Windows Forms

Table des matières

[1 Introduction 2](#_Toc480843662)

[2 Création d’une application Winforms 2](#_Toc480843663)

[2.1 Création d’un nouveau projet 2](#_Toc480843664)

[2.2 Le designer 3](#_Toc480843665)

[2.3 Structure du projet 5](#_Toc480843666)

[3 Abonnement aux évènements 5](#_Toc480843667)

[3.1 Génération automatique du code par VS 5](#_Toc480843668)

[3.2 Ecriture manuelle du code 6](#_Toc480843669)

[4 Les paramètres d’application 6](#_Toc480843670)

[4.1 Liste des paramètres 7](#_Toc480843671)

[4.2 Création de paramètres depuis les propriétés d’un contrôle 7](#_Toc480843672)

[4.3 Stockage des paramètres 8](#_Toc480843673)

[4.4 Accès aux paramètres par le code 9](#_Toc480843674)

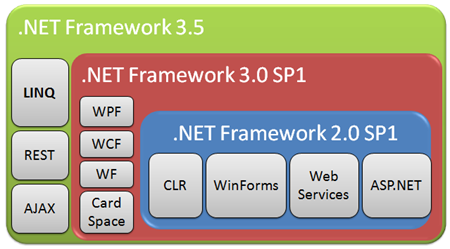
[5 Les ressources 10](#_Toc480843675)

[5.1 Rendre une application localisable 10](#_Toc480843676)

[5.2 Ressources de portée application 11](#_Toc480843677)

# Introduction

Windows Forms (WinForms) est une librairie de classes graphiques du framework .Net, permettant de créer des applications de bureau avec couche cliente riche (par opposition aux applications avec couche cliente légère telles que les applications web).



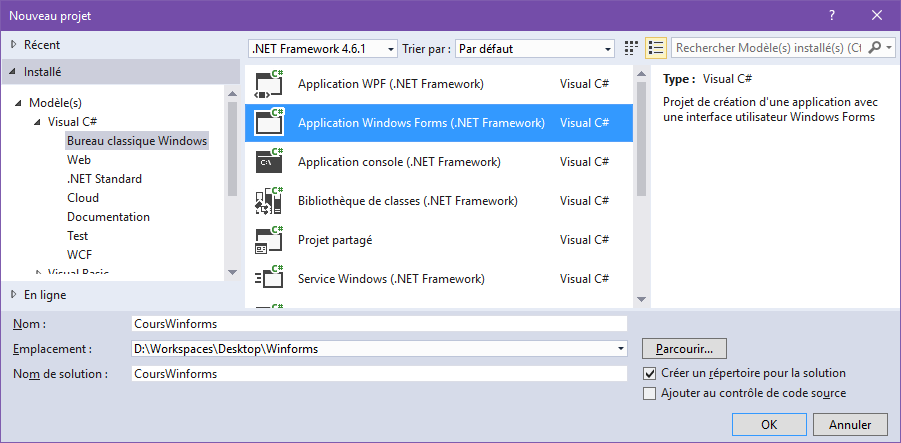
Une application Windows Forms est basée sur le modèle évènementiel, c’est-à-dire que son code est essentiellement exécuté en réponse à des évènements déclenchés par l’utilisateur, via son interaction avec l’interface graphique.

Winforms a été introduite dès la première version du framework.net. Il s’agit donc d’une technologie déjà ancienne, qui est depuis 2014 en mode maintenance, c’est-à-dire qu’elle n’évolue plus fonctionnellement. Les autres technologies d’interfaces graphiques plus récentes de Microsoft sont WPF et xaml (sorties en 2006), et celle fournie par UWP, introduite avec Windows 10, et qui s’appuie aussi sur le XAML.

# Création d’une application Winforms

## Création d’un nouveau projet

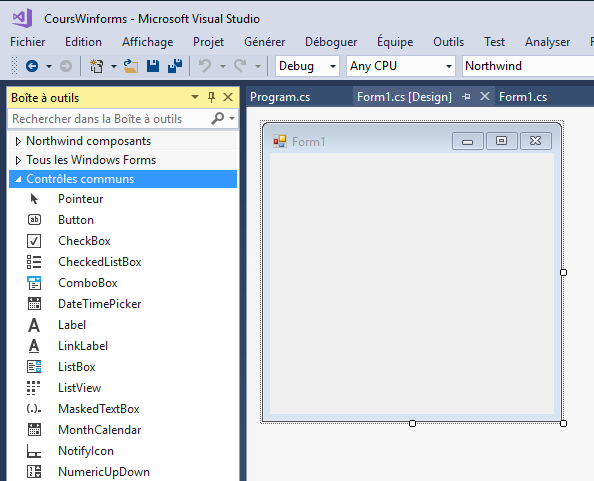
Pour créer une application Winforms avec Visual Studio, il faut ouvrir la fenêtre de création d’un nouveau projet et choisir le modèle « Application Windows Forms » dans la branche « Bureau Classique Windows ».



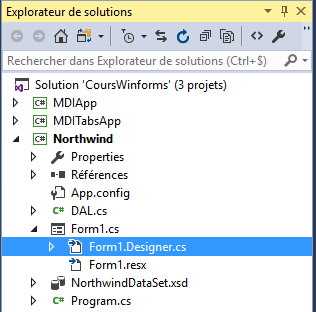
## Le designer

L’interface graphique d’une application Winforms est composée de fenêtres, appelées **formulaires** (forms), sur lesquels sont disposés des **contrôles**, tels que des libellés, zones de saisies, boutons…etc.

Cette interface est créée dans Visual Studio avec un outil graphique appelé Designer, et une boîte à outils contenant les contrôles :



Tout ce qui est créé par l’intermédiaire du Designer est traduit en code C#, dans des fichiers avec les extensions cs et Designer.cs, comme on peut le voir dans l’explorateur de solution :



## Structure du projet

Par défaut un nouveau projet Winforms contient les éléments suivants :

* Un fichier Program.cs, qui contient la fonction Main, qui est le point d’entrée de l’application. Celle-ci contient par défaut un code minimal, qui permet de d’afficher la fenêtre principale au lancement de l’application
* Une fenêtre principale nommée Form1. Celle-ci est représentée par :
  + Un fichier Form1.Designer.cs qui est l’implémentation de ce qui est affiché visuellement dans le designer.
  + Un fichier Forms1.cs, dans lequel on ajoutera notre code d’interaction avec le formulaire.
  + Un fichier resx, appelé fichier de ressources, qui contient les ressources utilisées par la fenêtre. Il s’agit typiquement de libellés traductibles, d’icônes, d’images…
* Dans la branche Properties :
  + Un fichier de ressources de niveau application
  + Un fichier de paramètres nommé Settings.settings, destiné à stocker des paramètres de niveau utilisateur (cf. paragraphe suivant)
* Un fichier App.config, qui contient des paramètres de niveau application (cf. paragraphe suivant)

# Abonnement aux évènements

Les applications fenêtrées utilisent abondamment les évènements. Il est donc important de savoir comment s’y abonner.

Chaque contrôle de l’interface utilisateur est susceptible d’émettre des évènements (ex : évènement clic d'un bouton). La déclaration et le déclenchement d'un événement sont faits dans la classe du contrôle.

Le code de branchement et d'implémentation du gestionnaire d'évènement, c’est-à-dire le code à exécuter en réponse à l'évènement, est dans la classe qui utilise le contrôle (ex : la classe de la fenêtre contenant le bouton).

## Génération automatique du code par VS

Le code de branchement d’un gestionnaire d’évènement peut être en partie généré automatiquement par Visual Studio. Pour cela, dans la fenêtre de propriétés du contrôle, il suffit de double cliquer en regard de l'évènement souhaité (ex : en regard de l'évènement Click du bouton). VS génère alors automatiquement, ce qui suit :

* Une méthode gestionnaire vide dans la classe de la fenêtre contenant le contrôle. Exemple :

private void btnOK\_Click(object sender, System.EventArgs e)

{

}

* Le code de branchement de ce gestionnaire dans la méthode InitializeComponent, présente dans le fichier .Designer.cs de la fenêtre. Exemple :

this.btnOK.Click += new System.EventHandler(this.btnOK\_Click);

Ce code utilise la syntaxe C# 1.00, pour des raisons de compatibilité.

System.EventHandler est un type délégué, et btnOK\_Click est une méthode gestionnaire, dont les paramètres et le type de retour respectent ceux du type délégué. En effet, si on regarde la définition du type délégué en faisant F12 dessus, on voit qu'il prend bien 2 paramètres de types object et System.EventArgs et qu'il a un type de retour void.

## Ecriture manuelle du code

On peut écrire nous-même le code de branchement et d'implémentation du gestionnaire d'évènement, ce qui apporte des avantages par rapport à la génération automatique :

* On peut utiliser la syntaxe C# 2.0, qui est plus simple et intuitive
* L'ensemble du code est placé dans le même fichier, ce qui est plus clair

De plus, il est bon de maîtriser ce type de syntaxe, qui est très utilisée dans l'implémentation des commandes relais en WPF.

L'exemple ci-dessus peut ainsi être implémenté de la façon suivante :

* Gestionnaire d'évènement : même code que précédemment
* Branchement du gestionnaire dans le constructeur de la fenêtre :  
  btnOK.Click += btnOK\_Click;

On peut facilement déterminer les types des paramètres et de retour à utiliser pour le gestionnaire d’évènement, en faisant F12 sur Click (on accède ainsi à l'évènement), puis à nouveau sur le type de cet évènement (on accède alors au type délégué qu'il utilise).

# Les paramètres d’application

Cf. [cette page MSDN](https://msdn.microsoft.com/fr-fr/library/a65txexh.aspx).

Les paramètres d’application répondent à plusieurs besoins :

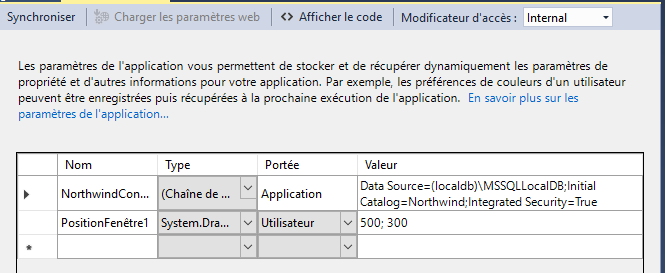
* Faciliter l’accès par le code à des informations utilisées partout dans l’application, et qu’on souhaite pouvoir modifier sans recompiler l’application. Ex : une chaîne de connexion à une base de données, des valeurs qui dépendent de différentes typologies de clients…etc
* Permettre aux utilisateurs de mémoriser leurs préférences (ex : choix de polices ou de couleurs, dernières positions des fenêtres à l’écran…etc.)

Une application .net gère deux types de paramètres :

* Des paramètres de portée application, qui ne sont pas modifiables par les utilisateurs durant l’exécution de l’appli.
* Des paramètres de portée utilisateur, modifiables par les utilisateurs durant l’exécution de l’appli

## Liste des paramètres

Dans Visual Studio, la liste des paramètres de l’application est consultable et modifiable depuis une interface visuelle spécifique présentée ci-dessous :



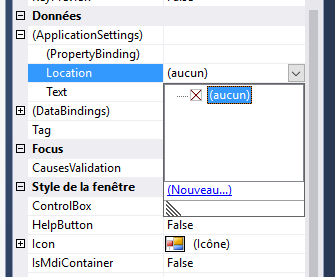
Cette interface est accessible de deux façons :

* Depuis la rubrique Paramètres des propriétés du projet (figure de gauche)
* En double-cliquant sur l’entrée Settings.settings dans l’explorateur de solution (figure de droite)

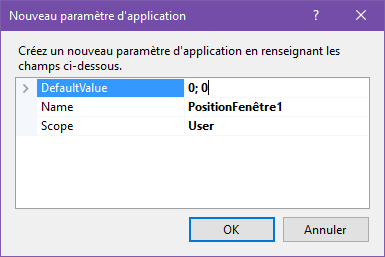
|  |  |
| --- | --- |
|  |  |

## Création de paramètres depuis les propriétés d’un contrôle

Par ailleurs, on peut créer des paramètres directement depuis le panneau de propriétés d’un formulaire ou de l’un de ses contrôles, comme illustré dans la capture d’écran ci-dessous :



Ici, on souhaite créer un paramètre pour mémoriser la dernière position d’une fenêtre à l’écran. Pour cela, sur la propriété Location, on déplie la liste déroulante et on clique sur le Lien [Nouveau…]. Ceci ouvre la boîte de dialogue ci-dessous, dans laquelle on peut saisir les caractéristiques du paramètre :



Le paramètre ainsi créé se retrouve bien entendu dans la liste des paramètres présentée plus haut.

## Stockage des paramètres

Les paramètres sont enregistrés dans le fichier App.config, visible dans l’explorateur de solution. Il s’agit d’un fichier au format xml, dont le contenu ressemble à ce qui suit :

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

    <configSections>

        <sectionGroup name="userSettings" type="System.Configuration.UserSettingsGroup, System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >

            <section name="Northwind.Properties.Settings" type="System.Configuration.ClientSettingsSection, System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" allowExeDefinition="MachineToLocalUser" requirePermission="false" />

        </sectionGroup>

    </configSections>

    <connectionStrings>

        <add name="NorthwindConnectionString" connectionString="Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Northwind;Integrated Security=True"

            providerName="System.Data.SqlClient" />

        <add name="Northwind.Properties.Settings.NorthwindConnectionString"

            connectionString="Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Northwind;Integrated Security=True"

            providerName="System.Data.SqlClient" />

    </connectionStrings>

    <startup>

        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.1" />

    </startup>

    <userSettings>

        <Northwind.Properties.Settings>

            <setting name="PositionFenêtre1" serializeAs="String">

                <value>500, 300</value>

            </setting>

        </Northwind.Properties.Settings>

    </userSettings>

</configuration>

* Les chaînes de connexions sont stockées dans une section spécifique ConnectionString. Elles peuvent être stockées de façon cryptée pour ne pas révéler les mots de passe éventuels.
* Les autres paramètres de portée application sont stockés dans la section applicationSettings, qui n’apparaît pas ici, car nous n’avons pas défini de tels paramètres
* Les paramètres de portée utilisateur sont stockés dans la section userSettings,

Ce fichier est copié à côté de l’exécutable au moment de la compilation de l’application

**Valeurs définies par l’utilisateur**

Les valeurs de tous les paramètres de portée utilisateur modifiés par l’utilisateur au moment de l’exécution, sont stockées dans un fichier user.config, placé dans un dossier caché. Les valeurs par défaut restent enregistrées dans le fichier app.config.

Il est

## Accès aux paramètres par le code

Chaque paramètre d’application est accessible de façon statique et fortement typée de la façon suivante :

Properties.Settings.Default.PositionFenêtre1 = ...

A noter que lorsqu’on modifie la valeur d’un paramètre de portée utilisateur, il faut appeler explicitement la méthode de sauvegarde des valeurs :

Properties.Settings.Default.Save();

# Les ressources

Les fichiers resx. Peuvent contenir des textes traductibles, des images, et d’autres types de fichiers. Nous présentons ci-dessous la façon de les utiliser pour rendre une application localisable, c’est-à-dire adaptable à différents pays / langues.

## Rendre une application localisable

Les contrôles placés dans une fenêtre peuvent contenir du texte fixe. C’est le cas par exemple des libellés, des boutons, des groupbox…etc.

Si l’application est distribuée dans plusieurs pays, on souhaite généralement que ces textes puissent être traduits et affichés dans la langue du pays.

Prenons l’exemple simple d’un bouton affichant le texte « Annuler », qui sera traduit en anglais en « Cancel » et en allemand en espagnol en « Anular ». Pour que le bouton affiche le texte adéquat en fonction de la langue du poste de l’utilisateur, il faut :

* Affecter la valeur true à la propriété Localizable de la fenêtre qui contient le bouton
* Affecter la valeur « Anglais (Etats-Unis) » à la propriété Language
* Modifier le texte du bouton en « Cancel »
* Affecter la valeur « Espagnol(Espagne) » à la propriété Language
* Modifier le texte du bouton en « Anular »
* Recompiler l’application

On constate que Visual Studio génère plusieurs fichiers resx correspondant aux différentes langues qu’on a choisies :

|  |  |
| --- | --- |
|  | Dans l’explorateur de fichiers Windows, dans le répertoire bin\debug : |

Dans le répertoire de sortie bin\debug, on voit que 3 sous-répertoires ont été générés, contenant chacun une dll qui contient les ressources traduites dans la langue correspondante.

Lors de l’exécution de l’application, un mécanisme du framework .net permet de sélectionner automatiquement la bonne dll de ressource pour afficher les textes dans la langue du poste sur lequel s’exécute l’application.

Pour tester le changement de langue, on peut ajouter le code suivant dans la méthode Main, avant l’affichage de la fenêtre principale :

Thread.CurrentThread.CurrentUICulture = new CultureInfo("es");

NB/ Ce code changer la langue de l’application en espagnol. Il nécessite l’ajout des using suivants :

using System.Globalization;

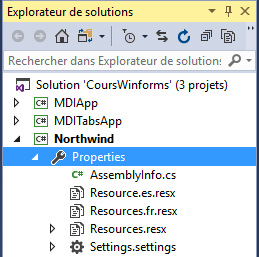
using System.Threading;

A l’exécution, on constate que le texte du bouton est bien affiché en espagnol

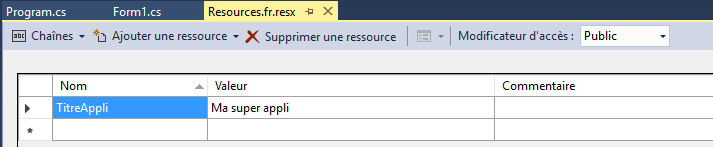
## Ressources de portée application

Un fichier Resources.resx est créé par défaut dans le répertoire Properties du projet. Il est destiné à contenir des ressources partagées par plusieurs formulaires / classes du projet.

Si on souhaite que ces ressources soient localisables, on peut créer nous-même de nouveaux fichiers resx avec les noms adéquats, comme ceux générés automatiquement par VS.



Ces fichiers sont éditables dans une interface visuelle de VS :



Si on veut que les ressources soient accessibles dans d’autres assemblies, il ne faut pas oublier de changer la valeur de la liste déroulante « Modificateur d’accès » (dans le coin supérieur droit de la fenêtre), en public.

Pour accéder à ces ressources dans le code, on utilisera une syntaxe du type :

Properties.Resources.TitreAppli;

NB/ On remarque la similitude avec l’accès aux paramètres d’application.