EventArgs .
MouseMove	Cet événement se produit lorsque le pointeur de la souris se déplace alors qu'il se trouve sur un contrôle. Le gestionnaire pour cet événement reçoit un argument de type MouseEventArgs .
MouseUp	Cet événement se produit lorsque le pointeur de la souris se trouve sur le contrôle et que l'utilisateur relâche un bouton de la souris. Le gestionnaire pour cet événement reçoit un argument de type MouseEventArgs .
MouseWheel	Cet événement se produit lorsque l'utilisateur fait tourner la roulette de la souris pendant que le contrôle a le focus. Le gestionnaire pour cet événement reçoit un argument de type MouseEventArgs . Vous pouvez utiliser la propriété Delta de MouseEventArgs pour déterminer la distance de défilement de la souris.

Modification de l'entrée de la souris et détection des paramètres système

Vous pouvez détecter et modifier la manière dont un contrôle gère l'entrée de la souris en dérivant du contrôle et à l'aide des méthodes [GetStyle](#) et [SetStyle](#). La méthode [SetStyle](#) prend une combinaison d'opérations de bits de valeurs [ControlStyles](#) pour déterminer si le contrôle aura un comportement de clic ou de double-clic standard ou si le contrôle gère son propre traitement de souris. En outre, la classe [SystemInformation](#) comprend des propriétés qui décrivent les fonctionnalités de la souris et spécifient comment la souris interagit avec le système d'exploitation. Le tableau suivant récapitule ces propriétés.

PROPRIÉTÉ	DESCRIPTION
DoubleClickSize	Obtient les dimensions, en pixels, de la zone dans laquelle l'utilisateur doit cliquer deux fois pour que le système d'exploitation considère deux clics comme un double-clic.
DoubleClickTime	Obtient le nombre maximal de millisecondes qui peuvent s'écouler entre un premier clic et un deuxième clic pour que le système d'exploitation considère l'action de la souris comme un double-clic.
MouseButtons	Obtient le nombre de boutons de la souris.
MouseButtonsSwapped	Obtient une valeur qui indique si les fonctions des boutons gauche et droit de la souris ont été permutées.
MouseHoverSize	Obtient les dimensions en pixels du rectangle dans lequel le pointeur de la souris doit rester pendant le délai de pointage de la souris avant qu'un message de pointage soit généré.
MouseHoverTime	Obtient le temps en millisecondes pendant lequel le pointeur doit rester dans le rectangle sélectionné automatiquement par pointage avec la souris avant qu'un message de pointage soit généré.
MousePresent	Obtient une valeur indiquant si une souris est installée.
MouseSpeed	Obtient une valeur indiquant la vitesse actuelle de la souris, comprise entre 1 et 20.
MouseWheelPresent	Obtient une valeur indiquant si une souris avec roulette est installée.
MouseWheelScrollDelta	Obtient le montant de la valeur delta de l'incrément d'une rotation de roulette de souris unique.
MouseWheelScrollLines	Obtient le nombre de lignes à faire défiler lors de la rotation de la roulette de la souris.

Voir aussi

- [Entrée de la souris dans une application Windows Forms](#)
- [Capture de la souris dans les Windows Forms](#)
- [Pointeurs de souris dans les Windows Forms](#)

Événements liés à la souris dans les Windows Forms

8 minutes to read • [Edit Online](#)

Quand vous gérez l'entrée de souris, vous souhaitez habituellement connaître l'emplacement du pointeur de la souris et l'état de ses boutons. Cette rubrique fournit des détails sur la façon d'obtenir des informations à partir des événements de souris et explique l'ordre dans lequel les événements de clic de souris sont déclenchés dans les contrôles Windows Forms. Pour obtenir la liste et la description de tous les événements de la souris, consultez [fonctionnement de l'entrée de la souris dans Windows Forms](#). Consultez également [vue d'ensemble des gestionnaires d'événements \(Windows Forms\)](#) et [vue d'ensemble des événements \(Windows Forms\)](#).

Informations sur la souris

Un [MouseEventArgs](#) est envoyé aux gestionnaires d'événements de souris liés aux clics de souris et au suivi des mouvements de souris. [MouseEventArgs](#) fournit des informations sur l'état actuel de la souris, y compris l'emplacement du pointeur de la souris sous forme de coordonnées clientes, les boutons de souris qui sont enfoncés et si la roulette a défilé. Plusieurs événements de souris, tels que ceux qui indiquent simplement si le pointeur de souris pénètre ou quitte les limites d'un contrôle, envoient un [EventArgs](#) au gestionnaire d'événements sans aucune information complémentaire.

Si vous souhaitez connaître l'état actuel des boutons de la souris ou l'emplacement du pointeur de la souris et que vous souhaitez éviter de gérer un événement de souris, vous pouvez aussi utiliser les propriétés [MouseButtons](#) et [MousePosition](#) de la classe [Control](#). [MouseButtons](#) retourne des informations sur les boutons de souris qui sont actuellement enfoncés. [MousePosition](#) retourne les coordonnées d'écran du pointeur de la souris et est équivalente à la valeur retournée par [Position](#).

Conversion entre les coordonnées clientes et d'écran

Étant donné que certaines informations sur l'emplacement de la souris sont spécifiées en coordonnées clientes et d'autres en coordonnées d'écran, vous devrez peut-être convertir un point d'un système de coordonnées en un autre. Cette opération peut être effectuée facilement à l'aide des méthodes [PointToClient](#) et [PointToScreen](#) disponibles sur la classe [Control](#).

Comportement d'événement de clic standard

Si vous souhaitez gérer les événements de clic de souris dans l'ordre approprié, vous devez connaître l'ordre dans lequel les événements de clic sont déclenchés dans les contrôles Windows Forms. Tous les contrôles Windows Forms déclenchent les événements de clic dans le même ordre quand un bouton de souris est enfoncé et relâché (quel que soit le bouton), sauf indication contraire mentionnée dans la liste suivante pour un contrôle spécifique. La liste ci-dessous indique l'ordre des événements déclenchés pour un clic sur un bouton de souris :

1. Événement [MouseDown](#) .
2. Événement [Click](#) .
3. Événement [MouseClick](#) .
4. Événement [MouseUp](#) .

Voici l'ordre des événements déclenchés pour un clic double-bouton de la souris :

1. Événement [MouseDown](#) .

2. Événement [Click](#) .
3. Événement [MouseClicked](#) .
4. Événement [MouseUp](#) .
5. Événement [MouseDown](#) .
6. Événement [DoubleClick](#) . (Cela peut varier, selon que le contrôle en question a la valeur [StandardDoubleClick](#) définie comme bit de style `true` . Pour plus d'informations sur la façon de définir un bit [ControlStyles](#), consultez la méthode [SetStyle](#).)
7. Événement [MouseDoubleClick](#) .
8. Événement [MouseUp](#) .

Pour obtenir un exemple de code illustrant l'ordre des événements de clic de souris, consultez [Comment : gérer des événements d'entrée d'utilisateur dans les contrôles de Windows Forms](#).

Contrôles spécifiques

Les contrôles suivants n'ont pas le comportement d'événement de clic de souris standard :

- [Button](#)
- [CheckBox](#)
- [ComboBox](#)
- [RadioButton](#)

NOTE

Pour le contrôle [ComboBox](#), le comportement d'événement détaillé plus loin se produit si l'utilisateur clique sur le champ d'édition, le bouton ou un élément dans la liste.

- Clic gauche : [Click](#), [MouseClicked](#)
 - Clic droit : aucun événement de clic déclenché
 - Double-clic gauche : [Click](#), [MouseClicked](#) ; [Click](#), [MouseClicked](#)
 - Double-clic droit : aucun événement de clic déclenché
- Contrôles [TextBox](#), [RichTextBox](#), [ListBox](#), [MaskedTextBox](#) et [CheckedListBox](#)

NOTE

Le comportement d'événement détaillé plus loin se produit quand l'utilisateur clique dans ces contrôles.

- Clic gauche : [Click](#), [MouseClicked](#)
 - Clic droit : aucun événement de clic déclenché
 - Double-clic gauche : [Click](#), [MouseClicked](#), [DoubleClick](#), [MouseDoubleClick](#)
 - Double-clic droit : aucun événement de clic déclenché
- Contrôle [ListView](#)

NOTE

Le comportement d'événement détaillé plus loin se produit quand l'utilisateur clique sur les éléments dans le contrôle [ListView](#). Aucun événement n'est déclenché si l'utilisateur clique ailleurs sur le contrôle. Outre les événements décrits plus loin, il existe les événements [BeforeLabelEdit](#) et [AfterLabelEdit](#), qui peuvent être utiles si vous souhaitez utiliser la validation avec le contrôle [ListView](#).

- Clic gauche : [Click](#), [MouseDown](#)
 - Clic droit : [Click](#), [MouseDown](#)
 - Double-clic gauche : [Click](#), [MouseDown](#) ; [DoubleClick](#), [MouseDownDoubleClick](#)
 - Double-clic droit : [Click](#), [MouseDown](#) ; [DoubleClick](#), [MouseDownDoubleClick](#)
- Contrôle [TreeView](#)

NOTE

Le comportement d'événement détaillé plus loin se produit uniquement quand l'utilisateur clique sur les éléments proprement dits ou à droite des éléments dans le contrôle [TreeView](#). Aucun événement n'est déclenché si l'utilisateur clique ailleurs sur le contrôle. Outre ceux décrits plus loin, il existe les événements [BeforeCheck](#), [BeforeSelect](#), [BeforeLabelEdit](#), [AfterSelect](#), [AfterCheck](#) et [AfterLabelEdit](#), qui peuvent être utiles si vous souhaitez utiliser la validation avec le contrôle [TreeView](#).

- Clic gauche : [Click](#), [MouseDown](#)
- Clic droit : [Click](#), [MouseDown](#)
- Double-clic gauche : [Click](#), [MouseDown](#) ; [DoubleClick](#), [MouseDownDoubleClick](#)
- Double-clic droit : [Click](#), [MouseDown](#) ; [DoubleClick](#), [MouseDownDoubleClick](#)

Comportement de peinture des contrôles Toggle

Les contrôles de basculement, tels que ceux dérivant de la classe [ButtonBase](#), présentent le comportement de peinture distinctif suivant en cas de combinaison avec des événements de clic de souris :

1. L'utilisateur appuie sur le bouton de la souris.
2. Le contrôle est peint à l'état enfoncé.
3. L'événement [MouseDown](#) est déclenché.
4. L'utilisateur relâche le bouton de la souris.
5. Le contrôle est peint à l'état déclenché.
6. L'événement [Click](#) est déclenché.
7. L'événement [MouseDown](#) est déclenché.
8. L'événement [MouseUp](#) est déclenché.

NOTE

Si l'utilisateur déplace le pointeur hors du contrôle de basculement alors que le bouton de la souris est enfoncé (par exemple en cas de déplacement de la souris hors du contrôle [Button](#) pendant qu'il est enfoncé), le contrôle de basculement est peint à l'état déclenché et seul l'événement [MouseUp](#) se produit. L'événement [Click](#) ou [MouseClicked](#) ne se produira pas dans cette situation.

Voir aussi

- [Entrée de la souris dans une application Windows Forms](#)

Procédure : faire la distinction entre les clics et les double-clics

10 minutes to read • [Edit Online](#)

En règle générale, un *clic* unique lance une action d'interface utilisateur et un *double-clic* étend l'action. Par exemple, un clic sélectionne habituellement un élément et un double-clic modifie l'élément sélectionné. Toutefois, les événements de clic Windows Forms ne gèrent pas facilement les scénarios dans lesquels un clic et un double-clic effectuent des actions incompatibles, car une action liée à l'événement [Click](#) ou [MouseClick](#) est effectuée avant l'action liée à l'événement [DoubleClick](#) ou [MouseDoubleClick](#). Cette rubrique illustre deux solutions à ce problème. Une solution consiste à gérer l'événement de double-clic et à annuler les actions dans la gestion de l'événement de clic. Dans de rares cas, vous devrez peut-être simuler le comportement de clic et de double-clic en gérant l'événement [MouseDown](#) et en utilisant les propriétés [DoubleClickTime](#) et [DoubleClickSize](#) de la classe [SystemInformation](#). Vous mesurez le délai entre les clics et si un deuxième clic se produit avant que la valeur de [DoubleClickTime](#) soit atteinte et que le clic se trouve dans un rectangle défini par [DoubleClickSize](#), vous exécutez l'action de double-clic ; sinon, vous exécutez l'action de clic.

Pour annuler une action de clic

- Assurez-vous que le contrôle que vous utilisez a un comportement de double-clic standard. Si ce n'est pas le cas, activez le contrôle avec la méthode [SetStyle](#). Gérez l'événement de double-clic et annulez les actions de clic et de double-clic. L'exemple de code suivant montre comment créer un bouton personnalisé avec double-clic activé et comment annuler l'action de clic dans le code de gestion de l'événement de double-clic.

```

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

namespace MouseRollBackSingleClick
{
    public class Form1 : Form
    {
        private DoubleClickButton button1;
        private FormBorderStyle initialStyle;

        public Form1()
        {
            initialStyle = this.FormBorderStyle;
            this.ClientSize = new System.Drawing.Size(292, 266);
            button1 = new DoubleClickButton();
            button1.Location = new Point (40,40);
            button1.Click += new EventHandler(button1_Click);
            button1.AutoSize = true;
            this.AllowDrop = true;
            button1.Text = "Click or Double Click";
            button1.DoubleClick += new EventHandler(button1_DoubleClick);
            this.Controls.Add(button1);
        }

        // Handle the double click event.
        void button1_DoubleClick(object sender, EventArgs e)
        {
            // Change the border style back to the initial style.
            this.FormBorderStyle = initialStyle;
            MessageBox.Show("Rolled back single click change.");
        }

        // Handle the click event.
        void button1_Click(object sender, EventArgs e)
        {
            this.FormBorderStyle = FormBorderStyle.FixedToolWindow;
        }

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

    public class DoubleClickButton : Button
    {
        public DoubleClickButton() : base()
        {
            // Set the style so a double click event occurs.
            SetStyle(ControlStyles.StandardClick |
                ControlStyles.StandardDoubleClick, true);
        }
    }
}

```

```

Imports System.ComponentModel
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms

Public Class Form1
    Inherits Form
    Private WithEvents button1 As DoubleClickButton
    Private initialStyle As FormBorderStyle

    Public Sub New()
        Me.SuspendLayout()
        initialStyle = Me.FormBorderStyle
        Me.ClientSize = New System.Drawing.Size(292, 266)
        button1 = New DoubleClickButton()
        button1.Location = New Point(40, 40)
        button1.AutoSize = True
        button1.Text = "Click or Double Click"
        Me.Controls.Add(button1)
        Me.Name = "Form1"
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub

    ' Handle the double click event.
    Private Sub button1_DoubleClick(ByVal sender As Object, ByVal e As EventArgs) _
        Handles button1.DoubleClick

        ' Change the border style back to the initial style.
        Me.FormBorderStyle = initialStyle
        MessageBox.Show("Rolled back single click change.")

    End Sub

    ' Handle the click event.
    Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs) _
        Handles button1.Click

        Me.FormBorderStyle = FormBorderStyle.FixedToolWindow

    End Sub

    <SThread()> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())

    End Sub
End Class

Public Class DoubleClickButton
    Inherits Button

    Public Sub New()
        ' Set the style so a double click event occurs.
        SetStyle(ControlStyles.StandardClick Or ControlStyles.StandardDoubleClick, True)

    End Sub
End Class

```

Pour faire la distinction entre les clics dans l'événement `MouseDown`

- Gérez l'événement `MouseDown` et déterminez l'emplacement et l'intervalle entre les clics à l'aide des propriétés `SystemInformation` appropriées et d'un composant `Timer`. Exécutez l'action appropriée selon qu'un clic ou un double-clic se produit. L'exemple de code suivant montre comment procéder.

```
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>
#using <System.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::Windows::Forms;

namespace SingleVersusDoubleClick
{
    public ref class Form1 : public Form
    {
    private:
        Rectangle hitTestRectangle;
    private:
        Rectangle doubleClickRectangle;
    private:
        TextBox^ outputBox;
    private:
        Timer^ doubleClickTimer;
    private:
        ProgressBar^ doubleClickBar;
    private:
        Label^ hitTestLabel;
    private:
        Label^ timerLabel;
    private:
        bool isFirstClick;
    private:
        bool isDoubleClick;
    private:
        int milliseconds;

    public:
        Form1()
        {
            hitTestRectangle = Rectangle();
            hitTestRectangle.Location = Point(30, 20);
            hitTestRectangle.Size = System::Drawing::Size(100, 40);

            doubleClickRectangle = Rectangle();

            outputBox = gcnew TextBox();
            outputBox->Location = Point(30, 120);
            outputBox->Size = System::Drawing::Size(200, 100);
            outputBox->AutoSize = false;
            outputBox->Multiline = true;

            doubleClickTimer = gcnew Timer();
            doubleClickTimer->Interval = 100;
            doubleClickTimer->Tick +=
                gcnew EventHandler(this, &Form1::doubleClickTimer_Tick);

            doubleClickBar = gcnew ProgressBar();
            doubleClickBar->Location = Point(30, 85);
            doubleClickBar->Minimum = 0;
            doubleClickBar->Maximum = SystemInformation::DoubleClickTime;

            hitTestLabel = gcnew Label();
            hitTestLabel->Location = Point(30, 5);
            hitTestLabel->Size = System::Drawing::Size(100, 15);
            hitTestLabel->Text = "Hit test rectangle:";
```

```

        timerLabel = gcnew Label();
        timerLabel->Location = Point(30, 70);
        timerLabel->Size = System::Drawing::Size(100, 15);
        timerLabel->Text = "Double click timer:";

        isFirstClick = true;

        this->Paint += gcnew PaintEventHandler(this, &Form1::Form1_Paint);
        this->MouseDown +=
            gcnew MouseEventHandler(this, &Form1::Form1_MouseDown);
        this->Controls->
            AddRange(gcnew array<Control^> { doubleClickBar, outputBox,
                hitTestLabel, timerLabel });
    }

    // Detect a valid single click or double click.
private:
    void Form1_MouseDown(Object^ sender, MouseEventArgs^ e)
    {
        // Verify that the mouse click is in the main hit
        // test rectangle.
        if (!hitTestRectangle.Contains(e->Location))
        {
            return;
        }

        // This is the first mouse click.
        if (isFirstClick)
        {
            isFirstClick = false;

            // Determine the location and size of the double click
            // rectangle area to draw around the cursor point.
            doubleClickRectangle = Rectangle(
                e->X - (SystemInformation::DoubleClickSize.Width / 2),
                e->Y - (SystemInformation::DoubleClickSize.Height / 2),
                SystemInformation::DoubleClickSize.Width,
                SystemInformation::DoubleClickSize.Height);
            Invalidate();

            // Start the double click timer.
            doubleClickTimer->Start();
        }

        // This is the second mouse click.
        else
        {
            // Verify that the mouse click is within the double click
            // rectangle and is within the system-defined double
            // click period.
            if (doubleClickRectangle.Contains(e->Location) &&
                milliseconds < SystemInformation::DoubleClickTime)
            {
                isDoubleClick = true;
            }
        }
    }

private:
    void doubleClickTimer_Tick(Object^ sender, EventArgs^ e)
    {
        milliseconds += 100;
        doubleClickBar->Increment(100);

        // The timer has reached the double click time limit.
        if (milliseconds >= SystemInformation::DoubleClickTime)
        {
            doubleClickTimer->Stop();
        }
    }
}

```

```

        doubleClickTimer->Stop();

        if (isDoubleClick)
        {
            outputBox->AppendText("Perform double click action");
            outputBox->AppendText(Environment::NewLine);
        }
        else
        {
            outputBox->AppendText("Perform single click action");
            outputBox->AppendText(Environment::NewLine);
        }

        // Allow the MouseDown event handler to process clicks again.
        isFirstClick = true;
        isDoubleClick = false;
        milliseconds = 0;
        doubleClickBar->Value = 0;
    }
}

// Paint the hit test and double click rectangles.
private:
void Form1_Paint(Object^ sender, PaintEventArgs^ e)
{
    // Draw the border of the main hit test rectangle.
    e->Graphics->DrawRectangle(Pens::Black, hitTestRectangle);

    // Fill in the double click rectangle.
    e->Graphics->FillRectangle(Brushes::Blue, doubleClickRectangle);
}
};
}

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run(gcnew SingleVersusDoubleClick::Form1);
}

```

```

using System;
using System.Drawing;
using System.Windows.Forms;

namespace SingleVersusDoubleClick
{
    class Form1 : Form
    {
        private Rectangle hitTestRectangle = new Rectangle();
        private Rectangle doubleClickRectangle = new Rectangle();
        private TextBox textBox1 = new TextBox();
        private Timer doubleClickTimer = new Timer();
        private ProgressBar doubleClickBar = new ProgressBar();
        private Label label1 = new Label();
        private Label label2 = new Label();
        private bool isFirstClick = true;
        private bool isDoubleClick = false;
        private int milliseconds = 0;

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

```

```

public Form1()
{
    label1.Location = new Point(30, 5);
    label1.Size = new Size(100, 15);
    label1.Text = "Hit test rectangle:";

    label2.Location = new Point(30, 70);
    label2.Size = new Size(100, 15);
    label2.Text = "Double click timer:";

    hitTestRectangle.Location = new Point(30, 20);
    hitTestRectangle.Size = new Size(100, 40);

    doubleClickTimer.Interval = 100;
    doubleClickTimer.Tick +=
        new EventHandler(doubleClickTimer_Tick);

    doubleClickBar.Location = new Point(30, 85);
    doubleClickBar.Minimum = 0;
    doubleClickBar.Maximum = SystemInformation.DoubleClickTime;

    textBox1.Location = new Point(30, 120);
    textBox1.Size = new Size(200, 100);
    textBox1.AutoSize = false;
    textBox1.Multiline = true;

    this.Paint += new PaintEventHandler(Form1_Paint);
    this.MouseDown += new MouseEventHandler(Form1_MouseDown);
    this.Controls.AddRange(new Control[] { doubleClickBar, textBox1,
        label1, label2 });
}

// Detect a valid single click or double click.
void Form1_MouseDown(object sender, MouseEventArgs e)
{
    // Verify that the mouse click is in the main hit
    // test rectangle.
    if (!hitTestRectangle.Contains(e.Location))
    {
        return;
    }

    // This is the first mouse click.
    if (isFirstClick)
    {
        isFirstClick = false;

        // Determine the location and size of the double click
        // rectangle area to draw around the cursor point.
        doubleClickRectangle = new Rectangle(
            e.X - (SystemInformation.DoubleClickSize.Width / 2),
            e.Y - (SystemInformation.DoubleClickSize.Height / 2),
            SystemInformation.DoubleClickSize.Width,
            SystemInformation.DoubleClickSize.Height);
        Invalidate();

        // Start the double click timer.
        doubleClickTimer.Start();
    }

    // This is the second mouse click.
    else
    {
        // Verify that the mouse click is within the double click
        // rectangle and is within the system-defined double
        // click period.
        if (doubleClickRectangle.Contains(e.Location) &&
            milliseconds < SystemInformation.DoubleClickTime)
        ,

```

```

        {
            isDoubleClick = true;
        }
    }
}

void doubleClickTimer_Tick(object sender, EventArgs e)
{
    milliseconds += 100;
    doubleClickBar.Increment(100);

    // The timer has reached the double click time limit.
    if (milliseconds >= SystemInformation.DoubleClickTime)
    {
        doubleClickTimer.Stop();

        if (isDoubleClick)
        {
            textBox1.AppendText("Perform double click action");
            textBox1.AppendText(Environment.NewLine);
        }
        else
        {
            textBox1.AppendText("Perform single click action");
            textBox1.AppendText(Environment.NewLine);
        }

        // Allow the MouseDown event handler to process clicks again.
        isFirstClick = true;
        isDoubleClick = false;
        milliseconds = 0;
        doubleClickBar.Value = 0;
    }
}

// Paint the hit test and double click rectangles.
void Form1_Paint(object sender, PaintEventArgs e)
{
    // Draw the border of the main hit test rectangle.
    e.Graphics.DrawRectangle(Pens.Black, hitTestRectangle);

    // Fill in the double click rectangle.
    e.Graphics.FillRectangle(Brushes.Blue, doubleClickRectangle);
}
}
}

```

```

Imports System.Drawing
Imports System.Windows.Forms

Namespace SingleVersusDoubleClick

    Class Form1
        Inherits Form
        Private hitTestRectangle As New Rectangle()
        Private doubleClickRectangle As New Rectangle()
        Private textBox1 As New TextBox()
        Private WithEvents doubleClickTimer As New Timer()
        Private doubleClickBar As New ProgressBar()
        Private label1 As New Label()
        Private label2 As New Label()
        Private isFirstClick As Boolean = True
        Private isDoubleClick As Boolean = False
        Private milliseconds As Integer = 0

        <STAThread(>> _
        Public Shared Sub Main()
            Application.EnableVisualStyles()

```



```

Application.EnableVisualStyles()
Application.Run(New Form1())
End Sub

Public Sub New()
    label1.Location = New Point(30, 5)
    label1.Size = New Size(100, 15)
    label1.Text = "Hit test rectangle:"

    label2.Location = New Point(30, 70)
    label2.Size = New Size(100, 15)
    label2.Text = "Double click timer:"

    hitTestRectangle.Location = New Point(30, 20)
    hitTestRectangle.Size = New Size(100, 40)
    doubleClickTimer.Interval = 100

    doubleClickBar.Location = New Point(30, 85)
    doubleClickBar.Minimum = 0
    doubleClickBar.Maximum = SystemInformation.DoubleClickTime

    textBox1.Location = New Point(30, 120)
    textBox1.Size = New Size(200, 100)
    textBox1.AutoSize = False
    textBox1.Multiline = True

    Me.Controls.Add(doubleClickBar)
    Me.Controls.Add(textBox1)
    Me.Controls.Add(label1)
    Me.Controls.Add(label2)
End Sub

' Detect a valid single click or double click.
Sub Form1_MouseDown(ByVal sender As Object, _
    ByVal e As MouseEventArgs) Handles Me.MouseDown

    ' Verify that the mouse click is in the main hit
    ' test rectangle.
    If Not hitTestRectangle.Contains(e.Location) Then
        Return
    End If

    ' This is the first mouse click.
    If isFirstClick = True Then
        isFirstClick = False

        ' Determine the location and size of the double click
        ' rectangle to draw around the cursor point.
        doubleClickRectangle = New Rectangle( _
            e.X - (SystemInformation.DoubleClickSize.Width / 2), _
            e.Y - (SystemInformation.DoubleClickSize.Height / 2), _
            SystemInformation.DoubleClickSize.Width, _
            SystemInformation.DoubleClickSize.Height)
        Invalidate()

        ' Start the double click timer.
        doubleClickTimer.Start()

    ' This is the second mouse click.
    Else
        ' Verify that the mouse click is within the double click
        ' rectangle and is within the system-defined double
        ' click period.
        If doubleClickRectangle.Contains(e.Location) And _
            milliseconds < SystemInformation.DoubleClickTime Then
            isDoubleClick = True
        End If
    End If
End Sub

```

```

Sub doubleClickTimer_Tick(ByVal sender As Object, _
    ByVal e As EventArgs) Handles doubleClickTimer.Tick

    milliseconds += 100
    doubleClickBar.Increment(100)

    ' The timer has reached the double click time limit.
    If milliseconds >= SystemInformation.DoubleClickTime Then
        doubleClickTimer.Stop()

        If isDoubleClick Then
            textBox1.AppendText("Perform double click action")
            textBox1.AppendText(Environment.NewLine)
        Else
            textBox1.AppendText("Perform single click action")
            textBox1.AppendText(Environment.NewLine)
        End If

        ' Allow the MouseDown event handler to process clicks again.
        isFirstClick = True
        isDoubleClick = False
        milliseconds = 0
        doubleClickBar.Value = 0
    End If
End Sub

' Paint the hit test and double click rectangles.
Sub Form1_Paint(ByVal sender As Object, _
    ByVal e As PaintEventArgs) Handles Me.Paint

    ' Draw the border of the main hit test rectangle.
    e.Graphics.DrawRectangle(Pens.Black, hitTestRectangle)

    ' Fill in the double click rectangle.
    e.Graphics.FillRectangle(Brushes.Blue, doubleClickRectangle)
End Sub
End Class
End Namespace

```

Compilation du code

Ces exemples nécessitent :

- Références aux assemblés System, System.Drawing et System.Windows.Forms.

Voir aussi

- [Entrée de la souris dans une application Windows Forms](#)

Pointeurs de souris dans les Windows Forms

3 minutes to read • [Edit Online](#)

Le *pointeur* de la souris, qui est parfois appelé curseur, est une image bitmap qui spécifie un point de focus sur l'écran pour l'entrée d'utilisateur avec la souris. Cette rubrique fournit une vue d'ensemble du pointeur de la souris dans Windows Forms et décrit quelques-unes des façons de modifier et de contrôler le pointeur de la souris.

Accès au pointeur de la souris

Le pointeur de la souris est représenté par la classe [Cursor](#), et chaque [Control](#) a une propriété [Control.Cursor](#) qui spécifie le pointeur pour ce contrôle. La classe [Cursor](#) contient des propriétés qui décrivent le pointeur, telles que les propriétés de [Position](#) et de [HotSpot](#), ainsi que des méthodes qui peuvent modifier l'apparence du pointeur, comme les méthodes [Show](#), [Hide](#) et [DrawStretched](#).

Contrôle du pointeur de la souris

Il peut arriver que vous souhaitiez limiter la zone dans laquelle le pointeur de la souris peut être utilisé ou modifier la position de la souris. Vous pouvez récupérer ou définir l'emplacement actuel de la souris à l'aide de la propriété [Position](#) de la [Cursor](#). En outre, vous pouvez limiter la zone à laquelle le pointeur de la souris peut être utilisé pour définir la propriété [Clip](#). La zone de découpage est, par défaut, la totalité de l'écran.

Modification du pointeur de la souris

La modification du pointeur de la souris est un moyen important de fournir des commentaires à l'utilisateur. Par exemple, le pointeur de la souris peut être modifié dans les gestionnaires du [MouseEnter](#) et [MouseLeave](#) des événements pour indiquer à l'utilisateur que les calculs se produisent et pour limiter l'interaction de l'utilisateur dans le contrôle. Parfois, le pointeur de la souris change en raison des événements système, par exemple quand votre application est impliquée dans une opération de glisser-déplacer.

Le principal moyen de modifier le pointeur de la souris consiste à définir la propriété [Control.Cursor](#) ou [DefaultCursor](#) d'un contrôle sur une nouvelle [Cursor](#). Pour obtenir des exemples de modification du pointeur de la souris, consultez l'exemple de code dans la classe [Cursor](#). En outre, la classe [Cursors](#) expose un ensemble d'objets [Cursor](#) pour de nombreux types différents de pointeurs, tels qu'un pointeur qui ressemble à une main. Pour afficher le pointeur d'attente, qui ressemble à un sablier, chaque fois que le pointeur de la souris se trouve sur le contrôle, utilisez la propriété [UseWaitCursor](#) de la classe [Control](#).

Voir aussi

- [Cursor](#)
- [Entrée de la souris dans une application Windows Forms](#)
- [Fonctionnalité de glisser-déposer dans les Windows Forms](#)

Capture de la souris dans les Windows Forms

2 minutes to read • [Edit Online](#)

La capture de la souris fait référence à lorsqu'un contrôle prend la commande de toutes les entrées de la souris. Lorsqu'un contrôle a capturé la souris, il reçoit l'entrée de la souris, que le pointeur se trouve dans ses limites ou non.

Configuration de la capture de la souris

Dans Windows Forms la souris est capturée par le contrôle lorsque l'utilisateur appuie sur un bouton de la souris sur un contrôle et que la souris est relâchée par le contrôle lorsque l'utilisateur relâche le bouton de la souris.

La propriété [Capture](#) de la classe [Control](#) spécifie si un contrôle a capturé la souris. Pour déterminer quand un contrôle perd la capture de la souris, gérez l'événement [MouseCaptureChanged](#).

Seule la fenêtre de premier plan peut capturer la souris. Quand une fenêtre d'arrière-plan tente de capturer la souris, la fenêtre reçoit des messages uniquement pour les événements de souris qui se produisent lorsque le pointeur de la souris se trouve dans la partie visible de la fenêtre. De même, même si la fenêtre de premier plan a capturé la souris, l'utilisateur peut toujours cliquer sur une autre fenêtre, en l'amenant au premier plan. Lorsque la souris est capturée, les touches de raccourci ne fonctionnent pas.

Voir aussi

- [Entrée de la souris dans une application Windows Forms](#)

Fonctionnalité de glisser-déplacer dans les Windows Forms

4 minutes to read • [Edit Online](#)

Windows Forms comprend un ensemble de méthodes, événements et classes qui implémentent le comportement de glisser-déposer. Cette rubrique offre une vue d'ensemble de la prise en charge du glisser-déposer dans Windows Forms. Consultez également [opérations de glisser-déplacer et prise en charge du presse-papiers](#).

Exécution d'opérations de glisser-déposer

Pour effectuer une opération de glisser-déposer, utilisez la méthode [DoDragDrop](#) de la classe [Control](#). Pour plus d'informations sur la façon d'exécuter une opération de glisser-déplacer, consultez [DoDragDrop](#). Pour obtenir le rectangle sur lequel le pointeur de la souris doit être déplacé pour qu'une opération de glisser-déplacer commence, utilisez la propriété [DragSize](#) de la classe [SystemInformation](#).

Événements liés aux opérations de glisser-déposer

Il existe deux catégories d'événements dans une opération de glisser-déplacer : les événements qui se produisent sur la cible actuelle de l'opération de glisser-déplacer et ceux qui se produisent sur la source de l'opération de glisser-déplacer.

Événements sur la cible actuelle

Le tableau suivant présente les événements qui se produisent sur la cible actuelle d'une opération de glisser-déplacer.

ÉVÉNEMENT DE SOURIS	DESCRIPTION
DragEnter	Cet événement se produit quand un objet est déplacé dans les limites d'un contrôle. Le gestionnaire pour cet événement reçoit un argument de type DragEventArgs .
DragOver	Cet événement se produit quand un objet est déplacé alors que le pointeur de la souris se trouve dans les limites du contrôle. Le gestionnaire pour cet événement reçoit un argument de type DragEventArgs .
DragDrop	Cet événement se produit quand une opération de glisser-déplacer est terminée. Le gestionnaire pour cet événement reçoit un argument de type DragEventArgs .
DragLeave	Cet événement se produit quand un objet est déplacé hors des limites d'un contrôle. Le gestionnaire pour cet événement reçoit un argument de type EventArgs .

La classe [DragEventArgs](#) fournit l'emplacement du pointeur de la souris, l'état actuel des boutons de la souris et des touches de modification du clavier, les données déplacées et des valeurs [DragDropEffects](#) qui spécifient les opérations autorisées par la source de l'événement de glisser et l'effet de déplacement cible pour cette opération.

Événements sur la source

Le tableau suivant présente les événements qui se produisent sur la source de l'opération de glisser-déposer.

ÉVÉNEMENT DE SOURIS	DESCRIPTION
GiveFeedback	Cet événement se produit pendant une opération glisser. Il permet de fournir une aide visuelle à l'utilisateur (par exemple la modification du pointeur de souris) pour signaler que l'opération de glisser-déplacer est en cours. Le gestionnaire pour cet événement reçoit un argument de type GiveFeedbackEventArgs .
QueryContinueDrag	Cet événement se produit pendant une opération de glisser-déposer et permet à la source de cette opération de déterminer si l'opération doit être annulée. Le gestionnaire pour cet événement reçoit un argument de type QueryContinueDragEventArgs .

La classe [QueryContinueDragEventArgs](#) fournit l'état actuel des boutons de la souris et des touches de modification du clavier, une valeur indiquant si la touche Échappement a été enfoncée et une valeur [DragAction](#) qui peut être définie pour spécifier si l'opération de glisser-déplacer doit continuer.

Voir aussi

- [Entrée de la souris dans une application Windows Forms](#)

Comment : simuler des événements de la souris et du clavier dans le code

12 minutes to read • [Edit Online](#)

Windows Forms fournit plusieurs options pour simuler par programmation l'entrée de souris et de clavier. Cette rubrique offre une vue d'ensemble de ces options.

Simulation de l'entrée de souris

Le meilleur moyen de simuler des événements de souris consiste à appeler la méthode `On EventName` qui déclenche l'événement de souris que vous souhaitez simuler. Cette option est généralement possible uniquement dans les contrôles et les formulaires personnalisés, car les méthodes qui déclenchent des événements sont protégées et ne sont pas accessibles en dehors du contrôle ou du formulaire. Par exemple, les étapes suivantes illustrent comment simuler un clic sur le bouton droit de la souris dans le code.

Pour cliquer sur le bouton droit de la souris par programmation

1. Créez un `MouseEventArgs` dont la propriété `Button` a la valeur `MouseButtons.Right`.
2. Appelez la méthode `OnMouseClicked` avec ce `MouseEventArgs` comme argument.

Pour plus d'informations sur les contrôles personnalisés, consultez [Développement de contrôles Windows Forms au moment du design](#).

Il existe d'autres manières de simuler l'entrée de souris. Par exemple, vous pouvez définir par programmation une propriété de contrôle qui représente un état qui est généralement défini par une entrée de souris (telle que la propriété `Checked` du contrôle `CheckBox`) ou vous pouvez appeler directement le délégué qui est attaché à l'événement que vous souhaitez simuler.

Simulation de l'entrée au clavier

Bien que vous puissiez simuler l'entrée au clavier à l'aide des stratégies abordées ci-dessus pour l'entrée de souris, Windows Forms fournit également la classe `SendKeys` pour envoyer des séquences de touches à l'application active.

Caution

Si votre application est prévue pour une utilisation internationale avec différents claviers, l'utilisation de `SendKeys.Send` peut produire des résultats imprévisibles et doit être évitée.

NOTE

La classe [SendKeys](#) a été mise à jour pour .NET Framework 3.0 pour pouvoir être utilisée dans les applications qui s'exécutent sur Windows Vista. La sécurité renforcée de Windows Vista (également appelée Contrôle de compte d'utilisateur) empêche le fonctionnement correct de l'implémentation précédente.

La classe [SendKeys](#) est vulnérable aux problèmes de synchronisation, que certains développeurs ont dû contourner. L'implémentation mise à jour est toujours vulnérable aux problèmes de synchronisation, mais est légèrement plus rapide et peut nécessiter certaines modifications des solutions de contournement. La classe [SendKeys](#) tente d'abord d'utiliser l'implémentation précédente et, en cas d'échec, utilise la nouvelle implémentation. Ainsi, la classe [SendKeys](#) peut se comporter différemment sur différents systèmes d'exploitation. En outre, quand la classe [SendKeys](#) utilise la nouvelle implémentation, la méthode [SendWait](#) n'attend pas que les messages soient traités quand ils sont envoyés à un autre processus.

Si votre application repose sur un comportement cohérent indépendamment du système d'exploitation, vous pouvez forcer la classe [SendKeys](#) à utiliser la nouvelle implémentation en ajoutant le paramètre d'application suivant à votre fichier app.config.

```
<appSettings>
  <add key="SendKeys" value="SendInput"/>
</appSettings>
```

Pour forcer la classe [SendKeys](#) à utiliser l'implémentation précédente, utilisez plutôt la valeur `"JournalHook"`.

Pour envoyer une séquence de touches à la même application

1. Appelez la méthode [Send](#) ou [SendWait](#) de la classe [SendKeys](#). Les séquences de touches spécifiées seront reçues par le contrôle actif de l'application. L'exemple de code suivant utilise [Send](#) pour simuler une pression sur la touche Entrée quand l'utilisateur double-clique sur la surface du formulaire. Cet exemple suppose que l'on a un [Form](#) avec un seul contrôle [Button](#) ayant un index de tabulation de 0.

```
// Send a key to the button when the user double-clicks anywhere
// on the form.
private:
void Form1_DoubleClick(Object^ sender, EventArgs^ e)
{
    // Send the enter key to the button, which triggers the click
    // event for the button. This works because the tab stop of
    // the button is 0.
    SendKeys::Send("{ENTER}");
}
```

```
// Send a key to the button when the user double-clicks anywhere
// on the form.
private void Form1_DoubleClick(object sender, EventArgs e)
{
    // Send the enter key to the button, which raises the click
    // event for the button. This works because the tab stop of
    // the button is 0.
    SendKeys.Send("{ENTER}");
}
```



```

' Send a key to the button when the user double-clicks anywhere
' on the form.
Private Sub Form1_DoubleClick(ByVal sender As Object, _
    ByVal e As EventArgs) Handles Me.DoubleClick

    ' Send the enter key to the button, which raises the click
    ' event for the button. This works because the tab stop of
    ' the button is 0.
    SendKeys.Send("{ENTER}")
End Sub

```

Pour envoyer une séquence de touches vers une autre application

1. Activez la fenêtre d'application qui recevra les séquences de touches, puis appelez la méthode [Send](#) ou [SendWait](#). Comme il n'existe aucune méthode managée permettant d'activer une autre application, vous devez utiliser des méthodes Windows natives pour forcer le focus sur d'autres applications. L'exemple de code suivant utilise l'appel de code non managé pour appeler les méthodes `FindWindow` et `SetForegroundWindow` pour activer la fenêtre de l'application Calculatrice, puis appelle [SendWait](#) pour émettre une série de calculs vers l'application Calculatrice.

NOTE

Les paramètres corrects de l'appel `FindWindow` qui recherche l'application Calculatrice varient en fonction de votre version de Windows. Le code suivant recherche l'application Calculator sur Windows 7. Sur Windows Vista, remplacez le premier paramètre par « SciCalc ». Vous pouvez utiliser l'outil Spy++, fourni avec Visual Studio, pour déterminer les paramètres corrects.

```

// Get a handle to an application window.
public:
    [DllImport("USER32.DLL", CharSet = CharSet::Unicode)]
    static IntPtr FindWindow(String^ lpClassName, String^ lpWindowName);
public:
    // Activate an application window.
    [DllImport("USER32.DLL")]
    static bool SetForegroundWindow(IntPtr hWnd);

    // Send a series of key presses to the Calculator application.
private:
    void button1_Click(Object^ sender, EventArgs^ e)
    {
        // Get a handle to the Calculator application. The window class
        // and window name were obtained using the Spy++ tool.
        IntPtr calculatorHandle = FindWindow("CalcFrame", "Calculator");

        // Verify that Calculator is a running process.
        if (calculatorHandle == IntPtr::Zero)
        {
            MessageBox::Show("Calculator is not running.");
            return;
        }

        // Make Calculator the foreground application and send it
        // a set of calculations.
        SetForegroundWindow(calculatorHandle);
        SendKeys::SendWait("111");
        SendKeys::SendWait("*");
        SendKeys::SendWait("11");
        SendKeys::SendWait("=");
    }
}

```

```

// Get a handle to an application window.
[DllImport("USER32.DLL", CharSet = CharSet.Unicode)]
public static extern IntPtr FindWindow(string lpClassName,
    string lpWindowName);

// Activate an application window.
[DllImport("USER32.DLL")]
public static extern bool SetForegroundWindow(IntPtr hWnd);

// Send a series of key presses to the Calculator application.
private void button1_Click(object sender, EventArgs e)
{
    // Get a handle to the Calculator application. The window class
    // and window name were obtained using the Spy++ tool.
    IntPtr calculatorHandle = FindWindow("CalcFrame", "Calculator");

    // Verify that Calculator is a running process.
    if (calculatorHandle == IntPtr.Zero)
    {
        MessageBox.Show("Calculator is not running.");
        return;
    }

    // Make Calculator the foreground application and send it
    // a set of calculations.
    SetForegroundWindow(calculatorHandle);
    SendKeys.SendWait("111");
    SendKeys.SendWait("*");
    SendKeys.SendWait("11");
    SendKeys.SendWait("=");
}

```

```

' Get a handle to an application window.
Declare Auto Function FindWindow Lib "USER32.DLL" ( _
    ByVal lpClassName As String, _
    ByVal lpWindowName As String) As IntPtr

' Activate an application window.
Declare Auto Function SetForegroundWindow Lib "USER32.DLL" _
    (ByVal hWnd As IntPtr) As Boolean

' Send a series of key presses to the Calculator application.
Private Sub button1_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles button1.Click

    ' Get a handle to the Calculator application. The window class
    ' and window name were obtained using the Spy++ tool.
    Dim calculatorHandle As IntPtr = FindWindow("CalcFrame", "Calculator")

    ' Verify that Calculator is a running process.
    If calculatorHandle = IntPtr.Zero Then
        MsgBox("Calculator is not running.")
        Return
    End If

    ' Make Calculator the foreground application and send it
    ' a set of calculations.
    SetForegroundWindow(calculatorHandle)
    SendKeys.SendWait("111")
    SendKeys.SendWait("*")
    SendKeys.SendWait("11")
    SendKeys.SendWait("=")
End Sub

```

Exemple

L'exemple de code suivant est l'application complète pour les exemples de code précédents.

```
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>
#using <System.dll>

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace System::Drawing;
using namespace System::Windows::Forms;

namespace SimulateKeyPress
{
    public ref class Form1 : public Form
    {
    public:
        Form1()
        {
            Button^ button1 = gcnew Button();
            button1->Location = Point(10, 10);
            button1->TabIndex = 0;
            button1->Text = "Click to automate Calculator";
            button1->AutoSize = true;
            button1->Click += gcnew EventHandler(this, &Form1::button1_Click);

            this->DoubleClick += gcnew EventHandler(this,
                &Form1::Form1_DoubleClick);
            this->Controls->Add(button1);
        }

        // Get a handle to an application window.
    public:
        [DllImport("USER32.DLL", CharSet = CharSet::Unicode)]
        static IntPtr FindWindow(String^ lpClassName, String^ lpWindowName);
    public:
        // Activate an application window.
        [DllImport("USER32.DLL")]
        static bool SetForegroundWindow(IntPtr hWnd);

        // Send a series of key presses to the Calculator application.
    private:
        void button1_Click(Object^ sender, EventArgs^ e)
        {
            // Get a handle to the Calculator application. The window class
            // and window name were obtained using the Spy++ tool.
            IntPtr calculatorHandle = FindWindow("CalcFrame", "Calculator");

            // Verify that Calculator is a running process.
            if (calculatorHandle == IntPtr::Zero)
            {
                MessageBox::Show("Calculator is not running.");
                return;
            }

            // Make Calculator the foreground application and send it
            // a set of calculations.
            SetForegroundWindow(calculatorHandle);
            SendKeys::SendWait("111");
            SendKeys::SendWait("*");
            SendKeys::SendWait("11");
            SendKeys::SendWait("=");
        }

        // Send a key to the button when the user double-clicks anywhere
```

```

        // on the form.
private:
    void Form1_DoubleClick(Object^ sender, EventArgs^ e)
    {
        // Send the enter key to the button, which triggers the click
        // event for the button. This works because the tab stop of
        // the button is 0.
        SendKeys::Send("{ENTER}");
    }
};

}

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run(gcnew SimulateKeyPress::Form1());
}

```

```

using System;
using System.Runtime.InteropServices;
using System.Drawing;
using System.Windows.Forms;

namespace SimulateKeyPress
{
    class Form1 : Form
    {
        private Button button1 = new Button();

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

        public Form1()
        {
            button1.Location = new Point(10, 10);
            button1.TabIndex = 0;
            button1.Text = "Click to automate Calculator";
            button1.AutoSize = true;
            button1.Click += new EventHandler(button1_Click);

            this.DoubleClick += new EventHandler(Form1_DoubleClick);
            this.Controls.Add(button1);
        }

        // Get a handle to an application window.
        [DllImport("USER32.DLL", CharSet = CharSet.Unicode)]
        public static extern IntPtr FindWindow(string lpClassName,
            string lpWindowName);

        // Activate an application window.
        [DllImport("USER32.DLL")]
        public static extern bool SetForegroundWindow(IntPtr hWnd);

        // Send a series of key presses to the Calculator application.
        private void button1_Click(object sender, EventArgs e)
        {
            // Get a handle to the Calculator application. The window class
            // and window name were obtained using the Spy++ tool.
            IntPtr calculatorHandle = FindWindow("CalcFrame", "Calculator");

            // Verify that Calculator is a running process.
            if (calculatorHandle == IntPtr.Zero)
            {

```

```

        {
            MessageBox.Show("Calculator is not running.");
            return;
        }

        // Make Calculator the foreground application and send it
        // a set of calculations.
        SetForegroundWindow(calculatorHandle);
        SendKeys.SendWait("111");
        SendKeys.SendWait("*");
        SendKeys.SendWait("11");
        SendKeys.SendWait("=");
    }

    // Send a key to the button when the user double-clicks anywhere
    // on the form.
    private void Form1_DoubleClick(object sender, EventArgs e)
    {
        // Send the enter key to the button, which raises the click
        // event for the button. This works because the tab stop of
        // the button is 0.
        SendKeys.Send("{ENTER}");
    }
}

```

```

Imports System.Runtime.InteropServices
Imports System.Drawing
Imports System.Windows.Forms

Namespace SimulateKeyPress

    Class Form1
        Inherits Form
        Private WithEvents button1 As New Button()

        <STAThread()> _
        Public Shared Sub Main()
            Application.EnableVisualStyles()
            Application.Run(New Form1())
        End Sub

        Public Sub New()
            button1.Location = New Point(10, 10)
            button1.TabIndex = 0
            button1.Text = "Click to automate Calculator"
            button1.AutoSize = True
            Me.Controls.Add(button1)
        End Sub

        ' Get a handle to an application window.
        Declare Auto Function FindWindow Lib "USER32.DLL" ( _
            ByVal lpClassName As String, _
            ByVal lpWindowName As String) As IntPtr

        ' Activate an application window.
        Declare Auto Function SetForegroundWindow Lib "USER32.DLL" _
            (ByVal hWnd As IntPtr) As Boolean

        ' Send a series of key presses to the Calculator application.
        Private Sub button1_Click(ByVal sender As Object, _
            ByVal e As EventArgs) Handles button1.Click

            ' Get a handle to the Calculator application. The window class
            ' and window name were obtained using the Spy++ tool.
            Dim calculatorHandle As IntPtr = FindWindow("CalcFrame", "Calculator")

            ' Verify that Calculator is a running process

```

```

        verify that calculator is a running process.
    If calculatorHandle = IntPtr.Zero Then
        MsgBox("Calculator is not running.")
        Return
    End If

    ' Make Calculator the foreground application and send it
    ' a set of calculations.
    SetForegroundWindow(calculatorHandle)
    SendKeys.SendWait("111")
    SendKeys.SendWait("*")
    SendKeys.SendWait("11")
    SendKeys.SendWait("=")
End Sub

' Send a key to the button when the user double-clicks anywhere
' on the form.
Private Sub Form1_DoubleClick(ByVal sender As Object, _
    ByVal e As EventArgs) Handles Me.DoubleClick

    ' Send the enter key to the button, which raises the click
    ' event for the button. This works because the tab stop of
    ' the button is 0.
    SendKeys.Send("{ENTER}")
End Sub

End Class
End Namespace

```

Compilation du code

Cet exemple nécessite :

- des références aux assemblies System, System.Drawing et System.Windows.Forms.

Voir aussi

- [Entrées d'utilisateur dans les Windows Forms](#)

Comment : gérer des événements d'entrée d'utilisateur dans les contrôles Windows Forms

27 minutes to read • [Edit Online](#)

Cet exemple montre comment gérer la plupart des événements de clavier, de souris, de focus et de validation qui peuvent se produire dans un contrôle Windows Forms. La zone de texte nommée `TextBoxInput` reçoit les événements quand elle a le focus et les informations relatives à chaque événement sont écrites dans la zone de texte nommée `TextBoxOutput` dans l'ordre dans lequel les événements sont déclenchés. L'application comprend également un ensemble de cases à cocher qui peuvent servir à filtrer les événements pour lesquels établir un rapport.

Exemple

```
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>
#using <System.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::ComponentModel;
using namespace System::Windows::Forms;

namespace UserInputWalkthrough
{
    public ref class Form1 : public Form
    {
        Label^ lblEvent;
        Label^ lblInput;

        TextBox^ TextBoxOutput;
        TextBox^ TextBoxInput;
        GroupBox^ GroupBoxEvents;
        Button^ ButtonClear;
        LinkLabel^ LinkLabelDrag;

        CheckBox^ CheckBoxToggleAll;
        CheckBox^ CheckBoxMouse;
        CheckBox^ CheckBoxMouseEnter;
        CheckBox^ CheckBoxMouseMove;
        CheckBox^ CheckBoxMousePoints;
        CheckBox^ CheckBoxMouseDrag;
        CheckBox^ CheckBoxMouseDragOver;
        CheckBox^ CheckBoxKeyboard;
        CheckBox^ CheckBoxKeyUpDown;
        CheckBox^ CheckBoxFocus;
        CheckBox^ CheckBoxValidation;

    public:
        Form1() : Form()
        {
            this->Load += gcnew EventHandler(this, &Form1::Form1_Load);

            lblEvent = gcnew Label();
            lblInput = gcnew Label();

            TextBoxOutput = gcnew TextBox();
            TextBoxInput = gcnew TextBox();
            GroupBoxEvents = gcnew GroupBox();
```

```

        ButtonClear = gcnew Button();
        LinkLabelDrag = gcnew LinkLabel();

        CheckBoxToggleAll = gcnew CheckBox();
        CheckBoxMouse = gcnew CheckBox();
        CheckBoxMouseEnter = gcnew CheckBox();
        CheckBoxMouseMove = gcnew CheckBox();
        CheckBoxMousePoints = gcnew CheckBox();
        CheckBoxMouseDrag = gcnew CheckBox();
        CheckBoxMouseDragOver = gcnew CheckBox();
        CheckBoxKeyboard = gcnew CheckBox();
        CheckBoxKeyUpDown = gcnew CheckBox();
        CheckBoxFocus = gcnew CheckBox();
        CheckBoxValidation = gcnew CheckBox();
    }

private:
    void Form1_Load(Object^ sender, EventArgs^ e)
    {
        this->GroupBoxEvents->SuspendLayout();
        this->SuspendLayout();

        lblEvent->Location = Point(232, 12);
        lblEvent->Size = System::Drawing::Size(98, 14);
        lblEvent->AutoSize = true;
        lblEvent->Text = "Generated Events:";

        lblInput->Location = Point(13, 12);
        lblInput->Size = System::Drawing::Size(95, 14);
        lblInput->AutoSize = true;
        lblInput->Text = "User Input Target:";

        TextBoxInput->Location = Point(13, 34);
        TextBoxInput->Size = System::Drawing::Size(200, 200);
        TextBoxInput->AllowDrop = true;
        TextBoxInput->AutoSize = false;
        TextBoxInput->Cursor = Cursors::Cross;
        TextBoxInput->Multiline = true;
        TextBoxInput->TabIndex = 1;

        LinkLabelDrag->AllowDrop = true;
        LinkLabelDrag->AutoSize = true;
        LinkLabelDrag->Location = Point(13, 240);
        LinkLabelDrag->Size = System::Drawing::Size(175, 14);
        LinkLabelDrag->TabIndex = 2;
        LinkLabelDrag->TabStop = true;
        LinkLabelDrag->Text = "Click here to use as a drag source";
        LinkLabelDrag->Links->Add(gcnew LinkLabel::Link(0,
            LinkLabelDrag->Text->Length));

        GroupBoxEvents->Location = Point(13, 281);
        GroupBoxEvents->Size = System::Drawing::Size(200, 302);
        GroupBoxEvents->Text = "Event Filter:";
        GroupBoxEvents->TabStop = true;
        GroupBoxEvents->TabIndex = 3;
        GroupBoxEvents->Controls->Add(CheckBoxMouseEnter);
        GroupBoxEvents->Controls->Add(CheckBoxToggleAll);
        GroupBoxEvents->Controls->Add(CheckBoxMousePoints);
        GroupBoxEvents->Controls->Add(CheckBoxKeyUpDown);
        GroupBoxEvents->Controls->Add(CheckBoxMouseDragOver);
        GroupBoxEvents->Controls->Add(CheckBoxMouseDrag);
        GroupBoxEvents->Controls->Add(CheckBoxValidation);
        GroupBoxEvents->Controls->Add(CheckBoxMouseMove);
        GroupBoxEvents->Controls->Add(CheckBoxFocus);
        GroupBoxEvents->Controls->Add(CheckBoxKeyboard);
        GroupBoxEvents->Controls->Add(CheckBoxMouse);

        CheckBoxToggleAll->AutoSize = true;
        CheckBoxToggleAll->Location = Point(7, 20);
    }

```



```

CheckBoxToggleAll->Size = System::Drawing::Size(122, 17);
CheckBoxToggleAll->TabIndex = 4;
CheckBoxToggleAll->Text = "Toggle All Events";

CheckBoxMouse->AutoSize = true;
CheckBoxMouse->Location = Point(7, 45);
CheckBoxMouse->Size = System::Drawing::Size(137, 17);
CheckBoxMouse->TabIndex = 5;
CheckBoxMouse->Text = "Mouse and Click Events";

CheckBoxMouseEnter->AutoSize = true;
CheckBoxMouseEnter->Location = Point(26, 69);
CheckBoxMouseEnter->Margin =
    System::Windows::Forms::Padding(3, 3, 3, 1);
CheckBoxMouseEnter->Size = System::Drawing::Size(151, 17);
CheckBoxMouseEnter->TabIndex = 6;
CheckBoxMouseEnter->Text = "Mouse Enter/Hover/Leave";

CheckBoxMouseMove->AutoSize = true;
CheckBoxMouseMove->Location = Point(26, 89);
CheckBoxMouseMove->Margin =
    System::Windows::Forms::Padding(3, 2, 3, 3);
CheckBoxMouseMove->Size = System::Drawing::Size(120, 17);
CheckBoxMouseMove->TabIndex = 7;
CheckBoxMouseMove->Text = "Mouse Move Events";

CheckBoxMousePoints->AutoSize = true;
CheckBoxMousePoints->Location = Point(26, 112);
CheckBoxMousePoints->Margin =
    System::Windows::Forms::Padding(3, 3, 3, 1);
CheckBoxMousePoints->Size = System::Drawing::Size(141, 17);
CheckBoxMousePoints->TabIndex = 8;
CheckBoxMousePoints->Text = "Draw Mouse Points";

CheckBoxMouseDrag->AutoSize = true;
CheckBoxMouseDrag->Location = Point(26, 135);
CheckBoxMouseDrag->Margin =
    System::Windows::Forms::Padding(3, 1, 3, 3);
CheckBoxMouseDrag->Size = System::Drawing::Size(151, 17);
CheckBoxMouseDrag->TabIndex = 9;
CheckBoxMouseDrag->Text = "Mouse Drag && Drop Events";

CheckBoxMouseDragOver->AutoSize = true;
CheckBoxMouseDragOver->Location = Point(44, 159);
CheckBoxMouseDragOver->Size = System::Drawing::Size(142, 17);
CheckBoxMouseDragOver->TabIndex = 10;
CheckBoxMouseDragOver->Text = "Mouse Drag Over Events";

CheckBoxKeyboard->AutoSize = true;
CheckBoxKeyboard->Location = Point(8, 184);
CheckBoxKeyboard->Size = System::Drawing::Size(103, 17);
CheckBoxKeyboard->TabIndex = 11;
CheckBoxKeyboard->Text = "Keyboard Events";

CheckBoxKeyUpDown->AutoSize = true;
CheckBoxKeyUpDown->Location = Point(26, 207);
CheckBoxKeyUpDown->Margin =
    System::Windows::Forms::Padding(3, 3, 3, 1);
CheckBoxKeyUpDown->Size = System::Drawing::Size(133, 17);
CheckBoxKeyUpDown->TabIndex = 12;
CheckBoxKeyUpDown->Text = "Key Up && Down Events";

CheckBoxFocus->AutoSize = true;
CheckBoxFocus->Location = Point(8, 233);
CheckBoxFocus->Margin =
    System::Windows::Forms::Padding(3, 2, 3, 3);
CheckBoxFocus->Size = System::Drawing::Size(146, 17);
CheckBoxFocus->TabIndex = 13;
CheckBoxFocus->Text = "Focus && Activation Events";

```

```

CheckBoxValidation->AutoSize = true;
CheckBoxValidation->Location = Point(8, 257);
CheckBoxValidation->Size = System::Drawing::Size(104, 17);
CheckBoxValidation->TabIndex = 14;
CheckBoxValidation->Text = "Validation Events";

TextBoxOutput->Location = Point(232, 34);
TextBoxOutput->Size = System::Drawing::Size(308, 510);
TextBoxOutput->Multiline = true;
TextBoxOutput->CausesValidation = false;
TextBoxOutput->ReadOnly = true;
TextBoxOutput->ScrollBars = ScrollBars::Vertical;
TextBoxOutput->TabIndex = 15;
TextBoxOutput->WordWrap = false;

ButtonClear->Location = Point(232, 560);
ButtonClear->Size = System::Drawing::Size(308, 23);
ButtonClear->TabIndex = 16;
ButtonClear->Text = "Clear Event List";

this->ClientSize = System::Drawing::Size(552, 595);
this->Controls->Add(LinkLabelDrag);
this->Controls->Add(ButtonClear);
this->Controls->Add(GroupBoxEvents);
this->Controls->Add(lblEvent);
this->Controls->Add(lblInput);
this->Controls->Add(TextBoxInput);
this->Controls->Add(TextBoxOutput);
this->Text = "User Input Events";

ButtonClear->Click +=
    gcnew EventHandler(this, &Form1::ButtonClear_Click);
TextBoxInput->KeyDown +=
    gcnew KeyEventHandler(this, &Form1::TextBoxInput_KeyDown);
TextBoxInput->KeyPress +=
    gcnew KeyPressEventHandler(this,
        &Form1::TextBoxInput_KeyPress);
TextBoxInput->KeyUp +=
    gcnew KeyEventHandler(this, &Form1::TextBoxInput_KeyUp);
TextBoxInput->Click +=
    gcnew EventHandler(this, &Form1::TextBoxInput_Click);
TextBoxInput->DoubleClick +=
    gcnew EventHandler(this, &Form1::TextBoxInput_DoubleClick);
TextBoxInput->MouseClick +=
    gcnew MouseEventHandler(this, &Form1::TextBoxInput_MouseClick);
TextBoxInput->MouseDoubleClick +=
    gcnew MouseEventHandler(this,
        &Form1::TextBoxInput_MouseDoubleClick);
TextBoxInput->MouseDown +=
    gcnew MouseEventHandler(this, &Form1::TextBoxInput_MouseDown);
TextBoxInput->MouseUp +=
    gcnew MouseEventHandler(this, &Form1::TextBoxInput_MouseUp);
TextBoxInput->MouseEnter +=
    gcnew EventHandler(this, &Form1::TextBoxInput_MouseEnter);
TextBoxInput->MouseHover +=
    gcnew EventHandler(this, &Form1::TextBoxInput_MouseHover);
TextBoxInput->MouseLeave +=
    gcnew EventHandler(this, &Form1::TextBoxInput_MouseLeave);
TextBoxInput->MouseWheel +=
    gcnew MouseEventHandler(this, &Form1::TextBoxInput_MouseWheel);
TextBoxInput->MouseMove +=
    gcnew MouseEventHandler(this, &Form1::TextBoxInput_MouseMove);
TextBoxInput->MouseCaptureChanged +=
    gcnew EventHandler(this,
        &Form1::TextBoxInput_MouseCaptureChanged);
TextBoxInput->DragEnter +=
    gcnew DragEventHandler(this, &Form1::TextBoxInput_DragEnter);
TextBoxInput->DragDrop +=

```

```

        gcnnew DragEventHandler(this, &Form1::TextBoxInput_DragDrop);
        TextBoxInput->DragOver +=
            gcnnew DragEventHandler(this, &Form1::TextBoxInput_DragOver);
        TextBoxInput->DragLeave +=
            gcnnew EventHandler(this, &Form1::TextBoxInput_DragLeave);
        TextBoxInput->Enter +=
            gcnnew EventHandler(this, &Form1::TextBoxInput_Enter);
        TextBoxInput->Leave +=
            gcnnew EventHandler(this, &Form1::TextBoxInput_Leave);
        TextBoxInput->GotFocus +=
            gcnnew EventHandler(this, &Form1::TextBoxInput_GotFocus);
        TextBoxInput->LostFocus +=
            gcnnew EventHandler(this, &Form1::TextBoxInput_LostFocus);
        TextBoxInput->Validated +=
            gcnnew EventHandler(this, &Form1::TextBoxInput_Validated);
        TextBoxInput->Validating +=
            gcnnew CancelEventHandler(this,
                &Form1::TextBoxInput_Validating);

        LinkLabelDrag->MouseDown +=
            gcnnew MouseEventHandler(this, &Form1::LinkLabelDrag_MouseDown);
        LinkLabelDrag->GiveFeedback +=
            gcnnew GiveFeedbackEventHandler(this,
                &Form1::LinkLabelDrag_GiveFeedback);

        CheckBoxToggleAll->CheckedChanged +=
            gcnnew EventHandler(this,
                &Form1::CheckBoxToggleAll_CheckedChanged);
        CheckBoxMouse->CheckedChanged +=
            gcnnew EventHandler(this, &Form1::CheckBoxMouse_CheckedChanged);
        CheckBoxMouseDrag->CheckedChanged +=
            gcnnew EventHandler(this,
                &Form1::CheckBoxMouseDrag_CheckedChanged);
        CheckBoxMouseEnter->CheckedChanged +=
            gcnnew EventHandler(this,
                &Form1::CheckBoxMouseMove_CheckedChanged);
        CheckBoxMouseMove->CheckedChanged +=
            gcnnew EventHandler(this,
                &Form1::CheckBoxMouseMove_CheckedChanged);
        CheckBoxKeyboard->CheckedChanged +=
            gcnnew EventHandler(this,
                &Form1::CheckBoxKeyboard_CheckedChanged);

        this->GroupBoxEvents->ResumeLayout(false);
        this->GroupBoxEvents->PerformLayout();
        this->ResumeLayout(false);
        this->PerformLayout();
        CheckAllChildCheckBoxes(this, true);
    }

    // Recursively search the form for all contained checkboxes and
    // initially check them
private:
    void CheckAllChildCheckBoxes(Control^ parent, bool value)
    {
        CheckBox^ box;
        for each (Control^ currentControl in parent->Controls)
        {
            if (dynamic_cast<CheckBox^>(currentControl))
            {
                box = (CheckBox^)currentControl;
                box->Checked = value;
            }

            // Recurse if control contains other controls
            if (currentControl->Controls->Count > 0)
            {
                CheckAllChildCheckBoxes(currentControl, value);
            }
        }
    }

```

```

    }
}

// All-purpose method for displaying a line of text in one of the
// text boxes.
private:
void DisplayLine(String^ line)
{
    TextBoxOutput->AppendText(line);
    TextBoxOutput->AppendText(Environment::NewLine);
}

// Click event handler for the button that clears the text box.
private:
void ButtonClear_Click(Object^ sender, EventArgs^ e)
{
    TextBoxOutput->Invalidate();
    TextBoxOutput->Clear();
}

private:
void TextBoxInput_KeyDown(Object^ sender, KeyEventArgs^ e)
{
    if (CheckBoxKeyUpDown->Checked)
    {
        DisplayLine("KeyDown: " + e->KeyData.ToString());
    }
}

private:
void TextBoxInput_KeyUp(Object^ sender, KeyEventArgs^ e)
{
    if (CheckBoxKeyUpDown->Checked)
    {
        DisplayLine("KeyUp: " + e->KeyData.ToString());
    }
}

private:
void TextBoxInput_KeyPress(Object^ sender,
    KeyPressEventArgs^ e)
{
    if (CheckBoxKeyboard->Checked)
    {
        if (Char::IsWhiteSpace(e->KeyChar))
        {
            DisplayLine("KeyPress: WS");
        }
        else
        {
            DisplayLine("KeyPress: " + e->KeyChar.ToString());
        }
    }
}

private:
void TextBoxInput_Click(Object^ sender, EventArgs^ e)
{
    if (CheckBoxMouse->Checked)
    {
        DisplayLine("Click event");
    }
}

private:
void TextBoxInput_DoubleClick(Object^ sender, EventArgs^ e)
{
    if (CheckBoxMouse->Checked)

```

```

        {
            DisplayLine("DoubleClick event");
        }
    }

private:
    void TextBoxInput_MouseClick(Object^ sender, MouseEventArgs^ e)
    {
        if (CheckBoxMouse->Checked)
        {
            DisplayLine("MouseClicked: " + e->Button.ToString() +
                " " + e->Location.ToString());
        }
    }

private:
    void TextBoxInput_MouseDoubleClick(Object^ sender,
        MouseEventArgs^ e)
    {
        if (CheckBoxMouse->Checked)
        {
            DisplayLine("MouseDoubleClick: " + e->Button.ToString() +
                " " + e->Location.ToString());
        }
    }

private:
    void TextBoxInput_MouseDown(Object^ sender,
        MouseEventArgs^ e)
    {
        if (CheckBoxMouse->Checked)
        {
            DisplayLine("MouseDown: " + e->Button.ToString() +
                " " + e->Location.ToString());
        }
    }

private:
    void TextBoxInput_MouseUp(Object^ sender,
        MouseEventArgs^ e)
    {
        if (CheckBoxMouse->Checked)
        {
            DisplayLine("MouseUp: " + e->Button.ToString() +
                " " + e->Location.ToString());
        }

        // The TextBox control was designed to change focus only on
        // the primary click, so force focus to avoid user confusion.
        if (!TextBoxInput->Focused)
        {
            TextBoxInput->Focus();
        }
    }

private:
    void TextBoxInput_MouseEnter(Object^ sender, EventArgs^ e)
    {
        if (CheckBoxMouseEnter->Checked)
        {
            DisplayLine("MouseEnter event");
        }
    }

private:
    void TextBoxInput_MouseHover(Object^ sender, EventArgs^ e)
    {
        if (CheckBoxMouseEnter->Checked)
        {

```

```

        DisplayLine("MouseHover event");
    }
}

private:
void TextBoxInput_MouseLeave(Object^ sender, EventArgs^ e)
{
    if (CheckBoxMouseEnter->Checked)
    {
        DisplayLine("MouseLeave event");
    }
}

private:
void TextBoxInput_MouseWheel(Object^ sender,
    MouseEventArgs^ e)
{
    if (CheckBoxMouse->Checked)
    {
        DisplayLine("MouseWheel: " + e->Delta.ToString() +
            " detents at " + e->Location.ToString());
    }
}

private:
void TextBoxInput_MouseMove(Object^ sender,
    MouseEventArgs^ e)
{
    if (CheckBoxMouseMove->Checked)
    {
        DisplayLine("MouseMove: " + e->Button.ToString() + " " +
            e->Location.ToString());
    }

    if (CheckBoxMousePoints->Checked)
    {
        Graphics^ g = TextBoxInput->CreateGraphics();
        g->FillRectangle(Brushes::Black, e->Location.X,
            e->Location.Y, 1, 1);
        delete g;
    }
}

private:
void TextBoxInput_MouseCaptureChanged(Object^ sender,
    EventArgs^ e)
{
    if (CheckBoxMouseDrag->Checked)
    {
        DisplayLine("MouseCaptureChanged event");
    }
}

private:
void TextBoxInput_DragEnter(Object^ sender, DragEventArgs^ e)
{
    if (CheckBoxMouseDrag->Checked)
    {
        Point^ pt = gcnew Point(e->X, e->Y);
        DisplayLine("DragEnter: " +
            CovertKeyStateToString(e->KeyState)
            + " at " + pt->ToString());
    }
}

private:
void TextBoxInput_DragDrop(Object^ sender, DragEventArgs^ e)
{
    if (CheckBoxMouseDrag->Checked)

```

```

        {
            Point^ pt = gcnew Point(e->X, e->Y);
            DisplayLine("DragDrop: " +
                CovertKeyStateToString(e->KeyState)
                + " at " + pt->ToString());
        }
    }

private:
    void TextBoxInput_DragOver(Object^ sender, DragEventArgs^ e)
    {
        if (CheckBoxMouseDragOver->Checked)
        {
            Point^ pt = gcnew Point(e->X, e->Y);
            DisplayLine("DragOver: " +
                CovertKeyStateToString(e->KeyState)
                + " at " + pt->ToString());
        }

        // Allow if drop data is of type string.
        if (!e->Data->GetDataPresent(String::typeid))
        {
            e->Effect = DragDropEffects::None;
        }
        else
        {
            e->Effect = DragDropEffects::Copy;
        }
    }

private:
    void TextBoxInput_DragLeave(Object^ sender,
        EventArgs^ e)
    {
        if (CheckBoxMouseDrag->Checked)
        {
            DisplayLine("DragLeave event");
        }
    }

private:
    static String^ CovertKeyStateToString(int keyState)
    {
        String^ keyString = "None";

        // Which button was pressed?
        if ((keyState & 1) == 1)
        {
            keyString = "Left";
        }
        else if ((keyState & 2) == 2)
        {
            keyString = "Right";
        }
        else if ((keyState & 16) == 16)
        {
            keyString = "Middle";
        }

        // Are one or more modifier keys also pressed?
        if ((keyState & 4) == 4)
        {
            keyString += "+SHIFT";
        }

        if ((keyState & 8) == 8)
        {
            keyString += "+CTRL";
        }
    }

```

```

        if ((keyState & 32) == 32)
        {
            keyString += "+ALT";
        }

        return keyString;
    }

private:
    void TextBoxInput_Enter(Object^ sender, EventArgs^ e)
    {
        if (CheckBoxFocus->Checked)
        {
            DisplayLine("Enter event");
        }
    }

private:
    void TextBoxInput_Leave(Object^ sender, EventArgs^ e)
    {
        if (CheckBoxFocus->Checked)
        {
            DisplayLine("Leave event");
        }
    }

private:
    void TextBoxInput_GotFocus(Object^ sender, EventArgs^ e)
    {
        if (CheckBoxFocus->Checked)
        {
            DisplayLine("GotFocus event");
        }
    }

private:
    void TextBoxInput_LostFocus(Object^ sender, EventArgs^ e)
    {
        if (CheckBoxFocus->Checked)
        {
            DisplayLine("LostFocus event");
        }
    }

private:
    void TextBoxInput_Validated(Object^ sender, EventArgs^ e)
    {
        if (CheckBoxValidation->Checked)
        {
            DisplayLine("Validated event");
        }
    }

private:
    void TextBoxInput_Validating(
        Object^ sender, CancelEventArgs^ e)
    {
        if (CheckBoxValidation->Checked)
        {
            DisplayLine("Validating event");
        }
    }

private:
    void CheckBoxToggleAll_CheckedChanged(
        Object^ sender, EventArgs^ e)
    {
        if (dynamic cast<CheckBox^>(sender))

```



```

        {
            CheckAllChildCheckBoxes(this, ((CheckBox^)sender)->Checked);
        }
    }

private:
    void CheckBoxMouse_CheckedChanged(
        Object^ sender, EventArgs^ e)
    {
        ConfigureCheckBoxSettings();
    }

private:
    void CheckBoxMouseDrag_CheckedChanged(
        Object^ sender, EventArgs^ e)
    {
        ConfigureCheckBoxSettings();
    }

private:
    void CheckBoxKeyboard_CheckedChanged(
        Object^ sender, EventArgs^ e)
    {
        ConfigureCheckBoxSettings();
    }

private:
    void CheckBoxMouseMove_CheckedChanged(
        Object^ sender, EventArgs^ e)
    {
        ConfigureCheckBoxSettings();
    }

    // Reconcile dependencies between the check box
    // selection choices.
private:
    void ConfigureCheckBoxSettings()
    {
        // CheckBoxMouse is a top-level check box.
        if (!CheckBoxMouse->Checked)
        {
            CheckBoxMouseEnter->Enabled = false;
            CheckBoxMouseMove->Enabled = false;
            CheckBoxMouseDrag->Enabled = false;
            CheckBoxMouseDragOver->Enabled = false;
            CheckBoxMousePoints->Enabled = false;
        }
        else
        {
            CheckBoxMouseEnter->Enabled = true;
            CheckBoxMouseMove->Enabled = true;
            CheckBoxMouseDrag->Enabled = true;
            CheckBoxMousePoints->Enabled = true;

            // Enable children depending on the state of the parent.
            if (!CheckBoxMouseDrag->Checked)
            {
                CheckBoxMouseDragOver->Enabled = false;
            }
            else
            {
                CheckBoxMouseDragOver->Enabled = true;
            }
        }

        if (!CheckBoxKeyboard->Checked)
        {
            CheckBoxKeyUpDown->Enabled = false;
        }
    }
}

```

```

        }
        else
        {
            CheckBoxKeyUpDown->Enabled = true;
        }
    }

private:
    void LinkLabelDrag_MouseDown(Object^ sender, MouseEventArgs^ e)
    {
        String^ data = "Sample Data";
        LinkLabelDrag->DoDragDrop(data, DragDropEffects::All);
    }

private:
    void LinkLabelDrag_GiveFeedback(Object^ sender,
        GiveFeedbackEventArgs^ e)
    {
        if ((e->Effect & DragDropEffects::Copy) ==
            DragDropEffects::Copy)
        {
            LinkLabelDrag->Cursor = Cursors::HSplit;
        }
        else
        {
            LinkLabelDrag->Cursor = Cursors::Default;
        }
    }
};
}

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run(gcnew UserInputWalkthrough::Form1());
}

```

```

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

namespace UserInputWalkthrough
{
    public class Form1 : Form
    {
        Label Label1 = new Label();
        Label Label2 = new Label();
        TextBox TextBoxOutput = new TextBox();
        TextBox TextBoxInput = new TextBox();
        GroupBox GroupBoxEvents = new GroupBox();
        Button ButtonClear = new Button();
        LinkLabel LinkLabelDrag = new LinkLabel();

        CheckBox CheckBoxToggleAll = new CheckBox();
        CheckBox CheckBoxMouse = new CheckBox();
        CheckBox CheckBoxMouseEnter = new CheckBox();
        CheckBox CheckBoxMouseMove = new CheckBox();
        CheckBox CheckBoxMousePoints = new CheckBox();
        CheckBox CheckBoxMouseDrag = new CheckBox();
        CheckBox CheckBoxMouseDragOver = new CheckBox();
        CheckBox CheckBoxKeyboard = new CheckBox();
        CheckBox CheckBoxKeyUpDown = new CheckBox();
        CheckBox CheckBoxFocus = new CheckBox();
        CheckBox CheckBoxValidation = new CheckBox();

        [STAThread]

```

```

[...
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}

public Form1()
    : base()
{
    this.Load += new EventHandler(Form1_Load);
}

private void Form1_Load(object sender, EventArgs e)
{
    this.GroupBoxEvents.SuspendLayout();
    this.SuspendLayout();

    Label1.Location = new Point(232, 12);
    Label1.Size = new Size(98, 14);
    Label1.AutoSize = true;
    Label1.Text = "Generated Events:";

    Label2.Location = new Point(13, 12);
    Label2.Size = new Size(95, 14);
    Label2.AutoSize = true;
    Label2.Text = "User Input Target:";

    TextBoxInput.Location = new Point(13, 34);
    TextBoxInput.Size = new Size(200, 200);
    TextBoxInput.AllowDrop = true;
    TextBoxInput.AutoSize = false;
    TextBoxInput.Cursor = Cursors.Cross;
    TextBoxInput.Multiline = true;
    TextBoxInput.TabIndex = 1;

    LinkLabelDrag.AllowDrop = true;
    LinkLabelDrag.AutoSize = true;
    LinkLabelDrag.Location = new Point(13, 240);
    LinkLabelDrag.Size = new Size(175, 14);
    LinkLabelDrag.TabIndex = 2;
    LinkLabelDrag.TabStop = true;
    LinkLabelDrag.Text = "Click here to use as a drag source";
    LinkLabelDrag.Links.Add(new LinkLabel.Link(0,
        LinkLabelDrag.Text.Length));

    GroupBoxEvents.Location = new Point(13, 281);
    GroupBoxEvents.Size = new Size(200, 302);
    GroupBoxEvents.Text = "Event Filter:";
    GroupBoxEvents.TabStop = true;
    GroupBoxEvents.TabIndex = 3;
    GroupBoxEvents.Controls.Add(CheckBoxMouseEnter);
    GroupBoxEvents.Controls.Add(CheckBoxToggleAll);
    GroupBoxEvents.Controls.Add(CheckBoxMousePoints);
    GroupBoxEvents.Controls.Add(CheckBoxKeyUpDown);
    GroupBoxEvents.Controls.Add(CheckBoxMouseDragOver);
    GroupBoxEvents.Controls.Add(CheckBoxMouseDrag);
    GroupBoxEvents.Controls.Add(CheckBoxValidation);
    GroupBoxEvents.Controls.Add(CheckBoxMouseMove);
    GroupBoxEvents.Controls.Add(CheckBoxFocus);
    GroupBoxEvents.Controls.Add(CheckBoxKeyboard);
    GroupBoxEvents.Controls.Add(CheckBoxMouse);

    CheckBoxToggleAll.AutoSize = true;
    CheckBoxToggleAll.Location = new Point(7, 20);
    CheckBoxToggleAll.Size = new Size(122, 17);
    CheckBoxToggleAll.TabIndex = 4;
    CheckBoxToggleAll.Text = "Toggle All Events";

    CheckBoxMouse.AutoSize = true;

```

```

CheckBoxMouse.AutoSize = true;
CheckBoxMouse.Location = new Point(7, 45);
CheckBoxMouse.Size = new Size(137, 17);
CheckBoxMouse.TabIndex = 5;
CheckBoxMouse.Text = "Mouse and Click Events";

CheckBoxMouseEnter.AutoSize = true;
CheckBoxMouseEnter.Location = new Point(26, 69);
CheckBoxMouseEnter.Margin = new Padding(3, 3, 3, 1);
CheckBoxMouseEnter.Size = new System.Drawing.Size(151, 17);
CheckBoxMouseEnter.TabIndex = 6;
CheckBoxMouseEnter.Text = "Mouse Enter/Hover/Leave";

CheckBoxMouseMove.AutoSize = true;
CheckBoxMouseMove.Location = new Point(26, 89);
CheckBoxMouseMove.Margin = new Padding(3, 2, 3, 3);
CheckBoxMouseMove.Size = new Size(120, 17);
CheckBoxMouseMove.TabIndex = 7;
CheckBoxMouseMove.Text = "Mouse Move Events";

CheckBoxMousePoints.AutoSize = true;
CheckBoxMousePoints.Location = new Point(26, 112);
CheckBoxMousePoints.Margin = new Padding(3, 3, 3, 1);
CheckBoxMousePoints.Size = new Size(141, 17);
CheckBoxMousePoints.TabIndex = 8;
CheckBoxMousePoints.Text = "Draw Mouse Points";

CheckBoxMouseDrag.AutoSize = true;
CheckBoxMouseDrag.Location = new Point(26, 135);
CheckBoxMouseDrag.Margin = new Padding(3, 1, 3, 3);
CheckBoxMouseDrag.Size = new Size(151, 17);
CheckBoxMouseDrag.TabIndex = 9;
CheckBoxMouseDrag.Text = "Mouse Drag && Drop Events";

CheckBoxMouseDragOver.AutoSize = true;
CheckBoxMouseDragOver.Location = new Point(44, 159);
CheckBoxMouseDragOver.Size = new Size(142, 17);
CheckBoxMouseDragOver.TabIndex = 10;
CheckBoxMouseDragOver.Text = "Mouse Drag Over Events";

CheckBoxKeyboard.AutoSize = true;
CheckBoxKeyboard.Location = new Point(8, 184);
CheckBoxKeyboard.Size = new Size(103, 17);
CheckBoxKeyboard.TabIndex = 11;
CheckBoxKeyboard.Text = "Keyboard Events";

CheckBoxKeyUpDown.AutoSize = true;
CheckBoxKeyUpDown.Location = new Point(26, 207);
CheckBoxKeyUpDown.Margin = new Padding(3, 3, 3, 1);
CheckBoxKeyUpDown.Size = new Size(133, 17);
CheckBoxKeyUpDown.TabIndex = 12;
CheckBoxKeyUpDown.Text = "Key Up && Down Events";

CheckBoxFocus.AutoSize = true;
CheckBoxFocus.Location = new Point(8, 233);
CheckBoxFocus.Margin = new Padding(3, 2, 3, 3);
CheckBoxFocus.Size = new Size(146, 17);
CheckBoxFocus.TabIndex = 13;
CheckBoxFocus.Text = "Focus && Activation Events";

CheckBoxValidation.AutoSize = true;
CheckBoxValidation.Location = new Point(8, 257);
CheckBoxValidation.Size = new Size(104, 17);
CheckBoxValidation.TabIndex = 14;
CheckBoxValidation.Text = "Validation Events";

TextBoxOutput.Location = new Point(232, 34);
TextBoxOutput.Size = new Size(308, 510);
TextBoxOutput.Multiline = true;

```

```

textBoxOutput.CausesValidation = false;
textBoxOutput.ReadOnly = true;
textBoxOutput.ScrollBars = ScrollBars.Vertical;
textBoxOutput.TabIndex = 15;
textBoxOutput.WordWrap = false;

buttonClear.Location = new Point(232, 560);
buttonClear.Size = new Size(308, 23);
buttonClear.TabIndex = 16;
buttonClear.Text = "Clear Event List";

this.ClientSize = new Size(552, 595);
this.Controls.Add(linkLabelDrag);
this.Controls.Add(buttonClear);
this.Controls.Add(groupBoxEvents);
this.Controls.Add(label1);
this.Controls.Add(label2);
this.Controls.Add(textBoxInput);
this.Controls.Add(textBoxOutput);
this.Text = "User Input Events";

buttonClear.Click +=
    new EventHandler(buttonClear_Click);
textBoxInput.KeyDown +=
    new KeyEventHandler(textBoxInput_KeyDown);
textBoxInput.KeyPress +=
    new KeyPressEventHandler(textBoxInput_KeyPress);
textBoxInput.KeyUp +=
    new KeyEventHandler(textBoxInput_KeyUp);
textBoxInput.Click +=
    new EventHandler(textBoxInput_Click);
textBoxInput.DoubleClick +=
    new EventHandler(textBoxInput_DoubleClick);
textBoxInput.MouseClick +=
    new MouseEventArgs(textBoxInput_MouseClick);
textBoxInput.MouseDoubleClick +=
    new MouseEventArgs(textBoxInput_MouseDoubleClick);
textBoxInput.MouseDown +=
    new MouseEventArgs(textBoxInput_MouseDown);
textBoxInput.MouseUp +=
    new MouseEventArgs(textBoxInput_MouseUp);
textBoxInput.MouseEnter +=
    new EventHandler(textBoxInput_MouseEnter);
textBoxInput.MouseHover +=
    new EventHandler(textBoxInput_MouseHover);
textBoxInput.MouseLeave +=
    new EventHandler(textBoxInput_MouseLeave);
textBoxInput.MouseWheel +=
    new MouseEventArgs(textBoxInput_MouseWheel);
textBoxInput.MouseMove +=
    new MouseEventArgs(textBoxInput_MouseMove);
textBoxInput.MouseCaptureChanged +=
    new EventHandler(textBoxInput_MouseCaptureChanged);
textBoxInput.DragEnter +=
    new DragEventHandler(textBoxInput_DragEnter);
textBoxInput.DragDrop +=
    new DragEventHandler(textBoxInput_DragDrop);
textBoxInput.DragOver +=
    new DragEventHandler(textBoxInput_DragOver);
textBoxInput.DragLeave +=
    new EventHandler(textBoxInput_DragLeave);
textBoxInput.Enter +=
    new EventHandler(textBoxInput_Enter);
textBoxInput.Leave +=
    new EventHandler(textBoxInput_Leave);
textBoxInput.GotFocus +=
    new EventHandler(textBoxInput_GotFocus);
textBoxInput.LostFocus +=
    new EventHandler(textBoxInput_LostFocus);

```

```

        TextBoxInput.Validated +=
            new EventHandler(TextBoxInput_Validated);
        TextBoxInput.Validating +=
            new CancelEventHandler(TextBoxInput_Validating);

        LinkLabelDrag.MouseDown +=
            new MouseEventHandler(LinkLabelDrag_MouseDown);
        LinkLabelDrag.GiveFeedback +=
            new GiveFeedbackEventHandler(LinkLabelDrag_GiveFeedback);

        CheckBoxToggleAll.CheckedChanged +=
            new EventHandler(CheckBoxToggleAll_CheckedChanged);
        CheckBoxMouse.CheckedChanged +=
            new EventHandler(CheckBoxMouse_CheckedChanged);
        CheckBoxMouseDrag.CheckedChanged +=
            new EventHandler(CheckBoxMouseDrag_CheckedChanged);
        CheckBoxMouseEnter.CheckedChanged +=
            new EventHandler(CheckBoxMouseMove_CheckedChanged);
        CheckBoxMouseMove.CheckedChanged +=
            new EventHandler(CheckBoxMouseMove_CheckedChanged);
        CheckBoxKeyboard.CheckedChanged +=
            new EventHandler(CheckBoxKeyboard_CheckedChanged);

        this.GroupBoxEvents.ResumeLayout(false);
        this.GroupBoxEvents.PerformLayout();
        this.ResumeLayout(false);
        this.PerformLayout();
        CheckAllChildCheckBoxes(this, true);
    }

    // Recursively search the form for all contained checkboxes and
    // initially check them
    private void CheckAllChildCheckBoxes(Control parent, bool value)
    {
        CheckBox box;
        foreach (Control currentControl in parent.Controls)
        {
            if (currentControl is CheckBox)
            {
                {
                    box = (CheckBox)currentControl;
                    box.Checked = value;
                }

                // Recurse if control contains other controls
                if (currentControl.Controls.Count > 0)
                {
                    CheckAllChildCheckBoxes(currentControl, value);
                }
            }
        }
    }

    // All-purpose method for displaying a line of text in one of the
    // text boxes.
    private void DisplayLine(string line)
    {
        TextBoxOutput.AppendText(line);
        TextBoxOutput.AppendText(Environment.NewLine);
    }

    // Click event handler for the button that clears the text box.
    private void ButtonClear_Click(object sender, EventArgs e)
    {
        TextBoxOutput.Invalidate();
        TextBoxOutput.Clear();
    }

    private void TextBoxInput_KeyDown(object sender, KeyEventArgs e)
    {
        if (CheckBoxKeyUpDown.Checked)

```

```

        {
            DisplayLine("KeyDown: " + e.KeyData.ToString());
        }
    }

    private void TextBoxInput_KeyUp(object sender, KeyEventArgs e)
    {
        if (CheckBoxKeyUpDown.Checked)
        {
            DisplayLine("KeyUp: " + e.KeyData.ToString());
        }
    }

    private void TextBoxInput_KeyPress(object sender,
        KeyPressEventArgs e)
    {
        if (CheckBoxKeyboard.Checked)
        {
            if (Char.IsWhiteSpace(e.KeyChar))
            {
                DisplayLine("KeyPress: WS");
            }
            else
            {
                DisplayLine("KeyPress: " + e.KeyChar.ToString());
            }
        }
    }

    private void TextBoxInput_Click(object sender, EventArgs e)
    {
        if (CheckBoxMouse.Checked)
        {
            DisplayLine("Click event");
        }
    }

    private void TextBoxInput_DoubleClick(object sender, EventArgs e)
    {
        if (CheckBoxMouse.Checked)
        {
            DisplayLine("DoubleClick event");
        }
    }

    private void TextBoxInput_MouseClick(object sender, MouseEventArgs e)
    {
        if (CheckBoxMouse.Checked)
        {
            DisplayLine("MouseClicked: " + e.Button.ToString() +
                " " + e.Location.ToString());
        }
    }

    private void TextBoxInput_MouseDoubleClick(object sender,
        MouseEventArgs e)
    {
        if (CheckBoxMouse.Checked)
        {
            DisplayLine("MouseDoubleClick: " + e.Button.ToString() +
                " " + e.Location.ToString());
        }
    }

    private void TextBoxInput_MouseDown(object sender,
        MouseEventArgs e)
    {
        if (CheckBoxMouse.Checked)
        {

```

```

        DisplayLine("MouseDown: " + e.Button.ToString() +
            " " + e.Location.ToString());
    }
}

private void TextBoxInput_MouseUp(object sender,
    MouseEventArgs e)
{
    if (CheckBoxMouse.Checked)
    {
        DisplayLine("MouseUp: " + e.Button.ToString() +
            " " + e.Location.ToString());
    }

    // The TextBox control was designed to change focus only on
    // the primary click, so force focus to avoid user confusion.
    if (!TextBoxInput.Focused)
    {
        TextBoxInput.Focus();
    }
}

private void TextBoxInput_MouseEnter(object sender, EventArgs e)
{
    if (CheckBoxMouseEnter.Checked)
    {
        DisplayLine("MouseEnter event");
    }
}

private void TextBoxInput_MouseHover(object sender, EventArgs e)
{
    if (CheckBoxMouseEnter.Checked)
    {
        DisplayLine("MouseHover event");
    }
}

private void TextBoxInput_MouseLeave(object sender, EventArgs e)
{
    if (CheckBoxMouseEnter.Checked)
    {
        DisplayLine("MouseLeave event");
    }
}

private void TextBoxInput_MouseWheel(object sender,
    MouseEventArgs e)
{
    if (CheckBoxMouse.Checked)
    {
        DisplayLine("MouseWheel: " + e.Delta.ToString() +
            " detents at " + e.Location.ToString());
    }
}

private void TextBoxInput_MouseMove(object sender,
    MouseEventArgs e)
{
    if (CheckBoxMouseMove.Checked)
    {
        DisplayLine("MouseMove: " + e.Button.ToString() + " " +
            e.Location.ToString());
    }

    if (CheckBoxMousePoints.Checked)
    {
        Graphics g = TextBoxInput.CreateGraphics();
        g.FillRectangle(Brushes.Black, e.Location.X,

```



```

        e.Location.Y, 1, 1);
        g.Dispose();
    }
}

private void TextBoxInput_MouseCaptureChanged(object sender,
    EventArgs e)
{
    if (CheckBoxMouseDrag.Checked)
    {
        DisplayLine("MouseCaptureChanged event");
    }
}

private void TextBoxInput_DragEnter(object sender,
    DragEventArgs e)
{
    if (CheckBoxMouseDrag.Checked)
    {
        Point pt = new Point(e.X, e.Y);
        DisplayLine("DragEnter: " +
            CovertKeyStateToString(e.KeyState)
            + " at " + pt.ToString());
    }
}

private void TextBoxInput_DragDrop(object sender,
    DragEventArgs e)
{
    if (CheckBoxMouseDrag.Checked)
    {
        Point pt = new Point(e.X, e.Y);
        DisplayLine("DragDrop: " +
            CovertKeyStateToString(e.KeyState)
            + " at " + pt.ToString());
    }
}

private void TextBoxInput_DragOver(object sender,
    DragEventArgs e)
{
    if (CheckBoxMouseDragOver.Checked)
    {
        Point pt = new Point(e.X, e.Y);
        DisplayLine("DragOver: " +
            CovertKeyStateToString(e.KeyState)
            + " at " + pt.ToString());
    }

    // Allow if drop data is of type string.
    if (!e.Data.GetDataPresent(typeof(String)))
    {
        e.Effect = DragDropEffects.None;
    }
    else
    {
        e.Effect = DragDropEffects.Copy;
    }
}

private void TextBoxInput_DragLeave(object sender,
    EventArgs e)
{
    if (CheckBoxMouseDrag.Checked)
    {
        DisplayLine("DragLeave event");
    }
}

```

```

private string CovertKeyStateToString(int keyState)
{
    string keyString = "None";

    // Which button was pressed?
    if ((keyState & 1) == 1)
    {
        keyString = "Left";
    }
    else if ((keyState & 2) == 2)
    {
        keyString = "Right";
    }
    else if ((keyState & 16) == 16)
    {
        keyString = "Middle";
    }

    // Are one or more modifier keys also pressed?
    if ((keyState & 4) == 4)
    {
        keyString += "+SHIFT";
    }

    if ((keyState & 8) == 8)
    {
        keyString += "+CTRL";
    }

    if ((keyState & 32) == 32)
    {
        keyString += "+ALT";
    }

    return keyString;
}

private void TextBoxInput_Enter(object sender, EventArgs e)
{
    if (CheckBoxFocus.Checked)
    {
        DisplayLine("Enter event");
    }
}

private void TextBoxInput_Leave(object sender, EventArgs e)
{
    if (CheckBoxFocus.Checked)
    {
        DisplayLine("Leave event");
    }
}

private void TextBoxInput_GotFocus(object sender, EventArgs e)
{
    if (CheckBoxFocus.Checked)
    {
        DisplayLine("GotFocus event");
    }
}

private void TextBoxInput_LostFocus(object sender, EventArgs e)
{
    if (CheckBoxFocus.Checked)
    {
        DisplayLine("LostFocus event");
    }
}

```

```

private void TextBoxInput_Validated(object sender, EventArgs e)
{
    if (CheckBoxValidation.Checked)
    {
        DisplayLine("Validated event");
    }
}

private void TextBoxInput_Validating(
    object sender, CancelEventArgs e)
{
    if (CheckBoxValidation.Checked)
    {
        DisplayLine("Validating event");
    }
}

private void CheckBoxToggleAll_CheckedChanged(
    object sender, EventArgs e)
{
    if (sender is CheckBox)
    {
        CheckAllChildCheckBoxes(this, ((CheckBox)sender).Checked);
    }
}

private void CheckBoxMouse_CheckedChanged(
    object sender, EventArgs e)
{
    ConfigureCheckBoxSettings();
}

private void CheckBoxMouseDrag_CheckedChanged(
    object sender, EventArgs e)
{
    ConfigureCheckBoxSettings();
}

private void CheckBoxKeyboard_CheckedChanged(
    object sender, EventArgs e)
{
    ConfigureCheckBoxSettings();
}

private void CheckBoxMouseMove_CheckedChanged(
    object sender, EventArgs e)
{
    ConfigureCheckBoxSettings();
}

// Reconcile dependencies between the check box
// selection choices.
private void ConfigureCheckBoxSettings()
{
    // CheckBoxMouse is a top-level check box.
    if (!CheckBoxMouse.Checked)
    {
        CheckBoxMouseEnter.Enabled = false;
        CheckBoxMouseMove.Enabled = false;
        CheckBoxMouseDrag.Enabled = false;
        CheckBoxMouseDragOver.Enabled = false;
        CheckBoxMousePoints.Enabled = false;
    }
    else
    {
        CheckBoxMouseEnter.Enabled = true;
        CheckBoxMouseMove.Enabled = true;
        CheckBoxMouseDrag.Enabled = true;
    }
}

```

```

        CheckBoxMousePoints.Enabled = true;

        // Enable children depending on the state of the parent.
        if (!CheckBoxMouseDrag.Checked)
        {
            CheckBoxMouseDragOver.Enabled = false;
        }
        else
        {
            CheckBoxMouseDragOver.Enabled = true;
        }
    }

    if (!CheckBoxKeyboard.Checked)
    {
        CheckBoxKeyUpDown.Enabled = false;
    }
    else
    {
        CheckBoxKeyUpDown.Enabled = true;
    }
}

private void LinkLabelDrag_MouseDown(object sender,
    MouseEventArgs e)
{
    string data = "Sample Data";
    LinkLabelDrag.DoDragDrop(data, DragDropEffects.All);
}

private void LinkLabelDrag_GiveFeedback(object sender,
    GiveFeedbackEventArgs e)
{
    if ((e.Effect & DragDropEffects.Copy) ==
        DragDropEffects.Copy)
    {
        LinkLabelDrag.Cursor = Cursors.HSplit;
    }
    else
    {
        LinkLabelDrag.Cursor = Cursors.Default;
    }
}
}
}

```

```

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

Namespace UserInputWalkthrough

    Public Class Form1
        Inherits Form

        Dim Label1 As New Label
        Dim Label2 As New Label
        Dim TextBoxOutput As New TextBox
        Dim WithEvents TextBoxInput As New TextBox
        Dim WithEvents GroupBoxEvents As New GroupBox
        Dim WithEvents ButtonClear As New Button
        Dim WithEvents LinkLabelDrag As New LinkLabel

        Dim WithEvents CheckBoxToggleAll As New CheckBox
        Dim WithEvents CheckBoxMouse As New CheckBox
        Dim WithEvents CheckBoxMouseEnter As New CheckBox
        Dim WithEvents CheckBoxMouseMove As New CheckBox
        Dim WithEvents CheckBoxMousePoints As New CheckBox
    End Class
End Namespace

```

```

Dim WithEvents CheckBoxMousePoints As New CheckBox
Dim WithEvents CheckBoxMouseDrag As New CheckBox
Dim WithEvents CheckBoxMouseDragOver As New CheckBox
Dim WithEvents CheckBoxKeyboard As New CheckBox
Dim WithEvents CheckBoxKeyUpDown As New CheckBox
Dim WithEvents CheckBoxFocus As New CheckBox
Dim WithEvents CheckBoxValidation As New CheckBox

<STAThread(>> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

Public Sub New()
    MyBase.New()
End Sub

Private Sub Form1_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    Me.GroupBoxEvents.SuspendLayout()
    Me.SuspendLayout()

    Label1.Location = New Point(232, 12)
    Label1.Size = New Size(98, 14)
    Label1.AutoSize = True
    Label1.Text = "Generated Events:"

    Label2.Location = New Point(13, 12)
    Label2.Size = New Size(95, 14)
    Label2.AutoSize = True
    Label2.Text = "User Input Target:"

    TextBoxInput.Location = New Point(13, 34)
    TextBoxInput.Size = New Size(200, 200)
    TextBoxInput.AllowDrop = True
    TextBoxInput.AutoSize = False
    TextBoxInput.Cursor = Cursors.Cross
    TextBoxInput.Multiline = True
    TextBoxInput.TabIndex = 1

    LinkLabelDrag.AllowDrop = True
    LinkLabelDrag.AutoSize = True
    LinkLabelDrag.Location = New Point(13, 240)
    LinkLabelDrag.Size = New Size(175, 14)
    LinkLabelDrag.TabIndex = 2
    LinkLabelDrag.TabStop = True
    LinkLabelDrag.Text = "Click here to use as a drag source"
    LinkLabelDrag.Links.Add(New LinkLabel.Link(0, _
        LinkLabelDrag.Text.Length))

    GroupBoxEvents.Location = New Point(13, 281)
    GroupBoxEvents.Size = New Size(200, 302)
    GroupBoxEvents.TabIndex = 3
    GroupBoxEvents.TabStop = False
    GroupBoxEvents.Text = "Event Filter:"
    GroupBoxEvents.Controls.Add(CheckBoxMouseEnter)
    GroupBoxEvents.Controls.Add(CheckBoxToggleAll)
    GroupBoxEvents.Controls.Add(CheckBoxMousePoints)
    GroupBoxEvents.Controls.Add(CheckBoxKeyUpDown)
    GroupBoxEvents.Controls.Add(CheckBoxMouseDragOver)
    GroupBoxEvents.Controls.Add(CheckBoxMouseDrag)
    GroupBoxEvents.Controls.Add(CheckBoxValidation)
    GroupBoxEvents.Controls.Add(CheckBoxMouseMove)
    GroupBoxEvents.Controls.Add(CheckBoxFocus)
    GroupBoxEvents.Controls.Add(CheckBoxKeyboard)
    GroupBoxEvents.Controls.Add(CheckBoxMouse)

```

```
CheckBoxToggleAll.AutoSize = True
CheckBoxToggleAll.Location = New Point(7, 20)
CheckBoxToggleAll.Size = New Size(122, 17)
CheckBoxToggleAll.TabIndex = 4
CheckBoxToggleAll.Text = "Toggle All Events"

CheckBoxMouse.AutoSize = True
CheckBoxMouse.Location = New Point(7, 45)
CheckBoxMouse.Size = New Size(137, 17)
CheckBoxMouse.TabIndex = 5
CheckBoxMouse.Text = "Mouse and Click Events"

CheckBoxMouseEnter.AutoSize = True
CheckBoxMouseEnter.Location = New Point(26, 69)
CheckBoxMouseEnter.Margin = New Padding(3, 3, 3, 1)
CheckBoxMouseEnter.Size = New System.Drawing.Size(151, 17)
CheckBoxMouseEnter.TabIndex = 6
CheckBoxMouseEnter.Text = "Mouse Enter/Hover/Leave"

CheckBoxMouseMove.AutoSize = True
CheckBoxMouseMove.Location = New Point(26, 89)
CheckBoxMouseMove.Margin = New Padding(3, 2, 3, 3)
CheckBoxMouseMove.Size = New Size(120, 17)
CheckBoxMouseMove.TabIndex = 7
CheckBoxMouseMove.Text = "Mouse Move Events"

CheckBoxMousePoints.AutoSize = True
CheckBoxMousePoints.Location = New Point(26, 112)
CheckBoxMousePoints.Margin = New Padding(3, 3, 3, 1)
CheckBoxMousePoints.Size = New Size(141, 17)
CheckBoxMousePoints.TabIndex = 8
CheckBoxMousePoints.Text = "Draw Mouse Points"

CheckBoxMouseDrag.AutoSize = True
CheckBoxMouseDrag.Location = New Point(26, 135)
CheckBoxMouseDrag.Margin = New Padding(3, 1, 3, 3)
CheckBoxMouseDrag.Size = New Size(151, 17)
CheckBoxMouseDrag.TabIndex = 9
CheckBoxMouseDrag.Text = "Mouse Drag && Drop Events"

CheckBoxMouseDragOver.AutoSize = True
CheckBoxMouseDragOver.Location = New Point(44, 159)
CheckBoxMouseDragOver.Size = New Size(142, 17)
CheckBoxMouseDragOver.TabIndex = 10
CheckBoxMouseDragOver.Text = "Mouse Drag Over Events"

CheckBoxKeyboard.AutoSize = True
CheckBoxKeyboard.Location = New Point(8, 184)
CheckBoxKeyboard.Size = New Size(103, 17)
CheckBoxKeyboard.TabIndex = 11
CheckBoxKeyboard.Text = "Keyboard Events"

CheckBoxKeyUpDown.AutoSize = True
CheckBoxKeyUpDown.Location = New Point(26, 207)
CheckBoxKeyUpDown.Margin = New Padding(3, 3, 3, 1)
CheckBoxKeyUpDown.Size = New Size(133, 17)
CheckBoxKeyUpDown.TabIndex = 12
CheckBoxKeyUpDown.Text = "Key Up && Down Events"

CheckBoxFocus.AutoSize = True
CheckBoxFocus.Location = New Point(8, 233)
CheckBoxFocus.Margin = New Padding(3, 2, 3, 3)
CheckBoxFocus.Size = New Size(146, 17)
CheckBoxFocus.TabIndex = 13
CheckBoxFocus.Text = "Focus && Activation Events"

CheckBoxValidation.AutoSize = True
CheckBoxValidation.Location = New Point(8, 257)
CheckBoxValidation.Size = New Size(104, 17)
```

```

CheckBoxValidation.TabIndex = 14
CheckBoxValidation.Text = "Validation Events"

TextBoxOutput.Location = New Point(232, 34)
TextBoxOutput.Size = New Size(308, 510)
TextBoxOutput.Multiline = True
TextBoxOutput.CausesValidation = False
TextBoxOutput.ReadOnly = True
TextBoxOutput.ScrollBars = ScrollBars.Vertical
TextBoxOutput.TabIndex = 15
TextBoxOutput.WordWrap = False

ButtonClear.Location = New Point(232, 560)
ButtonClear.Size = New Size(308, 23)
ButtonClear.TabIndex = 16
ButtonClear.Text = "Clear Event List"

Me.ClientSize = New Size(552, 595)
Me.Controls.Add(LinkLabelDrag)
Me.Controls.Add(ButtonClear)
Me.Controls.Add(GroupBoxEvents)
Me.Controls.Add(Label1)
Me.Controls.Add(Label2)
Me.Controls.Add(TextBoxInput)
Me.Controls.Add(TextBoxOutput)
Me.Text = "User Input Events"

Me.GroupBoxEvents.ResumeLayout(False)
Me.GroupBoxEvents.PerformLayout()
Me.ResumeLayout(False)
Me.PerformLayout()
CheckAllChildCheckBoxes(Me, True)
End Sub

' Recursively search the form for all contained checkboxes and
' initially check them
Private Sub CheckAllChildCheckBoxes(ByRef parent As Control, _
    ByVal value As Boolean)

    Dim currentControl As Control
    Dim box As CheckBox
    For Each currentControl In parent.Controls
        box = TryCast(currentControl, CheckBox)
        If box IsNot Nothing Then
            box.Checked = value
        End If

        'Recurse if control contains other controls
        If currentControl.Controls.Count > 0 Then
            CheckAllChildCheckBoxes(currentControl, value)
        End If
    Next
End Sub

' All-purpose method for displaying a line of text in one of the
' text boxes.
Private Sub DisplayLine(ByRef line As String)

    TextBoxOutput.AppendText(line)
    TextBoxOutput.AppendText(ControlChars.NewLine)
End Sub

' Click event handler for the button that clears the text box.
Private Sub ButtonClear_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles ButtonClear.Click

    TextBoxOutput.Invalidate()
    TextBoxOutput.Clear()
End Sub

```

```

Private Sub TextBoxInput_KeyDown(ByVal sender As Object, _
    ByVal e As KeyEventArgs) Handles TextBoxInput.KeyDown

    If CheckBoxKeyUpDown.Checked = True Then
        DisplayLine("KeyDown: " + e.KeyData.ToString())
    End If
End Sub

Private Sub TextBoxInput_KeyUp(ByVal sender As Object, _
    ByVal e As KeyEventArgs) Handles TextBoxInput.KeyUp

    If CheckBoxKeyUpDown.Checked = True Then
        DisplayLine("KeyUp: " + e.KeyData.ToString())
    End If
End Sub

Private Sub TextBoxInput_KeyPress(ByVal sender As Object, _
    ByVal e As KeyPressEventArgs) Handles TextBoxInput.KeyPress

    If CheckBoxKeyboard.Checked = True Then

        If Char.IsWhiteSpace(e.KeyChar) Then
            Dim keyVal As Integer
            keyVal = AscW(e.KeyChar)
            DisplayLine("KeyPress: WS-" + keyVal.ToString())
        Else
            DisplayLine("KeyPress: " + e.KeyChar.ToString())
        End If
    End If
End Sub

Private Sub TextBoxInput_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.Click

    If CheckBoxMouse.Checked = True Then
        DisplayLine("Click event")
    End If
End Sub

Private Sub TextBoxInput_DoubleClick(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.DoubleClick

    If CheckBoxMouse.Checked = True Then
        DisplayLine("DoubleClick event")
    End If
End Sub

Private Sub TextBoxInput_MouseClick(ByVal sender As Object, _
    ByVal e As MouseEventArgs) Handles TextBoxInput.MouseClick

    If CheckBoxMouse.Checked = True Then
        DisplayLine("MouseClicked: " + e.Button.ToString() + _
            " " + e.Location.ToString())
    End If
End Sub

Private Sub TextBoxInput_MouseDoubleClick(ByVal sender As Object, _
    ByVal e As MouseEventArgs) Handles TextBoxInput.MouseDoubleClick

    If CheckBoxMouse.Checked = True Then
        DisplayLine("MouseDoubleClick: " + e.Button.ToString() + _
            " " + e.Location.ToString())
    End If
End Sub

Private Sub TextBoxInput_MouseDown(ByVal sender As System.Object, _
    ByVal e As MouseEventArgs) Handles TextBoxInput.MouseDown

```



```

        If CheckBoxMouse.Checked = True Then
            DisplayLine("MouseDown: " + e.Button.ToString() + _
                " " + e.Location.ToString())
        End If
    End Sub

Private Sub TextBoxInput_MouseUp(ByVal sender As Object, _
    ByVal e As MouseEventArgs) Handles TextBoxInput.MouseUp

    If CheckBoxMouse.Checked = True Then
        DisplayLine("MouseUp: " + e.Button.ToString() + _
            " " + e.Location.ToString())
    End If

    ' The TextBox control was designed to change focus only on
    ' the primary click, so force focus to avoid user confusion.
    If TextBoxInput.Focused = False Then
        TextBoxInput.Focus()
    End If
End Sub

Private Sub TextBoxInput_MouseEnter(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.MouseEnter

    If CheckBoxMouseEnter.Checked = True Then
        DisplayLine("MouseEnter event")
    End If
End Sub

Private Sub TextBoxInput_MouseHover(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.MouseHover

    If CheckBoxMouseEnter.Checked = True Then
        DisplayLine("MouseHover event")
    End If
End Sub

Private Sub TextBoxInput_MouseLeave(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.MouseLeave

    If CheckBoxMouseEnter.Checked = True Then
        DisplayLine("MouseLeave event")
    End If
End Sub

Private Sub TextBoxInput_MouseWheel(ByVal sender As Object, _
    ByVal e As MouseEventArgs) Handles TextBoxInput.MouseWheel

    If CheckBoxMouse.Checked = True Then
        DisplayLine("MouseWheel: " + e.Delta.ToString() + _
            " detents at " + e.Location.ToString())
    End If
End Sub

Private Sub TextBoxInput_MouseMove(ByVal sender As Object, _
    ByVal e As MouseEventArgs) Handles TextBoxInput.MouseMove

    If CheckBoxMouseMove.Checked = True Then
        DisplayLine("MouseMove: " + e.Button.ToString() + " " + _
            e.Location.ToString())
    End If

    If CheckBoxMousePoints.Checked = True Then
        Dim g As Graphics = TextBoxInput.CreateGraphics()
        g.FillRectangle(Brushes.Black, e.Location.X, _
            e.Location.Y, 1, 1)
        g.Dispose()
    End If
End Sub

```

End Sub

```
Private Sub TextBoxInput_MouseCaptureChanged( _  
    ByVal sender As Object, ByVal e As EventArgs) _  
    Handles TextBoxInput.MouseCaptureChanged  
  
    If CheckBoxMouseDrag.Checked = True Then  
        DisplayLine("MouseCaptureChanged event")  
    End If  
End Sub
```

```
Private Sub TextBoxInput_DragEnter(ByVal sender As Object, _  
    ByVal e As DragEventArgs) Handles TextBoxInput.DragEnter  
  
    Dim pt As Point  
    If CheckBoxMouseDrag.Checked = True Then  
        pt = New Point(e.X, e.Y)  
        DisplayLine("DragEnter: " + _  
            CovertKeyStateToString(e.KeyState) _  
            + " at " + pt.ToString())  
    End If  
End Sub
```

```
Private Sub TextBoxInput_DragDrop(ByVal sender As Object, _  
    ByVal e As DragEventArgs) Handles TextBoxInput.DragDrop  
  
    If CheckBoxMouseDrag.Checked = True Then  
        Dim pt As Point  
        pt = New Point(e.X, e.Y)  
  
        DisplayLine("DragDrop: " + _  
            CovertKeyStateToString(e.KeyState) _  
            + " at " + pt.ToString())  
    End If  
End Sub
```

```
Private Sub TextBoxInput_DragOver(ByVal sender As Object, _  
    ByVal e As DragEventArgs) Handles TextBoxInput.DragOver  
  
    If CheckBoxMouseDragOver.Checked = True Then  
        Dim pt As Point  
        pt = New Point(e.X, e.Y)  
        DisplayLine("DragOver: " + _  
            CovertKeyStateToString(e.KeyState) _  
            + " at " + pt.ToString())  
    End If  
  
    ' Allow if drop data is of type string.  
    If Not (e.Data.GetDataPresent(GetType(String))) Then  
        e.Effect = DragDropEffects.None  
    Else  
        e.Effect = DragDropEffects.Copy  
    End If  
End Sub
```

```
Private Sub TextBoxInput_DragLeave(ByVal sender As Object, _  
    ByVal e As EventArgs) Handles TextBoxInput.DragLeave  
  
    If CheckBoxMouseDrag.Checked = True Then  
        DisplayLine("DragLeave event")  
    End If  
End Sub
```

```
Private Function CovertKeyStateToString(ByVal keyState As Integer) _  
    As String  
  
    Dim keyString As String = "None"  
  
    ' Which button was pressed?  
    If (keyState And 1) = 1 Then
```

```

    If ((keyState And 1) = 1) Then
        keyString = "Left"
    ElseIf ((keyState And 2) = 2) Then
        keyString = "Right"
    ElseIf ((keyState And 16) = 16) Then
        keyString = "Middle"
    End If

    ' Are one or more modifier keys also pressed?
    If ((keyState And 4) = 4) Then
        keyString += "+SHIFT"
    End If
    If ((keyState And 8) = 8) Then
        keyString += "+CTRL"
    End If
    If ((keyState And 32) = 32) Then
        keyString += "+ALT"
    End If

    Return keyString
End Function

Private Sub TextBoxInput_Enter(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.Enter

    If CheckBoxFocus.Checked = True Then
        DisplayLine("Enter event")
    End If
End Sub

Private Sub TextBoxInput_Leave(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.Leave

    If CheckBoxFocus.Checked = True Then
        DisplayLine("Leave event")
    End If
End Sub

Private Sub TextBoxInput_GotFocus(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.GotFocus

    If CheckBoxFocus.Checked = True Then
        DisplayLine("GotFocus event")
    End If
End Sub

Private Sub TextBoxInput_LostFocus(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.LostFocus

    If CheckBoxFocus.Checked = True Then
        DisplayLine("LostFocus event")
    End If
End Sub

Private Sub TextBoxInput_Validated(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.Validated

    If CheckBoxValidation.Checked = True Then
        DisplayLine("Validated event")
    End If
End Sub

Private Sub TextBoxInput_Validating(ByVal sender As Object, _
    ByVal e As CancelEventArgs) _
    Handles TextBoxInput.Validating

    If CheckBoxValidation.Checked = True Then
        DisplayLine("Validating event")
    End If

```

```

End Sub

Private Sub CheckBoxToggleAll_CheckedChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
    Handles CheckBoxToggleAll.CheckedChanged

    Dim cb As CheckBox
    cb = TryCast(sender, CheckBox)
    If cb IsNot Nothing Then
        CheckAllChildCheckBoxes(Me, cb.Checked)
    End If
End Sub

Private Sub CheckBoxMouse_CheckedChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
    Handles CheckBoxMouse.CheckedChanged

    ConfigureCheckBoxSettings()
End Sub

Private Sub CheckBoxMouseDown_CheckedChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
    Handles CheckBoxMouseDown.CheckedChanged

    ConfigureCheckBoxSettings()
End Sub

Private Sub CheckBoxKeyboard_CheckedChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
    Handles CheckBoxKeyboard.CheckedChanged

    ConfigureCheckBoxSettings()
End Sub

Private Sub CheckBoxMouseMove_CheckedChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
    Handles CheckBoxMouseEnter.CheckedChanged, _
        CheckBoxMouseMove.CheckedChanged

    ConfigureCheckBoxSettings()
End Sub

' Reconcile dependencies between the check box
' selection choices.
Private Sub ConfigureCheckBoxSettings()

    ' CheckBoxMouse is a top-level check box.
    If CheckBoxMouse.Checked = False Then
        CheckBoxMouseEnter.Enabled = False
        CheckBoxMouseMove.Enabled = False
        CheckBoxMouseDown.Enabled = False
        CheckBoxMouseDragOver.Enabled = False
        CheckBoxMousePoints.Enabled = False
    Else
        CheckBoxMouseEnter.Enabled = True
        CheckBoxMouseMove.Enabled = True
        CheckBoxMouseDown.Enabled = True
        CheckBoxMousePoints.Enabled = True

        ' Enable children depending on the state of the parent.
        If CheckBoxMouseDrag.Checked = False Then
            CheckBoxMouseDragOver.Enabled = False
        Else
            CheckBoxMouseDragOver.Enabled = True
        End If
    End If

    If CheckBoxKeyboard.Checked = False Then
        CheckBoxKeyUpDown.Enabled = False
    End If
End Sub

```

```

        Else
            CheckBoxKeyUpDown.Enabled = True
        End If
    End Sub

    Private Sub LinkLabelDrag_MouseDown(ByVal sender As Object, _
        ByVal e As MouseEventArgs) Handles LinkLabelDrag.MouseDown

        Dim data As String = "Sample Data"
        LinkLabelDrag.DoDragDrop(data, DragDropEffects.All)
    End Sub

    Private Sub LinkLabelDrag_GiveFeedback(ByVal sender As Object, _
        ByVal e As GiveFeedbackEventArgs) _
        Handles LinkLabelDrag.GiveFeedback

        If ((e.Effect And DragDropEffects.Copy) = _
            DragDropEffects.Copy) Then
            LinkLabelDrag.Cursor = Cursors.HSplit
        Else
            LinkLabelDrag.Cursor = Cursors.Default
        End If
    End Sub

End Class
End Namespace

```

Compilation du code

Cet exemple nécessite :

- des références aux assemblés System, System.Drawing et System.Windows.Forms.

Voir aussi

- [Entrée d'utilisateur dans Windows Forms](#)

Validation des entrées d'utilisateur dans les Windows Forms

13 minutes to read • [Edit Online](#)

Lorsque les utilisateurs entrent des données dans votre application, vous pouvez vérifier que les données sont valides avant que votre application ne l'utilise. Vous pouvez exiger que certains champs de texte ne soient pas de longueur nulle, qu'un champ soit mis en forme comme un numéro de téléphone ou un autre type de données bien formées, ou qu'une chaîne ne contienne pas de caractères non sécurisés susceptibles d'être utilisés pour compromettre la sécurité d'une base de données. Windows Forms offre plusieurs méthodes pour valider l'entrée dans votre application.

Validation avec le contrôle MaskedTextBox

Si vous devez obliger les utilisateurs à entrer des données dans un format bien défini, par exemple un numéro de téléphone ou un numéro de référence, vous pouvez le faire rapidement et avec un minimum de code à l'aide du contrôle [MaskedTextBox](#). Un *masque* est une chaîne composée de caractères issus d'un langage de masquage qui spécifie les caractères qui peuvent être entrés à une position donnée dans la zone de texte. Le contrôle affiche un ensemble d'invites à l'utilisateur. Si l'utilisateur tape une entrée incorrecte, par exemple, l'utilisateur tape une lettre quand un chiffre est requis, le contrôle rejette automatiquement l'entrée.

Le langage de masquage utilisé par [MaskedTextBox](#) est très flexible. Elle vous permet de spécifier des caractères obligatoires, des caractères facultatifs, des caractères littéraux, tels que des traits d'Union et des parenthèses, des caractères monétaires et des séparateurs de date. Le contrôle fonctionne également bien lorsqu'il est lié à une source de données. L'événement [Format](#) sur une liaison de données peut être utilisé pour reformater les données entrantes en fonction du masque, et l'événement [Parse](#) peut être utilisé pour reformater les données sortantes en fonction des spécifications du champ de données.

Pour plus d'informations, consultez [MaskedTextBox Control](#).

Validation pilotée par les événements

Si vous souhaitez un contrôle total de la validation par programmation ou si vous devez effectuer des contrôles de validation complexes, vous devez utiliser les événements de validation intégrés à la plupart des contrôles Windows Forms. Chaque contrôle qui accepte les entrées utilisateur de forme libre a un événement [Validating](#) qui se produit chaque fois que le contrôle requiert la validation de données. Dans la [Validating](#) méthode de gestion des événements, vous pouvez valider l'entrée utilisateur de plusieurs façons. Par exemple, si vous avez une zone de texte qui doit contenir un code postal, vous pouvez effectuer la validation des manières suivantes :

- Si le code postal doit appartenir à un groupe spécifique de codes postaux, vous pouvez effectuer une comparaison de chaînes sur l'entrée pour valider les données entrées par l'utilisateur. Par exemple, si le code postal doit se trouver dans le jeu {10001, 10002, 10003}, vous pouvez utiliser une comparaison de chaînes pour valider les données.
- Si le code postal doit être dans un formulaire spécifique, vous pouvez utiliser des expressions régulières pour valider les données entrées par l'utilisateur. Par exemple, pour valider le formulaire `#####` ou `#####-####`, vous pouvez utiliser l'expression régulière `^(\d{5})(-\d{4})?$`. Pour valider le `A#A #A#` de formulaire, vous pouvez utiliser l'expression régulière `[A-Z]\d[A-Z] \d[A-Z]\d`. Pour plus d'informations sur les expressions régulières, consultez [.NET Framework des expressions régulières](#) et des exemples d'expressions [régulières](#).

- Si le code postal doit être un code postal États-Unis valide, vous pouvez appeler un service Web de code postal pour valider les données entrées par l'utilisateur.

L'événement [Validating](#) est fourni à un objet de type [CancelEventArgs](#). Si vous déterminez que les données du contrôle ne sont pas valides, vous pouvez annuler l'événement [Validating](#) en affectant à la propriété [Cancel](#) de cet objet la valeur `true`. Si vous ne définissez pas la propriété [Cancel](#), Windows Forms supposera que la validation a réussi pour ce contrôle et déclenchera l'événement [Validated](#).

Pour obtenir un exemple de code qui valide une adresse de messagerie dans une [TextBox](#), consultez [Validating](#).

Liaison de données et validation pilotée par les événements

La validation est très utile lorsque vous avez lié vos contrôles à une source de données, telle qu'une table de base de données. À l'aide de la validation, vous pouvez vous assurer que les données de votre contrôle respectent le format requis par la source de données, et qu'il ne contient pas de caractères spéciaux tels que des guillemets et des barres obliques inverses qui peuvent ne pas être sécurisés.

Lorsque vous utilisez la liaison de données, les données de votre contrôle sont synchronisées avec la source de données lors de l'exécution de l'événement [Validating](#). Si vous annulez l'événement [Validating](#), les données ne seront pas synchronisées avec la source de données.

IMPORTANT

Si vous avez une validation personnalisée qui a lieu après l'événement [Validating](#), elle n'affecte pas la liaison de données. Par exemple, si vous avez du code dans un événement [Validated](#) qui tente d'annuler la liaison de données, la liaison de données se produira encore. Dans ce cas, pour effectuer la validation dans l'événement [Validated](#), modifiez la propriété **mode de mise à jour** de la source de données du contrôle (**sous (DataBindings) \ (avancé)**) de **OnValidation** à **Never**, puis ajoutez le contrôle `.DataBindings[" <YOURFIELD> ".WriteValue()` à votre code de validation.

Validation implicite et explicite

Quand les données d'un contrôle sont-elles validées ? C'est à vous, le développeur. Vous pouvez utiliser une validation implicite ou explicite, selon les besoins de votre application.

Validation implicite

L'approche de validation implicite valide les données au fur et à mesure que l'utilisateur les entre. Vous pouvez valider les données au fur et à mesure que les données sont entrées dans un contrôle en lisant les touches au fur et à mesure qu'elles sont enfoncées, ou plus souvent chaque fois que l'utilisateur éloigne le focus d'entrée d'un contrôle et passe au suivant. Cette approche est utile lorsque vous souhaitez fournir à l'utilisateur des commentaires immédiats sur les données au fur et à mesure de leur travail.

Si vous souhaitez utiliser la validation implicite pour un contrôle, vous devez définir la propriété [AutoValidate](#) de ce contrôle sur [EnablePreventFocusChange](#) ou [EnableAllowFocusChange](#). Si vous annulez l'événement [Validating](#), le comportement du contrôle est déterminé par la valeur que vous avez assignée à [AutoValidate](#). Si vous avez assigné [EnablePreventFocusChange](#), l'annulation de l'événement entraîne la non-exécution de l'événement [Validated](#). Le focus d'entrée reste sur le contrôle actuel jusqu'à ce que l'utilisateur modifie les données en une entrée valide. Si vous avez assigné [EnableAllowFocusChange](#), l'événement [Validated](#) ne se produit pas lorsque vous annulez l'événement, mais le focus reste toujours sur le contrôle suivant.

L'affectation de [Disable](#) à la propriété [AutoValidate](#) empêche la validation implicite. Pour valider vos contrôles, vous devez utiliser la validation explicite.

Validation explicite

L'approche de validation explicite valide les données à un moment donné. Vous pouvez valider les données en réponse à une action de l'utilisateur, par exemple en cliquant sur un bouton enregistrer ou sur un lien suivant. Lorsque l'action de l'utilisateur se produit, vous pouvez déclencher la validation explicite de l'une des manières suivantes :

- Appelez [Validate](#) pour valider que le dernier contrôle a perdu le focus.
- Appelez [ValidateChildren](#) pour valider tous les contrôles enfants dans un formulaire ou un contrôle conteneur.
- Appelez une méthode personnalisée pour valider les données dans les contrôles manuellement.

Comportement de validation implicite par défaut pour les contrôles Windows Forms

Les différents contrôles de Windows Forms ont des valeurs par défaut différentes pour leur propriété [AutoValidate](#). Le tableau suivant présente les contrôles les plus courants et leurs valeurs par défaut.

CONTRÔLE	COMPORTEMENT DE VALIDATION PAR DÉFAUT
ContainerControl	Inherit
Form	EnableAllowFocusChange
PropertyGrid	Propriété non exposée dans Visual Studio
ToolStripContainer	Propriété non exposée dans Visual Studio
SplitContainer	Inherit
UserControl	EnableAllowFocusChange

Fermeture du formulaire et substitution de la validation

Lorsqu'un contrôle gère le focus parce que les données qu'il contient ne sont pas valides, il est impossible de fermer le formulaire parent de l'une des manières habituelles :

- En cliquant sur le bouton **Fermer** .
- En sélectionnant **Fermer** dans le menu **système** .
- En appelant la méthode [Close](#) par programmation.

Toutefois, dans certains cas, vous souhaitez peut-être laisser l'utilisateur fermer le formulaire, que les valeurs des contrôles soient valides ou non. Vous pouvez remplacer la validation et fermer un formulaire qui contient encore des données non valides en créant un gestionnaire pour l'événement [FormClosing](#) du formulaire. Dans l'événement, affectez à la propriété [Cancel](#) la valeur `false` . Cela force la fermeture du formulaire. Pour plus d'informations et pour obtenir un exemple, consultez [Form.FormClosing](#).

NOTE

Si vous forcez le formulaire à se fermer de cette manière, toutes les données des contrôles du formulaire qui n'ont pas encore été enregistrées sont perdues. En outre, les formulaires modaux ne valident pas le contenu des contrôles lorsqu'ils sont fermés. Vous pouvez toujours utiliser la validation de contrôle pour verrouiller le focus sur un contrôle, mais vous n'avez pas à vous soucier du comportement associé à la fermeture du formulaire.

Voir aussi

- [Control.Validating](#)
- [Form.FormClosing](#)
- [System.Windows.Forms.FormClosingEventArgs](#)
- [MaskedTextBox](#), contrôle

- Exemples d'expressions régulières

Boîtes de dialogue dans les Windows Forms

2 minutes to read • [Edit Online](#)

Les boîtes de dialogue servent à interagir avec l'utilisateur et à récupérer des informations. En termes simples, une boîte de dialogue est un formulaire dont la propriété d'énumération `FormBorderStyle` a la valeur `FixedDialog`. Vous pouvez créer vos propres boîtes de dialogue personnalisées à l'aide de l'Concepteur Windows Forms dans Visual Studio. Ajoutez des contrôles tels que `Label`, `Textbox` et `Button` pour personnaliser les boîtes de dialogue en fonction de vos besoins spécifiques. La .NET Framework comprend également des boîtes de dialogue prédéfinies, telles que les boîtes de message et les **fichiers ouverts**, que vous pouvez adapter pour vos propres applications. Pour plus d'informations, consultez [contrôles et composants de boîte de dialogue](#).

Dans cette section

[Guide pratique pour afficher des boîtes de dialogue pour les Windows Forms](#)

Explique comment afficher des boîtes de dialogue.

Sections connexes

[Contrôles et composants de boîte de dialogue](#)

Répertorie les contrôles de boîtes de dialogue prédéfinis.

[Modification de l'apparence des Windows Forms](#)

Contient des liens vers des rubriques qui montrent comment modifier l'apparence des applications Windows Forms.

[Vue d'ensemble du contrôle TabControl](#)

Explique comment insérer le contrôle d'onglet dans une boîte de dialogue.

Comment : afficher des boîtes de dialogue pour les Windows Forms

2 minutes to read • [Edit Online](#)

Vous affichez une boîte de dialogue de la même manière que vous affichez n'importe quel autre formulaire dans une application. Le formulaire de démarrage se charge automatiquement lorsque l'application est exécutée. Pour faire apparaître un deuxième formulaire ou une boîte de dialogue dans l'application, écrivez du code pour le charger et l'afficher. De même, pour faire disparaître le formulaire ou la boîte de dialogue, écrivez du code pour le décharger ou le cacher.

Pour afficher une boîte de dialogue

1. Naviguez vers le gestionnaire d'événements avec lequel vous souhaitez ouvrir la boîte de dialogue. Cela peut se produire lorsqu'une commande de menu est sélectionnée, lorsqu'un bouton est cliqué, ou lorsque tout autre événement se produit.
2. Dans le gestionnaire d'événement, ajoutez du code pour ouvrir la boîte de dialogue. Dans cet exemple, un événement bouton-clic est utilisé pour afficher la boîte de dialogue:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles  
    Button1.Click  
        Dim dlg1 as new Form()  
        dlg1.ShowDialog()  
    End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)  
{  
    Form dlg1 = new Form();  
    dlg1.ShowDialog();  
}
```

```
private:  
void button1_Click(System::Object ^ sender,  
    System::EventArgs ^ e)  
{  
    Form ^ dlg1 = gcnew Form();  
    dlg1->ShowDialog();  
}
```

Liaison de données Windows Forms

4 minutes to read • [Edit Online](#)

La liaison de données dans les Windows Forms vous permet d'afficher et de modifier les informations d'une source de données dans les contrôles du formulaire. Vous pouvez établir une liaison à une source de données traditionnelle et à presque toute structure qui contient des données.

Dans cette section

[Liaison de données et Windows Forms](#)

Fournit une vue d'ensemble de la liaison de données dans les Windows Forms.

[Sources de données prises en charge par les Windows Forms](#)

Décrit les sources de données qui peuvent être utilisées avec Windows Forms.

[Interfaces participant à la liaison de données](#)

Décrit plusieurs interfaces utilisées avec la liaison de données Windows Forms.

[Guide pratique pour naviguer au sein des données dans les Windows Forms](#)

Montre comment parcourir les éléments dans une source de données.

[Notification de modifications dans la liaison de données Windows Forms](#)

Décrit les différents types de notifications de modifications pour la liaison de données Windows Forms.

[Guide pratique pour implémenter l'interface INotifyPropertyChanged](#)

Montre comment implémenter l'interface `INotifyPropertyChanged`. L'interface communique à un contrôle dépendant les modifications apportées aux propriétés d'un objet métier.

[Guide pratique pour appliquer le modèle PropertyChanged](#)

Montre comment appliquer le modèle `PropertyChanged` aux propriétés d'un contrôle utilisateur Windows Forms.

[Guide pratique pour implémenter l'interface IListSource](#)

Montre comment activer la découverte du schéma pour une liste pouvant être liée en implémentant l'interface `IListSource`.

[Guide pratique pour implémenter l'interface IListSource](#)

Montre comment implémenter l'interface `IListSource` pour créer une classe pouvant être liée qui n'implémente pas `IList` mais fournit une liste à partir d'un autre emplacement.

[Guide pratique pour s'assurer que plusieurs contrôles liés à la même source de données restent synchronisés](#)

Montre comment gérer l'événement `BindingComplete` pour garantir que tous les contrôles liés à une source de données restent synchronisés.

[Guide pratique pour s'assurer que la ligne sélectionnée dans une table enfant reste au bon emplacement](#)

Montre comment s'assurer que la ligne sélectionnée dans une table enfant ne change pas quand une modification est apportée à un champ de la table parente.

Consultez également les [interfaces relatives à la liaison de données](#), [Comment : naviguer parmi les données dans Windows Forms](#) et [Comment : créer un contrôle à liaison simple dans un Windows Form](#).

Référence

[System.Windows.Forms.Binding](#)

Décrit la classe qui représente la liaison entre un composant pouvant être lié et une source de données.

[System.Windows.Forms.BindingSource](#)

Décrit la classe qui encapsule une source de données pour la liaison à des contrôles.

Sections connexes

[BindingSource, composant](#)

Contient une liste de rubriques qui expliquent comment utiliser le composant [BindingSource](#).

[DataGridView, contrôle](#)

Fournit une liste de rubriques qui expliquent comment utiliser un contrôle datagrid pouvant être lié.

Consultez également [accès aux données dans Visual Studio](#).

Liaison de données et Windows Forms

8 minutes to read • [Edit Online](#)

Dans Windows Forms, vous pouvez créer des liaisons avec des sources de données traditionnelles, mais aussi avec quasiment toute structure contenant des données. Vous pouvez créer une liaison avec un tableau de valeurs que vous calculez au moment de l'exécution, que vous lisez depuis un fichier ou que vous dérivez de valeurs d'autres contrôles.

De plus, vous pouvez lier n'importe quelle propriété de n'importe quel contrôle à la source de données. Dans une liaison de données traditionnelle, vous liez généralement la propriété d'affichage (par exemple, la propriété `Text` d'un contrôle `TextBox`) à la source de données. Avec la .NET Framework, vous avez également la possibilité de définir d'autres propriétés via la liaison. Vous pouvez utiliser des liaisons pour effectuer les tâches suivantes :

- Définir le graphisme d'un contrôle d'image
- Définir la couleur d'arrière-plan d'un ou plusieurs contrôles
- Définition de la taille des contrôles

La liaison de données permet de définir automatiquement toute propriété d'un contrôle de formulaire accessible au moment de l'exécution.

Types de liaisons de données

Windows Forms peut utiliser deux types de liaisons de données : les liaisons simples et les liaisons complexes. Chaque type a ses propres avantages.

TYPE DE LIAISON DE DONNÉES	DESCRIPTION
Liaison de données simple	<p>Capacité d'un contrôle à créer une liaison avec un élément de données, tel qu'une valeur dans une colonne d'une table de dataset. Il s'agit du type de liaison typique des contrôles tels que <code>TextBox</code> ou <code>Label</code>, qui n'affichent généralement qu'une seule valeur. En réalité, n'importe quelle propriété d'un contrôle peut être liée à un champ d'une base de données. Il existe une prise en charge étendue de cette fonctionnalité dans Visual Studio.</p> <p>Pour plus d'informations, consultez les pages suivantes :</p> <p>interfaces de - liées à la liaison de données</p> <ul style="list-style-type: none">- Comment : parcourir des données dans Windows Forms- Comment : créer un contrôle à liaison simple dans un Windows Form
Liaison de données complexe	<p>Capacité d'un contrôle à lier plusieurs éléments de données, généralement plusieurs enregistrements d'une base de données. La liaison complexe est également appelée liaison basée sur des listes. Des contrôles qui prennent en charge la liaison complexe sont, par exemple, <code>DataGridView</code>, <code>ListBox</code> et <code>ComboBox</code>. Pour obtenir un exemple de liaison de données complexe, consultez Comment : lier un contrôle ComboBox ou ListBox Windows Forms à des données.</p>

Composant BindingSource

Pour simplifier la liaison de données, Windows Forms permet de lier une source de données au composant [BindingSource](#), puis de lier les contrôles à [BindingSource](#). Vous pouvez utiliser [BindingSource](#) dans des scénarios de liaison simple ou complexe. Dans les deux cas, [BindingSource](#) joue le rôle d'intermédiaire entre la source de données et les contrôles liés qui fournissent des services de notification de modifications et de gestion des devises, entre autres services.

Scénarios courants impliquant des liaisons de données

Quasiment toutes les applications commerciales utilisent des informations lues à partir de sources de données d'un type quelconque, généralement au moyen de liaisons de données. La liste suivante présente quelques exemples de scénarios courants qui utilisent la liaison de données comme méthode de présentation et de manipulation des données.

SCÉNARIO	DESCRIPTION
Reporting	Les rapports permettant d'afficher et de synthétiser vos données dans un document imprimé. Il est très courant de créer un rapport à partir du contenu sélectionné d'une source de données en vue de l'afficher à l'écran ou de l'imprimer. Les rapports les plus couramment créés sont les listes, les factures et les résumés. Les éléments sont généralement organisés sous forme de colonnes de listes, avec des sous-éléments sous chacun d'eux. Toutefois, vous devez choisir la disposition qui convient le mieux à vos données.
Entrée de données	Les formulaires de saisie de données sont couramment utilisés pour entrer de grandes quantités de données associées ou pour demander aux utilisateurs de saisir des informations. Les utilisateurs peuvent entrer des informations ou choisir parmi des options au moyen de cases à cocher, de cases d'option, de listes déroulantes et de zones de texte. Les informations sont ensuite envoyées, puis stockées dans une base de données, dont la structure est basée sur les informations entrées.
Relation Maître/Détail	Une application Maître/Détail est un format qui permet de consulter des données associées. Il s'agit de deux tables de données liées par une relation. Un exemple classique est celui de la table "Clients" et de la table "Commandes" qui sont liées par une relation permettant d'associer les clients aux commandes qu'ils ont passées. Pour plus d'informations sur la création d'une application maître/détail avec deux Windows Forms DataGridView contrôles, consultez Comment : créer un formulaire maître/détail à l'aide de deux contrôles DataGridView Windows Forms

SCÉNARIO	DESCRIPTION
Table de choix	<p>Un autre scénario courant de manipulation et de présentation des données est celui de la table de choix. Souvent, dans le cadre d'un affichage de données plus important, un contrôle ComboBox est utilisé pour afficher et manipuler des données. Ce qu'il faut retenir, c'est que les données affichées dans le contrôle ComboBox sont différentes de celles écrites dans la base de données. Par exemple, si vous avez un contrôle ComboBox qui affiche les articles disponibles dans une épicerie, vous préférerez sans doute voir le nom des produits (pain, lait, œufs). Toutefois, afin de faciliter l'extraction d'informations et dans un objectif de normalisation des bases de données, vous stockeriez probablement les informations relatives aux éléments d'une commande donnée sous forme de numéros d'article (#501, #603, etc.). Par conséquent, il existe un lien implicite entre le "nom convivial" de l'article dans le contrôle ComboBox de votre formulaire, et le numéro d'article associé qui est présent dans la commande. C'est tout l'intérêt d'une recherche dans une table. Pour plus d'informations, consultez procédure : création d'une table de recherche avec le composant Windows Forms BindingSource.</p>

Voir aussi

- [Binding](#)
- [Liaison de données Windows Forms](#)
- [Guide pratique pour lier le contrôle DataGridView Windows Forms à une source de données](#)
- [BindingSource, composant](#)

Sources de données prises en charge par les Windows Forms

7 minutes to read • [Edit Online](#)

Traditionnellement, la liaison de données a été utilisée dans les applications pour tirer parti des données stockées dans les bases de données. Avec Windows Forms la liaison de données, vous pouvez accéder aux données des bases de données ainsi qu'aux données d'autres structures, telles que les tableaux et les collections, tant que certaines exigences minimales ont été respectées.

Structures à lier

Dans Windows Forms, vous pouvez lier à une grande variété de structures, à partir d'objets simples (liaison simple) à des listes complexes, telles que des tables de données ADO.NET (liaison complexe). Pour une liaison simple, Windows Forms prend en charge la liaison aux propriétés publiques sur l'objet simple. Windows Forms la liaison basée sur une liste requiert généralement que l'objet prenne en charge l'interface [IList](#) ou l'interface [IListSource](#). En outre, si vous effectuez une liaison avec via un composant [BindingSource](#), vous pouvez effectuer une liaison à un objet qui prend en charge l'interface [IEnumerable](#). Pour plus d'informations sur les interfaces associées à la liaison de données, consultez [interfaces associées à la liaison de données](#).

La liste suivante répertorie les structures auxquelles vous pouvez établir une liaison dans Windows Forms.

[BindingSource](#)

Une [BindingSource](#) est la source de données Windows Forms la plus courante et agit un proxy entre une source de données et les contrôles de Windows Forms. Le modèle d'utilisation générale [BindingSource](#) consiste à lier vos contrôles à la [BindingSource](#) et à lier les [BindingSource](#) à la source de données (par exemple, une table de données ADO.NET ou un objet métier). Le [BindingSource](#) fournit des services qui activent et améliorent le niveau de prise en charge de la liaison de données. Par exemple, Windows Forms des contrôles basés sur la liste, tels que les [DataGridView](#) et les [ComboBox](#) ne prennent pas directement en charge la liaison à des sources de données [IEnumerable](#), vous pouvez activer ce scénario par liaison à l'aide d'une [BindingSource](#). Dans ce cas, le [BindingSource](#) convertira la source de données en [IList](#).

Objets simples

Windows Forms prend en charge les propriétés de contrôle de liaison de données dans les propriétés publiques de l'instance d'un objet à l'aide du type de [Binding](#). Windows Forms prend également en charge la liaison de contrôles basés sur des listes, tels qu'un [ListControl](#) à une instance d'objet lorsqu'un [BindingSource](#) est utilisé.

tableau ou collection

Pour agir en tant que source de données, une liste doit implémenter l'interface [IList](#) ; un tableau qui est une instance de la classe [Array](#) en est un exemple. Pour plus d'informations sur les tableaux, consultez [Comment : créer un tableau d'objets \(Visual Basic\)](#).

En général, vous devez utiliser [BindingList<T>](#) lorsque vous créez des listes d'objets pour la liaison de données. [BindingList<T>](#) est une version générique de l'interface [IBindingList](#). L'interface [IBindingList](#) étend l'interface [IList](#) en ajoutant des propriétés, des méthodes et des événements nécessaires pour la liaison de données bidirectionnelle.

[IEnumerable](#)

Windows Forms contrôles peuvent être liés à des sources de données qui ne prennent en charge que l'interface [IEnumerable](#) si elles sont liées via un composant [BindingSource](#).

Objets de données ADO.NET

ADO.NET fournit un certain nombre de structures de données adaptées à la liaison. Chacun varie en fonction de sa complexité et de son sophistication.

- [DataColumn](#). Un [DataColumn](#) est le bloc de construction essentiel d'un [DataTable](#), dans le fait qu'un certain nombre de colonnes comportent une table. Chaque [DataColumn](#) a une propriété [DataType](#) qui détermine le type de données que contient la colonne (par exemple, la marque d'une automobile dans une table décrivant des voitures). Vous pouvez facilement lier un contrôle (par exemple, la propriété [Text](#) d'un contrôle [TextBox](#)) à une colonne dans une table de données.
- [DataTable](#). Une [DataTable](#) est la représentation d'une table, avec des lignes et des colonnes, dans ADO.NET. Une table de données contient deux collections : [DataColumn](#), représentant les colonnes de données d'une table donnée (qui déterminent en fin de compte les types de données qui peuvent être entrées dans cette table) et [DataRow](#), représentant les lignes de données d'une table donnée. Vous pouvez lier un contrôle de manière complexe aux informations contenues dans une table de données (par exemple, en liant le contrôle [DataGridView](#) à une table de données). Toutefois, lorsque vous effectuez une liaison à une [DataTable](#), vous êtes véritablement lié à la vue par défaut de la table.
- [DataView](#). Une [DataView](#) est une vue personnalisée d'une table de données qui peut être filtrée ou triée. Une vue de données est l'instantané de données utilisé par les contrôles liés complexes. Vous pouvez effectuer une liaison simple ou une liaison complexe aux données d'une vue de données, mais sachez que vous êtes lié à une « image » fixe des données plutôt qu'à une source de données propre et en cours de mise à jour.
- [DataSet](#). Une [DataSet](#) est une collection de tables, de relations et de contraintes de données dans une base de données. Vous pouvez effectuer une liaison simple ou une liaison complexe aux données d'un [DataSet](#), mais sachez que vous êtes lié au [DataViewManager](#) par défaut pour le [DataSet](#) (voir le point à puce suivant).
- [DataViewManager](#). Une [DataViewManager](#) est une vue personnalisée du [DataSet](#) entier, analogue à un [DataView](#), mais avec des relations incluses. Avec une collection [DataViewSettings](#), vous pouvez définir des filtres et des options de tri par défaut pour toutes les vues que l' [DataViewManager](#) a pour une table donnée.

Voir aussi

- [Notification de modifications dans la liaison de données Windows Forms](#)
- [Liaison de données et Windows Forms](#)
- [Liaison de données Windows Forms](#)

Interfaces participant à la liaison de données

15 minutes to read • [Edit Online](#)

Avec ADO.NET, vous pouvez créer de nombreuses structures de données différentes pour répondre aux besoins de liaison de votre application et des données que vous utilisez. Vous souhaitez peut-être créer vos propres classes qui fourniront ou utiliseront des données dans les Windows Forms. Ces objets peuvent offrir différents niveaux de fonctionnalités et de complexité, de la liaison de données basique à la fourniture d'une prise en charge au moment du design, en passant par la vérification des erreurs, la notification des modifications ou même la prise en charge de la restauration structurée des modifications apportées aux données elles-mêmes.

Utilisateurs d'interfaces de liaison de données

Les sections suivantes décrivent deux groupes d'objets d'interface. Le premier groupe répertorie les interfaces implémentées sur les sources de données par les auteurs de sources de données. Ces interfaces sont conçues pour être utilisées par les consommateurs de sources de données, qui sont dans la plupart des cas des contrôles ou des composants Windows Forms. Le second groupe répertorie les interfaces conçues pour être utilisées par les auteurs de composants. Les auteurs de composants utilisent ces interfaces lorsqu'ils créent un composant prenant en charge la liaison de données pour une utilisation par le moteur de liaison de données Windows Forms. Vous pouvez implémenter ces interfaces dans les classes associées à votre formulaire pour activer la liaison de données. Chaque cas présente une classe qui implémente une interface permettant l'interaction avec les données. Les outils d'expérience de conception de données de Visual Studio Rapid Application Development (RAD) tirent déjà parti de cette fonctionnalité.

Interfaces pour implémentation par des auteurs de sources de données

Les interfaces suivantes sont conçues pour être utilisées par les contrôles Windows Forms :

- interface [IList](#)

Une classe qui implémente l'interface [IList](#) peut être une [Array](#), [ArrayList](#) ou [CollectionBase](#). Il s'agit de listes indexées d'éléments de type [Object](#). Ces listes doivent contenir des types homogènes, car le premier élément de l'index détermine le type. [IList](#) n'est disponible pour la liaison qu'au moment de l'exécution.

NOTE

Si vous souhaitez créer une liste d'objets métier pour la liaison avec Windows Forms, vous devez envisager d'utiliser l' [BindingList<T>](#). Le [BindingList<T>](#) est une classe extensible qui implémente les interfaces principales requises pour la liaison de données bidirectionnelle Windows Forms.

- interface [IBindingList](#)

Une classe qui implémente l'interface [IBindingList](#) fournit un niveau de fonctionnalité de liaison de données bien supérieur. Cette implémentation fournit des fonctionnalités de tri de base et une notification des modifications, à la fois lorsque la liste des éléments change (par exemple, le champ Adresse du troisième élément d'une liste de clients change) et lorsque la liste elle-même change (par exemple, augmentation ou diminution du nombre d'éléments de la liste). La notification des modifications est importante si vous prévoyez de lier plusieurs contrôles aux mêmes données et si vous souhaitez que les modifications apportées aux données d'un contrôle se propagent à d'autres contrôles liés.

NOTE

La notification de modification est activée pour l'interface [IBindingList](#) via la propriété [SupportsChangeNotification](#) qui, lorsque `true`, déclenche un événement [ListChanged](#), indiquant que la liste a été modifiée ou qu'un élément de la liste a été modifié.

Le type de modification est décrit par la propriété [ListChangedType](#) du paramètre [ListChangedEventArgs](#). Par conséquent, à chaque mise à jour du modèle de données, toutes les vues dépendantes, comme les autres contrôles liés à la même source de données, seront également mises à jour. Toutefois, les objets contenus dans la liste devront notifier la liste lorsqu'ils changent afin que la liste puisse déclencher l'événement [ListChanged](#).

NOTE

L' [BindingList<T>](#) fournit une implémentation générique de l'interface [IBindingList](#).

- interface [IBindingListView](#)

Une classe qui implémente l'interface [IBindingListView](#) fournit toutes les fonctionnalités d'une implémentation de [IBindingList](#), ainsi que des fonctionnalités de filtrage et de tri avancé. Cette implémentation offre un filtrage basé sur les chaînes ainsi que le tri multicolonne avec des paires direction-descripteur de propriété.

- interface [IEditableObject](#)

Une classe qui implémente l'interface [IEditableObject](#) permet à un objet de contrôler le moment où les modifications apportées à cet objet sont rendues permanentes. Cette implémentation vous permet d'utiliser les méthodes [BeginEdit](#), [EndEdit](#) et [CancelEdit](#), qui vous permettent de restaurer les modifications apportées à l'objet. Vous trouverez ci-dessous une brève explication du fonctionnement des méthodes [BeginEdit](#), [EndEdit](#) et [CancelEdit](#), ainsi que de la façon dont elles fonctionnent conjointement les unes avec les autres pour permettre une restauration possible des modifications apportées aux données :

- La méthode [BeginEdit](#) signale le début d'une modification sur un objet. Un objet qui implémente cette interface doit stocker les mises à jour après l'appel de la méthode [BeginEdit](#) de manière à ce que les mises à jour puissent être ignorées si la méthode [CancelEdit](#) est appelée. Dans Windows Forms de liaison de données, vous pouvez appeler [BeginEdit](#) plusieurs fois dans l'étendue d'une seule transaction de modification (par exemple, [BeginEdit](#), [BeginEdit](#), [EndEdit](#)). Les implémentations de [IEditableObject](#) doivent suivre si [BeginEdit](#) a déjà été appelé et ignorer les appels suivants à [BeginEdit](#). Étant donné que cette méthode peut être appelée plusieurs fois, il est important que les appels suivants à celle-ci soient non destructifs ; autrement dit, les appels de [BeginEdit](#) suivants ne peuvent pas détruire les mises à jour effectuées ou modifier les données qui ont été enregistrées lors du premier appel de [BeginEdit](#).
- La méthode [EndEdit](#) envoie toutes les modifications depuis l'appel de [BeginEdit](#) dans l'objet sous-jacent, si l'objet est actuellement en mode édition.
- La méthode [CancelEdit](#) ignore toutes les modifications apportées à l'objet.

Pour plus d'informations sur le fonctionnement des méthodes [BeginEdit](#), [EndEdit](#) et [CancelEdit](#), consultez [enregistrer des données dans la base de données](#).

Cette notion transactionnelle des fonctionnalités de données est utilisée par le contrôle [DataGridView](#).

- interface [ICancelAddNew](#)

Une classe qui implémente l'interface [ICancelAddNew](#) implémente généralement l'interface [IBindingList](#)

et vous permet de restaurer un ajout apporté à la source de données avec la méthode [AddNew](#). Si votre source de données implémente l'interface [IBindingList](#), vous devez également l'implémenter dans l'interface [ICancelAddNew](#).

- interface [IDataErrorInfo](#)

Une classe qui implémente l'interface [IDataErrorInfo](#) permet aux objets d'offrir des informations d'erreur personnalisées aux contrôles liés :

- La propriété [Error](#) retourne le texte général du message d'erreur (par exemple, « une erreur s'est produite »).
- La propriété [Item\[String\]](#) retourne une chaîne avec le message d'erreur spécifique de la colonne (par exemple, « la valeur dans la colonne `State` n'est pas valide »).

- interface [IEnumerable](#)

Une classe qui implémente l'interface [IEnumerable](#) est généralement consommée par ASP.NET. La prise en charge de Windows Forms pour cette interface est disponible uniquement via le composant [BindingSource](#).

NOTE

Le composant [BindingSource](#) copie tous les éléments [IEnumerable](#) dans une liste distincte à des fins de liaison.

- interface [ITypedList](#)

Une classe de collections qui implémente l'interface [ITypedList](#) offre la possibilité de contrôler l'ordre et l'ensemble des propriétés exposées au contrôle lié.

NOTE

Lorsque vous implémentez la méthode [GetItemProperties](#) et que le tableau de [PropertyDescriptor](#) n'est pas null, la dernière entrée du tableau est le descripteur de propriété qui décrit la propriété de liste qui est une autre liste d'éléments.

- interface [ICustomTypeDescriptor](#)

Une classe qui implémente l'interface [ICustomTypeDescriptor](#) fournit des informations dynamiques à propos de lui-même. Cette interface est similaire à [ITypedList](#), mais elle est utilisée pour les objets plutôt que pour les listes. Cette interface est utilisée par [DataRowView](#) pour projeter le schéma des lignes sous-jacentes. Une implémentation simple de [ICustomTypeDescriptor](#) est fournie par la classe [CustomTypeDescriptor](#).

NOTE

Pour prendre en charge la liaison au moment du design aux types qui implémentent [ICustomTypeDescriptor](#), le type doit également implémenter [IComponent](#) et exister en tant qu'instance sur le formulaire.

- interface [IListSource](#)

Une classe qui implémente l'interface [IListSource](#) active la liaison basée sur les listes sur les objets non listés. La méthode [GetList](#) de [IListSource](#) est utilisée pour retourner une liste pouvant être liée à partir d'un objet qui n'hérite pas de [IList](#). [IListSource](#) est utilisé par la classe [DataSet](#).

- interface [IRaiseItemChangedEvents](#)

Une classe qui implémente l'interface [IRaisesItemChangedEvents](#) est une liste pouvant être liée qui implémente également l'interface [IBindingList](#). Cette interface est utilisée pour indiquer si votre type déclenche [ListChanged](#) événements de type [ItemChanged](#) par le biais de sa propriété [RaisesItemChangedEvents](#).

NOTE

Vous devez implémenter le [IRaisesItemChangedEvents](#) si votre source de données fournit la conversion d'événement de la propriété à la liste décrite précédemment et interagit avec le composant [BindingSource](#). Dans le cas contraire, le [BindingSource](#) effectuera également la propriété pour répertorier la conversion d'événements, ce qui ralentit les performances.

- interface [ISupportInitialize](#)

Un composant qui implémente l'interface [ISupportInitialize](#) tire parti des optimisations par lots pour définir des propriétés et initialiser des propriétés codépendantes. Le [ISupportInitialize](#) contient deux méthodes :

- [BeginInit](#) signale que l'initialisation de l'objet démarre.
- [EndInit](#) signale la fin de l'initialisation de l'objet.

- interface [ISupportInitializeNotification](#)

Un composant qui implémente l'interface [ISupportInitializeNotification](#) implémente également l'interface [ISupportInitialize](#). Cette interface vous permet de notifier d'autres composants de [ISupportInitialize](#) que l'initialisation est terminée. L'interface [ISupportInitializeNotification](#) contient deux membres :

- [IsInitialized](#) retourne une valeur `boolean` indiquant si le composant est initialisé.
- [Initialized](#) se produit lors de l'appel de [EndInit](#).

- interface [INotifyPropertyChanged](#)

Une classe qui implémente cette interface est un type qui déclenche un événement lorsqu'une de ses valeurs de propriété change. Cette interface est conçue pour remplacer le modèle d'événement de modification pour chaque propriété d'un contrôle. Lorsqu'il est utilisé dans un [BindingList<T>](#), un objet métier doit implémenter l'interface [INotifyPropertyChanged](#) et le [BindingList](#) convertit [PropertyChanged](#) événements en [ListChanged](#) événements de type [ItemChanged](#).

NOTE

Pour que la notification de modification se produise dans une liaison entre un client lié et une source de données, votre type de source de données liée doit implémenter l'interface [INotifyPropertyChanged](#) (par défaut) ou vous pouvez fournir `propertyName` `Changed` événements pour le type lié, mais vous ne devez pas effectuer les deux.

Interfaces pour implémentation par des auteurs de composants

Les interfaces suivantes sont conçues pour une utilisation par le moteur de liaison de données Windows Forms :

- interface [IBindableComponent](#)

Une classe qui implémente cette interface est un composant qui n'est pas un contrôle et qui prend en charge la liaison de données. Cette classe retourne les liaisons de données et le contexte de liaison du composant via les propriétés [DataBindings](#) et [BindingContext](#) de cette interface.

NOTE

Si votre composant hérite de [Control](#), il n'est pas nécessaire d'implémenter l'interface [IBindableComponent](#).

- interface [ICurrencyManagerProvider](#)

Une classe qui implémente l'interface [ICurrencyManagerProvider](#) est un composant qui fournit son propre [CurrencyManager](#) pour gérer les liaisons associées à ce composant particulier. L'accès à l'[CurrencyManager](#) personnalisé est fourni par la propriété [CurrencyManager](#).

NOTE

Une classe qui hérite de [Control](#) gère automatiquement les liaisons par le biais de sa propriété [BindingContext](#), de sorte que les cas dans lesquels vous devez implémenter les [ICurrencyManagerProvider](#) sont relativement rares.

Voir aussi

- [Liaison de données et Windows Forms](#)
- [Guide pratique pour créer un contrôle à liaison simple dans un Windows Form](#)
- [Liaison de données Windows Forms](#)

Notification de modifications dans la liaison de données Windows Forms

5 minutes to read • [Edit Online](#)

L'un des concepts les plus importants de la liaison de données Windows Forms est la *notification de modification*. Pour vous assurer que votre source de données et vos contrôles liés possèdent toujours les données les plus récentes, vous devez ajouter la notification de modification pour la liaison de données. En particulier, vous souhaitez vous assurer que les contrôles liés sont avertis des modifications apportées à leur source de données, et la source de données est avertie des modifications apportées aux propriétés liées d'un contrôle.

Il existe différents types de notifications de modification, selon le type de liaison de données :

- Liaison simple, dans laquelle une propriété de contrôle unique est liée à une seule instance d'un objet.
- Liaison basée sur une liste, qui peut inclure une propriété de contrôle unique liée à la propriété d'un élément dans une liste ou une propriété de contrôle liée à une liste d'objets.

En outre, si vous créez Windows Forms contrôles que vous souhaitez utiliser pour la liaison de données, vous devez appliquer le modèle de *PropertyNameChanged* aux contrôles, afin que les modifications apportées à la propriété liée d'un contrôle soient propagées à la source de données.

Notification de modification pour liaison simple

Pour une liaison simple, les objets métier doivent fournir une notification de modification lorsque la valeur d'une propriété liée change. Pour ce faire, vous pouvez exposer un événement *PropertyNameChanged* pour chaque propriété de votre objet métier et lier l'objet métier à des contrôles avec le [BindingSource](#) ou à la méthode préférée dans laquelle votre objet métier implémente l'interface [INotifyPropertyChanged](#) et déclenche un événement [PropertyChanged](#) quand la valeur d'une propriété change. Pour plus d'informations, consultez [Comment : implémenter l'interface INotifyPropertyChanged](#). Lorsque vous utilisez des objets qui implémentent l'interface [INotifyPropertyChanged](#), il n'est pas nécessaire d'utiliser la [BindingSource](#) pour lier l'objet à un contrôle, mais l'utilisation de la [BindingSource](#) est recommandée.

Notification de modification pour la liaison basée sur une liste

Windows Forms dépend d'une liste liée pour fournir une modification de propriété (modification de la valeur d'une propriété d'élément de liste) et d'une liste modifiée (un élément est supprimé ou ajouté à la liste) à des contrôles liés. Par conséquent, les listes utilisées pour la liaison de données doivent implémenter le [IBindingList](#), qui fournit les deux types de notification de modification. Le [BindingList<T>](#) est une implémentation générique de [IBindingList](#) et est conçu pour être utilisé avec Windows Forms la liaison de données. Vous pouvez créer un [BindingList<T>](#) qui contient un type d'objet métier qui implémente [INotifyPropertyChanged](#) et la liste convertit automatiquement les événements [PropertyChanged](#) en [ListChanged](#) événements. Si la liste liée n'est pas un [IBindingList](#), vous devez lier la liste d'objets à Windows Forms contrôles à l'aide du composant [BindingSource](#). Le composant [BindingSource](#) fournit une conversion de propriété en liste similaire à celle de l' [BindingList<T>](#). Pour plus d'informations, consultez [Comment : déclencher des notifications de modifications à l'aide d'un BindingSource et de l'interface INotifyPropertyChanged](#).

Notification de modification pour les contrôles personnalisés

Enfin, du côté du contrôle, vous devez exposer un événement *PropertyNameChanged* pour chaque propriété conçue pour être liée aux données. Les modifications apportées à la propriété de contrôle sont ensuite propagées

vers la source de données liée. Pour plus d'informations, consultez [Comment : appliquer le modèle PropertyChangedEventArgs](#)

Voir aussi

- [BindingSource](#)
- [INotifyPropertyChanged](#)
- [BindingList<T>](#)
- [Liaison de données Windows Forms](#)
- [Sources de données prises en charge par les Windows Forms](#)
- [Liaison de données et Windows Forms](#)

Procédure : appliquer le modèle PropertyNameChanged

2 minutes to read • [Edit Online](#)

L'exemple de code suivant montre comment appliquer le *PropertyName* modèle a été modifié pour un contrôle personnalisé. Appliquer ce modèle lorsque vous implémentez des contrôles personnalisés qui sont utilisés avec le moteur de liaison de données Windows Forms.

Exemple

```

// This class implements a simple user control
// that demonstrates how to apply the propertyNameChanged pattern.
[ComplexBindingProperties("DataSource", "DataMember")]
public class CustomerControl : UserControl
{
    private DataGridView dataGridView1;
    private Label label1;
    private DateTime lastUpdate = DateTime.Now;

    public EventHandler DataSourceChanged;

    public object DataSource
    {
        get
        {
            return this.dataGridView1.DataSource;
        }
        set
        {
            if (DataSource != value)
            {
                this.dataGridView1.DataSource = value;
                OnDataSourceChanged();
            }
        }
    }

    public string DataMember
    {
        get { return this.dataGridView1.DataMember; }

        set { this.dataGridView1.DataMember = value; }
    }

    private void OnDataSourceChanged()
    {
        if (DataSourceChanged != null)
        {
            DataSourceChanged(this, new EventArgs());
        }
    }

    public CustomerControl()
    {
        this.dataGridView1 = new System.Windows.Forms.DataGridView();
        this.label1 = new System.Windows.Forms.Label();
        this.dataGridView1.ColumnHeadersHeightSizeMode =
            System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
        this.dataGridView1.ImeMode = System.Windows.Forms.ImeMode.Disable;
        this.dataGridView1.Location = new System.Drawing.Point(19, 55);
        this.dataGridView1.Size = new System.Drawing.Size(350, 150);
        this.dataGridView1.TabIndex = 1;
        this.label1.AutoSize = true;
        this.label1.Location = new System.Drawing.Point(19, 23);
        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(76, 13);
        this.label1.TabIndex = 2;
        this.label1.Text = "Customer List:";
        this.Controls.Add(this.label1);
        this.Controls.Add(this.dataGridView1);
        this.Size = new System.Drawing.Size(350, 216);
    }
}

```

```

' This class implements a simple user control
' that demonstrates how to apply the propertyNameChanged pattern.
<ComplexBindingProperties("DataSource", "DataMember")> _
Public Class CustomerControl
    Inherits UserControl
    Private dataGridView1 As DataGridView
    Private label1 As Label
    Private lastUpdate As DateTime = DateTime.Now

    Public DataSourceChanged As EventHandler

    Public Property DataSource() As Object
        Get
            Return Me.dataGridView1.DataSource
        End Get
        Set
            If DataSource IsNot Value Then
                Me.dataGridView1.DataSource = Value
                OnDataSourceChanged()
            End If
        End Set
    End Property

    Public Property DataMember() As String
        Get
            Return Me.dataGridView1.DataMember
        End Get
        Set
            Me.dataGridView1.DataMember = value
        End Set
    End Property

    Private Sub OnDataSourceChanged()
        If (DataSourceChanged IsNot Nothing) Then
            DataSourceChanged(Me, New EventArgs())
        End If
    End Sub

    Public Sub New()
        Me.dataGridView1 = New System.Windows.Forms.DataGridView()
        Me.label1 = New System.Windows.Forms.Label()
        Me.dataGridView1.ColumnHeadersHeightSizeMode = _
            System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize
        Me.dataGridView1.ImeMode = System.Windows.Forms.ImeMode.Disable
        Me.dataGridView1.Location = New System.Drawing.Point(19, 55)
        Me.dataGridView1.Size = New System.Drawing.Size(350, 150)
        Me.dataGridView1.TabIndex = 1
        Me.label1.AutoSize = True
        Me.label1.Location = New System.Drawing.Point(19, 23)
        Me.label1.Name = "label1"
        Me.label1.Size = New System.Drawing.Size(76, 13)
        Me.label1.TabIndex = 2
        Me.label1.Text = "Customer List:"
        Me.Controls.Add(Me.label1)
        Me.Controls.Add(Me.dataGridView1)
        Me.Size = New System.Drawing.Size(350, 216)
    End Sub
End Class

```

Compilation du code

Pour compiler l'exemple de code précédent :

- Collez le code dans un fichier de code vide. Vous devez utiliser le contrôle personnalisé dans un formulaire Windows qui contient un `Main` (méthode).

Voir aussi

- [Guide pratique pour Implémenter l'Interface INotifyPropertyChanged](#)
- [Notification de modifications dans la liaison de données Windows Forms](#)
- [Liaison de données Windows Forms](#)

Procédure : créer un contrôle lié et mettre en forme les données affichées

4 minutes to read • [Edit Online](#)

Avec Windows Forms la liaison de données, vous pouvez mettre en forme les données affichées dans un contrôle lié aux données à l'aide de la boîte de dialogue **mise en forme et liaison avancée**.

Pour lier un contrôle et mettre en forme les données affichées

1. Connectez-vous à une source de données. Pour plus d'informations, consultez [connexion à une source de données](#).
2. Dans Visual Studio, sélectionnez le contrôle sur le formulaire, puis ouvrez la fenêtre **Propriétés**.
3. Développez la propriété (**DataBindings**), puis dans la zone (**avancé**), cliquez sur le bouton de Visual Studio) pour afficher les **options de mise en forme et avancé**. La boîte de dialogue liaison, qui contient la liste complète des propriétés de ce contrôle.
4. Sélectionnez la propriété que vous souhaitez lier, puis sélectionnez la flèche **liaison**.

Une liste des sources de données disponibles s'affiche.

5. Développez la source de données avec laquelle vous voulez établir une liaison jusqu'à trouver l'élément de données souhaité.

Par exemple, si vous établissez une liaison à une valeur de colonne dans une table de dataset, développez le nom du dataset, puis développez le nom de la table pour afficher les noms des colonnes.

6. Sélectionnez le nom d'un élément à lier.
7. Dans la zone **type de format**, sélectionnez le format que vous souhaitez appliquer aux données affichées dans le contrôle.

Dans tous les cas, vous pouvez spécifier la valeur affichée dans le contrôle si la source de données contient **DBNull**. Sinon, les options varient légèrement en fonction du type de format que vous choisissez. Le tableau suivant présente les options et les types de format.

TYPE DE FORMAT	OPTION DE MISE EN FORME
Aucune mise en forme	Aucune option.
Numeric	Spécifiez le nombre de décimales à l'aide du contrôle de nombre de décimales.
Devise	Spécifiez le nombre de décimales à l'aide du contrôle de nombre de décimales.
Date et heure	Sélectionnez le mode d'affichage de la date et de l'heure en sélectionnant l'un des éléments dans la zone de sélection type .
Scientifique	Spécifiez le nombre de décimales à l'aide du contrôle de nombre de décimales.

TYPE DE FORMAT	OPTION DE MISE EN FORME
Personnalisé	<p>Spécifiez une chaîne de format personnalisée.</p> <p>Pour plus d'informations, consultez Mise en forme des types. Remarque : Il n'est pas garanti que les chaînes de format personnalisées puissent effectuer un aller-retour entre la source de données et le contrôle dépendant. Gérez plutôt l'événement Parse ou Format pour la liaison et appliquez la mise en forme personnalisée dans le code de gestion d'événements.</p>

8. Sélectionnez **OK** pour fermer la boîte de dialogue **mise en forme et liaison avancée** et revenir au fenêtre Propriétés.

Voir aussi

- [Guide pratique pour Créer un contrôle à liaison simple dans un Windows Form](#)
- [Validation des entrées d'utilisateur dans les Windows Forms](#)
- [Liaison de données Windows Forms](#)

Procédure : Créer un contrôle à liaison simple dans un Windows Form


3 minutes to read • [Edit Online](#)

Avec une *liaison simple*, vous pouvez afficher un élément de données unique, tel qu'une valeur de colonne à partir d'une table de DataSet, dans un contrôle. Vous pouvez facilement lier n'importe quelle propriété d'un contrôle à une valeur de données.

Pour lier un contrôle de façon simple

1. Connectez-vous à une source de données. Pour plus d'informations, consultez [connexion à une source de données](#).
2. Dans Visual Studio, sélectionnez le contrôle dans le formulaire et affichez la fenêtre **Propriétés**.
3. Développez la propriété (**DataBindings**).

Les propriétés les plus souvent liées sont affichées sous la propriété (**DataBindings**). Par exemple, dans la plupart des contrôles, la propriété **Text** est la plus souvent liée.

4. Si la propriété que vous souhaitez lier ne fait pas partie des propriétés couramment liées, cliquez sur le bouton de  Visual Studio). dans la zone (**avancé**) pour afficher la **Boîte de dialogue mise en forme et liaison avancée** avec la liste complète des propriétés de ce contrôle.
5. Sélectionnez la propriété que vous souhaitez lier, puis cliquez sur la flèche déroulante sous **liaison**.

Une liste des sources de données disponibles s'affiche.

6. Développez la source de données avec laquelle vous voulez établir une liaison jusqu'à trouver l'élément de données souhaité. Par exemple, si vous établissez une liaison à une valeur de colonne dans une table de dataset, développez le nom du dataset, puis développez le nom de la table pour afficher les noms des colonnes.
7. Cliquez sur le nom d'un élément avec lequel établir une liaison.
8. Si vous travaillez dans la boîte de dialogue **mise en forme et liaison avancée**, cliquez sur **OK** pour revenir à la fenêtre **Propriétés**.
9. Si vous souhaitez lier des propriétés supplémentaires du contrôle, répétez les étapes 3 à 7.

NOTE

Étant donné que les contrôles à liaison simple n'affichent qu'un seul élément de données, il est très courant d'inclure la logique de navigation dans un Windows Form avec des contrôles à liaison simple.

Voir aussi

- [Binding](#)
- [Liaison de données Windows Forms](#)
- [Liaison de données et Windows Forms](#)

Procédure : vérifier que plusieurs contrôles liés à la même source de données restent synchronisés

5 minutes to read • [Edit Online](#)

Souvent, lorsque vous travaillez avec la liaison de données dans les Windows Forms, plusieurs contrôles sont liés à la même source de données. Dans certains cas, il peut être nécessaire de prendre des mesures supplémentaires pour garantir que les propriétés liées des contrôles restent synchronisées entre eux et à la source de données. Ces étapes sont nécessaires dans deux situations :

- Si la source de données n'implémente pas [IBindingList](#)et ainsi générer [ListChanged](#) événements de type [ItemChanged](#).
- Si la source de données implémente [IEditableObject](#).

Dans le premier cas, vous pouvez utiliser un [BindingSource](#) pour lier la source de données aux contrôles. Dans ce cas, vous utilisez un [BindingSource](#) et gérer le [BindingComplete](#) événements et les appels [EndCurrentEdit](#) sur associé [BindingManagerBase](#).

Exemple

L'exemple de code suivant montre comment lier les trois contrôles : deux contrôles de zone de texte et un [DataGridView](#) contrôle, à la même colonne dans un [DataSet](#) à l'aide un [BindingSource](#) composant. Cet exemple montre comment gérer les [BindingComplete](#) événements et vérifiez que, quand la valeur de texte d'une zone de texte est modifiée, la zone de texte supplémentaires et la [DataGridView](#) contrôle sont mis à jour avec la valeur correcte.

L'exemple utilise un [BindingSource](#) pour lier la source de données et les contrôles. Ou bien, vous pouvez lier les contrôles directement à la source de données et récupérer la [BindingManagerBase](#) pour la liaison à partir du formulaire [BindingContext](#), puis gérer la [BindingComplete](#) événement pour le [BindingManagerBase](#). Pour obtenir un exemple de procédure à suivre, consultez la page d'aide la [BindingComplete](#) événement de [BindingManagerBase](#).

```
// Declare the controls to be used.
private BindingSource bindingSource1;
private TextBox textBox1;
private TextBox textBox2;
private DataGridView dataGridView1;

private void InitializeControlsAndDataSource()
{
    // Initialize the controls and set location, size and
    // other basic properties.
    this.dataGridView1 = new DataGridView();
    this.bindingSource1 = new BindingSource();
    this.textBox1 = new TextBox();
    this.textBox2 = new TextBox();
    this.dataGridView1.ColumnHeadersHeightSizeMode =
        DataGridViewColumnHeadersHeightSizeMode.AutoSize;
    this.dataGridView1.Dock = DockStyle.Top;
    this.dataGridView1.Location = new Point(0, 0);
    this.dataGridView1.Size = new Size(292, 150);
    this.textBox1.Location = new Point(132, 156);
    this.textBox1.Size = new Size(100, 20);
    this.textBox2.Location = new Point(12, 156);
```

```

        this.textBox2.Size = new Size(100, 20);
        this.ClientSize = new Size(292, 266);
        this.Controls.Add(this.textBox2);
        this.Controls.Add(this.textBox1);
        this.Controls.Add(this.dataGridView1);

        // Declare the DataSet and add a table and column.
        DataSet set1 = new DataSet();
        set1.Tables.Add("Menu");
        set1.Tables[0].Columns.Add("Beverages");

        // Add some rows to the table.
        set1.Tables[0].Rows.Add("coffee");
        set1.Tables[0].Rows.Add("tea");
        set1.Tables[0].Rows.Add("hot chocolate");
        set1.Tables[0].Rows.Add("milk");
        set1.Tables[0].Rows.Add("orange juice");

        // Set the data source to the DataSet.
        bindingSource1.DataSource = set1;

        //Set the DataMember to the Menu table.
        bindingSource1.DataMember = "Menu";

        // Add the control data bindings.
        dataGridView1.DataSource = bindingSource1;
        textBox1.DataBindings.Add("Text", bindingSource1,
            "Beverages", true, DataSourceUpdateMode.OnPropertyChanged);
        textBox2.DataBindings.Add("Text", bindingSource1,
            "Beverages", true, DataSourceUpdateMode.OnPropertyChanged);
        bindingSource1.BindingComplete +=
            new BindingCompleteEventHandler(bindingSource1_BindingComplete);
    }

    private void bindingSource1_BindingComplete(object sender, BindingCompleteEventArgs e)
    {
        // Check if the data source has been updated, and that no error has occurred.
        if (e.BindingCompleteContext ==
            BindingCompleteContext.DataSourceUpdate && e.Exception == null)

            // If not, end the current edit.
            e.Binding.BindingManagerBase.EndCurrentEdit();
    }

```

```

' Declare the controls to be used.
Private WithEvents bindingSource1 As BindingSource
Private WithEvents textBox1 As TextBox
Private WithEvents textBox2 As TextBox
Private WithEvents dataGridView1 As DataGridView

Private Sub InitializeControlsAndDataSource()
    ' Initialize the controls and set location, size and
    ' other basic properties.
    Me.dataGridView1 = New DataGridView()
    Me.bindingSource1 = New BindingSource()
    Me.textBox1 = New TextBox()
    Me.textBox2 = New TextBox()
    Me.dataGridView1.ColumnHeadersHeightSizeMode = DataGridViewColumnHeadersHeightSizeMode.AutoSize
    Me.dataGridView1.Dock = DockStyle.Top
    Me.dataGridView1.Location = New Point(0, 0)
    Me.dataGridView1.Size = New Size(292, 150)
    Me.textBox1.Location = New Point(132, 156)
    Me.textBox1.Size = New Size(100, 20)
    Me.textBox2.Location = New Point(12, 156)
    Me.textBox2.Size = New Size(100, 20)

```

```

Me.ClientSize = New Size(292, 266)
Me.Controls.Add(Me.textBox2)
Me.Controls.Add(Me.textBox1)
Me.Controls.Add(Me.dataGridView1)

' Declare the DataSet and add a table and column.
Dim set1 As New DataSet()
set1.Tables.Add("Menu")
set1.Tables(0).Columns.Add("Beverages")

' Add some rows to the table.
set1.Tables(0).Rows.Add("coffee")
set1.Tables(0).Rows.Add("tea")
set1.Tables(0).Rows.Add("hot chocolate")
set1.Tables(0).Rows.Add("milk")
set1.Tables(0).Rows.Add("orange juice")

' Set the data source to the DataSet.
bindingSource1.DataSource = set1

'Set the DataMember to the Menu table.
bindingSource1.DataMember = "Menu"

' Add the control data bindings.
dataGridView1.DataSource = bindingSource1
textBox1.DataBindings.Add("Text", bindingSource1, "Beverages", _
    True, DataSourceUpdateMode.OnPropertyChanged)
textBox2.DataBindings.Add("Text", bindingSource1, "Beverages", _
    True, DataSourceUpdateMode.OnPropertyChanged)

End Sub

Private Sub bindingSource1_BindingComplete(ByVal sender As Object, _
    ByVal e As BindingCompleteEventArgs) Handles bindingSource1.BindingComplete

    ' Check if the data source has been updated, and that no error has occurred.
    If e.BindingCompleteContext = BindingCompleteContext.DataSourceUpdate _
        AndAlso e.Exception Is Nothing Then

        ' If not, end the current edit.
        e.Binding.BindingManagerBase.EndCurrentEdit()
    End If

End Sub

End Sub

```

Compilation du code

- Cet exemple de code nécessite
- des références aux assemblés [System](#), [System.Windows.Forms](#) et [System.Drawing](#).
- Un formulaire avec le [Load](#) événement géré et un appel à la `InitializeControlsAndDataSource` méthode dans l'exemple à partir du formulaire [Load](#) Gestionnaire d'événements.

Voir aussi

- [Guide pratique pour Partager des données liées entre des formulaires à l'aide du composant BindingSource](#)
- [Notification de modifications dans la liaison de données Windows Forms](#)
- [Interfaces participant à la liaison de données](#)
- [Liaison de données Windows Forms](#)

Procédure : vérifier que la ligne sélectionnée dans une table enfant reste à la bonne position

14 minutes to read • [Edit Online](#)

Souvent, lors de l'utilisation de la liaison de données dans des Windows Forms, vous affichez des données dans ce que l'on appelle une vue parent/enfant ou maître/détails. Cela fait référence à un scénario de liaison de données où les données de la même source sont affichées dans deux contrôles. La modification de la sélection dans un contrôle entraîne la modification des données affichées dans le deuxième contrôle. Par exemple, le premier contrôle peut contenir une liste de clients et le second une liste de commandes associées au client sélectionné dans le premier contrôle.

À compter de .NET Framework version 2.0, quand vous affichez des données dans une vue parent/enfant, vous devrez peut-être effectuer des étapes supplémentaires pour vous assurer que la ligne actuellement sélectionnée dans la table enfant n'est pas réinitialisée à la première ligne de la table. Pour cela, vous devez mettre en cache la position de la table enfant et la réinitialiser une fois que la table parente a changé. En général, la réinitialisation de l'enfant se produit la première fois qu'un champ sur une ligne de la table parente change.

Pour mettre en cache la position actuelle de l'enfant

1. Déclarez une variable entière pour stocker la position de liste de l'enfant et une variable booléenne pour stocker une valeur indiquant s'il faut mettre en cache la position de l'enfant.

```
private int cachedPosition = -1;
private bool cacheChildPosition = true;
```

```
Private cachedPosition As Integer = - 1
Private cacheChildPosition As Boolean = True
```

2. Gérez l'événement [ListChanged](#) pour le [CurrencyManager](#) de la liaison et vérifiez s'il y a un [ListChangedType](#) égal à [Reset](#).
3. Vérifiez la position actuelle du [CurrencyManager](#). Si elle est supérieure à la première entrée de la liste (en général 0), enregistrez-la dans une variable.

```
void relatedCM_ListChanged(object sender, ListChangedEventArgs e)
{
    // Check to see if this is a caching situation.
    if (cacheChildPosition && cachePositionCheckBox.Checked)
    {
        // If so, check to see if it is a reset situation, and the current
        // position is greater than zero.
        CurrencyManager relatedCM = sender as CurrencyManager;
        if (e.ListChangedType == ListChangedType.Reset && relatedCM.Position > 0)

            // If so, cache the position of the child table.
            cachedPosition = relatedCM.Position;
    }
}
```

```

Private Sub relatedCM_ListChanged(ByVal sender As Object, _
    ByVal e As ListChangedEventArgs)
    ' Check to see if this is a caching situation.
    If cacheChildPosition AndAlso cachePositionCheckBox.Checked Then
        ' If so, check to see if it is a reset situation, and the current
        ' position is greater than zero.
        Dim relatedCM As CurrencyManager = sender
        If e.ListChangedType = ListChangedType.Reset _
            AndAlso relatedCM.Position > 0 Then

            ' If so, cache the position of the child table.
            cachedPosition = relatedCM.Position
        End If
    End If
End Sub

```

4. Gérez l'événement [CurrentChanged](#) de la liste parente pour le gestionnaire de devise parent. Dans le gestionnaire, définissez la valeur booléenne pour indiquer qu'il ne s'agit pas d'un scénario de mise en cache. Si l'événement [CurrentChanged](#) se produit, la modification du parent est une modification de position dans la liste et non une modification de valeur d'élément.

```

void bindingSource1_CurrentChanged(object sender, EventArgs e)
{
    // If the CurrentChanged event occurs, this is not a caching
    // situation.
    cacheChildPosition = false;
}

```

```

' Handle the current changed event. This event occurs when
' the current item is changed, but not when a field of the current
' item is changed.
Private Sub bindingSource1_CurrentChanged(ByVal sender As Object, _
    ByVal e As EventArgs) Handles bindingSource1.CurrentChanged
    ' If the CurrentChanged event occurs, this is not a caching
    ' situation.
    cacheChildPosition = False
End Sub

```

Pour réinitialiser la position de l'enfant

1. Gérez l'événement [PositionChanged](#) pour le [CurrencyManager](#) de la liaison enfant.
2. Réinitialisez la position de la table enfant à la position mise en cache enregistrée lors de la procédure précédente.

```

void relatedCM_PositionChanged(object sender, EventArgs e)
{
    // Check to see if this is a caching situation.
    if (cacheChildPosition && cachePositionCheckBox.Checked)
    {
        CurrencyManager relatedCM = sender as CurrencyManager;

        // If so, check to see if the current position is
        // not equal to the cached position and the cached
        // position is not out of bounds.
        if (relatedCM.Position != cachedPosition && cachedPosition
            > 0 && cachedPosition < relatedCM.Count)
        {
            relatedCM.Position = cachedPosition;
            cachedPosition = -1;
        }
    }
}

```

```

Private Sub relatedCM_PositionChanged(ByVal sender As Object, ByVal e As EventArgs)
    ' Check to see if this is a caching situation.
    If cacheChildPosition AndAlso cachePositionCheckBox.Checked Then
        Dim relatedCM As CurrencyManager = sender

        ' If so, check to see if the current position is
        ' not equal to the cached position and the cached
        ' position is not out of bounds.
        If relatedCM.Position <> cachedPosition AndAlso _
            cachedPosition > 0 AndAlso cachedPosition < _
            relatedCM.Count Then
            relatedCM.Position = cachedPosition
            cachedPosition = -1
        End If
    End If
End Sub

```

Exemple

L'exemple suivant montre comment enregistrer la position actuelle sur le [CurrencyManager](#) pour une table enfant et réinitialiser la position après qu'une modification a été effectuée sur la table parente. Cet exemple contient deux contrôles [DataGridView](#) liés à deux tables dans un [DataSet](#) à l'aide d'un composant [BindingSource](#). Une relation est établie entre les deux tables et la relation est ajoutée au [DataSet](#). La position dans la table enfant est initialement définie sur la troisième ligne à des fins de démonstration.

```

using System;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace BT2
{
    public class Form1 : Form
    {
        public Form1()
        {
            InitializeControlsAndDataSource();
        }

        // Declare the controls to be used.
        private BindingSource bindingSource1;

```

```

private DataGridView dataGridView1;
private Button button1;
private DataGridView dataGridView2;
private CheckBox cachePositionCheckBox;
public DataSet set1;

private void InitializeControlsAndDataSource()
{
    // Initialize the controls and set location, size and
    // other basic properties.
    this.dataGridView1 = new DataGridView();
    this.bindingSource1 = new BindingSource();
    this.button1 = new Button();
    this.dataGridView2 = new DataGridView();
    this.cachePositionCheckBox = new System.Windows.Forms.CheckBox();
    this.dataGridView1.ColumnHeadersHeightSizeMode =
        DataGridViewColumnHeadersHeightSizeMode.AutoSize;
    this.dataGridView1.Dock = DockStyle.Top;
    this.dataGridView1.Location = new Point(0, 20);
    this.dataGridView1.Size = new Size(292, 170);
    this.button1.Location = new System.Drawing.Point(18, 175);
    this.button1.Size = new System.Drawing.Size(125, 23);

    button1.Text = "Clear Parent Field";
    this.button1.Click += new System.EventHandler(this.button1_Click);
    this.dataGridView2.ColumnHeadersHeightSizeMode =
        System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
    this.dataGridView2.Location = new System.Drawing.Point(0, 225);
    this.dataGridView2.Size = new System.Drawing.Size(309, 130);
    this.cachePositionCheckBox.AutoSize = true;
    this.cachePositionCheckBox.Checked = true;
    this.cachePositionCheckBox.Location = new System.Drawing.Point(150, 175);
    this.cachePositionCheckBox.Name = "radioButton1";
    this.cachePositionCheckBox.Size = new System.Drawing.Size(151, 17);
    this.cachePositionCheckBox.Text = "Cache and restore position";
    this.ClientSize = new System.Drawing.Size(325, 420);
    this.Controls.Add(this.dataGridView1);
    this.Controls.Add(this.cachePositionCheckBox);
    this.Controls.Add(this.dataGridView2);
    this.Controls.Add(this.button1);

    // Initialize the data.
    set1 = InitializeDataSet();

    // Set the data source to the DataSet.
    bindingSource1.DataSource = set1;

    //Set the DataMember to the Menu table.
    bindingSource1.DataMember = "Customers";

    // Add the control data bindings.
    dataGridView1.DataSource = bindingSource1;

    // Set the data source and member for the second DataGridView.
    dataGridView2.DataSource = bindingSource1;
    dataGridView2.DataMember = "custOrders";

    // Get the currency manager for the customer orders binding.
    CurrencyManager relatedCM =
        bindingSource1.GetRelatedCurrencyManager("custOrders");

    // Set the position in the child table for demonstration purposes.
    relatedCM.Position = 3;

    // Handle the current changed event. This event occurs when
    // the current item is changed, but not when a field of the current
    // item is changed.
    bindingSource1.CurrentChanged +=
        new EventHandler(bindingSource1_CurrentChanged);

```

```

        // Handle the two events for caching and resetting the position.
        relatedCM.ListChanged += new ListChangedEventHandler(relatedCM_ListChanged);
        relatedCM.PositionChanged
            += new EventHandler(relatedCM_PositionChanged);

        // Set caching to true in case current changed event
        // occurred on set up.
        cacheChildPosition = true;
    }

    // Establish the data set with two tables and a relationship
    // between them.
    private DataSet InitializeDataSet()
    {
        set1 = new DataSet();
        // Declare the DataSet and add a table and column.
        set1.Tables.Add("Customers");
        set1.Tables[0].Columns.Add("CustomerID");
        set1.Tables[0].Columns.Add("Customer Name");
        set1.Tables[0].Columns.Add("Contact Name");

        // Add some rows to the table.
        set1.Tables["Customers"].Rows.Add("c1", "Fabrikam, Inc.", "Ellen Adams");
        set1.Tables[0].Rows.Add("c2", "Lucerne Publishing", "Don Hall");
        set1.Tables[0].Rows.Add("c3", "Northwind Traders", "Lori Penor");
        set1.Tables[0].Rows.Add("c4", "Tailspin Toys", "Michael Patten");
        set1.Tables[0].Rows.Add("c5", "Woodgrove Bank", "Jyothi Pai");

        // Declare the DataSet and add a table and column.
        set1.Tables.Add("Orders");
        set1.Tables[1].Columns.Add("CustomerID");
        set1.Tables[1].Columns.Add("OrderNo");
        set1.Tables[1].Columns.Add("OrderDate");

        // Add some rows to the table.
        set1.Tables[1].Rows.Add("c1", "119", "10/04/2006");
        set1.Tables[1].Rows.Add("c1", "149", "10/10/2006");
        set1.Tables[1].Rows.Add("c1", "159", "10/12/2006");
        set1.Tables[1].Rows.Add("c2", "169", "10/10/2006");
        set1.Tables[1].Rows.Add("c2", "179", "10/10/2006");
        set1.Tables[1].Rows.Add("c2", "189", "10/12/2006");
        set1.Tables[1].Rows.Add("c3", "122", "10/04/2006");
        set1.Tables[1].Rows.Add("c4", "130", "10/10/2006");
        set1.Tables[1].Rows.Add("c5", "1.29", "10/14/2006");

        DataRelation dr = new DataRelation("custOrders",
            set1.Tables["Customers"].Columns["CustomerID"],
            set1.Tables["Orders"].Columns["CustomerID"]);
        set1.Relations.Add(dr);
        return set1;
    }

    private int cachedPosition = -1;
    private bool cacheChildPosition = true;

    void relatedCM_ListChanged(object sender, ListChangedEventArgs e)
    {
        // Check to see if this is a caching situation.
        if (cacheChildPosition && cachePositionCheckBox.Checked)
        {
            // If so, check to see if it is a reset situation, and the current
            // position is greater than zero.
            CurrencyManager relatedCM = sender as CurrencyManager;
            if (e.ListChangedType == ListChangedType.Reset && relatedCM.Position > 0)

                // If so, cache the position of the child table.
                cachedPosition = relatedCM.Position;
        }
    }
}

```



```

    ,
    void bindingSource1_CurrentChanged(object sender, EventArgs e)
    {
        // If the CurrentChanged event occurs, this is not a caching
        // situation.
        cacheChildPosition = false;
    }
    void relatedCM_PositionChanged(object sender, EventArgs e)
    {
        // Check to see if this is a caching situation.
        if (cacheChildPosition && cachePositionCheckBox.Checked)
        {
            CurrencyManager relatedCM = sender as CurrencyManager;

            // If so, check to see if the current position is
            // not equal to the cached position and the cached
            // position is not out of bounds.
            if (relatedCM.Position != cachedPosition && cachedPosition
                > 0 && cachedPosition < relatedCM.Count)
            {
                relatedCM.Position = cachedPosition;
                cachedPosition = -1;
            }
        }
    }
    int count = 0;
    private void button1_Click(object sender, EventArgs e)
    {
        // For demo purposes--modifies a value in the first row of the
        // parent table.
        DataRow row1 = set1.Tables[0].Rows[0];
        row1[1] = DBNull.Value;
    }

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

```

```

Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    Public Sub New()
        InitializeControlsAndDataSource()

    End Sub

    ' Declare the controls to be used.
    Private WithEvents bindingSource1 As BindingSource
    Private dataGridView1 As DataGridView
    Private WithEvents button1 As Button
    Private dataGridView2 As DataGridView
    Private cachePositionCheckBox As CheckBox
    Public set1 As DataSet

```

```

Private Sub InitializeControlsAndDataSource()
    ' Initialize the controls and set location, size and
    ' other basic properties.
    Me.dataGridView1 = New DataGridView()
    Me.bindingSource1 = New BindingSource()
    Me.button1 = New Button()
    Me.dataGridView2 = New DataGridView()
    Me.cachePositionCheckBox = New System.Windows.Forms.CheckBox()
    Me.dataGridView1.ColumnHeadersHeightSizeMode = _
        DataGridViewColumnHeadersHeightSizeMode.AutoSize
    Me.dataGridView1.Dock = DockStyle.Top
    Me.dataGridView1.Location = New Point(0, 20)
    Me.dataGridView1.Size = New Size(292, 170)
    Me.button1.Location = New System.Drawing.Point(18, 175)
    Me.button1.Size = New System.Drawing.Size(125, 23)

    button1.Text = "Clear Parent Field"

    Me.dataGridView2.ColumnHeadersHeightSizeMode = _
        System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize
    Me.dataGridView2.Location = New System.Drawing.Point(0, 225)
    Me.dataGridView2.Size = New System.Drawing.Size(309, 130)
    Me.cachePositionCheckBox.AutoSize = True
    Me.cachePositionCheckBox.Checked = True
    Me.cachePositionCheckBox.Location = New System.Drawing.Point(150, 175)
    Me.cachePositionCheckBox.Name = "radioButton1"
    Me.cachePositionCheckBox.Size = New System.Drawing.Size(151, 17)
    Me.cachePositionCheckBox.Text = "Cache and restore position"
    Me.ClientSize = New System.Drawing.Size(325, 420)
    Me.Controls.Add(Me.dataGridView1)
    Me.Controls.Add(Me.cachePositionCheckBox)
    Me.Controls.Add(Me.dataGridView2)
    Me.Controls.Add(Me.button1)

    ' Initialize the data.
    set1 = InitializeDataSet()

    ' Set the data source to the DataSet.
    bindingSource1.DataSource = set1

    'Set the DataMember to the Menu table.
    bindingSource1.DataMember = "Customers"

    ' Add the control data bindings.
    dataGridView1.DataSource = bindingSource1

    ' Set the data source and member for the second DataGridView.
    dataGridView2.DataSource = bindingSource1
    dataGridView2.DataMember = "custOrders"

    ' Get the currency manager for the customer orders binding.
    Dim relatedCM As CurrencyManager = _
        bindingSource1.GetRelatedCurrencyManager("custOrders")

    ' Handle the two events for caching and resetting the position.
    AddHandler relatedCM.ListChanged, AddressOf relatedCM_ListChanged
    AddHandler relatedCM.PositionChanged, AddressOf relatedCM_PositionChanged

    ' Set the position in the child table for demonstration purposes.
    relatedCM.Position = 3

    ' Set cacheing to true in case current changed event
    ' occurred on set up.
    cacheChildPosition = True

```

End Sub

```

' Establish the data set with two tables and a relationship
' between them.
Private Function InitializeDataSet() As DataSet
    set1 = New DataSet()
    ' Declare the DataSet and add a table and column.
    set1.Tables.Add("Customers")
    set1.Tables(0).Columns.Add("CustomerID")
    set1.Tables(0).Columns.Add("Customer Name")
    set1.Tables(0).Columns.Add("Contact Name")

    ' Add some rows to the table.
    set1.Tables("Customers").Rows.Add("c1", "Fabrikam, Inc.", _
        "Ellen Adams")
    set1.Tables(0).Rows.Add("c2", "Lucerne Publishing", "Don Hall")
    set1.Tables(0).Rows.Add("c3", "Northwind Traders", "Lori Penor")
    set1.Tables(0).Rows.Add("c4", "Tailspin Toys", "Michael Patten")
    set1.Tables(0).Rows.Add("c5", "Woodgrove Bank", "Jyothi Pai")

    ' Declare the DataSet and add a table and column.
    set1.Tables.Add("Orders")
    set1.Tables(1).Columns.Add("CustomerID")
    set1.Tables(1).Columns.Add("OrderNo")
    set1.Tables(1).Columns.Add("OrderDate")

    ' Add some rows to the table.
    set1.Tables(1).Rows.Add("c1", "119", "10/04/2006")
    set1.Tables(1).Rows.Add("c1", "149", "10/10/2006")
    set1.Tables(1).Rows.Add("c1", "159", "10/12/2006")
    set1.Tables(1).Rows.Add("c2", "169", "10/10/2006")
    set1.Tables(1).Rows.Add("c2", "179", "10/10/2006")
    set1.Tables(1).Rows.Add("c2", "189", "10/12/2006")
    set1.Tables(1).Rows.Add("c3", "122", "10/04/2006")
    set1.Tables(1).Rows.Add("c4", "130", "10/10/2006")
    set1.Tables(1).Rows.Add("c5", "1.29", "10/14/2006")

    Dim dr As New DataRelation("custOrders", _
        set1.Tables("Customers").Columns("CustomerID"), _
        set1.Tables("Orders").Columns("CustomerID"))
    set1.Relations.Add(dr)
    Return set1

End Function '
Private cachedPosition As Integer = - 1
Private cacheChildPosition As Boolean = True

Private Sub relatedCM_ListChanged(ByVal sender As Object, _
    ByVal e As ListChangedEventArgs)
    ' Check to see if this is a caching situation.
    If cacheChildPosition AndAlso cachePositionCheckBox.Checked Then
        ' If so, check to see if it is a reset situation, and the current
        ' position is greater than zero.
        Dim relatedCM As CurrencyManager = sender
        If e.ListChangedType = ListChangedType.Reset _
            AndAlso relatedCM.Position > 0 Then

            ' If so, cache the position of the child table.
            cachedPosition = relatedCM.Position
        End If
    End If

End Sub

' Handle the current changed event. This event occurs when
' the current item is changed, but not when a field of the current
' item is changed.
Private Sub bindingSource1_CurrentChanged(ByVal sender As Object, _
    ByVal e As EventArgs) Handles bindingSource1_CurrentChanged

```

```

        ByVal e As EventArgs) Handles BindingSource1.CurrentChanged
        ' If the CurrentChanged event occurs, this is not a caching
        ' situation.
        cacheChildPosition = False

    End Sub

    Private Sub relatedCM_PositionChanged(ByVal sender As Object, ByVal e As EventArgs)
        ' Check to see if this is a caching situation.
        If cacheChildPosition AndAlso cachePositionCheckBox.Checked Then
            Dim relatedCM As CurrencyManager = sender

            ' If so, check to see if the current position is
            ' not equal to the cached position and the cached
            ' position is not out of bounds.
            If relatedCM.Position <> cachedPosition AndAlso _
                cachedPosition > 0 AndAlso cachedPosition < _
                relatedCM.Count Then
                relatedCM.Position = cachedPosition
                cachedPosition = -1
            End If
        End If
    End Sub

    Private count As Integer = 0

    Private Sub button1_Click(ByVal sender As Object, _
        ByVal e As EventArgs) Handles button1.Click
        ' For demo purposes--modifies a value in the first row of the
        ' parent table.
        Dim row1 As DataRow = set1.Tables(0).Rows(0)
        row1(1) = DBNull.Value
    End Sub

    <STAThread> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.SetCompatibleTextRenderingDefault(False)
        Application.Run(New Form1())
    End Sub
End Class

```

Pour tester l'exemple de code, procédez comme suit :

1. Exécutez l'exemple.
2. Assurez-vous que la case **Cache et réinitialisation de position** est cochée.
3. Cliquez sur le bouton **Effacer le champ parent** pour provoquer une modification dans un champ de la table parent. Notez que la ligne sélectionnée dans la table enfant ne change pas.
4. Fermez puis réexécutez l'exemple. Cette opération est nécessaire car le comportement de réinitialisation se produit uniquement lors de la première modification sur la ligne parente.
5. Décochez la case **Cache et réinitialisation de position**.
6. Cliquez sur le bouton **Effacer le champ parent**. Notez que la ligne sélectionnée dans la table enfant passe en première ligne.

Compilation du code

Cet exemple nécessite :

- Références aux assemblés System, System.Data, System.Drawing, System.Windows.Forms et System.Xml.

Voir aussi

- [Guide pratique pour S'assurer que plusieurs contrôles liés à la même Source de données restent synchronisés](#)
- [BindingSource, composant](#)
- [Liaison de données et Windows Forms](#)

Procédure : implémenter l'interface IListSource

7 minutes to read • [Edit Online](#)

Implémenter le [IListSource](#) interface permettant de créer une classe pouvant être liée qui n'implémente pas [IList](#) mais plutôt une liste à partir d'un autre emplacement.

Exemple

L'exemple de code suivant montre comment implémenter la [IListSource](#) interface. Un composant nommé

`EmployeeListSource` expose un [IList](#) pour la liaison de données en implémentant le [GetList](#) (méthode).

```
using System;
using System.Collections.Generic;
using System.Text;
using System.ComponentModel;

namespace IListSourceCS
{
    public class EmployeeListSource : Component, IListSource
    {
        public EmployeeListSource() {}

        public EmployeeListSource(IContainer container)
        {
            container.Add(this);
        }

        #region IListSource Members

        bool IListSource.ContainsListCollection
        {
            get { return false; }
        }

        System.Collections.IList IListSource.GetList()
        {
            BindingList<Employee> ble = new BindingList<Employee>();

            if (!this.DesignMode)
            {
                ble.Add(new Employee("Aaberg, Jesper", 26000000));
                ble.Add(new Employee("Cajhen, Janko", 19600000));
                ble.Add(new Employee("Furse, Kari", 19000000));
                ble.Add(new Employee("Langhorn, Carl", 16000000));
                ble.Add(new Employee("Todorov, Teodor", 15700000));
                ble.Add(new Employee("Verebélyi, Ágnes", 15700000));
            }

            return ble;
        }

        #endregion
    }
}
```

Imports System.ComponentModel

Public Class EmployeeListSource
Inherits Component

```

Implements IListSource

<System.Diagnostics.DebuggerNonUserCode()> _
Public Sub New(ByVal Container As System.ComponentModel.IContainer)
    MyClass.New()

    'Required for Windows.Forms Class Composition Designer support
    Container.Add(Me)

End Sub

<System.Diagnostics.DebuggerNonUserCode()> _
Public Sub New()
    MyBase.New()

    'This call is required by the Component Designer.
    InitializeComponent()

End Sub

'Component overrides dispose to clean up the component list.
<System.Diagnostics.DebuggerNonUserCode()> _
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing AndAlso components IsNot Nothing Then
        components.Dispose()
    End If
    MyBase.Dispose(disposing)
End Sub

'Required by the Component Designer
Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Component Designer
'It can be modified using the Component Designer.
'Do not modify it using the code editor.
<System.Diagnostics.DebuggerStepThrough()> _
Private Sub InitializeComponent()
    components = New System.ComponentModel.Container()
End Sub

#Region "IListSource Members"

    Public ReadOnly Property ContainsListCollection() As Boolean Implements
System.ComponentModel.IListSource.ContainsListCollection
        Get
            Return False
        End Get
    End Property

    Public Function GetList() As System.Collections.IList Implements System.ComponentModel.IListSource.GetList

        Dim ble As New BindingList(Of Employee)

        If Not Me.DesignMode Then
            ble.Add(New Employee("Aaberg, Jesper", 26000000))
            ble.Add(New Employee("Cajhen, Janko", 19600000))
            ble.Add(New Employee("Furse, Kari", 19000000))
            ble.Add(New Employee("Langhorn, Carl", 16000000))
            ble.Add(New Employee("Todorov, Teodor", 15700000))
            ble.Add(New Employee("Verebelyi, Ágnes", 15700000))
        End If

        Return ble

    End Function

#End Region

End Class

```

```

using System;
using System.Collections.Generic;
using System.Text;
using System.ComponentModel;

namespace IListSourceCS
{
    public class Employee : BusinessObjectBase
    {
        private string      _id;
        private string      _name;
        private Decimal      parkingId;

        public Employee() : this(string.Empty, 0) {}
        public Employee(string name) : this(name, 0) {}

        public Employee(string name, Decimal parkingId) : base()
        {
            this._id = System.Guid.NewGuid().ToString();

            // Set values
            this.Name = name;
            this.ParkingID = parkingId;
        }

        public string ID
        {
            get { return _id; }
        }

        const string NAME = "Name";
        public string Name
        {
            get { return _name; }
            set
            {
                if (_name != value)
                {
                    _name = value;

                    // Raise the PropertyChanged event.
                    OnPropertyChanged(NAME);
                }
            }
        }

        const string PARKING_ID = "Salary";
        public Decimal ParkingID
        {
            get { return parkingId; }
            set
            {
                if (parkingId != value)
                {
                    parkingId = value;

                    // Raise the PropertyChanged event.
                    OnPropertyChanged(PARKING_ID);
                }
            }
        }
    }
}

```



```

Imports System.ComponentModel

Public Class Employee
    Inherits BusinessObjectBase

    Private _id As String
    Private _name As String
    Private _parkingId As Decimal

    Public Sub New(ByVal name As String, ByVal parkId As Decimal)
        MyBase.New()
        Me._id = System.Guid.NewGuid().ToString()
        ' Set values
        Me.Name = name
        Me.ParkingID = parkId
    End Sub

    Public ReadOnly Property ID() As String
        Get
            Return _id
        End Get
    End Property

    Const NAME_Const As String = "Name"

    Public Property Name() As String
        Get
            Return _name
        End Get
        Set(ByVal value As String)
            If _name <> value Then
                _name = value
                OnPropertyChanged(NAME_Const)
            End If
        End Set
    End Property

    Const PARKINGID_Const As String = "ParkingID"

    Public Property ParkingID() As Decimal
        Get
            Return _parkingId
        End Get
        Set(ByVal value As Decimal)
            If _parkingId <> value Then
                _parkingId = value
                OnPropertyChanged(PARKINGID_Const)
            End If
        End Set
    End Property

End Class

```

```

using System;
using System.Collections.Generic;
using System.Text;
using System.ComponentModel;
using System.Diagnostics;

namespace IListSourceCS
{
    public class BusinessObjectBase : INotifyPropertyChanged
    {
        #region INotifyPropertyChanged Members

        public event PropertyChangedEventHandler PropertyChanged;

        protected virtual void OnPropertyChanged(string propertyName)
        {
            OnPropertyChanged(new PropertyChangedEventArgs(propertyName));
        }

        private void OnPropertyChanged(PropertyChangedEventArgs e)
        {
            if (null != PropertyChanged)
            {
                PropertyChanged(this, e);
            }
        }

        #endregion
    }
}

```

```

Imports System.ComponentModel

Public Class BusinessObjectBase
    Implements INotifyPropertyChanged

    #Region "INotifyPropertyChanged Members"

    Public Event PropertyChanged(ByVal sender As Object, ByVal e As
System.ComponentModel.PropertyChangedEventArgs) Implements
System.ComponentModel.INotifyPropertyChanged.PropertyChanged

    Protected Overridable Sub OnPropertyChanged(ByVal propertyName As String)
        OnPropertyChanged(New PropertyChangedEventArgs(propertyName))
    End Sub

    Private Sub OnPropertyChanged(ByVal e As PropertyChangedEventArgs)
        RaiseEvent PropertyChanged(Me, e)
    End Sub

    #End Region

End Class

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace IListSourceCS
{
    public class BusinessObjectBase : INotifyPropertyChanged
    {
        #region INotifyPropertyChanged Members

```

```

public class Form1 : Form
{
    private System.ComponentModel.IContainer components = null;
    private FlowLayoutPanel flowLayoutPanel1;
    private Label label2;
    private DataGridView dataGridView1;
    private DataGridViewTextBoxColumn nameDataGridViewTextBoxColumn;
    private DataGridViewTextBoxColumn salaryDataGridViewTextBoxColumn;
    private DataGridViewTextBoxColumn idDataGridViewTextBoxColumn;
    private EmployeeListSource employeeListSource1;

    public Form1()
    {
        InitializeComponent();
    }

    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    #region Windows Form Designer generated code

    private void InitializeComponent()
    {
        this.components = new System.ComponentModel.Container();
        System.Windows.Forms.DataGridViewCellStyle dataGridViewCellStyle1 = new
System.Windows.Forms.DataGridViewCellStyle();
        System.Windows.Forms.DataGridViewCellStyle dataGridViewCellStyle2 = new
System.Windows.Forms.DataGridViewCellStyle();
        this.flowLayoutPanel1 = new System.Windows.Forms.FlowLayoutPanel();
        this.label2 = new System.Windows.Forms.Label();
        this.dataGridView1 = new System.Windows.Forms.DataGridView();
        this.nameDataGridViewTextBoxColumn = new System.Windows.Forms.DataGridViewTextBoxColumn();
        this.salaryDataGridViewTextBoxColumn = new System.Windows.Forms.DataGridViewTextBoxColumn();
        this.idDataGridViewTextBoxColumn = new System.Windows.Forms.DataGridViewTextBoxColumn();
        this.employeeListSource1 = new EmployeeListSource(this.components);
        this.flowLayoutPanel1.SuspendLayout();
        ((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).BeginInit();
        this.SuspendLayout();
        //
        // flowLayoutPanel1
        //
        this.flowLayoutPanel1.AutoSize = true;
        this.flowLayoutPanel1.Controls.Add(this.label2);
        this.flowLayoutPanel1.Dock = System.Windows.Forms.DockStyle.Top;
        this.flowLayoutPanel1.Location = new System.Drawing.Point(0, 0);
        this.flowLayoutPanel1.Name = "flowLayoutPanel1";
        this.flowLayoutPanel1.Size = new System.Drawing.Size(416, 51);
        this.flowLayoutPanel1.TabIndex = 11;
        //
        // label2
        //
        this.label2.AutoSize = true;
        this.label2.Location = new System.Drawing.Point(3, 6);
        this.label2.Margin = new System.Windows.Forms.Padding(3, 6, 3, 6);
        this.label2.Name = "label2";
        this.label2.Size = new System.Drawing.Size(408, 39);
        this.label2.TabIndex = 0;
        this.label2.Text = "This sample demonstrates how to implement the IListSource interface. In this
sam" +
        "ple, a DataGridView is bound at design time to a Component (employeeListSource1)" +
        " that implements IListSource.";
        //
        // dataGridView1

```

```

//
this.dataGridView1.AllowUserToAddRows = false;
this.dataGridView1.AllowUserToDeleteRows = false;
dataGridViewCellStyle1.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(255)))), ((int)
(((byte)(255)))), ((int)(((byte)(192))))));
this.dataGridView1.AlternatingRowsDefaultCellStyle = dataGridViewCellStyle1;
this.dataGridView1.AutoGenerateColumns = false;
this.dataGridView1.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
this.dataGridView1.Columns.AddRange(new System.Windows.Forms.DataGridViewColumn[] {
this.nameDataGridViewTextBoxColumn,
this.salaryDataGridViewTextBoxColumn,
this.iDDataGridViewTextBoxColumn});
this.dataGridView1.DataSource = this.employeeListSource1;
this.dataGridView1.Dock = System.Windows.Forms.DockStyle.Fill;
this.dataGridView1.Location = new System.Drawing.Point(0, 51);
this.dataGridView1.Name = "dataGridView1";
this.dataGridView1.RowHeadersVisible = false;
this.dataGridView1.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
this.dataGridView1.Size = new System.Drawing.Size(416, 215);
this.dataGridView1.TabIndex = 12;
//
// nameDataGridViewTextBoxColumn
//
this.nameDataGridViewTextBoxColumn.DataPropertyName = "Name";
this.nameDataGridViewTextBoxColumn.FillWeight = 131.7987F;
this.nameDataGridViewTextBoxColumn.HeaderText = "Name";
this.nameDataGridViewTextBoxColumn.Name = "nameDataGridViewTextBoxColumn";
//
// salaryDataGridViewTextBoxColumn
//
this.salaryDataGridViewTextBoxColumn.DataPropertyName = "ParkingID";
this.salaryDataGridViewTextBoxColumn.DefaultCellStyle = dataGridViewCellStyle2;
this.salaryDataGridViewTextBoxColumn.FillWeight = 121.8274F;
this.salaryDataGridViewTextBoxColumn.HeaderText = "Parking ID";
this.salaryDataGridViewTextBoxColumn.Name = "salaryDataGridViewTextBoxColumn";
//
// iDDataGridViewTextBoxColumn
//
this.iDDataGridViewTextBoxColumn.AutoSizeMode =
System.Windows.Forms.DataGridViewAutoSizeColumnMode.Fill;
this.iDDataGridViewTextBoxColumn.DataPropertyName = "ID";
this.iDDataGridViewTextBoxColumn.FillWeight = 46.37391F;
this.iDDataGridViewTextBoxColumn.HeaderText = "ID";
this.iDDataGridViewTextBoxColumn.Name = "iDDataGridViewTextBoxColumn";
this.iDDataGridViewTextBoxColumn.ReadOnly = true;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(416, 266);
this.Controls.Add(this.dataGridView1);
this.Controls.Add(this.flowLayoutPanel1);
this.Name = "Form1";
this.Text = "IListSource Sample";
this.flowLayoutPanel1.ResumeLayout(false);
this.flowLayoutPanel1.PerformLayout();
((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).EndInit();
this.ResumeLayout(false);
this.PerformLayout();
}

#endregion

}

static class Program
{
    [STAThread]

```

```

        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}

```

```

Imports System.ComponentModel
Imports System.Windows.Forms

```

```
Public Class Form1
```

```
    Inherits System.Windows.Forms.Form
```

```
    Friend WithEvents flowLayoutPanel1 As FlowLayoutPanel
```

```
    Friend WithEvents label2 As Label
```

```
    Friend WithEvents dataGridView1 As DataGridView
```

```
    Friend WithEvents nameDataGridViewTextBoxColumn As DataGridViewTextBoxColumn
```

```
    Friend WithEvents salaryDataGridViewTextBoxColumn As DataGridViewTextBoxColumn
```

```
    Friend WithEvents idDataGridViewTextBoxColumn As DataGridViewTextBoxColumn
```

```
    Friend WithEvents employeeListSource1 As EmployeeListSource
```

```
    Public Sub New()
```

```
        MyBase.New()
```

```
        Me.InitializeComponent()
```

```
    End Sub
```

```
    'Form overrides dispose to clean up the component list.
```

```
    <System.Diagnostics.DebuggerNonUserCode()> _
```

```
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
```

```
        If disposing AndAlso components IsNot Nothing Then
            components.Dispose()

```

```
        End If
```

```
        MyBase.Dispose(disposing)
```

```
    End Sub
```

```
    'Required by the Windows Form Designer
```

```
    Private components As System.ComponentModel.IContainer
```

```
    'NOTE: The following procedure is required by the Windows Form Designer
```

```
    'It can be modified using the Windows Form Designer.
```

```
    'Do not modify it using the code editor.
```

```
    <System.Diagnostics.DebuggerStepThrough()> _
```

```
    Private Sub InitializeComponent()
```

```
        components = New System.ComponentModel.Container()
```

```
        Dim dataGridViewCellStyle1 = New System.Windows.Forms.DataGridViewCellStyle()
```

```
        Dim dataGridViewCellStyle2 = New System.Windows.Forms.DataGridViewCellStyle()
```

```
        Me.flowLayoutPanel1 = New System.Windows.Forms.FlowLayoutPanel()
```

```
        Me.label2 = New System.Windows.Forms.Label()
```

```
        Me.dataGridView1 = New System.Windows.Forms.DataGridView()
```

```
        Me.nameDataGridViewTextBoxColumn = New System.Windows.Forms.DataGridViewTextBoxColumn()
```

```
        Me.salaryDataGridViewTextBoxColumn = New System.Windows.Forms.DataGridViewTextBoxColumn()
```

```
        Me.idDataGridViewTextBoxColumn = New System.Windows.Forms.DataGridViewTextBoxColumn()
```

```
        Me.employeeListSource1 = New EmployeeListSource(Me.components)
```

```
        Me.flowLayoutPanel1.SuspendLayout()
```

```
        CType(Me.dataGridView1, System.ComponentModel.ISupportInitialize).BeginInit()
```

```
        Me.SuspendLayout()
```

```
        '
```

```
        ' flowLayoutPanel1
```

```
        '
```

```
        Me.flowLayoutPanel1.AutoSize = True
```

```
        Me.flowLayoutPanel1.Controls.Add(Me.label2)
```

```
        Me.flowLayoutPanel1.Dock = System.Windows.Forms.DockStyle.Top
```

```
        Me.flowLayoutPanel1.Location = New System.Drawing.Point(0, 0)
```

```

Me.flowLayoutPanel1.Name = "flowLayoutPanel1"
Me.flowLayoutPanel1.Size = New System.Drawing.Size(416, 51)
Me.flowLayoutPanel1.TabIndex = 11
'
' label2
'
Me.label2.AutoSize = True
Me.label2.Location = New System.Drawing.Point(3, 6)
Me.label2.Margin = New System.Windows.Forms.Padding(3, 6, 3, 6)
Me.label2.Name = "label2"
Me.label2.Size = New System.Drawing.Size(408, 39)
Me.label2.TabIndex = 0
Me.label2.Text = "This sample demonstrates how to implement the IListSource interface. In this sam" +
-
    "ple, a DataGridView is bound at design time to a Component (employeeListSource1)" + _
    " that implements IListSource."
'
' dataGridView1
'
Me.dataGridView1.AllowUserToAddRows = False
Me.dataGridView1.AllowUserToDeleteRows = False
dataGridViewCellStyle1.BackColor = System.Drawing.Color.FromArgb(255, 255, 192)
Me.dataGridView1.AlternatingRowsDefaultCellStyle = dataGridViewCellStyle1
Me.dataGridView1.AutoGenerateColumns = False
Me.dataGridView1.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize
Me.dataGridView1.Columns.AddRange(New System.Windows.Forms.DataGridViewColumn() { _
Me.nameDataGridViewTextBoxColumn, Me.salaryDataGridViewTextBoxColumn, Me.iDDataGridViewTextBoxColumn})
Me.dataGridView1.DataSource = Me.employeeListSource1
Me.dataGridView1.Dock = System.Windows.Forms.DockStyle.Fill
Me.dataGridView1.Location = New System.Drawing.Point(0, 51)
Me.dataGridView1.Name = "dataGridView1"
Me.dataGridView1.RowHeadersVisible = False
Me.dataGridView1.SelectionMode = System.Windows.Forms.DataGridViewSelectionMode.FullRowSelect
Me.dataGridView1.Size = New System.Drawing.Size(416, 215)
Me.dataGridView1.TabIndex = 12
'
' nameDataGridViewTextBoxColumn
'
Me.nameDataGridViewTextBoxColumn.DataPropertyName = "Name"
Me.nameDataGridViewTextBoxColumn.FillWeight = 131.7987F
Me.nameDataGridViewTextBoxColumn.HeaderText = "Name"
Me.nameDataGridViewTextBoxColumn.Name = "nameDataGridViewTextBoxColumn"
'
' salaryDataGridViewTextBoxColumn
'
Me.salaryDataGridViewTextBoxColumn.DataPropertyName = "ParkingID"
Me.salaryDataGridViewTextBoxColumn.DefaultCellStyle = dataGridViewCellStyle2
Me.salaryDataGridViewTextBoxColumn.FillWeight = 121.8274F
Me.salaryDataGridViewTextBoxColumn.HeaderText = "Parking ID"
Me.salaryDataGridViewTextBoxColumn.Name = "salaryDataGridViewTextBoxColumn"
'
' iDDataGridViewTextBoxColumn
'
Me.iDDataGridViewTextBoxColumn.AutoSizeMode = System.Windows.Forms.DataGridViewAutoSizeColumnMode.Fill
Me.iDDataGridViewTextBoxColumn.DataPropertyName = "ID"
Me.iDDataGridViewTextBoxColumn.FillWeight = 46.37391F
Me.iDDataGridViewTextBoxColumn.HeaderText = "ID"
Me.iDDataGridViewTextBoxColumn.Name = "iDDataGridViewTextBoxColumn"
Me.iDDataGridViewTextBoxColumn.ReadOnly = True
'
' Form1
'
Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0F, 13.0F)
Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
Me.ClientSize = New System.Drawing.Size(416, 266)
Me.Controls.Add(Me.dataGridView1)
Me.Controls.Add(Me.flowLayoutPanel1)
Me.Name = "Form1"

```

```
Me.Text = "IListSource Sample"
Me.flowLayoutPanel1.ResumeLayout(False)
Me.flowLayoutPanel1.PerformLayout()
CType(Me.dataGridView1, System.ComponentModel.ISupportInitialize).EndInit()
Me.ResumeLayout(False)
Me.PerformLayout()

End Sub

Shared Sub Main()
    Application.Run(New Form1())
End Sub

End Class
```

Compilation du code

Cet exemple nécessite :

- Références aux assemblies System.Drawing et System.Windows.Forms.

Voir aussi

- [IListSource](#)
- [ITypedList](#)
- [BindingList<T>](#)
- [IBindingList](#)
- [Liaison de données et Windows Forms](#)

Procédure : implémenter l'interface INotifyPropertyChanged

5 minutes to read • [Edit Online](#)

L'exemple de code suivant montre comment implémenter la [INotifyPropertyChanged](#) interface. Implémentez cette interface sur les objets métier qui sont utilisés dans la liaison de données Windows Forms. En cas d'implémentation, l'interface communique à un contrôle dépendant les modifications des propriétés sur un objet métier.

Exemple

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Runtime.CompilerServices;
using System.Windows.Forms;

// Change the namespace to the project name.
namespace TestNotifyPropertyChangedCS
{
    // This form demonstrates using a BindingSource to bind
    // a list to a DataGridView control. The list does not
    // raise change notifications. However the DemoCustomer type
    // in the list does.
    public partial class Form1 : Form
    {
        // This button causes the value of a list element to be changed.
        private Button changeItemBtn = new Button();

        // This DataGridView control displays the contents of the list.
        private DataGridView customersDataGridView = new DataGridView();

        // This BindingSource binds the list to the DataGridView control.
        private BindingSource customersBindingSource = new BindingSource();

        public Form1()
        {
            InitializeComponent();

            // Set up the "Change Item" button.
            this.changeItemBtn.Text = "Change Item";
            this.changeItemBtn.Dock = DockStyle.Bottom;
            this.changeItemBtn.Click +=
                new EventHandler(changeItemBtn_Click);
            this.Controls.Add(this.changeItemBtn);

            // Set up the DataGridView.
            customersDataGridView.Dock = DockStyle.Top;
            this.Controls.Add(customersDataGridView);

            this.Size = new Size(400, 200);
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Create and populate the list of DemoCustomer objects
            // which will supply data to the DataGridView.
            BindingList<DemoCustomer> customerList = new BindingList<DemoCustomer>();
```



```

        customerList.Add(DemoCustomer.CreateNewCustomer());
        customerList.Add(DemoCustomer.CreateNewCustomer());
        customerList.Add(DemoCustomer.CreateNewCustomer());

        // Bind the list to the BindingSource.
        this.customersBindingSource.DataSource = customerList;

        // Attach the BindingSource to the DataGridView.
        this.customersDataGridView.DataSource =
            this.customersBindingSource;
    }

    // Change the value of the CompanyName property for the first
    // item in the list when the "Change Item" button is clicked.
    void changeItemBtn_Click(object sender, EventArgs e)
    {
        // Get a reference to the list from the BindingSource.
        BindingList<DemoCustomer> customerList =
            this.customersBindingSource.DataSource as BindingList<DemoCustomer>;

        // Change the value of the CompanyName property for the
        // first item in the list.
        customerList[0].CustomerName = "Tailspin Toys";
        customerList[0].PhoneNumber = "(708)555-0150";
    }
}

// This is a simple customer class that
// implements the IPropertyChange interface.
public class DemoCustomer : INotifyPropertyChanged
{
    // These fields hold the values for the public properties.
    private Guid idValue = Guid.NewGuid();
    private string customerNameValue = String.Empty;
    private string phoneNumberValue = String.Empty;

    public event PropertyChangedEventHandler PropertyChanged;

    // This method is called by the Set accessor of each property.
    // The CallerMemberName attribute that is applied to the optional propertyName
    // parameter causes the property name of the caller to be substituted as an argument.
    private void NotifyPropertyChanged([CallerMemberName] String propertyName = "")
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }

    // The constructor is private to enforce the factory pattern.
    private DemoCustomer()
    {
        customerNameValue = "Customer";
        phoneNumberValue = "(312)555-0100";
    }

    // This is the public factory method.
    public static DemoCustomer CreateNewCustomer()
    {
        return new DemoCustomer();
    }

    // This property represents an ID, suitable
    // for use as a primary key in a database.
    public Guid ID
    {
        get
        {
            return this.idValue;

```

```

    }
}

public string CustomerName
{
    get
    {
        return this.customerNameValue;
    }

    set
    {
        if (value != this.customerNameValue)
        {
            this.customerNameValue = value;
            NotifyPropertyChanged();
        }
    }
}

public string PhoneNumber
{
    get
    {
        return this.phoneNumberValue;
    }

    set
    {
        if (value != this.phoneNumberValue)
        {
            this.phoneNumberValue = value;
            NotifyPropertyChanged();
        }
    }
}
}
}
}

```

```

Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Drawing
Imports System.Runtime.CompilerServices
Imports System.Windows.Forms

' This form demonstrates using a BindingSource to bind
' a list to a DataGridView control. The list does not
' raise change notifications. However the DemoCustomer type
' in the list does.

Public Class Form1
    Inherits System.Windows.Forms.Form
    ' This button causes the value of a list element to be changed.
    Private changeItemBtn As New Button()

    ' This DataGridView control displays the contents of the list.
    Private customersDataGridView As New DataGridView()

    ' This BindingSource binds the list to the DataGridView control.
    Private customersBindingSource As New BindingSource()

    Public Sub New()
        InitializeComponent()

        ' Set up the "Change Item" button.
        Me.changeItemBtn.Text = "Change Item"
        Me.changeItemBtn.Dock = DockStyle.Bottom
    End Sub
End Class

```

```

AddHandler Me.changeItemBtn.Click, AddressOf changeItemBtn_Click
Me.Controls.Add(Me.changeItemBtn)

' Set up the DataGridView.
customersDataGridView.Dock = DockStyle.Top
Me.Controls.Add(customersDataGridView)

Me.Size = New Size(400, 200)
End Sub

Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Me.Load

    ' Create and populate the list of DemoCustomer objects
    ' which will supply data to the DataGridView.
    Dim customerList As New BindingList(Of DemoCustomer)

    customerList.Add(DemoCustomer.CreateNewCustomer())
    customerList.Add(DemoCustomer.CreateNewCustomer())
    customerList.Add(DemoCustomer.CreateNewCustomer())

    ' Bind the list to the BindingSource.
    Me.customersBindingSource.DataSource = customerList

    ' Attach the BindingSource to the DataGridView.
    Me.customersDataGridView.DataSource = Me.customersBindingSource
End Sub

' This event handler changes the value of the CompanyName
' property for the first item in the list.
Private Sub changeItemBtn_Click(ByVal sender As Object, ByVal e As EventArgs)
    ' Get a reference to the list from the BindingSource.
    Dim customerList As BindingList(Of DemoCustomer) = _
        CType(customersBindingSource.DataSource, BindingList(Of DemoCustomer))

    ' Change the value of the CompanyName property for the
    ' first item in the list.
    customerList(0).CustomerName = "Tailspin Toys"
    customerList(0).PhoneNumber = "(708)555-0150"
End Sub
End Class

' This class implements a simple customer type
' that implements the IPropertyChange interface.
Public Class DemoCustomer
    Implements INotifyPropertyChanged

    ' These fields hold the values for the public properties.
    Private idValue As Guid = Guid.NewGuid()
    Private customerNameValue As String = String.Empty
    Private phoneNumberValue As String = String.Empty

    Public Event PropertyChanged As PropertyChangedEventHandler _
        Implements INotifyPropertyChanged.PropertyChanged

    ' This method is called by the Set accessor of each property.
    ' The CallerMemberName attribute that is applied to the optional propertyName
    ' parameter causes the property name of the caller to be substituted as an argument.
    Private Sub NotifyPropertyChanged(<CallerMemberName()> Optional ByVal propertyName As String = Nothing)
        RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs(propertyName))
    End Sub

    ' The constructor is private to enforce the factory pattern.
    Private Sub New()
        customerNameValue = "Customer"
        phoneNumberValue = "(312)555-0100"
    End Sub

    ' This is the public factory method.

```

```

' This is the proper way to create a new customer
Public Shared Function CreateNewCustomer() As DemoCustomer
    Return New DemoCustomer()
End Function

' This property represents an ID, suitable
' for use as a primary key in a database.
Public ReadOnly Property ID() As Guid
    Get
        Return Me.idValue
    End Get
End Property

Public Property CustomerName() As String
    Get
        Return Me.customerNameValue
    End Get

    Set(ByVal value As String)
        If Not (value = customerNameValue) Then
            Me.customerNameValue = value
            NotifyPropertyChanged()
        End If
    End Set
End Property

Public Property PhoneNumber() As String
    Get
        Return Me.phoneNumberValue
    End Get

    Set(ByVal value As String)
        If Not (value = phoneNumberValue) Then
            Me.phoneNumberValue = value
            NotifyPropertyChanged()
        End If
    End Set
End Property
End Class

```

Voir aussi

- [Guide pratique pour Appliquer le modèle PropertyChanged](#)
- [Liaison de données Windows Forms](#)
- [Guide pratique pour Générer des Notifications de modification à l'aide d'un BindingSource et l'Interface INotifyPropertyChanged](#)
- [Notification de modifications dans la liaison de données Windows Forms](#)

Procédure : implémenter l'interface IListed

9 minutes to read • [Edit Online](#)

Implémentez le [IListed](#) interface pour activer la découverte du schéma pour une liste pouvant être liée.

Exemple

L'exemple de code suivant montre comment implémenter la [IListed](#) interface. Un type générique nommé `SortableBindingList` dérive le `BindingList<T>` classe et implémente la [IListed](#) interface. Une classe simple nommée `Customer` fournit des données, qui sont liées à l'en-tête d'un `DataGridView` contrôle.

```

using System;
using System.Collections.Generic;
using System.Text;
using System.ComponentModel;
using System.Windows.Forms;
using System.Collections;
using System.Reflection;

namespace ITypedListCS
{
    [Serializable()]
    public class SortableBindingList<T> : BindingList<T>, ITypedList
    {
        [NonSerialized()]
        private PropertyDescriptorCollection properties;

        public SortableBindingList() : base()
        {
            // Get the 'shape' of the list.
            // Only get the public properties marked withBrowsable = true.
            PropertyDescriptorCollection pdc = TypeDescriptor.GetProperties(
                typeof(T),
                new Attribute[] { new BrowsableAttribute(true) });

            // Sort the properties.
            properties = pdc.Sort();
        }

        #region ITypedList Implementation

        public PropertyDescriptorCollection GetItemProperties(PropertyDescriptor[] listAccessors)
        {
            PropertyDescriptorCollection pdc;

            if (listAccessors != null && listAccessors.Length > 0)
            {
                // Return child list shape.
                pdc = ListBindingHelper.GetListItemProperties(listAccessors[0].PropertyType);
            }
            else
            {
                // Return properties in sort order.
                pdc = properties;
            }

            return pdc;
        }

        // This method is only used in the design-time framework
        // and by the obsolete DataGrid control.
        public string GetListName(PropertyDescriptor[] listAccessors)
        {
            return typeof(T).Name;
        }

        #endregion
    }
}

```

```

Imports System.ComponentModel
Imports System.Collections.Generic
Imports System.Windows.Forms

<Serializable()> _
Public Class SortableBindingList(Of TKey)
    Inherits BindingList(Of TKey)
    Implements IList

    <NonSerialized()> _
    Private properties As PropertyDescriptorCollection

    Public Sub New()
        MyBase.New()

        ' Get the 'shape' of the list.
        ' Only get the public properties marked withBrowsable = true.
        Dim pdc As PropertyDescriptorCollection = TypeDescriptor.GetProperties(GetType(TKey), New Attribute()
{NewBrowsableAttribute(True)})

        ' Sort the properties.
        properties = pdc.Sort()

    End Sub

#Region "IList Implementation"

    Public Function GetItemProperties(ByVal listAccessors() As System.ComponentModel.PropertyDescriptor) As
System.ComponentModel.PropertyDescriptorCollection Implements
System.ComponentModel.IList.GetItemProperties

        Dim pdc As PropertyDescriptorCollection

        If (Not (listAccessors Is Nothing)) And (listAccessors.Length > 0) Then
            ' Return child list shape
            pdc = ListBindingHelper.GetListItemProperties(listAccessors(0).PropertyType)
        Else
            ' Return properties in sort order
            pdc = properties
        End If

        Return pdc

    End Function

    ' This method is only used in the design-time framework
    ' and by the obsolete DataGridView control.
    Public Function GetListName( _
        ByVal listAccessors() As PropertyDescriptor) As String _
        Implements System.ComponentModel.IList.GetListName

        Return GetType(TKey).Name

    End Function

#End Region

End Class

```

```

using System;
using System.Collections.Generic;
using System.Text;
using System.ComponentModel;

namespace IListCS
{

```

```

class Customer : INotifyPropertyChanged
{
    public Customer() {}

    public Customer(int id, string name, string company, string address, string city, string state, string
zip)
    {
        this._id = id;
        this._name = name;
        this._company = company;
        this._address = address;
        this._city = city;
        this._state = state;
        this._zip = zip;
    }

    #region Public Properties

    private int _id;

    public int ID
    {
        get { return _id; }
        set
        {
            if (_id != value)
            {
                _id = value;
                OnPropertyChanged(new PropertyChangedEventArgs("ID"));
            }
        }
    }

    private string _name;

    public string Name
    {
        get { return _name; }
        set
        {
            if (_name != value)
            {
                _name = value;
                OnPropertyChanged(new PropertyChangedEventArgs("Name"));
            }
        }
    }

    private string _company;

    public string Company
    {
        get { return _company; }
        set
        {
            if (_company != value)
            {
                _company = value;
                OnPropertyChanged(new PropertyChangedEventArgs("Company"));
            }
        }
    }

    private string _address;

    public string Address
    {
        get { return _address; }
        set
    }

```



```

        {
            if (_address != value)
            {
                _address = value;
                OnPropertyChanged(new PropertyChangedEventArgs("Address"));
            }
        }
    }

    private string _city;

    public string City
    {
        get { return _city; }
        set
        {
            if (_city != value)
            {
                _city = value;
                OnPropertyChanged(new PropertyChangedEventArgs("City"));
            }
        }
    }

    private string _state;

    public string State
    {
        get { return _state; }
        set
        {
            if (_state != value)
            {
                _state = value;
                OnPropertyChanged(new PropertyChangedEventArgs("State"));
            }
        }
    }

    private string _zip;

    public string ZipCode
    {
        get { return _zip; }
        set
        {
            if (_zip != value)
            {
                _zip = value;
                OnPropertyChanged(new PropertyChangedEventArgs("ZipCode"));
            }
        }
    }

    #endregion

    #region INotifyPropertyChanged Members

    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged(PropertyChangedEventArgs e)
    {
        if (null != PropertyChanged)
        {
            PropertyChanged(this, e);
        }
    }

    #endregion

```

```
}  
}
```

```
Imports System.ComponentModel  
  
Public Class Customer  
    Implements INotifyPropertyChanged  
  
    Public Sub New()  
  
    End Sub  
  
    Public Sub New(ByVal id As Integer, ByVal name As String, ByVal company As String, ByVal address As  
String, ByVal city As String, ByVal state As String, ByVal zip As String)  
        Me._id = id  
        Me._name = name  
        Me._company = company  
        Me._address = address  
        Me._city = city  
        Me._state = state  
        Me._zip = zip  
  
    End Sub  
  
    #Region "Public Properties"  
  
    Private _id As Integer  
    Public Property ID() As Integer  
        Get  
            Return _id  
        End Get  
        Set(ByVal value As Integer)  
            If _id <> value Then  
                _id = value  
                OnPropertyChanged(New PropertyChangedEventArgs("ID"))  
            End If  
        End Set  
    End Property  
  
    Private _name As String  
  
    Public Property Name() As String  
        Get  
            Return _name  
        End Get  
        Set(ByVal value As String)  
            If _name <> value Then  
                _name = value  
                OnPropertyChanged(New PropertyChangedEventArgs("Name"))  
            End If  
        End Set  
    End Property  
  
    Private _company As String  
  
    Public Property Company() As String  
        Get  
            Return _company  
        End Get  
        Set(ByVal value As String)  
            If _company <> value Then  
                _company = value  
                OnPropertyChanged(New PropertyChangedEventArgs("Company"))  
            End If  
        End Set  
    End Property
```

```
Private _address As String
```

```
Public Property Address() As String
```

```
Get
```

```
Return _address
```

```
End Get
```

```
Set(ByVal value As String)
```

```
If _address <> value Then
```

```
_address = value
```

```
OnPropertyChanged(New PropertyChangedEventArgs("Address"))
```

```
End If
```

```
End Set
```

```
End Property
```

```
Private _city As String
```

```
Public Property City() As String
```

```
Get
```

```
Return _city
```

```
End Get
```

```
Set(ByVal value As String)
```

```
If _city <> value Then
```

```
_city = value
```

```
OnPropertyChanged(New PropertyChangedEventArgs("City"))
```

```
End If
```

```
End Set
```

```
End Property
```

```
Private _state As String
```

```
Public Property State() As String
```

```
Get
```

```
Return _state
```

```
End Get
```

```
Set(ByVal value As String)
```

```
If _state <> value Then
```

```
_state = value
```

```
OnPropertyChanged(New PropertyChangedEventArgs("State"))
```

```
End If
```

```
End Set
```

```
End Property
```

```
Private _zip As String
```

```
Public Property ZipCode() As String
```

```
Get
```

```
Return _zip
```

```
End Get
```

```
Set(ByVal value As String)
```

```
If _zip <> value Then
```

```
_zip = value
```

```
OnPropertyChanged(New PropertyChangedEventArgs("ZipCode"))
```

```
End If
```

```
End Set
```

```
End Property
```

```
#End Region
```

```
#Region "INotifyPropertyChanged Members"
```

```
Public Event PropertyChanged(ByVal sender As Object, ByVal e As  
System.ComponentModel.PropertyChangedEventArgs) Implements  
System.ComponentModel.INotifyPropertyChanged.PropertyChanged
```

```
Protected Overridable Sub OnPropertyChanged(ByVal e As PropertyChangedEventArgs)
```

```

        RaiseEvent PropertyChanged(Me, e)
    End Sub

#End Region

End Class

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace ITypedListCS
{
    public partial class Form1 : Form
    {
        private SortableBindingList<Customer> sortableBindingListOfCustomers;
        private BindingList<Customer> bindingListOfCustomers;

        private System.ComponentModel.IContainer components = null;
        private System.Windows.Forms.FlowLayoutPanel flowLayoutPanel1;
        private System.Windows.Forms.Label label2;
        private DataGridView dataGridView1;
        private Button button1;
        private Button button2;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.sortableBindingListOfCustomers = new SortableBindingList<Customer>();
            this.bindingListOfCustomers = new BindingList<Customer>();

            this.dataGridView1.DataSource = this.bindingListOfCustomers;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.dataGridView1.DataSource = null;
            this.dataGridView1.DataSource = this.sortableBindingListOfCustomers;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.dataGridView1.DataSource = null;
            this.dataGridView1.DataSource = this.bindingListOfCustomers;
        }

        protected override void Dispose(bool disposing)
        {
            {
                if (disposing && (components != null))
                {
                    components.Dispose();
                }
                base.Dispose(disposing);
            }
        }

        #region Windows Form Designer generated code

        private void InitializeComponent()
        {

```

```

        System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form1));
        this.flowLayoutPanel1 = new System.Windows.Forms.FlowLayoutPanel();
        this.label2 = new System.Windows.Forms.Label();
        this.dataGridView1 = new System.Windows.Forms.DataGridView();
        this.button1 = new System.Windows.Forms.Button();
        this.button2 = new System.Windows.Forms.Button();
        this.flowLayoutPanel1.SuspendLayout();
        ((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).BeginInit();
        this.SuspendLayout();
        //
        // flowLayoutPanel1
        //
        this.flowLayoutPanel1.AutoSize = true;
        this.flowLayoutPanel1.Controls.Add(this.label2);
        this.flowLayoutPanel1.Dock = System.Windows.Forms.DockStyle.Top;
        this.flowLayoutPanel1.Location = new System.Drawing.Point(0, 0);
        this.flowLayoutPanel1.Name = "flowLayoutPanel1";
        this.flowLayoutPanel1.Size = new System.Drawing.Size(566, 51);
        this.flowLayoutPanel1.TabIndex = 13;
        //
        // label2
        //
        this.label2.AutoSize = true;
        this.label2.Location = new System.Drawing.Point(3, 6);
        this.label2.Margin = new System.Windows.Forms.Padding(3, 6, 3, 6);
        this.label2.Name = "label2";
        this.label2.Size = new System.Drawing.Size(558, 39);
        this.label2.TabIndex = 0;
        this.label2.Text = "This sample demonstrates how to implement the ITypedList interface. Clicking
on the 'Sort Columns' button will bind the DataGridView to a sub-classed BindingList<T> that implements
ITypedList to provide a sorted list of columns. Clicking on the 'Reset' button will bind the DataGridView to
a normal BindingList<T>.";
        //
        // dataGridView1
        //
        this.dataGridView1.AllowUserToAddRows = false;
        this.dataGridView1.AllowUserToDeleteRows = false;
        this.dataGridView1.Anchor = ((System.Windows.Forms.AnchorStyles)
((((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Bottom)
| System.Windows.Forms.AnchorStyles.Left)
| System.Windows.Forms.AnchorStyles.Right)));
        this.dataGridView1.AutoSizeColumnsMode =
System.Windows.Forms.DataGridViewAutoSizeColumnsMode.Fill;
        this.dataGridView1.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
        this.dataGridView1.Location = new System.Drawing.Point(6, 57);
        this.dataGridView1.Name = "dataGridView1";
        this.dataGridView1.ReadOnly = true;
        this.dataGridView1.RowHeadersVisible = false;
        this.dataGridView1.Size = new System.Drawing.Size(465, 51);
        this.dataGridView1.TabIndex = 14;
        //
        // button1
        //
        this.button1.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top
| System.Windows.Forms.AnchorStyles.Right)));
        this.button1.Location = new System.Drawing.Point(477, 57);
        this.button1.Name = "button1";
        this.button1.Size = new System.Drawing.Size(82, 23);
        this.button1.TabIndex = 15;
        this.button1.Text = "Sort Columns";
        this.button1.UseVisualStyleBackColor = true;
        this.button1.Click += new System.EventHandler(this.button1_Click);
        //
        // button2
        //
        this.button2.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top
| System.Windows.Forms.AnchorStyles.Right)));

```

```

        this.button2.Location = new System.Drawing.Point(477, 86);
        this.button2.Name = "button2";
        this.button2.Size = new System.Drawing.Size(82, 23);
        this.button2.TabIndex = 16;
        this.button2.Text = "Reset";
        this.button2.UseVisualStyleBackColor = true;
        this.button2.Click += new System.EventHandler(this.button2_Click);
        //
        // Form1
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(566, 120);
        this.Controls.Add(this.button2);
        this.Controls.Add(this.button1);
        this.Controls.Add(this.dataGridView1);
        this.Controls.Add(this.flowLayoutPanel1);
        this.Name = "Form1";
        this.Text = "ITypedList Sample";
        this.Load += new System.EventHandler(this.Form1_Load);
        this.flowLayoutPanel1.ResumeLayout(false);
        this.flowLayoutPanel1.PerformLayout();
        ((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).EndInit();
        this.ResumeLayout(false);
        this.PerformLayout();

    }

    #endregion
}

static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
}
}

```

```

Imports System.ComponentModel
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    Friend WithEvents flowLayoutPanel1 As FlowLayoutPanel
    Friend WithEvents label2 As System.Windows.Forms.Label
    Friend WithEvents dataGridView1 As DataGridView
    Friend WithEvents button1 As Button
    Friend WithEvents button2 As Button

    Dim sortableBindingListOfCustomers As SortableBindingList(Of Customer)
    Dim bindingListOfCustomers As BindingList(Of Customer)

    Public Sub New()
        MyBase.New()

        Me.InitializeComponent()
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

```

```

        sortableBindingListOfCustomers = New SortableBindingList(Of Customer)()
        bindingListOfCustomers = New BindingList(Of Customer)()

        Me.dataGridView1.DataSource = bindingListOfCustomers
    End Sub

    Private Sub button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
button1.Click
        Me.dataGridView1.DataSource = Nothing
        Me.dataGridView1.DataSource = sortableBindingListOfCustomers
    End Sub

    Private Sub button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
button2.Click
        Me.dataGridView1.DataSource = Nothing
        Me.dataGridView1.DataSource = bindingListOfCustomers
    End Sub

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing AndAlso components IsNot Nothing Then
            components.Dispose()
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Dim resources As System.ComponentModel.ComponentResourceManager = New
System.ComponentModel.ComponentResourceManager(GetType(Form1))
        Me.flowLayoutPanel1 = New System.Windows.Forms.FlowLayoutPanel
        Me.label2 = New System.Windows.Forms.Label
        Me.dataGridView1 = New System.Windows.Forms.DataGridView
        Me.button1 = New System.Windows.Forms.Button
        Me.button2 = New System.Windows.Forms.Button
        Me.flowLayoutPanel1.SuspendLayout()
        CType(Me.dataGridView1, System.ComponentModel.ISupportInitialize).BeginInit()
        Me.SuspendLayout()
        '
        'flowLayoutPanel1
        '
        Me.flowLayoutPanel1.AutoSize = True
        Me.flowLayoutPanel1.Controls.Add(Me.label2)
        Me.flowLayoutPanel1.Dock = System.Windows.Forms.DockStyle.Top
        Me.flowLayoutPanel1.Location = New System.Drawing.Point(0, 0)
        Me.flowLayoutPanel1.Name = "flowLayoutPanel1"
        Me.flowLayoutPanel1.Size = New System.Drawing.Size(566, 51)
        Me.flowLayoutPanel1.TabIndex = 13
        '
        'label2
        '
        Me.label2.AutoSize = True
        Me.label2.Location = New System.Drawing.Point(3, 6)
        Me.label2.Margin = New System.Windows.Forms.Padding(3, 6, 3, 6)
        Me.label2.Name = "label2"
        Me.label2.Size = New System.Drawing.Size(558, 39)
        Me.label2.TabIndex = 0
        Me.label2.Text = "This sample demonstrates how to implement the ITypedList interface. Clicking on the
'Sort Columns' button will bind the DataGridView to a sub-classed BindingList<T> that implements ITypedlist to

```

Sort Columns button will bind the DataGridView to a sub-classed BindingList<T> that implements ITypedList to provide a sorted list of columns. Clicking on the 'Reset' button will bind the DataGridView to a normal BindingList<T>."

```
,
    'dataGridView1
,
    Me.dataGridView1.AllowUserToAddRows = False
    Me.dataGridView1.AllowUserToDeleteRows = False
    Me.dataGridView1.Anchor = CType((((System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom) _
        Or System.Windows.Forms.AnchorStyles.Left) _
        Or System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
    Me.dataGridView1.AutoSizeColumnsMode = System.Windows.Forms.DataGridViewAutoSizeColumnsMode.Fill
    Me.dataGridView1.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize
    Me.dataGridView1.Location = New System.Drawing.Point(6, 57)
    Me.dataGridView1.Name = "dataGridView1"
    Me.dataGridView1.ReadOnly = True
    Me.dataGridView1.RowHeadersVisible = False
    Me.dataGridView1.Size = New System.Drawing.Size(465, 51)
    Me.dataGridView1.TabIndex = 14
,
    'button1
,
    Me.button1.Anchor = CType((System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
    Me.button1.Location = New System.Drawing.Point(477, 57)
    Me.button1.Name = "button1"
    Me.button1.Size = New System.Drawing.Size(82, 23)
    Me.button1.TabIndex = 15
    Me.button1.Text = "Sort Columns"
    Me.button1.UseVisualStyleBackColor = True
,
    'button2
,
    Me.button2.Location = New System.Drawing.Point(477, 86)
    Me.button2.Name = "button2"
    Me.button2.Size = New System.Drawing.Size(82, 23)
    Me.button2.TabIndex = 16
    Me.button2.Text = "Reset"
    Me.button2.UseVisualStyleBackColor = True
,
    'Form1
,
    Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
    Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
    Me.ClientSize = New System.Drawing.Size(566, 120)
    Me.Controls.Add(Me.button2)
    Me.Controls.Add(Me.button1)
    Me.Controls.Add(Me.dataGridView1)
    Me.Controls.Add(Me.flowLayoutPanel1)
    Me.Name = "Form1"
    Me.Text = "ITypedList Sample"
    Me.flowLayoutPanel1.ResumeLayout(False)
    Me.flowLayoutPanel1.PerformLayout()
    CType(Me.dataGridView1, System.ComponentModel.ISupportInitialize).EndInit()
    Me.ResumeLayout(False)
    Me.PerformLayout()
End Sub

Shared Sub Main()
    Application.Run(New Form1())
End Sub

End Class
```


Compilation du code

Cet exemple nécessite :

- Références aux assemblys System.Drawing et System.Windows.Forms.

Voir aussi

- [ITypedList](#)
- [BindingList<T>](#)
- [IBindingList](#)
- [Liaison de données et Windows Forms](#)

Comment : naviguer au sein des données dans les Windows Forms

4 minutes to read • [Edit Online](#)

Dans une application Windows, le moyen le plus simple de parcourir les enregistrements d'une source de données consiste à lier un composant [BindingSource](#) à la source de données, puis à lier les contrôles à la [BindingSource](#). Vous pouvez ensuite utiliser la méthode de navigation intégrée sur le [BindingSource](#) un tel [MoveNext](#), [MoveLast](#), [MovePrevious](#) et [MoveFirst](#). L'utilisation de ces méthodes ajuste les propriétés [Position](#) et [Current](#) de la [BindingSource](#) de manière appropriée. Vous pouvez également rechercher un élément et le définir en tant qu'élément actuel en définissant la propriété [Position](#).

Pour incrémenter la position dans une source de données

1. Définissez la propriété [Position](#) du [BindingSource](#) pour vos données liées à la position d'enregistrement à atteindre. L'exemple suivant illustre l'utilisation de la méthode [MoveNext](#) de la [BindingSource](#) pour incrémenter la propriété [Position](#) lorsque l'utilisateur clique sur le `nextButton`. Le [BindingSource](#) est associé à la table `Customers` d'un `Northwind` de DataSet.

NOTE

La définition de la propriété [Position](#) sur une valeur au-delà du premier ou du dernier enregistrement n'entraîne pas d'erreur, car le .NET Framework ne vous permet pas de définir la position sur une valeur en dehors des limites de la liste. Si, dans votre application, il est important de savoir si vous avez dépassé le premier ou le dernier enregistrement, incluez la logique permettant de tester si vous dépassez le nombre d'éléments de données.

```
private void nextButton_Click(object sender, System.EventArgs e)
{
    this.customersBindingSource.MoveNext();
}
```

```
Private Sub nextButton_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles nextButton.Click
    Me.customersBindingSource.MoveNext()
End Sub
```

Pour vérifier si vous avez dépassé la fin ou le début

1. Créez un gestionnaire d'événements pour l'événement [PositionChanged](#). Dans le gestionnaire, vous pouvez vérifier si la valeur de position proposée a dépassé le nombre réel d'éléments de données.

L'exemple suivant montre comment vous pouvez vérifier si vous avez atteint le dernier élément de données. Dans l'exemple, si vous êtes au dernier élément, le bouton **suivant** du formulaire est désactivé.

NOTE

Sachez que, si vous modifiez la liste que vous naviguez dans le code, vous devez réactiver le bouton **suivant** afin que les utilisateurs puissent parcourir la totalité de la nouvelle liste. En outre, sachez que l'événement [PositionChanged](#) ci-dessus pour le [BindingSource](#) spécifique avec lequel vous travaillez doit être associé à sa méthode de gestion d'événements. Voici un exemple de méthode de gestion de l'événement [PositionChanged](#) :

```

void customersBindingSource_PositionChanged(object sender, EventArgs e)
{
    if (customersBindingSource.Position == customersBindingSource.Count - 1)
        nextButton.Enabled = false;
    else
        nextButton.Enabled = true;
}

```

```

Sub customersBindingSource_PositionChanged(ByVal sender As Object, _
    ByVal e As EventArgs)

    If customersBindingSource.Position = _
        customersBindingSource.Count - 1 Then
        nextButton.Enabled = False
    Else
        nextButton.Enabled = True
    End If
End Sub

```

Pour rechercher un élément et le définir comme élément actuel

1. Recherchez l'enregistrement que vous souhaitez définir en tant qu'élément actuel. Vous pouvez le faire à l'aide de la méthode [Find](#) de la [BindingSource](#), si votre source de données implémente [IBindingList](#). [BindingList<T>](#) et [DataView](#) sont des exemples de sources de données qui implémentent [IBindingList](#).

```

void findButton_Click(object sender, EventArgs e)
{
    int foundIndex = customersBindingSource.Find("CustomerID", "ANTON");
    customersBindingSource.Position = foundIndex;
}

```

```

Sub findButton_Click(ByVal sender As Object, ByVal e As EventArgs) _
    Handles findButton.Click
    Dim foundIndex As Integer = customersBindingSource.Find("CustomerID", _
        "ANTON")
    customersBindingSource.Position = foundIndex
End Sub

```

Voir aussi

- [Sources de données prises en charge par les Windows Forms](#)
- [Notification de modifications dans la liaison de données Windows Forms](#)
- [Liaison de données et Windows Forms](#)
- [Liaison de données Windows Forms](#)

Sécurité de Windows Forms

3 minutes to read • [Edit Online](#)

Windows Forms propose un modèle de sécurité basé sur du code (les niveaux de sécurité sont définis pour le code, quel que soit l'utilisateur qui exécute le code). Cela s'ajoute à tous les schémas de sécurité qui peuvent être déjà en place sur votre système informatique. Ceux-ci peuvent inclure ceux du navigateur (par exemple, la sécurité basée sur une zone disponible dans Internet Explorer) ou le système d'exploitation (par exemple, la sécurité basée sur les informations d'identification de Windows NT).

Dans cette section

[Vue d'ensemble de la sécurité dans les Windows Forms](#)

Explique brièvement le modèle de sécurité .NET Framework et les étapes de base nécessaires pour garantir la sécurité des Windows Forms dans votre application.

[Accès plus sécurisé aux fichiers et aux données dans les Windows Forms](#)

Décrit comment accéder aux fichiers et aux données dans un environnement de confiance partielle.

[Impression plus sécurisée dans les Windows Forms](#)

Décrit comment accéder aux fonctionnalités d'impression dans un environnement de confiance partielle.

[Considérations supplémentaires sur la sécurité des Windows Forms](#)

Décrit comment manipuler des fenêtres, utiliser le presse-papiers et effectuer des appels au code non managé dans un environnement de confiance partielle.

Sections connexes

[Stratégie de sécurité par défaut](#)

Répertorie les autorisations par défaut accordées dans les jeux d'autorisations confiance totale, Intranet local et Internet.

[Administration générale de la stratégie de sécurité](#)

Donne des informations sur l'administration de la stratégie de sécurité .NET Framework et l'élévation d'autorisations.

[Autorisations et administration de stratégie dangereuses](#)

Présente quelques-unes des autorisations de l'infrastructure the.NET qui peuvent permettre le contournement du système de sécurité.

[Instructions de codage sécurisé](#)

Liens vers des rubriques qui expliquent les meilleures pratiques pour écrire du code en toute sécurité sur le .NET Framework.

[Demande d'autorisations](#)

Traite de l'utilisation d'attributs pour permettre au runtime de savoir quelles autorisations votre code doit exécuter.

[Concepts fondamentaux sur la sécurité](#)

Liens vers des rubriques qui traitent des aspects fondamentaux de la sécurité du code.

[Notions fondamentales de la sécurité d'accès du code](#)

Décrit les principes de base de l'utilisation de la stratégie de sécurité .NET Framework Runtime.

[Détermination de la modification de la stratégie de sécurité](#)

Explique comment déterminer si vos applications doivent s'écarter de la stratégie de sécurité par défaut.

Déploiement de la stratégie de sécurité

Décrit la meilleure façon de déployer les modifications de stratégie de sécurité.

Vue d'ensemble de la sécurité dans les Windows Forms

18 minutes to read • [Edit Online](#)

Avant la publication de la .NET Framework, tout le code qui s'exécute sur l'ordinateur d'un utilisateur avait les mêmes droits ou autorisations pour accéder aux ressources que l'utilisateur de l'ordinateur avait. Par exemple, si l'utilisateur était autorisé à accéder au système de fichiers, le code était autorisé à accéder au système de fichiers. Si l'utilisateur était autorisé à accéder à une base de données, le code était autorisé à accéder à cette base de données. Bien que ces droits ou autorisations puissent être acceptables pour le code dans les exécutables que l'utilisateur a explicitement installé sur l'ordinateur local, ils peuvent ne pas être acceptables pour le code potentiellement malveillant provenant d'Internet ou d'un intranet local. Ce code ne doit pas pouvoir accéder aux ressources de l'ordinateur de l'utilisateur sans autorisation.

Le .NET Framework introduit une infrastructure appelée sécurité d'accès du code qui vous permet de différencier les autorisations, ou droits, que ce code a des droits de l'utilisateur. Par défaut, le code provenant d'Internet et de l'intranet peut uniquement s'exécuter dans un mode qui porte le nom de confiance partielle. La confiance partielle soumet une application à une série de restrictions : entre autres, une application ne peut pas accéder au disque dur local et ne peut pas exécuter de code non managé. Le .NET Framework contrôle les ressources auxquelles le code est autorisé à accéder en fonction de l'identité de ce code : son origine, s'il a des [assemblys avec nom fort](#), s'il est signé avec un certificat, et ainsi de suite.

La technologie ClickOnce, qui vous permet de déployer des applications Windows Forms, facilite le développement d'applications qui s'exécutent en confiance partielle, en confiance totale ou en confiance partielle avec des autorisations élevées. ClickOnce fournit des fonctionnalités telles que l'élévation d'autorisations et le déploiement d'applications approuvées pour que votre application puisse demander une confiance totale ou des autorisations élevées à l'utilisateur local de manière responsable.

Présentation de la sécurité dans .NET Framework

La sécurité d'accès du code permet au code d'avoir un niveau de confiance à différents degrés, en fonction de son origine et d'autres aspects de son identité. Pour plus d'informations sur les preuves utilisées par le Common Language Runtime afin de déterminer la stratégie de sécurité, consultez [Preuve](#). Elle vous aide à protéger les systèmes contre les programmes malveillants et à empêcher que le code approuvé ne compromette la sécurité de manière intentionnelle ou accidentelle. La sécurité d'accès du code vous permet aussi de mieux contrôler les actions que votre application peut effectuer, car vous pouvez spécifier uniquement les autorisations dont elle a besoin. La sécurité d'accès du code affecte tout le code managé qui cible le Common Language Runtime, même si ce code ne fait aucune vérification d'autorisation de sécurité d'accès du code. Pour plus d'informations sur la sécurité dans le .NET Framework, consultez [concepts de sécurité clés](#) et notions de base de la [sécurité d'accès du code](#).

Si l'utilisateur exécute un fichier exécutable Windows Forms directement à partir d'un serveur web ou d'un partage de fichiers, le degré de confiance accordé à votre application dépend de l'emplacement du code et de la façon dont il est démarré. Quand une application s'exécute, elle est évaluée automatiquement et le Common Language Runtime lui affecte un jeu d'autorisations nommé. Par défaut, le code sur l'ordinateur local dispose du jeu d'autorisations Confiance totale, le code provenant d'un réseau local dispose du jeu d'autorisations Intranet local et le code provenant d'Internet dispose du jeu d'autorisations Internet.

NOTE

Dans la .NET Framework version 1,0 Service Pack 1 et Service Pack 2, le groupe de codes de la zone Internet reçoit le jeu d'autorisations rien. Dans toutes les autres versions de la .NET Framework, le groupe de codes de la zone Internet reçoit le jeu d'autorisations Internet.

Les autorisations par défaut accordées dans chacun de ces jeux d'autorisations sont répertoriées dans la rubrique [Stratégie de sécurité par défaut](#). En fonction des autorisations reçues par l'application, elle s'exécute correctement ou génère une exception de sécurité.

De nombreuses applications de Windows Forms seront déployées à l'aide de ClickOnce. Les outils utilisés pour générer un déploiement ClickOnce ont des paramètres de sécurité par défaut différents de ceux abordés précédemment. Pour plus d'informations, voir la discussion suivante.

Les autorisations réellement accordées à votre application peuvent être différentes des valeurs par défaut, car la stratégie de sécurité peut être modifiée. Cela signifie que votre application peut avoir une autorisation sur un ordinateur, mais pas sur un autre.

Développement d'une application Windows Forms plus sécurisée

La sécurité est importante lors de toutes les étapes du développement d'application. Commencez par examiner et suivre les [Instructions de codage sécurisé](#).

Ensuite, décidez si votre application doit s'exécuter avec une confiance totale ou partielle. L'exécution de votre application avec une confiance totale facilite l'accès aux ressources sur l'ordinateur local, mais expose votre application et ses utilisateurs à des risques de sécurité élevés si vous ne concevez et ne développez pas votre application en respectant strictement les consignes fournies dans la rubrique Instructions de codage sécurisé. L'exécution de votre application avec une confiance partielle simplifie le développement d'une application plus sécurisée et réduit considérablement les risques, mais nécessite davantage de planification quant à l'implémentation de certaines fonctionnalités.

Si vous choisissez la confiance partielle (autrement dit, le jeu d'autorisation Internet ou Intranet local), choisissez comment votre application doit se comporter dans cet environnement. Windows Forms fournit des méthodes alternatives et plus sûres pour implémenter des fonctionnalités dans un environnement de confiance partielle. Certaines parties de votre application, comme l'accès aux données, peuvent être conçues et écrites différemment pour les environnements de confiance partielle et de confiance totale. Certaines fonctionnalités de Windows Forms, telles que les paramètres d'application, sont conçues pour fonctionner en mode de confiance partielle. Pour plus d'informations, consultez [Vue d'ensemble des paramètres d'application](#)

Si votre application a besoin de davantage d'autorisations que celles offertes par la confiance partielle mais que vous ne souhaitez pas l'exécuter avec une confiance totale, vous pouvez l'exécuter avec une confiance partielle en déclarant uniquement les autorisations supplémentaires dont vous avez besoin. Par exemple, si vous souhaitez l'exécuter avec une confiance partielle mais que vous devez l'autoriser à accéder en lecture seule à un répertoire dans le système de fichiers de l'utilisateur, vous pouvez demander [FileIOPermission](#) uniquement pour ce répertoire. Utilisée correctement, cette approche peut procurer à votre application des fonctionnalités supplémentaires et minimiser les risques de sécurité pour vos utilisateurs.

Quand vous développez une application qui s'exécute avec une confiance partielle, effectuez le suivi des autorisations dont votre application doit disposer et de celles qu'elle pourrait éventuellement utiliser. Quand toutes les autorisations sont connues, vous devez effectuer une demande déclarative pour disposer de l'autorisation au niveau de l'application. La demande d'autorisations indique au .NET Framework moment de l'exécution les autorisations dont votre application a besoin et les autorisations qu'elle ne souhaite pas spécifiquement. Pour plus d'informations sur les demandes d'autorisations, consultez [Demande d'autorisations](#).

Quand vous demandez des autorisations facultatives, vous devez gérer les exceptions de sécurité qui seront

générées si votre application effectue une action qui nécessite des autorisations qui ne lui ont pas été accordées. Une gestion appropriée de [SecurityException](#) garantit que votre application peut continuer à fonctionner. Votre application peut utiliser l'exception pour déterminer si une fonctionnalité doit être désactivée pour l'utilisateur. Par exemple, une application peut désactiver l'option de menu **Enregistrer** si l'autorisation de fichier nécessaire n'est pas accordée.

Il est parfois difficile de savoir si vous avez déclaré toutes les autorisations requises. Par exemple, un appel de méthode apparemment anodin peut accéder au système de fichiers à un moment donné de son exécution. Si vous ne déployez pas votre application avec toutes les autorisations requises, les tests peuvent réussir lors du débogage sur votre bureau, mais échouer lors du déploiement. Le kit de développement logiciel (SDK) .NET Framework 2,0 et Visual Studio 2005 contiennent des outils permettant de calculer les autorisations dont une application a besoin : l'outil en ligne de commande MT. exe et la fonctionnalité calculer les autorisations de Visual Studio, respectivement.

Les rubriques suivantes décrivent les fonctionnalités de sécurité supplémentaires de Windows Forms.

RUBRIQUE	DESCRIPTION
- Accès plus sécurisé aux fichiers et aux données dans les Windows Forms	Décrit comment accéder aux fichiers et aux données dans un environnement de confiance partielle.
- Impression plus sécurisée dans les Windows Forms	Décrit comment accéder aux fonctionnalités d'impression dans un environnement de confiance partielle.
- Considérations supplémentaires sur la sécurité des Windows Forms	Décrit comment manipuler des fenêtres, utiliser le Presse-papiers et effectuer des appels au code non managé dans un environnement de confiance partielle.

Déploiement d'une application avec les autorisations appropriées

La méthode la plus courante pour déployer une application Windows Forms sur un ordinateur client consiste à utiliser ClickOnce, une technologie de déploiement qui décrit tous les composants dont votre application a besoin pour s'exécuter. ClickOnce utilise des fichiers XML appelés manifestes pour décrire les assemblys et les fichiers qui composent votre application, ainsi que les autorisations dont votre application a besoin.

ClickOnce a deux technologies pour demander des autorisations élevées sur un ordinateur client. Ces deux technologies reposent sur l'utilisation de certificats Authenticode. Les certificats permettent de garantir à vos utilisateurs que l'application provient d'une source approuvée.

Le tableau suivant décrit ces technologies.

TECHNOLOGIE D'ÉLEVATION D'AUTORISATIONS	DESCRIPTION
Élévation d'autorisations	Affiche une boîte de dialogue de sécurité la première fois que votre application s'exécute. La boîte de dialogue Élévation d'autorisations indique à l'utilisateur le nom de la personne qui a publié l'application, pour qu'il puisse décider en connaissance de cause s'il doit lui accorder une confiance supplémentaire
Déploiement d'applications approuvées	Exige qu'un administrateur système effectue une installation ponctuelle du certificat Authenticode de l'éditeur sur un ordinateur client. Après cela, toutes les applications signées avec le certificat sont considérées comme approuvées et peuvent s'exécuter avec une confiance totale sur l'ordinateur local sans invite supplémentaire.

La technologie adoptée dépendra de votre environnement de déploiement. Pour plus d'informations, consultez

Choix d'une stratégie de déploiement ClickOnce.

Par défaut, les applications ClickOnce déployées à l'aide de Visual Studio ou des outils du kit de développement logiciel (SDK) .NET Framework (Mage.exe et MageUI.exe) sont configurées pour s'exécuter sur un ordinateur client qui dispose de la confiance totale. Si vous déployez votre application avec une confiance partielle ou en utilisant uniquement certaines autorisations supplémentaires, vous devez modifier ce comportement par défaut. Pour ce faire, vous pouvez utiliser Visual Studio ou l'outil .NET Framework SDK MageUI.exe lorsque vous configurez votre déploiement. Pour plus d'informations sur l'utilisation de MageUI.exe, consultez [procédure pas à pas : déploiement manuel d'une application ClickOnce](#). Consultez également [Comment : définir des autorisations personnalisées pour une application ClickOnce](#) ou [Comment : définir des autorisations personnalisées pour une application ClickOnce](#).

Pour plus d'informations sur les aspects de sécurité de ClickOnce et de l'élévation d'autorisations, consultez [sécurisation des applications ClickOnce](#). Pour plus d'informations sur le déploiement d'applications approuvées, consultez [Vue d'ensemble du déploiement d'applications approuvées](#).

Test de l'application

Si vous avez déployé votre application Windows Forms à l'aide de Visual Studio, vous pouvez activer le débogage avec un niveau de confiance partielle ou un jeu d'autorisations restreint à partir de l'environnement de développement. Consultez également [Comment : déboguer une application ClickOnce avec des autorisations restreintes](#).

Voir aussi

- [Sécurité de Windows Forms](#)
- [Notions fondamentales de la sécurité d'accès du code](#)
- [Sécurité et déploiement ClickOnce](#)
- [Vue d'ensemble du déploiement d'applications approuvées](#)
- [Mage.exe \(outil Manifest Generation and Editing\)](#)
- [MageUI.exe \(outil Manifest Generation and Editing, client graphique\)](#)

Accès plus sécurisé aux fichiers et aux données dans les Windows Forms

16 minutes to read • [Edit Online](#)

Le cadre .NET utilise les autorisations pour aider à protéger les ressources et les données. L'emplacement où votre application peut lire ou écrire des données dépend des autorisations qui lui sont accordées. Quand votre application s'exécute dans un environnement de confiance partielle, vous n'avez peut-être pas accès à vos données ou vous devrez peut-être modifier la manière dont vous accédez aux données.

Quand vous rencontrez une restriction de sécurité, vous avez deux options : déclarer l'autorisation (en supposant qu'elle a été accordée à votre application) ou utiliser une version de la fonctionnalité écrite pour fonctionner en mode de confiance partielle. Les sections suivantes décrivent comment gérer l'accès aux fichiers, aux bases de données et au Registre à partir d'applications qui s'exécutent dans un environnement de confiance partielle.

NOTE

Par défaut, les outils qui génèrent des déploiements ClickOnce par défaut de ces déploiements pour demander Full Trust à partir des ordinateurs sur lesquels ils s'exécutent. Si vous décidez que vous voulez les avantages supplémentaires de sécurité de l'exécution en confiance partielle, vous devez changer cette valeur dans Visual Studio ou l'un des outils Windows SDK (Mage.exe ou MageUI.exe). Pour plus d'informations sur la sécurité des formulaires Windows, et sur la façon de déterminer le niveau de confiance approprié pour votre application, voir [Sécurité dans Windows Forms Aperçu](#).

Accès aux fichiers

La [FileIOPermission](#) classe contrôle l'accès au fichier et au dossier dans le cadre .NET. Par défaut, le système de sécurité n'accorde pas [FileIOPermission](#) aux environnements de confiance partielle tels que les zones Intranet local et Internet. Cependant, une application qui nécessite l'accès aux fichiers peut quand même fonctionner dans ces environnements si vous modifiez la conception de votre application ou si vous utilisez des méthodes différentes pour accéder aux fichiers. Par défaut, la zone Intranet local est autorisée à accéder au même site et au même répertoire, à se reconnecter au site de son origine et à lire à partir de son répertoire d'installation. Par défaut, la zone Internet est autorisée uniquement à se reconnecter au site de son origine.

Fichiers spécifiés par l'utilisateur

L'une des façons de gérer le fait de ne pas avoir l'autorisation d'accès aux fichiers consiste à inviter l'utilisateur à fournir des informations de fichiers spécifiques à l'aide de la classe [OpenFileDialog](#) ou [SaveFileDialog](#). Cette interaction utilisateur permet de fournir une garantie que l'application ne peut pas remplacer des fichiers importants ou charger des fichiers privés à des fins malveillantes. Les méthodes [OpenFile](#) et [OpenFile](#) fournissent un accès en lecture et en écriture aux fichiers en ouvrant le flux de fichier pour le fichier spécifié par l'utilisateur. Ces méthodes aident aussi à protéger les fichiers de l'utilisateur en masquant leur chemin d'accès.

NOTE

Ces autorisations diffèrent selon que votre application est dans la zone Internet ou Intranet. Les applications de la zone Internet peuvent uniquement utiliser [OpenFileDialog](#), alors que les applications de la zone Intranet ont une autorisation Boîte de dialogue Fichier illimitée.

La classe [FileDialogPermission](#) spécifie le type de boîte de dialogue Fichier que votre application peut utiliser. Le tableau suivant indique la valeur que vous devez avoir pour utiliser chaque classe [FileDialog](#).

CLASSE	VALEUR D'ACCÈS NÉCESSAIRE
OpenFileDialog	Open
SaveFileDialog	Save

NOTE

L'autorisation spécifique n'est pas demandée tant que la méthode [OpenFile](#) n'est pas appelée.

L'autorisation d'afficher une boîte de dialogue Fichier n'accorde pas à votre application un accès total à tous les membres des classes [FileDialog](#), [OpenFileDialog](#) et [SaveFileDialog](#). Pour les autorisations exactes qui sont nécessaires pour appeler chaque méthode, voir le sujet de référence pour cette méthode dans la documentation de la bibliothèque de classe cadre .NET.

L'exemple de code suivant utilise la méthode [OpenFile](#) pour ouvrir un fichier spécifié par l'utilisateur dans un contrôle [RichTextBox](#). L'exemple exige [FileDialogPermission](#) et la valeur d'énumération [Open](#) associée. Il illustre comment gérer [SecurityException](#) pour déterminer si la fonctionnalité d'enregistrement doit être désactivée. Cet exemple exige que votre [Form](#) ait un contrôle [Button](#) nommé `ButtonOpen` et un contrôle [RichTextBox](#) nommé

`RtfBoxMain`.

NOTE

La logique de programmation pour la fonctionnalité d'enregistrement n'est pas illustrée dans l'exemple.

```

Private Sub ButtonOpen_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles ButtonOpen.Click

    Dim editingFileName as String = ""
    Dim saveAllowed As Boolean = True

    ' Displays the OpenFileDialog.
    If (OpenFileDialog1.ShowDialog() = DialogResult.OK) Then
        Dim userStream as System.IO.Stream
        Try
            ' Opens the file stream for the file selected by the user.
            userStream =OpenFileDialog1.OpenFile()
            Me.RtfBoxMain.LoadFile(userStream, _
                RichTextBoxStreamType.PlainText)
        Finally
            userStream.Close()
        End Try

        ' Tries to get the file name selected by the user.
        ' Failure means that the application does not have
        ' unrestricted permission to the file.
        Try
            editingFileName = OpenFileDialog1.FileName
        Catch ex As Exception
            If TypeOf ex Is System.Security.SecurityException Then
                ' The application does not have unrestricted permission
                ' to the file so the save feature will be disabled.
                saveAllowed = False
            Else
                Throw ex
            End If
        End Try
    End If
End Sub

```

```

private void ButtonOpen_Click(object sender, System.EventArgs e)
{
    String editingFileName = "";
    Boolean saveAllowed = true;

    // Displays the OpenFileDialog.
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        // Opens the file stream for the file selected by the user.
        using (System.IO.Stream userStream = openFileDialog1.OpenFile())
        {
            this.RtfBoxMain.LoadFile(userStream,
                RichTextBoxStreamType.PlainText);
            userStream.Close();
        }

        // Tries to get the file name selected by the user.
        // Failure means that the application does not have
        // unrestricted permission to the file.
        try
        {
            editingFileName = openFileDialog1.FileName;
        }
        catch (Exception ex)
        {
            if (ex is System.Security.SecurityException)
            {
                // The application does not have unrestricted permission
                // to the file so the save feature will be disabled.
                saveAllowed = false;
            }
            else
            {
                throw ex;
            }
        }
    }
}

```

NOTE

Dans Visual CMD, assurez-vous d'ajouter du code pour activer le gestionnaire d'événements. En utilisant le code de l'exemple précédent, le code suivant montre comment activer le gestionnaire d'événements.

```
this.ButtonOpen.Click += new System.Windows.Forms.EventHandler(this.ButtonOpen_Click);
```

Autres fichiers

Vous devrez parfois lire ou écrire dans des fichiers que l'utilisateur ne spécifie pas, par exemple quand vous devrez conserver les paramètres d'application. Dans les zones Intranet local et Internet, votre application ne sera pas autorisée à stocker des données dans un fichier local. Toutefois, elle pourra stocker des données dans un stockage isolé. Le stockage isolé est un compartiment de données abstrait (et non un emplacement de stockage spécifique) composé d'au moins un fichier de stockage isolé, appelé magasin, contenant les emplacements de répertoire réels où sont stockées les données. Les autorisations d'accès aux fichiers telles que [FileIOPermission](#) ne sont pas nécessaires. Au lieu de cela, la classe [IsolatedStoragePermission](#) contrôle les autorisations pour le stockage isolé. Par défaut, les applications qui s'exécutent dans les zones Intranet local et Internet peuvent stocker des données à l'aide du stockage isolé. Toutefois, des paramètres tels que les quotas de disque peuvent varier. Pour plus d'informations sur le stockage isolé, voir [Stockage isolé](#).

L'exemple suivant utilise le stockage isolé pour écrire des données dans un fichier situé dans un magasin. Cet exemple nécessite [IsolatedStorageFilePermission](#) et la valeur d'énumération [DomainIsolationByUser](#). Il illustre comment lire et écrire certaines valeurs de propriétés du contrôle [Button](#) dans un fichier dans du stockage isolé.

La fonction `Read` est appelée après le démarrage de l'application et la fonction `Write` est appelée avant la fin de l'application. L'exemple exige `Read` `Write` que les et les `Form` fonctions `Button` existent `MainButton` en tant que membres d'un qui contient un contrôle nommé .

```
' Reads the button options from the isolated storage. Uses Default values
' for the button if the options file does not exist.
Public Sub Read()
    Dim isoStore As System.IO.IsolatedStorage.IsolatedStorageFile = _
        System.IO.IsolatedStorage.IsolatedStorageFile. _
            GetUserStoreForDomain()

    Dim filename As String = "options.txt"
    Try
        ' Checks to see if the options.txt file exists.
        If (isoStore.GetFileNames(filename).GetLength(0) <> 0) Then

            ' Opens the file because it exists.
            Dim isos As New System.IO.IsolatedStorage.IsolatedStorageFileStream _
                (filename, IO.FileMode.Open, isoStore)
            Dim reader as System.IO.StreamReader
            Try
                reader = new System.IO.StreamReader(isos)

                ' Reads the values stored.
                Dim converter As System.ComponentModel.TypeConverter
                converter = System.ComponentModel.TypeDescriptor.GetConverter _
                    (GetType(Color))

                Me.MainButton.BackColor = _
                    CType(converter.ConvertFromString _
                        (reader.ReadLine()), Color)
                me.MainButton.ForeColor = _
                    CType(converter.ConvertFromString _
                        (reader.ReadLine()), Color)

                converter = System.ComponentModel.TypeDescriptor.GetConverter _
                    (GetType(Font))
                me.MainButton.Font = _
                    CType(converter.ConvertFromString _
                        (reader.ReadLine()), Font)

            Catch ex As Exception
                Debug.WriteLine("Cannot read options " + _
                    ex.ToString())
            Finally
                reader.Close()
            End Try
        End If

    Catch ex As Exception
        Debug.WriteLine("Cannot read options " + ex.ToString())
    End Try
End Sub

' Writes the button options to the isolated storage.
Public Sub Write()
    Dim isoStore As System.IO.IsolatedStorage.IsolatedStorageFile = _
        System.IO.IsolatedStorage.IsolatedStorageFile. _
            GetUserStoreForDomain()

    Dim filename As String = "options.txt"
    Try
        ' Checks if the file exists, and if it does, tries to delete it.
        If (isoStore.GetFileNames(filename).GetLength(0) <> 0) Then
            isoStore.DeleteFile(filename)
        End If
    Catch ex As Exception
```

```

        Debug.WriteLine("Cannot delete file " + ex.ToString())
    End Try

    ' Creates the options.txt file and writes the button options to it.
    Dim writer As System.IO.StreamWriter
    Try
        Dim isos As New System.IO.IsolatedStorage.IsolatedStorageFileStream _
            (filename, IO.FileMode.CreateNew, isoStore)

        writer = New System.IO.StreamWriter(isos)
        Dim converter As System.ComponentModel.TypeConverter

        converter = System.ComponentModel.TypeDescriptor.GetConverter _
            (GetType(Color))
        writer.WriteLine(converter.ConvertToString( _
            Me.MainButton.BackColor))
        writer.WriteLine(converter.ConvertToString( _
            Me.MainButton.ForeColor))

        converter = System.ComponentModel.TypeDescriptor.GetConverter _
            (GetType(Font))
        writer.WriteLine(converter.ConvertToString( _
            Me.MainButton.Font))

    Catch ex As Exception
        Debug.WriteLine("Cannot write options " + ex.ToString())
    End Try

    Finally
        writer.Close()
    End Try
End Sub

```

```

// Reads the button options from the isolated storage. Uses default values
// for the button if the options file does not exist.
public void Read()
{
    System.IO.IsolatedStorage.IsolatedStorageFile isoStore =
        System.IO.IsolatedStorage.IsolatedStorageFile.
            GetUserStoreForDomain();

    string filename = "options.txt";
    try
    {
        // Checks to see if the options.txt file exists.
        if (isoStore.GetFileNames(filename).GetLength(0) != 0)
        {
            // Opens the file because it exists.
            System.IO.IsolatedStorage.IsolatedStorageFileStream isos =
                new System.IO.IsolatedStorage.IsolatedStorageFileStream
                    (filename, System.IO.FileMode.Open, isoStore);
            System.IO.StreamReader reader = null;
            try
            {
                reader = new System.IO.StreamReader(isos);

                // Reads the values stored.
                TypeConverter converter ;
                converter = TypeDescriptor.GetConverter(typeof(Color));

                this.MainButton.BackColor =
                    (Color)(converter.ConvertFromString(reader.ReadLine()));
                this.MainButton.ForeColor =
                    (Color)(converter.ConvertFromString(reader.ReadLine()));

                converter = TypeDescriptor.GetConverter(typeof(Font));
                this.MainButton.Font =
                    (Font)(converter.ConvertFromString(reader.ReadLine()));
            }
            catch { }
        }
    }
}

```

```

        }
        catch (Exception ex)
        {
            System.Diagnostics.Debug.WriteLine
                ("Cannot read options " + ex.ToString());
        }
        finally
        {
            reader.Close();
        }
    }
}
catch (Exception ex)
{
    System.Diagnostics.Debug.WriteLine
        ("Cannot read options " + ex.ToString());
}
}

// Writes the button options to the isolated storage.
public void Write()
{
    System.IO.IsolatedStorage.IsolatedStorageFile isoStore =
        System.IO.IsolatedStorage.IsolatedStorageFile.
            GetUserStoreForDomain();

    string filename = "options.txt";
    try
    {
        // Checks if the file exists and, if it does, tries to delete it.
        if (isoStore.GetFileNames(filename).GetLength(0) != 0)
        {
            isoStore.DeleteFile(filename);
        }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine
            ("Cannot delete file " + ex.ToString());
    }

    // Creates the options file and writes the button options to it.
    System.IO.StreamWriter writer = null;
    try
    {
        System.IO.IsolatedStorage.IsolatedStorageFileStream isos = new
            System.IO.IsolatedStorage.IsolatedStorageFileStream(filename,
                System.IO.FileMode.CreateNew, isoStore);

        writer = new System.IO.StreamWriter(isos);
        TypeConverter converter ;

        converter = TypeDescriptor.GetConverter(typeof(Color));
        writer.WriteLine(converter.ConvertToString(
            this.MainButton.BackColor));
        writer.WriteLine(converter.ConvertToString(
            this.MainButton.ForeColor));

        converter = TypeDescriptor.GetConverter(typeof(Font));
        writer.WriteLine(converter.ConvertToString(
            this.MainButton.Font));

    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine
            ("Cannot write options " + ex.ToString());
    }
    finally

```



```
{  
    writer.Close();  
}
```

Accès à la base de données

Les autorisations nécessaires pour accéder à une base de données varient en fonction du fournisseur de base de données. Toutefois, seules les applications qui s'exécutent avec les autorisations appropriées peuvent accéder à une base de données via une connexion de données. Pour plus d'informations sur les autorisations requises pour accéder à une base de données, voir [Code Access Security et ADO.NET](#).

Si vous ne pouvez pas accéder directement à une base de données car vous souhaitez que votre application s'exécute en mode de confiance partielle, vous pouvez utiliser un service web comme alternative pour accéder à vos données. Un service web est un élément logiciel accessible par programmation sur un réseau. Avec les services web, les applications peuvent partager des données entre des zones de groupe de code. Par défaut, les applications des zones Intranet local et Internet sont autorisées à accéder à leurs sites d'origine, ce qui leur permet d'appeler un service web hébergé sur le même serveur. Pour plus d'informations, consultez [les services Web dans ASP.NET AJAX](#) ou Windows Communication [Foundation](#).

Accès au Registre

La classe [RegistryPermission](#) contrôle l'accès au Registre du système d'exploitation. Par défaut, seules les applications qui s'exécutent localement peuvent accéder au Registre. [RegistryPermission](#) accorde uniquement à une application le droit de tenter d'accéder au Registre. Il ne garantit pas la réussite de l'accès, car le système d'exploitation applique quand même la sécurité sur le Registre.

Étant donné que vous ne pouvez pas accéder au Registre avec une confiance partielle, vous devrez peut-être trouver d'autres méthodes pour stocker vos données. Quand vous stockez des paramètres d'application, utilisez le stockage isolé plutôt que le Registre. Vous pouvez aussi utiliser le stockage isolé pour stocker d'autres fichiers spécifiques à l'application. Vous pouvez stocker des informations d'application globales relatives au serveur ou au site d'origine, car par défaut une application a le droit d'accéder au site de son origine.

Voir aussi

- [Impression plus sécurisée dans les Windows Forms](#)
- [Considérations supplémentaires sur la sécurité des Windows Forms](#)
- [Vue d'ensemble de la sécurité dans les Windows Forms](#)
- [Sécurité des Windows Forms](#)
- [Mage.exe \(outil Manifest Generation and Editing\)](#)
- [MageUI.exe \(outil Manifest Generation and Editing, client graphique\)](#)

Impression plus sécurisée dans les Windows Forms

2 minutes to read • [Edit Online](#)

Windows Forms applications incluent souvent des fonctionnalités d'impression. L' .NET Framework utilise la classe [PrintingPermission](#) pour contrôler l'accès aux fonctionnalités d'impression et la valeur d'énumération [PrintingPermissionLevel](#) associée pour indiquer le niveau d'accès. Par défaut, l'impression est activée par défaut dans les zones Intranet local et Internet. Toutefois, le niveau d'accès est limité dans les deux zones. Le fait que votre application puisse imprimer, nécessite une interaction de l'utilisateur ou ne peut pas imprimer dépend de la valeur d'autorisation accordée à l'application. Par défaut, la zone Intranet local reçoit [DefaultPrinting](#) accès et la zone intranet reçoit [SafePrinting](#) accès.

Le tableau suivant présente les fonctionnalités disponibles à chaque niveau d'autorisation d'impression.

PRINTINGPERMISSIONLEVEL	DESCRIPTION
AllPrinting	Fournit un accès complet à toutes les imprimantes installées.
DefaultPrinting	Active l'impression par programmation sur l'imprimante par défaut et l'impression plus sécurisée via une boîte de dialogue d'impression restrictive. DefaultPrinting est un sous-ensemble de AllPrinting .
SafePrinting	Permet d'imprimer uniquement à partir d'une boîte de dialogue plus restreinte. SafePrinting est un sous-ensemble de DefaultPrinting .
NoPrinting	Interdit l'accès aux imprimantes. NoPrinting est un sous-ensemble de SafePrinting .

Voir aussi

- [Accès plus sécurisé aux fichiers et aux données dans les Windows Forms](#)
- [Considérations supplémentaires sur la sécurité des Windows Forms](#)
- [Vue d'ensemble de la sécurité dans les Windows Forms](#)
- [Sécurité de Windows Forms](#)

Considérations supplémentaires sur la sécurité des Windows Forms

13 minutes to read • [Edit Online](#)

.NET Framework paramètres de sécurité peuvent entraîner une exécution différente de votre application dans un environnement de confiance partielle que sur votre ordinateur local. Le .NET Framework restreint l'accès à ces ressources locales critiques comme le système de fichiers, le réseau et les API non managées, entre autres choses. Les paramètres de sécurité affectent la possibilité d'appeler l'API Microsoft Windows ou d'autres API qui ne peuvent pas être vérifiées par le système de sécurité. La sécurité affecte également les autres aspects de votre application, y compris l'accès aux fichiers et aux données, ainsi que l'impression. Pour plus d'informations sur l'accès aux fichiers et aux données dans un environnement à confiance partielle, consultez [Accès plus sécurisé aux fichiers et aux données dans les Windows Forms](#). Pour plus d'informations sur l'impression dans un environnement à confiance partielle, consultez [Impression plus sécurisée dans les Windows Forms](#).

Les sections suivantes expliquent comment utiliser le presse-papiers, effectuer une manipulation de fenêtre et appeler l'API Windows à partir d'applications qui s'exécutent dans un environnement de confiance partielle.

Accès au presse-papiers

La classe [UIPermission](#) contrôle l'accès au Presse-papiers, et la valeur d'énumération [UIPermissionClipboard](#) associée indique le niveau d'accès. Le tableau suivant présente les différents niveaux d'autorisation.

VALEUR UIPERMISSIONCLIPBOARD	DESCRIPTION
AllClipboard	Le Presse-papiers peut être utilisé sans aucune restriction.
OwnClipboard	Le Presse-papiers peut être utilisé avec certaines restrictions. La possibilité de placer des données dans le Presse-papiers (opérations de commande copier ou couper) n'est pas restreinte. Les contrôles intrinsèques qui acceptent le collage, comme les zones de texte, peuvent accepter les données du Presse-papiers, mais les contrôles utilisateur ne peuvent pas lire le contenu du Presse-papiers par programme.
NoClipboard	Impossible d'utiliser le Presse-papiers.

Par défaut, la zone Intranet local reçoit [AllClipboard](#) accès et la zone Internet reçoit [OwnClipboard](#) accès. Cela signifie que l'application peut copier des données dans le Presse-papiers, mais pas les coller par programme vers ou lire à partir du Presse-papiers. Ces restrictions empêchent les programmes sans confiance totale de lire les données copiées dans le Presse-papiers par une autre application. Si votre application nécessite l'accès total au Presse-papiers mais que vous n'avez pas les autorisations, vous devez élever les autorisations pour votre application. Pour plus d'informations sur l'élévation d'autorisations, consultez [Administration de la stratégie de sécurité générale](#).

Manipulation de fenêtres

La classe [UIPermission](#) contrôle également l'autorisation d'effectuer une manipulation de fenêtre et d'autres actions liées à l'interface utilisateur, et la valeur d'énumération [UIPermissionWindow](#) associée indique le niveau d'accès. Le tableau suivant présente les différents niveaux d'autorisation.

Par défaut, la zone Intranet local reçoit [AllWindows](#) accès et la zone Internet reçoit [SafeTopLevelWindows](#) accès. Cela signifie que dans la zone Internet, l'application peut effectuer la plupart des actions d'interface utilisateur et liées aux fenêtres, mais l'aspect de la fenêtre sera modifié. La fenêtre modifiée affiche une bulle de notification lors du démarrage de l'exécution, contient le texte de barre de titre modifié et requiert un bouton Fermer dans la barre de titre. La bulle de notification et la barre de titre indiquent à l'utilisateur de l'application que l'application s'exécute en confiance partielle.

VALEUR UIPERMISSIONWINDOW	DESCRIPTION
AllWindows	Les utilisateurs peuvent utiliser toutes les fenêtres et tous les événements d'entrée d'utilisateur sans restriction.
SafeTopLevelWindows	Les utilisateurs peuvent utiliser uniquement des fenêtres de niveau supérieur sûres et des sous-fenêtres sûres pour le dessin, et peuvent utiliser uniquement les événements d'entrée d'utilisateur pour l'interface utilisateur dans ces fenêtres de niveau supérieur et ces sous-fenêtres. Ces fenêtres sûres sont clairement indiquées et ont des restrictions de taille minimale et maximale. Les restrictions empêchent les attaques d'usurpation d'identité potentiellement dangereuses, telles que l'imitation des écrans d'ouverture de session système ou le Bureau du système, et limitent l'accès par programme aux fenêtres parentes, les API liées au focus et l'utilisation du contrôle ToolTip ,
SafeSubWindows	Les utilisateurs peuvent utiliser uniquement des sous-fenêtres sûres pour le dessin, et peuvent utiliser uniquement les événements d'entrée d'utilisateur pour l'interface utilisateur dans ces sous-fenêtres. Un contrôle affiché dans un navigateur est un exemple de sous-fenêtre sûre.
NoWindows	Les utilisateurs ne peuvent pas utiliser d'événements d'interface utilisateur ou Windows. Aucune interface utilisateur ne peut être utilisée.

Chaque niveau d'autorisation identifié par l'énumération [UIPermissionWindow](#) autorise moins d'actions que le niveau supérieur. Les tableaux suivants indiquent les actions qui sont limitées par les valeurs [SafeTopLevelWindows](#) et [SafeSubWindows](#). Pour plus d'informations sur les autorisations nécessaires pour chaque membre, consultez la rubrique de référence correspondant à ce membre dans la documentation de la bibliothèque de classes .NET Framework.

[SafeTopLevelWindows](#) autorisation restreint les actions énumérées dans le tableau suivant.

COMPOSANT	ACTIONS RESTREINTES
Application	- Définition de la propriété SafeTopLevelCaptionFormat .
Control	<ul style="list-style-type: none"> -Obtention de la propriété Parent. - Définition de la propriété Region. -Appel de la méthode FindForm, Focus, FromChildHandle et FromHandle, PreProcessMessage, ReflectMessage ou SetTopLevel. -Appel de la méthode GetChildAtPoint si le contrôle retourné n'est pas un enfant du contrôle appelant. - Modifier le focus du contrôle à l'intérieur d'un contrôle conteneur.

COMPOSANT	ACTIONS RESTREINTES
Cursor	<ul style="list-style-type: none"> - Définition de la propriété Clip. - Appel de la méthode Hide.
DataGrid	<ul style="list-style-type: none"> - Appel de la méthode ProcessTabKey.
Form	<ul style="list-style-type: none"> - Obtention de la propriété ActiveForm ou MdiParent. - Définition de la propriété ControlBox, ShowInTaskbar ou TopMost. - Définition de la propriété Opacity inférieure à 50%. - Définition de la propriété WindowState sur Minimized par programmation. - Appel de la méthode Activate. - À l'aide des valeurs d'énumération None, FixedToolWindow et SizableToolWindowFormBorderStyle.
NotifyIcon	<ul style="list-style-type: none"> - L'utilisation du composant NotifyIcon est complètement restreinte.

La valeur [SafeSubWindows](#) restreint les actions énumérées dans le tableau suivant, en plus des restrictions placées par la valeur [SafeTopLevelWindows](#).

COMPOSANT	ACTIONS RESTREINTES
CommonDialog	<ul style="list-style-type: none"> - Avec une boîte de dialogue dérivée de la classe CommonDialog.
Control	<ul style="list-style-type: none"> - Appel de la méthode CreateGraphics. - Définition de la propriété Cursor.
Cursor	<ul style="list-style-type: none"> - Définition de la propriété Current.
MessageBox	<ul style="list-style-type: none"> - Appel de la méthode Show.

Hébergement de contrôles tiers

Un autre type de manipulation de fenêtre peut se produire si vos formulaires hébergent des contrôles tiers. Un contrôle tiers est un [UserControl](#) personnalisé que vous n'avez pas développé et compilé vous-même. Bien que le scénario d'hébergement soit difficile à exploiter, il est théoriquement possible pour un contrôle tiers de développer sa surface de rendu pour couvrir la zone entière de votre formulaire. Ce contrôle peut alors reproduire une boîte de dialogue critique et demander des informations telles que des combinaisons nom d'utilisateur/mot de passe ou des numéros de compte bancaire de vos utilisateurs.

Pour limiter ce risque, utilisez des contrôles tiers uniquement auprès de fournisseurs auxquels vous pouvez faire confiance. Si vous utilisez des contrôles tiers que vous avez téléchargés à partir d'une source non vérifiable, nous vous recommandons de vérifier le code source à la recherche d'attaques éventuelles. Après avoir vérifié que la source est non malveillante, vous devez compiler l'assembly vous-même pour garantir que la source correspond à l'assembly.

Appels d'API Windows

Si la conception de votre application requiert l'appel d'une fonction à partir de l'API Windows, vous accédez au code non managé. Dans ce cas, les actions du code vers la fenêtre ou le système d'exploitation ne peuvent pas être déterminées lorsque vous utilisez des appels ou des valeurs de l'API Windows. La classe [SecurityPermission](#) et la valeur [UnmanagedCode](#) de l'énumération [SecurityPermissionFlag](#) contrôlent l'accès au code non managé. Une

application peut accéder à du code non géré uniquement quand l'autorisation [UnmanagedCode](#) est accordée. Par défaut, seules les applications qui s'exécutent localement peuvent appeler du code non géré.

Certains membres de Windows Forms fournissent un accès non géré qui nécessite l'autorisation [UnmanagedCode](#). Le tableau suivant répertorie les membres de l'espace de noms [System.Windows.Forms](#) qui requièrent l'autorisation. Pour plus d'informations sur les autorisations requises pour un membre, consultez la documentation de la bibliothèque de classes .NET Framework.

COMPOSANT	MEMBRE
Application	Méthode - AddMessageFilter - CurrentInputLanguage propriété Méthode - Exit Méthode - ExitThread événement - ThreadException
CommonDialog	Méthode - HookProc - OwnerWndProc , méthode Méthode - Reset Méthode - RunDialog
Control	Méthode - CreateParams Méthode - DefWndProc Méthode - DestroyHandle Méthode - WndProc
Help	méthodes de ShowHelp - Méthode - ShowHelpIndex
NativeWindow	classe - NativeWindow
Screen	Méthode - FromHandle
SendKeys	Méthode - Send Méthode - SendWait

Si votre application n'a pas l'autorisation d'appeler du code non géré, votre application doit demander [UnmanagedCode](#) autorisation, ou vous devez envisager d'autres méthodes d'implémentation des fonctionnalités. dans de nombreux cas, Windows Forms fournit une alternative gérée aux fonctions de l'API Windows. En cas d'absence d'alternative et si l'application doit accéder à du code non géré, vous devez élever les autorisations pour l'application.

L'autorisation d'appeler du code non géré permet à une application d'exécuter pratiquement n'importe quoi. Par conséquent, l'autorisation d'appeler du code non géré doit uniquement être accordée pour les applications qui proviennent d'une source approuvée. Selon l'application, la fonctionnalité qui effectue l'appel au code non géré peut également être facultative ou activée uniquement dans l'environnement à confiance totale. Pour plus d'informations sur les autorisations, consultez [Autorisations dangereuses et administration de stratégie](#). Pour plus d'informations sur l'élévation d'autorisations, consultez [Administration de la stratégie de sécurité générale](#).

Voir aussi

- [Accès plus sécurisé aux fichiers et aux données dans les Windows Forms](#)
- [Impression plus sécurisée dans les Windows Forms](#)
- [Vue d'ensemble de la sécurité dans les Windows Forms](#)
- [Sécurité de Windows Forms](#)

- [Sécurisation des applications ClickOnce](#)

Déploiement ClickOnce pour les Windows Forms

4 minutes to read • [Edit Online](#)

Les rubriques suivantes décrivent ClickOnce, une technologie permettant de déployer facilement des applications Windows Forms sur des ordinateurs clients.

Sections connexes

[Choix d'une stratégie de déploiement ClickOnce](#)

Présente plusieurs options pour le déploiement d'applications ClickOnce.

[Choix d'une stratégie de mise à jour ClickOnce](#)

Présente plusieurs options de mise à jour des applications ClickOnce.

[Sécurisation des applications ClickOnce](#)

Explique les implications en matière de sécurité du déploiement ClickOnce.

[Dépannage des déploiements ClickOnce](#)

Décrit différents problèmes qui peuvent se produire lors du déploiement d'applications ClickOnce et documente les messages d'erreur de niveau supérieur que ClickOnce peut générer.

[ClickOnce et paramètres d'application](#)

Décrit le fonctionnement du déploiement ClickOnce avec les paramètres de l'application, qui stocke les paramètres de l'application et de l'utilisateur pour une récupération ultérieure.

[Vue d'ensemble du déploiement d'applications approuvées](#)

Décrit une fonctionnalité ClickOnce qui permet aux applications approuvées de s'exécuter avec un niveau supérieur d'autorisation sur les ordinateurs clients.

[ClickOnce et Authenticode](#)

Décrit l'utilisation de la technologie Authenticode lors du déploiement d'applications approuvées.

[Procédure pas à pas : déploiement manuel d'une application ClickOnce](#)

Montre comment utiliser les outils de ligne de commande et du kit de développement logiciel (SDK) pour déployer une application ClickOnce sans utiliser Visual Studio.

[Guide pratique pour ajouter un éditeur approuvé à un ordinateur client pour les applications ClickOnce](#)

Décrit la configuration ponctuelle d'ordinateurs clients nécessaire pour le déploiement d'applications approuvées.

[Guide pratique pour spécifier un autre emplacement pour les mises à jour du déploiement](#)

Illustre la configuration d'une application ClickOnce, à l'aide des outils du kit de développement logiciel (SDK), pour vérifier un emplacement différent pour les nouvelles versions d'une application.

[Procédure pas à pas : téléchargement d'assemblies à la demande avec l'API de déploiement ClickOnce](#)

Illustre l'utilisation d'appels d'API pour récupérer un assembly la première fois que votre application essaie de le charger.

[Guide pratique pour récupérer les informations de chaîne de requête dans une application ClickOnce en ligne](#)

Illustre la récupération de paramètres à partir de l'URL utilisée pour exécuter une application ClickOnce.

[Vue d'ensemble du cache ClickOnce](#)

Décrit le cache utilisé pour stocker les applications ClickOnce sur l'ordinateur local.

[Accès aux données locales et distantes dans les applications ClickOnce](#)

Décrit comment accéder aux fichiers de données locaux et aux sources de données distantes à partir d'une application ClickOnce.

[How to: Include a Data File in a ClickOnce Application](#)

Montre comment marquer un fichier pour qu'il soit disponible dans le répertoire de données ClickOnce.

Voir aussi

- [Vue d'ensemble des paramètres d'application](#)
- [Publication d'applications ClickOnce](#)
- [Génération d'applications ClickOnce à partir de la ligne de commande](#)
- [Débogage des applications ClickOnce qui utilisent System.Deployment.Application](#)
- [Déploiement de composants COM avec ClickOnce](#)
- [Guide pratique pour publier une application ClickOnce à l'aide de l'Assistant Publication](#)

Comment : accéder à des collections à clé dans les Windows Forms

2 minutes to read • [Edit Online](#)

- Vous pouvez accéder à des éléments de collecte individuels par clé. Cette fonctionnalité a été ajoutée à de nombreuses classes de collection qui sont généralement utilisées par les applications de Windows Forms. La liste suivante répertorie certaines des classes de collection qui ont des collections à clé accessibles :

- [ListView.ListViewItemCollection](#)
- [ListViewItem.ListViewSubItemCollection](#)
- [Control.ControlCollection](#)
- [TabControl.TabPageCollection](#)
- [TreeNodeCollection](#)

La clé associée à un élément dans une collection est généralement le nom de l'élément. Les procédures suivantes vous montrent comment utiliser des classes de collection pour effectuer des tâches courantes.

Pour rechercher et attribuer le focus à un contrôle imbriqué dans une collection de contrôles

- Utilisez les méthodes [Find](#) et [Focus](#) pour spécifier le nom du contrôle à rechercher et lui attribuer le focus.

```
OrderForm OrderForm1 = new OrderForm();
OrderForm1.Show();
OrderForm1.Controls.Find("textBox1", true)[0].Focus();
```

```
Dim OrderForm1 As New OrderForm()
OrderForm1.Show()
OrderForm1.Controls.Find("textBox1", True)(0).Focus()
```

Pour accéder à une image dans une collection d'images

- Utilisez la propriété [Item\[Int32\]](#) pour spécifier le nom de l'image à laquelle vous souhaitez accéder.

```
this.BackgroundImage = imageList1.Images["logo.gif"];
```

```
Me.BackgroundImage = imageList1.Images("logo.gif")
```

Pour définir une page d'onglets en tant qu'onglet sélectionné

- Utilisez la propriété [SelectedTab](#) avec la propriété [Item\[String\]](#) pour spécifier le nom de la page d'onglets à définir comme onglet sélectionné.

```
tabControl1.SelectedTab = tabControl1.TabPages["shippingOptions"];
```

```
tabControl1.SelectedTab = tabControl1.TabPages("shippingOptions")
```

Voir aussi

- [Bien démarrer avec Windows Forms](#)
- [Guide pratique pour ajouter ou supprimer des images avec le composant ImageList Windows Forms](#)

Amélioration des applications Windows Forms

3 minutes to read • [Edit Online](#)

Windows Forms contient de nombreuses fonctionnalités que vous pouvez utiliser pour améliorer vos applications Windows pour répondre aux besoins spécifiques de vos utilisateurs. Les rubriques suivantes décrivent ces fonctionnalités et comment les utiliser.

Dans cette section

[Graphiques et dessins dans Windows Forms](#)

Contient des liens vers des rubriques qui décrivent et illustrent comment utiliser l'interface graphique dans les Windows Forms.

[Paramètres d'application pour les Windows Forms.](#)

Contient des liens vers des rubriques qui décrivent et illustrent comment utiliser la fonctionnalité **Paramètres d'application**.

[Prise en charge de l'impression dans les Windows Forms](#)

Contient des liens vers des rubriques qui décrivent et illustrent comment imprimer des fichiers à partir d'applications Windows Forms.

[Opérations glisser-déposer et prise en charge du Presse-papiers](#)

Contient des liens vers des rubriques qui décrivent et illustrent comment utiliser la fonctionnalité de glisser-déplacer et le Presse-papiers dans les Windows Forms.

[Mise en réseau dans les applications Windows Forms](#)

Contient des liens vers des rubriques qui décrivent et illustrent comment utiliser la mise en réseau dans les Windows Forms.

[Globalisation des applications Windows Forms](#)

Contient des liens vers des rubriques qui montrent comment globaliser des applications Windows Forms.

[Windows Forms and Unmanaged Applications](#)

Contient des liens vers des rubriques qui décrivent et illustrent comment accéder aux composants COM à partir d'applications Windows Form.

[Informations système et Windows Forms](#)

Décrit comment utiliser les informations système dans les Windows Forms.

[Gestion de l'alimentation dans Windows Forms](#)

Décrit comment gérer l'alimentation dans les applications Windows Forms.

[Héritage visuel des Windows Forms](#)

Décrit comment hériter d'un formulaire de base.

[Applications d'interface multidocument \(MDI, Multiple Document Interface\)](#)

Décrit comment créer des applications d'interface multidocument (MDI).

[Intégration de l'aide d'utilisateur dans les Windows Forms](#)

Décrit comment intégrer l'aide utilisateur dans vos applications.

[Accessibilité des Windows Forms](#)

Décrit comment rendre vos applications accessibles à un large éventail d'utilisateurs.

[Utilisation de contrôles WPF](#)

Décrit comment utiliser des contrôles WPF dans vos applications Windows Forms.

Sections connexes

[Systèmes d'aide dans les applications Windows Forms](#)

Contient des liens vers des rubriques qui décrivent et illustrent comment fournir de l'aide aux utilisateurs dans des applications Windows Forms.

[Bien démarrer avec Windows Forms](#)

Contient des liens vers des rubriques qui décrivent comment utiliser les fonctionnalités de base des Windows Forms.