



Créer sa première application VB.NET

<http://msdn.microsoft.com/vbasic>

SOMMAIRE

1	INTRODUCTION	3
1.1	CONTEXTE FONCTIONNEL	3
1.2	CONTEXTE TECHNIQUE.....	6
2	CRÉER LE PROJET ET LA FENÊTRE PRINCIPALE	7
2.1	CRÉER LA SOLUTION DE PROJET.....	8
2.2	CONTRÔLER LE DÉMARRAGE DE L'APPLICATION	12
2.3	COMPRENDRE LE FONCTIONNEMENT D'UN FORMULAIRE	26
2.4	CONTRÔLER L'AFFICHAGE ET L'ARRÊT DE L'APPLICATION	40
3	TRAVAILLER À BASE DE CONTRÔLES (COMPOSANTS)	42
3.1	CONFIGURER LES CARACTÉRISTIQUES DE LA FENÊTRE PRINCIPALE	44
3.2	CONSTRUIRE LE MENU DE L'APPLICATION.....	48
3.3	CODER LA FERMETURE DE L'APPLICATION.....	57
3.4	AFFICHER L'APPLICATION DANS LA ZONE DE NOTIFICATION	73
4	POUR ALLER PLUS LOIN.....	94

INTRODUCTION

CONTEXTE FONCTIONNEL

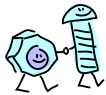
Rappel du contexte fonctionnel du tutorial du coach VB

L'objectif du tutorial du Coach VB est d'accompagner les développeurs à la découverte et la prise en main du langage Visual Basic (VB) pour la construction d'applications avec une approche orientée objet.

Pour rappel, vous pouvez repérer facilement deux caractéristiques importantes du langage à l'aide des logos suivants en marge :



Ce logo marque une fonctionnalité de VB ou de Visual Studio qui permet de développer vite (et juste 😊).



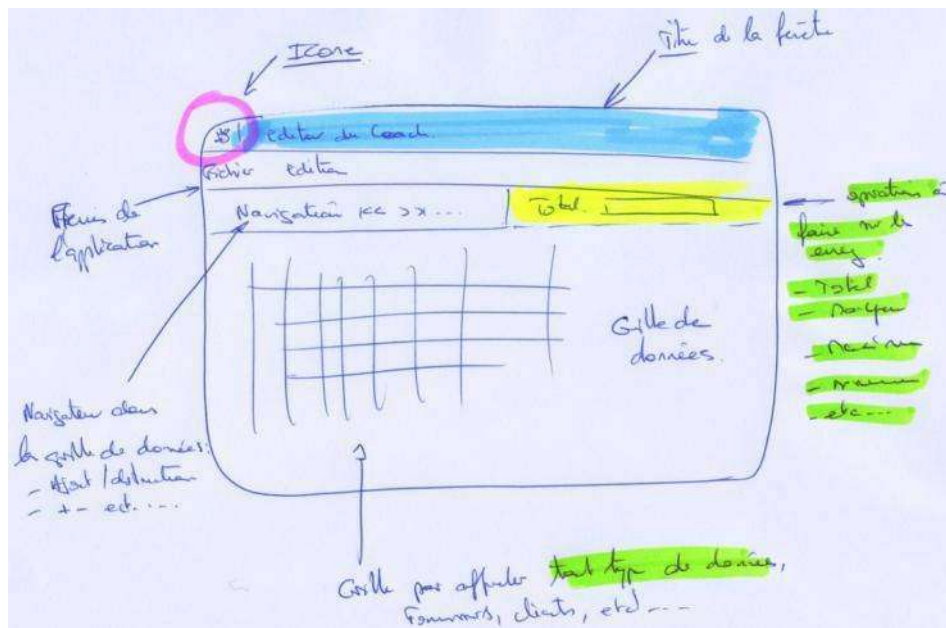
Ce logo met en évidence une caractéristique de la programmation orientée objet.

Contexte fonctionnel du deuxième atelier

Ce deuxième atelier décrit la création d'une première application Visual Basic de type Windows.

L'objectif est de construire une application présentant une fenêtre avec une grille de travail sur des données. Nous l'appellerons **Editeur du coach VB**.

Au mieux vous devez disposer d'un bout de feuille issu d'une réunion qui décrit l'interface de l'application que vous devez réaliser, sur laquelle a abouti l'analyse fonctionnelle. Cela ressemblerait à ceci :



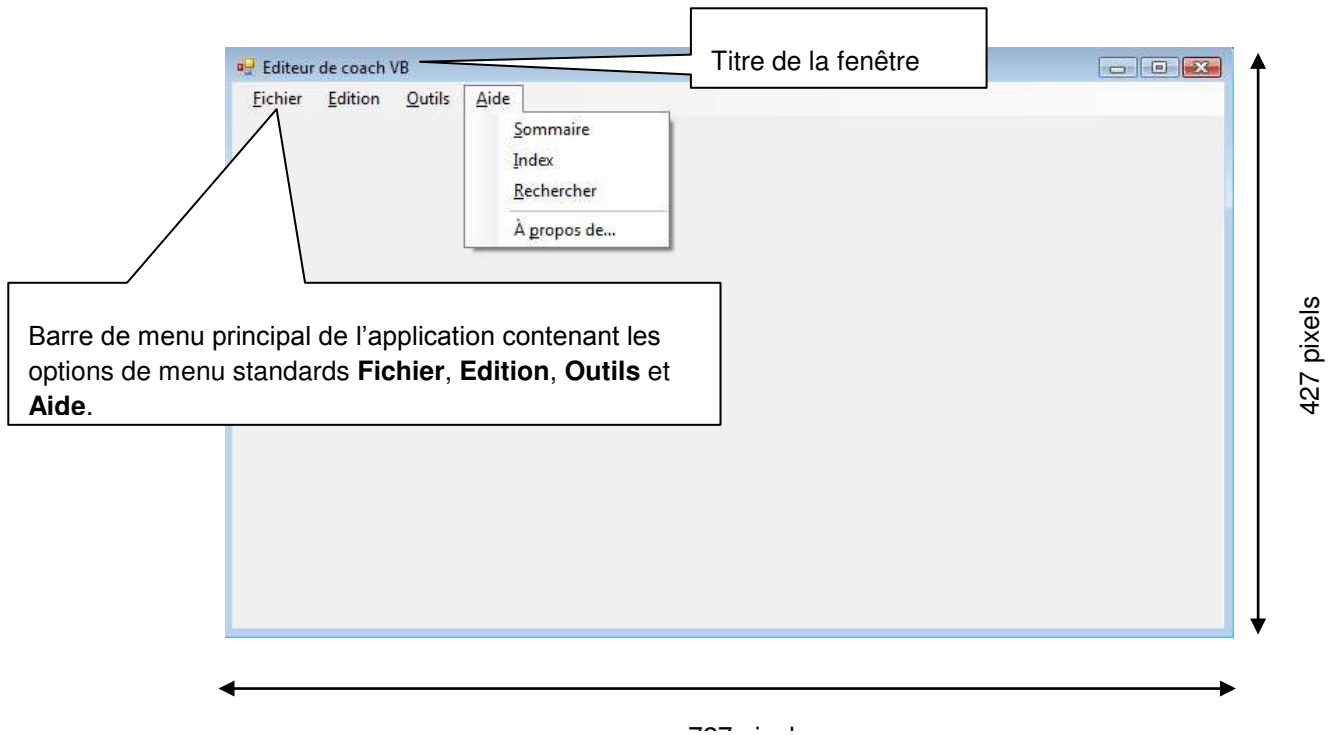
Ca va encore être simple ☺...

Pour l'instant dans cet atelier, nous nous attacherons à construire la « charpente » de l'application sans nous préoccuper de l'affichage des données.

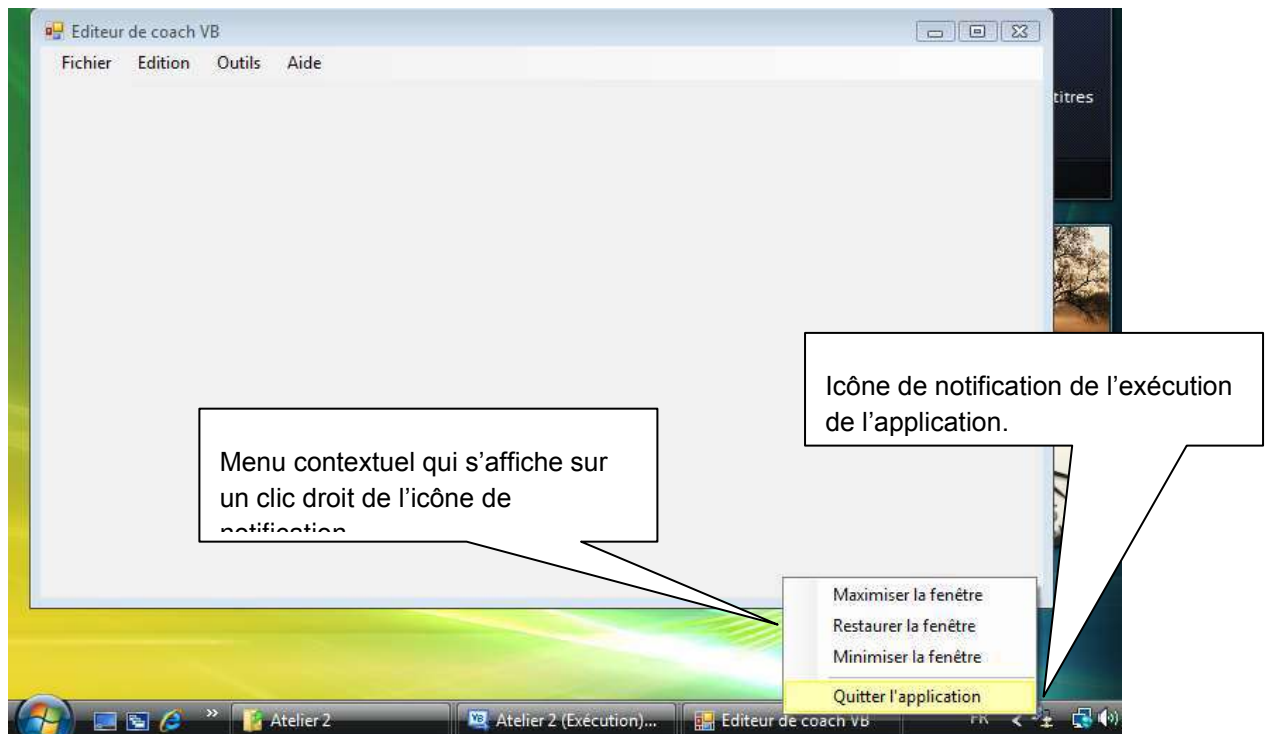
Au lancement de l'application, nous allons afficher un court instant un écran de démarrage donnant le titre, la version et le propriétaire de l'application :



Ensuite l'application devra afficher le formulaire principal de l'application avec la taille définie et une barre de menu contenant les éléments standards que l'on trouve dans les applications Windows.



Enfin, nous mettrons en évidence l'exécution de l'application par une petite icône dans la zone de notification d'état de la barre des tâches de Windows, sur laquelle nous accrocherons un menu contextuel pour permettre à l'utilisateur de fermer l'application.



CONTEXTE TECHNIQUE

Dans cet atelier nous allons mettre de côté (momentanément car nous traiterons le sujet dans les prochains ateliers de ce tutorial) le traitement des données pour nous concentrer essentiellement sur la mise en place de la structure de base d'une application Windows écrite en Visual Basic.

Nous allons créer le formulaire principal, configurer son mode de démarrage ainsi que la façon dont s'arrête l'application. Nous en profiterons pour aborder les principes de programmation essentiels de la programmation dite *évènementielle*.

A la fin de cet atelier, vous saurez comment :

- Créer une application simple à base de formulaire,
- Développer des gestionnaires d'évènement associés à différents types d'évènement,
- Concevoir un formulaire en utilisant des contrôles et des composants Windows Form,
- Utiliser la boîte d'outils et la fenêtre de propriétés de Visual Studio,

- Programmer la zone de notification de Windows,
- Utiliser des classes partielles.

La solution de cet atelier est disponible dans le répertoire **..\Atelier 2\Solution**. Les fichiers utiles, auxquels font référence les exercices sont disponibles dans le répertoire **..\Atelier 2\Fichiers utiles**.

CREER LE PROJET ET LA FENETRE PRINCIPALE

Dans cet exercice, vous allez apprendre à :

-
- Créer un projet de type Application Windows Forms,
 - Créer un écran de démarrage,
 - Utiliser le Concepteur de projets de Visual Studio pour configurer le démarrage et l'arrêt d'une application,
 - Visualiser les différents fichiers qui constituent un formulaire Windows Form,
 - Utiliser les classes partielles.
-

Objectif

L'objectif de ce premier exercice est de démarrer un premier projet de développement de type *Application Windows Forms* et d'en comprendre les principes de base.

Contexte fonctionnel

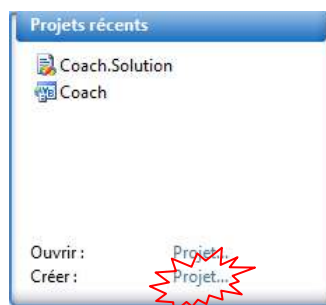
Cet dans ce premier exercice que nous allons créer un écran de démarrage qui s'affichera quelques secondes seulement avant l'affichage du formulaire principal de l'application :



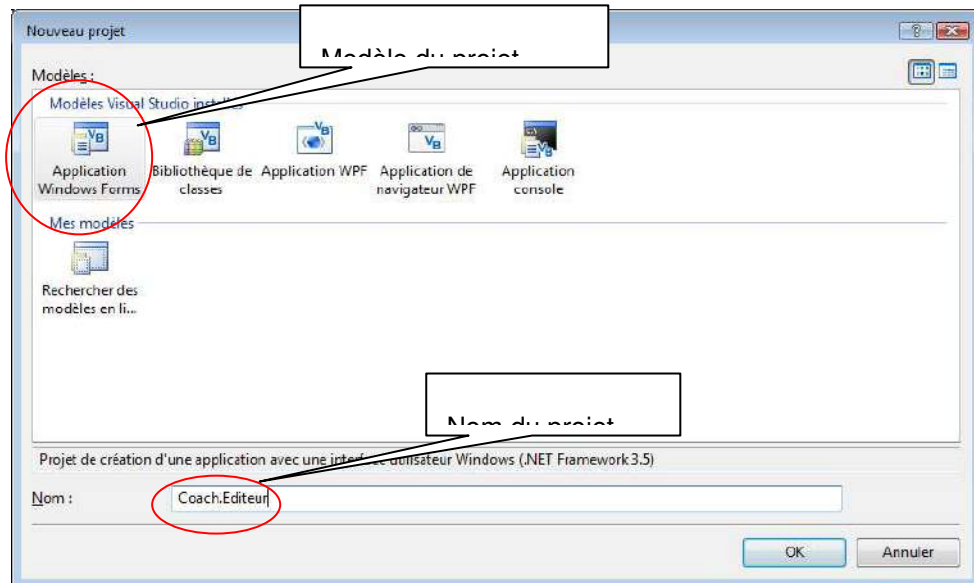
CREER LA SOLUTION DE PROJET

Déroulement de l'exercice :

1. Créez une solution de développement nommée Atelier 2 avec un projet de type Application Windows :
 - Lancez Visual Studio à partir du menu **Démarrer > Tous les programmes > Microsoft Visual Basic 2008 Express Edition**.
 - Créez un nouveau projet depuis l'option **Créer : > Projet...** de la page de démarrage ou à partir du menu **Fichier > Nouveau projet...**



- Dans la fenêtre **Nouveau projet**, sélectionnez le modèle de projet **Application Windows Forms** et indiquez **Coach.Editeur** comme nom de projet.



Ce modèle de projet fait référence aux *Windows Forms*. Savez-vous en quoi consiste cette technologie ?

Dans *Windows Forms*, il y a :

- le mot **Windows** qui fait référence aux applications de bureau riches et interactives que l'on déploie sur un poste client,
- et le mot **Forms** qui fait référence à la notion de formulaire.

Qu'est-ce qu'on entend par *formulaire* ?

Un formulaire est une fenêtre Windows dans laquelle l'utilisateur peut consulter et saisir des informations de manière simple et interactive.

Donc très succinctement, développer une application Windows Forms revient à dessiner un ou plusieurs formulaires puis à coder des traitements pour répondre aux différentes actions réalisées par l'utilisateur avec la souris ou le clavier sur ces formulaires.

Pour vous aider dans cette tâche, le Framework .NET vous fournit toute la « mécanique de base » de fonctionnement de ce type de formulaire de façon à ce que vous perdiez le moins de temps possible à les dessiner et surtout pour que vous puissiez vous concentrer sur la logique métier de votre application. Ce sont les **Windows Forms** comprenant de multiples classes, un Concepteur de formulaire et tout ce qui peut vous aider dans les tâches de programmation courantes de ce type d'application.



Pour avoir une vue d'ensemble des Windows Forms :

<http://msdn.microsoft.com/fr-fr/library/8bxy49h.aspx>

A ne pas confondre avec les Web Forms qui permettent de développer un autre type de formulaire pour les applications web. Pour comparer les deux technologies, rendez-vous à l'adresse suivante :

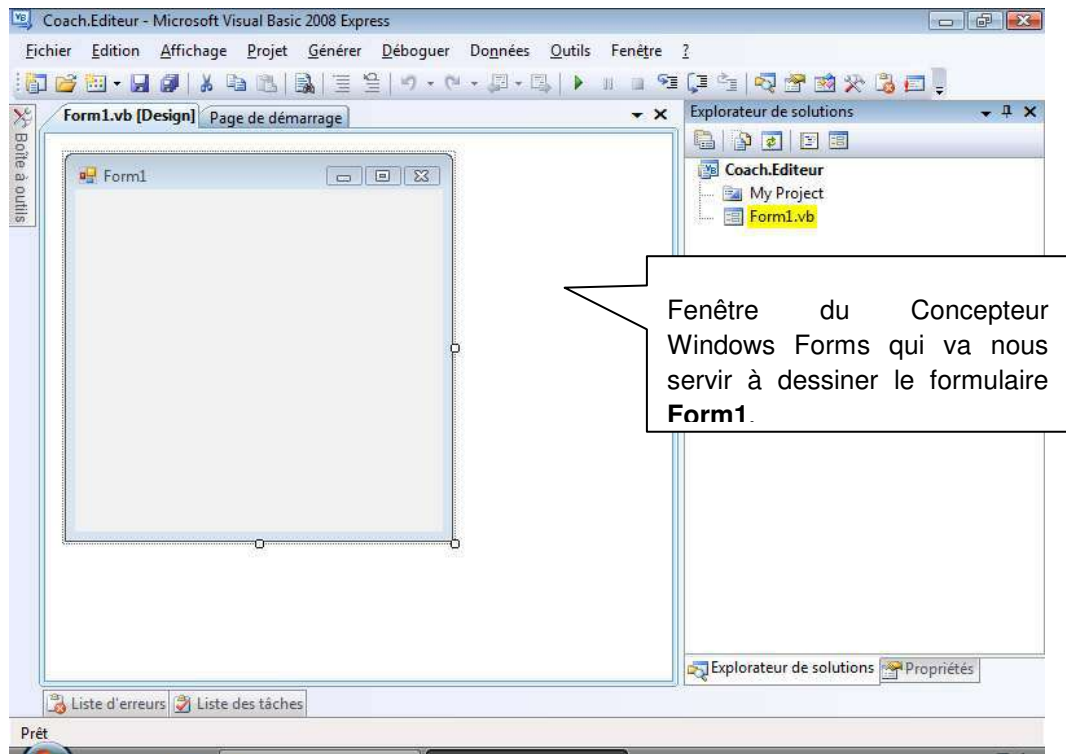
<http://msdn.microsoft.com/fr-fr/library/5t6z562c.aspx>

- Validez par **OK**.




Que contient ce modèle de projet ?

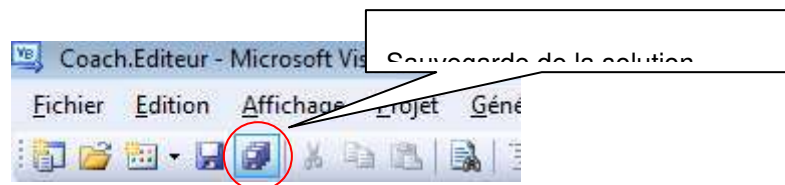
Un projet généré sur la base du modèle **Application Windows Forms** comprend un formulaire vide nommé **Form1** par défaut. Vous pouvez voir dans la surface de travail sa représentation visuelle dans l'onglet **Form1.vb [Design]** généré par le **Concepteur Windows Forms** de Visual Studio.



Pour en savoir plus sur l'outil de design de formulaires clients **Concepteur Windows Forms** :

<http://msdn.microsoft.com/fr-fr/library/e06hs424.aspx>

- Sauvegardez tout de suite le projet et créer une solution en cliquant sur l'icône  dans la barre d'outils standard de Visual Studio.



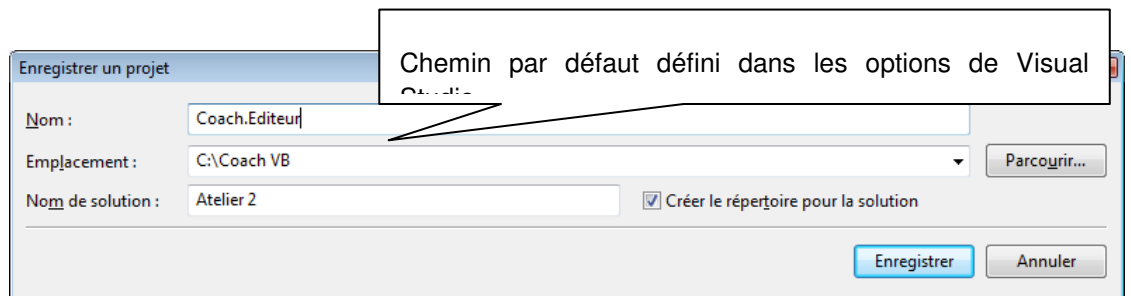
- Dans la boîte de dialogue **Enregistrer un projet**, indiquez votre répertoire de travail.



Par défaut vous devez retrouver le chemin que vous avez spécifié dans la boîte de dialogue d'options de Visual Studio dans l'exercice 3.1 de l'atelier 1 du tutorial.

C'est aussi à ce moment là que vous pouvez demander la création d'une solution en cochant la case **Créer le répertoire pour la solution**.

- Cochez la case **Créer le répertoire pour la solution** et saisissez un nom pour la solution par exemple : **Atelier 2**.



- Cliquez sur **Enregistrer**.

CONTRÔLER LE DÉMARRAGE DE L'APPLICATION



Dans ce type de projet, où est l'indispensable procédure *Main* qui constitue le point d'entrée du programme ?

C'est vrai que nous avons dit à l'exercice 2 de l'atelier 1 de ce tutorial que le Framework .NET appelle la procédure **Main** du programme lorsqu'il a chargé l'application. Elle constitue le point de départ de l'exécution.

Dans le cas d'une application basée sur des formulaires, en général il est surtout intéressant de démarrer l'exécution directement par l'affichage d'un formulaire du projet. En théorie, il faudrait donc écrire une procédure **Main** qui crée une instance du formulaire puis l'affiche.

Pour simplifier, le compilateur Visual Basic génère automatiquement cette procédure pour vous si bien que vous n'avez pas à vous soucier de l'écrire. En revanche, c'est à vous d'indiquer au compilateur quel est le formulaire du projet sur lequel vous souhaitez

démarrer.

Déroulement de l'exercice :

1. Configurez le formulaire de démarrage de l'application :

- Dans l'**Explorateur de Solutions**, double cliquez **My Project** pour faire apparaître le Concepteur de projets de Visual Studio.



Pour rappel, cette fenêtre que nous avons déjà eu l'occasion d'afficher dans l'atelier précédent de ce tutorial, centralise l'ensemble des propriétés et paramètres de votre projet. Dans l'atelier 1, nous l'avons utilisé pour ajouter au projet Console Windows une référence à la dll du Calculateur ou pour configurer des options de compilation.



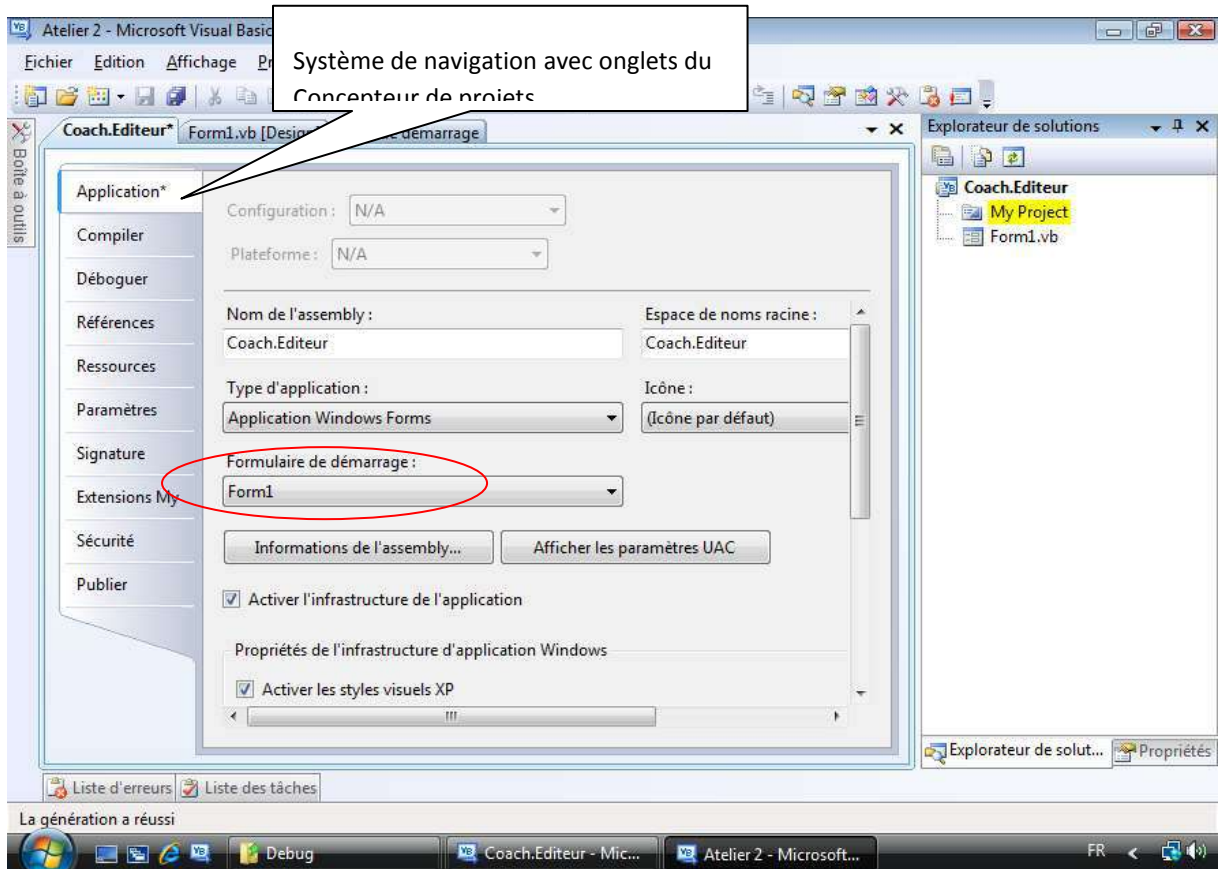
Vous pouvez également afficher le Concepteur de projets en faisant un clic droit à la racine du projet dans l'Explorateur de solutions > **Propriétés**.



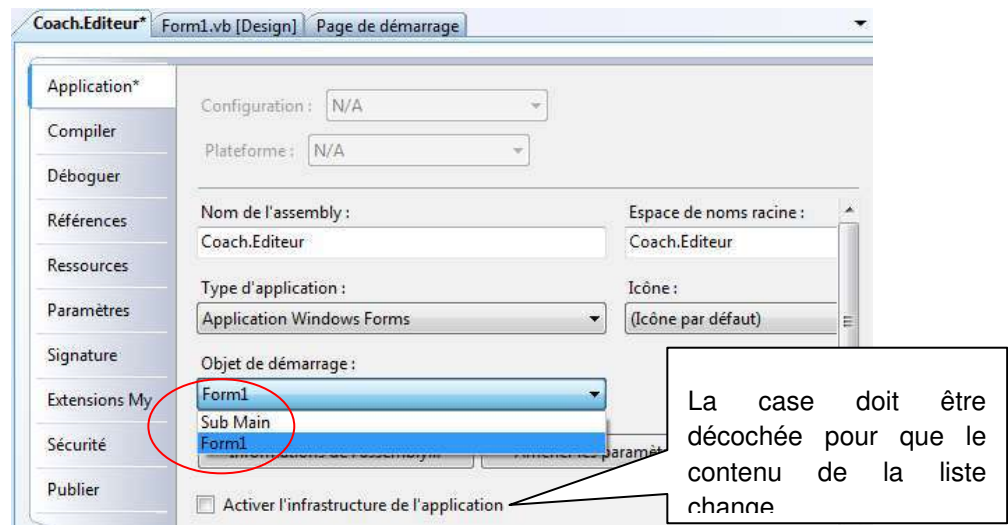
Pour en savoir plus sur le **Concepteur de projets** de Visual Studio :

<http://msdn.microsoft.com/fr-fr/library/bb1aa8f1.aspx>

- Dans l'onglet **Application**, vérifiez que le formulaire de démarrage est correctement configuré sur **Form1**.



Notez que dans l'état actuel du projet qui ne comporte qu'un seul formulaire, il n'y a pas d'autre possibilité dans la liste des formulaires de démarrage. Mais si vous désactivez la case **Activer l'infrastructure de l'application**, une nouvelle option est disponible dans cette même liste qui s'intitule alors **Objet de démarrage**.



L'option supplémentaire **Sub Main** serait utile pour écrire une fonction de démarrage personnalisée à votre convenance. Celle-ci pourrait être définie dans une classe (moyennant le mot clé **Shared**) ou dans un module. Pour afficher le formulaire **Form1** à partir d'une telle méthode utiliser la méthode **Run()** de la classe **Application** :

Code VB

```
Sub Main()  
    Application.Run(New Form1())  
    'Coder ici les autres actions à exécuter au lancement de l'application  
    ...  
End Sub
```

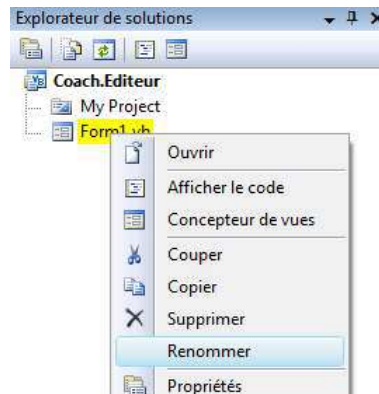


Pour en savoir plus sur la méthode **Run** de la classe **Application** :

[http://msdn.microsoft.com/fr-fr/library/ms157902\(VS.85\).aspx](http://msdn.microsoft.com/fr-fr/library/ms157902(VS.85).aspx)


2. Renommez la fenêtre principale du projet pour qu'elle soit clairement reconnaissable comme étant le point d'entrée de l'application :

- Dans l'**Explorateur de Solutions**, faites un **clic-droit** sur le fichier **Form1.vb**, et sélectionnez le menu **Renommer**.



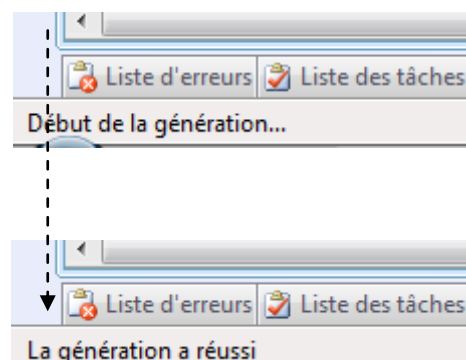
- Changez le nom de **Form1.vb** en **Main.vb**.
- Appuyez sur la touche **Entrée**.

3. Testez le comportement de l'application en mode d'exécution :

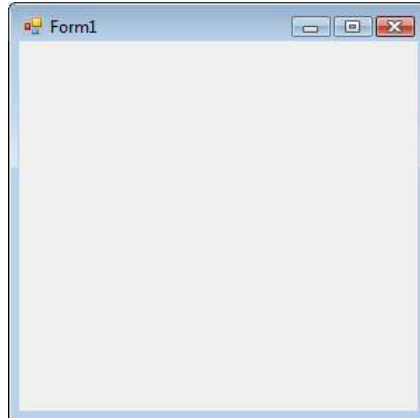
- Lancez l'application en utilisant le menu **Débugage**, ou la flèche  dans la barre d'outils standard ou via le raccourci **F5**.




Visual Studio commence par générer l'application. Vous pouvez observer en bas à gauche de l'écran les messages relatifs au processus de génération :



Puis le formulaire **Main** s'affiche :



- Arrêter l'exécution de l'application en cliquant l'icône  du formulaire.

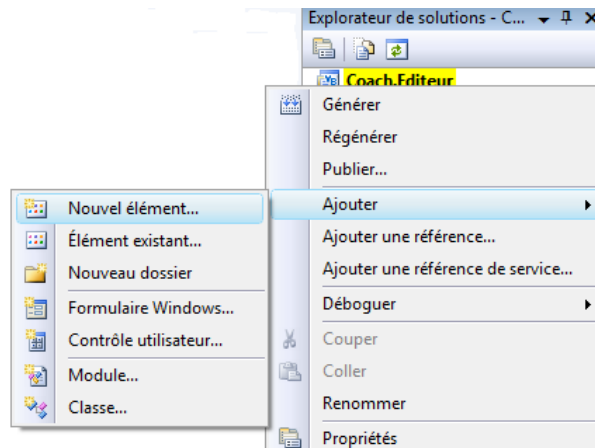


Une alternative à la configuration de démarrage que nous venons de mettre en place consiste à afficher un petit écran de démarrage l'espace de quelques instants *avant* d'afficher le formulaire principal de l'application. C'est d'ailleurs comme cela que démarre Visual Studio.

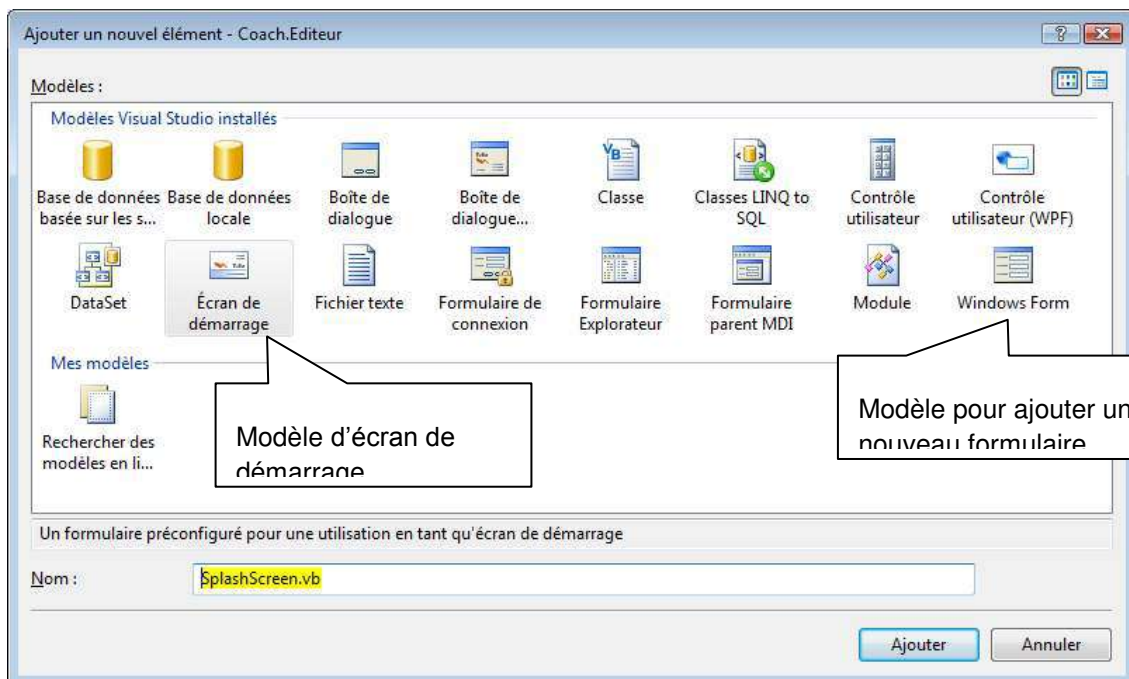
Le temps d'affichage de cet écran intermédiaire peut servir à couvrir le chargement ou l'initialisation de données, à ouvrir des fichiers etc...

4. Ajoutez un écran de démarrage au projet Coach.Editeur :

- Dans l'**Explorateur de Solutions**, faites un **clic-droit** sur la racine du projet **Coach.Editeur** > **Ajouter** > **Nouvel élément...**



- Dans la fenêtre **Ajouter un nouvel élément**, sélectionnez le modèle **Ecran de démarrage**.
- Nommez le formulaire **SplashScreen.vb**.

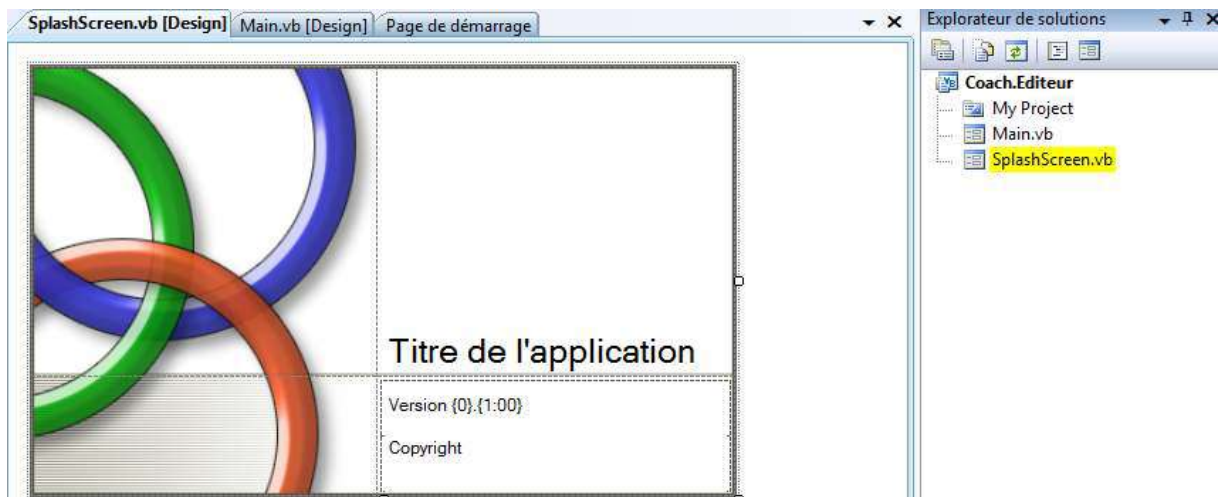




Notez que le modèle **Windows Form** sur lequel est basé votre formulaire **Main** est proposé toute à droite de la fenêtre par Visual Studio. Mais ne perdez pas de vue qu'un projet Windows Forms peut contenir aussi beaucoup d'autres types d'éléments, ce qui explique le nombre de modèles que propose cette fenêtre !

Vous y trouvez par exemple un modèle de création de classe, dont nous verrons l'utilité par la suite dans ce tutorial ; et pourquoi pas aussi un modèle d'**Ecran de démarrage**, basé lui-même sur le modèle Windows Form... Il s'agit d'un formulaire dessiné par défaut avec les dimensions et les zones qui caractérisent ce type de fenêtre.

- Validez en cliquant **Ajouter**.



Vous obtenez en effet un écran constitué par défaut d'une image, d'une zone de titre de l'application et des informations de version et de copyright.



Comment faire pour afficher ce formulaire pendant une durée limitée au démarrage de l'application ?

Vous avez deux possibilités :

- La première, est de vous reposer sur Visual Studio qui, ne l'oublions pas,

s'attache à nous simplifier la vie le plus possible. Puisqu'il sait afficher un formulaire au démarrage via un simple paramétrage dans le Concepteur de projets, pourquoi ne serait-il pas capable de nous afficher aussi un écran intermédiaire l'espace de quelques secondes ?

Cette première approche est dite *déclarative* dans la mesure où nous n'avons qu'à *déclarer* (ou *paramétrer*) les objets et le comportement attendu à Visual Studio pour qu'il sache ce qu'il faut faire. C'est cette option que nous allons mettre en œuvre.

- Mais gardez toujours à l'esprit que tout ce que vous faites avec l'aide de Visual Studio, peut évidemment être fait sans lui, tout simplement en programmant directement les ordres équivalents par code.

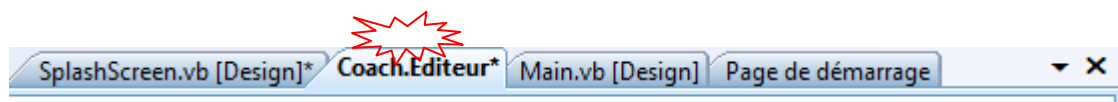
Cette autre approche *par code* est généralement un peu plus longue mais permet d'affiner bien davantage le résultat. Et grâce aux innombrables classes fournies par le .NET Framework, il n'est pas utile de tout réécrire de zéro 😊.

5. Associez le formulaire SplashScreen en tant qu'écran de démarrage du projet Coach.Editeur à l'aide du Concepteur de projets de Visual Studio :

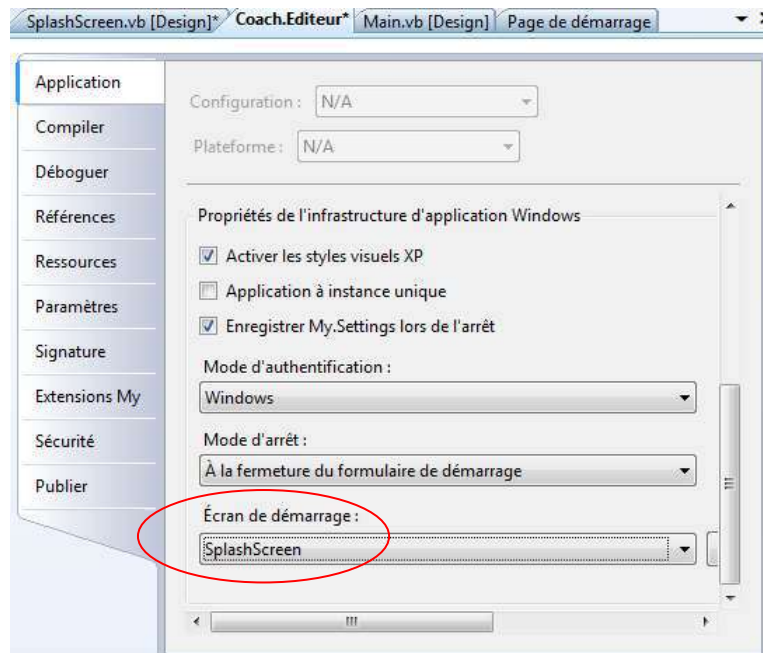
- Dans l'**Explorateur de Solutions**, double cliquez sur **My Project** pour afficher à nouveau le **Concepteur de projets**.




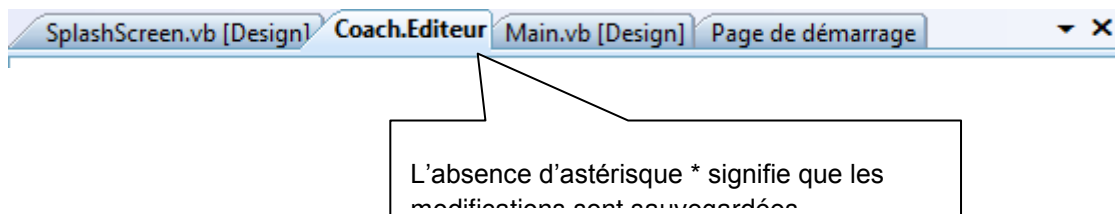
Vous pouvez aussi cliquer sur l'onglet **Coach.Editeur** sur la surface de travail si vous n'avez pas fermé le **Concepteur de projets** auparavant :



- Dans l'onglet **Application**, sélectionnez le formulaire **SplashScreen** dans la liste de la propriété **Ecran de démarrage**.



- Enregistrez vos changements en cliquant sur l'icône  dans la barre d'outils standard de Visual Studio.

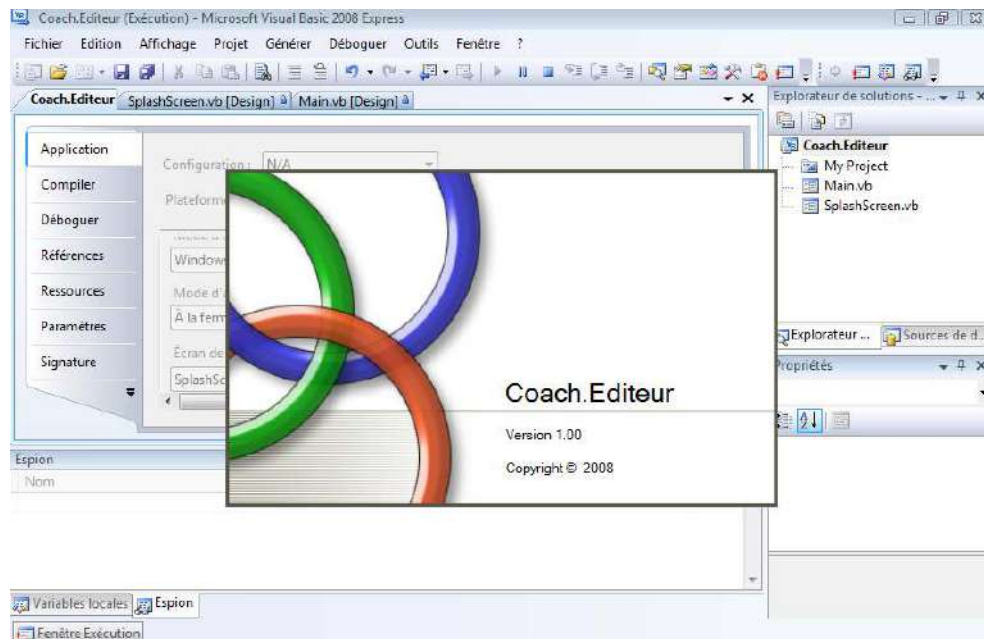


6. Testez le comportement de l'application en mode d'exécution :

- Relancez l'application (F5).



Visual Studio affiche pendant quelques secondes notre formulaire de démarrage. C'est cool !



Mais vous avez pu constater que si le paramétrage de l'écran de démarrage ne nous a demandé que quelques minutes, il ne nous est pas possible en revanche de régler par exemple la durée d'affichage de l'écran. La solution est de la programmer en vous aidant du Framework .NET. Nous aurons bien sûr l'occasion d'utiliser l'approche par code tout au long de ce tutorial.

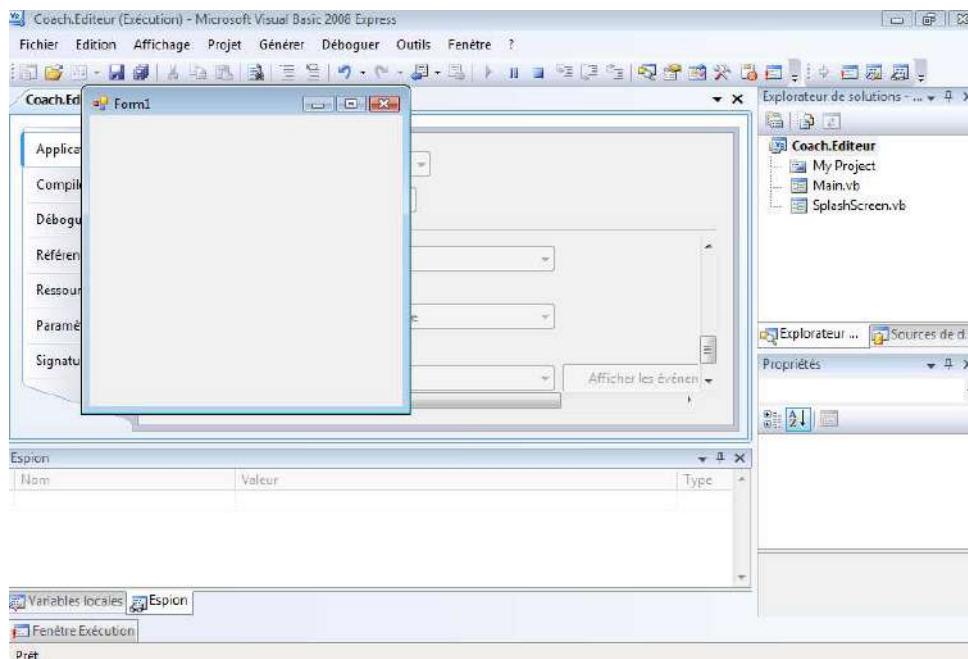



Pour apprendre à contrôler le temps d'affichage de l'écran de démarrage, rendez-vous sur :

[http://msdn.microsoft.com/fr-fr/library/ms234874\(en-us,VS.85\).aspx](http://msdn.microsoft.com/fr-fr/library/ms234874(en-us,VS.85).aspx)



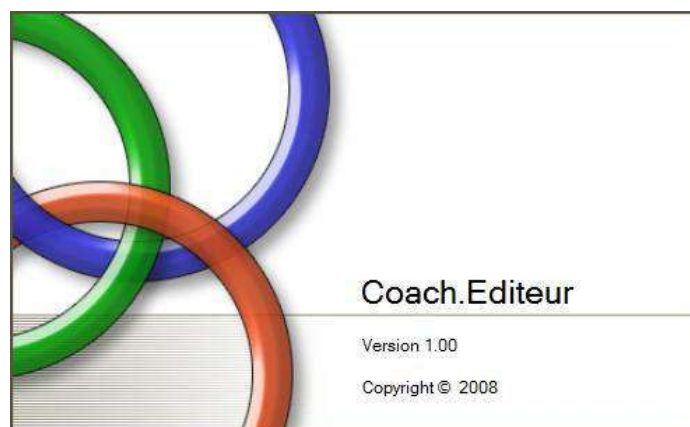
Enfin, au bout de quelques secondes, c'est au tour du formulaire principal de notre projet de s'afficher :



- Arrêter l'exécution de l'application en cliquant l'icône  du formulaire Form1.



Juste une petite question : est-ce que vous êtes certain que c'était bien votre formulaire **SplashScreen** qui s'est affiché au démarrage ?



Comment se fait-il que le titre de l'application soit **Coach.Editeur**, que l'écran indique un numéro de version **1.00** et un copyright avec l'année **2008** ?

Pour rappel, votre formulaire ressemblait à ceci lorsque que vous l'avez dessiné :



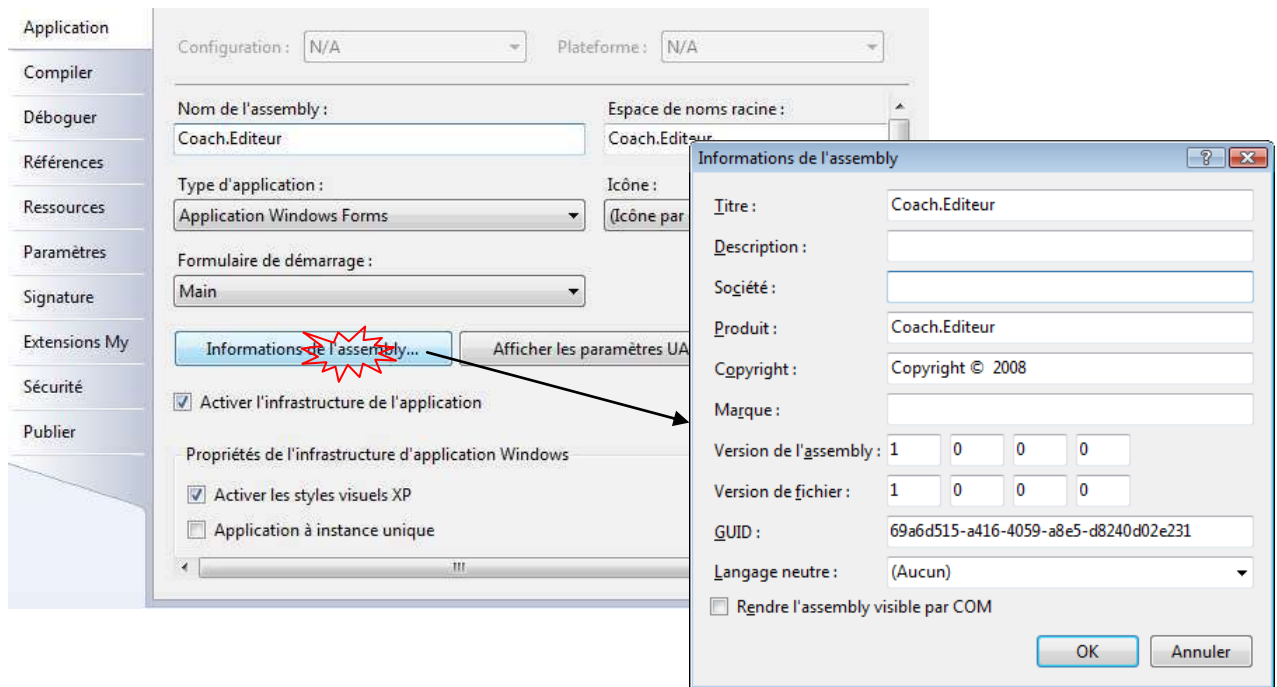
L'explication est simple : le modèle d'élément que nous avons utilisé pour construire l'écran de démarrage ne s'est pas contenté de *dessiner* un formulaire de démarrage type. Il vous a fourni en même temps le *code de traitement* qui contrôle le comportement du formulaire. Ce dernier consiste à afficher les informations de l'application dans les zones correspondantes de l'écran.



Où le code trouve-t-il les informations du programme ?

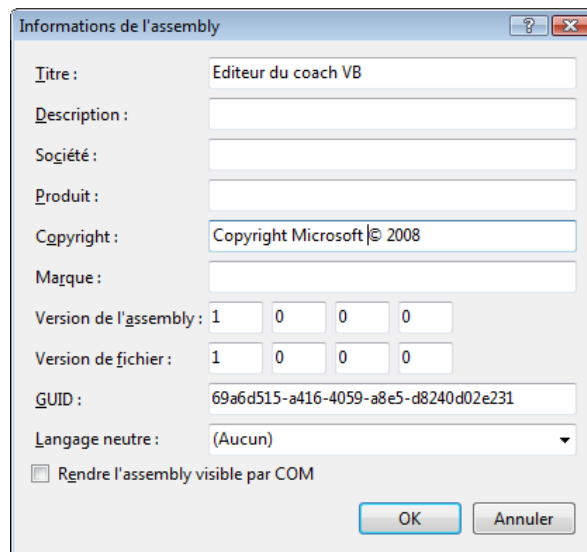
Celles-ci constituent des propriétés du projet configurables directement dans le Concepteur de projets.

- Dans l'**Explorateur de Solutions**, double cliquez sur **My Project** pour afficher à nouveau le **Concepteur de projets** ou basculez sur l'onglet **Coach.Editeur** s'il est encore ouvert.
- Dans l'onglet **Application**, cliquez sur le bouton **Informations de l'assembly...**



Les informations de titre, de version et de copyright récupérées par le formulaire de démarrage sont remplies ici par défaut en s'inspirant du nom de votre projet.

- Remplacez par exemple le titre de l'application par **Editeur du Coach VB**.
- Rajoutez le nom de votre société dans le Copyright.



Informations de l'assembly

Titre : Editeur du coach VB

Description :

Société :

Produit :

Copyright : Copyright Microsoft © 2008

Marque :

Version de l'assembly : 1 0 0 0

Version de fichier : 1 0 0 0

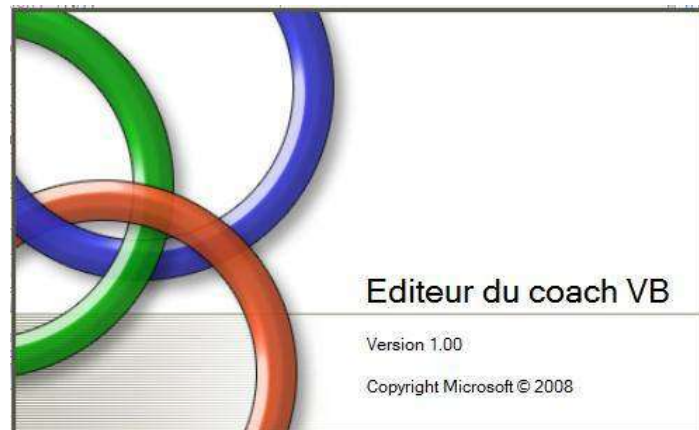
GUID : 69a6d515-a416-4059-a8e5-d8240d02e231

Langage neutre : (Aucun)

Rendre l'assembly visible par COM

OK Annuler

- Validez par **OK**.
- Enregistrez vos modifications dans le Concepteur de projets.
- Validez le nouvel écran de démarrage en relançant l'exécution de l'application (F5) :



COMPRENDRE LE FONCTIONNEMENT D'UN FORMULAIRE

En fait, un formulaire Windows n'est rien d'autre qu'une classe d'objet. Nous allons voir de quelle manière nous pouvons travailler sur cette classe pour l'enrichir.

Déroulement de l'exercice :

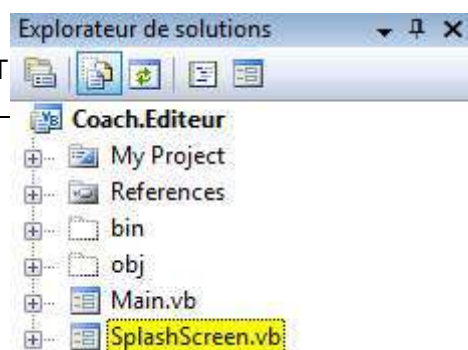
1. Observez la structure du formulaire de l'écran de démarrage SplashScreen :
 - Dans l'**Explorateur de solutions**, double cliquez sur le fichier **SplashScreen.vb**.

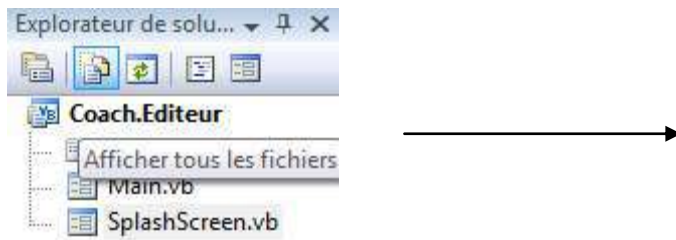


Que se passe-t-il ?

Le Concepteur de formulaire (ou concepteur de vues) de Visual Studio vous affiche l'interprétation graphique du code de définition de la classe. Cette représentation WISIWIG est très utile pour dessiner rapidement le formulaire mais dans les coulisses, tout ce que vous dessinez est automatiquement transcrit au niveau du fichier de code de la classe.

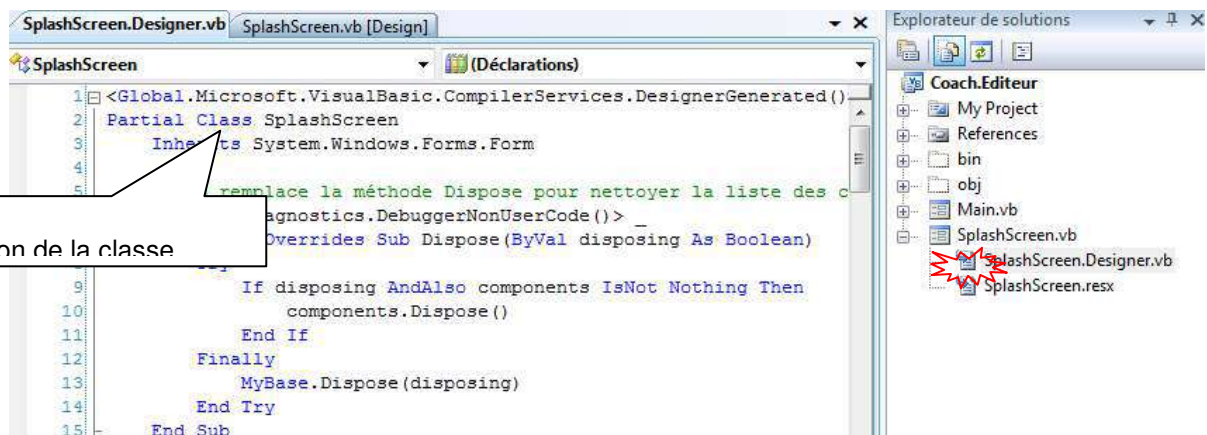
- Pour afficher le fichier de code contenant la définition design du formulaire, cliquez sur l'icône **Afficher tous les fichiers** dans la barre d'outils de l'**Explorateur de solutions**.





Pour éviter de se disperser, l'**Explorateur de solutions** de Visual Studio ne vous affiche pas par défaut l'intégralité de tous les fichiers dans la structure du projet. Par exemple, les dossiers **bin** et **obj** ne sont pas nécessairement très utiles au moment du développement donc sont cachés par défaut. Il en est de même pour les fichiers de code des formulaires générés automatiquement par le Concepteur de formulaire.

- Étendez le nœud  du fichier **SplashScreen.vb** et double cliquez sur le fichier **SplashScreen.Designer.vb**.



Dans ce fichier de code, vous trouvez la définition de la classe du formulaire nommée **SplashScreen** et toutes les informations nécessaires pour construire le panneau contenant l'image, les zones d'affichage du titre de l'application, du numéro de version et

du Copyright.

Exemple de la définition des propriétés de la zone d'affichage du numéro de version

```
72|      '  
73|      'Version  
74|      '  
75|      Me.Version.Anchor = System.Windows.Forms.AnchorStyles.None  
76|      Me.Version.BackColor = System.Drawing.Color.Transparent  
77|      Me.Version.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.0!, S:  
78|      Me.Version.Location = New System.Drawing.Point(3, 3)  
79|      Me.Version.Name = "Version"  
80|      Me.Version.Size = New System.Drawing.Size(241, 20)  
81|      Me.Version.TabIndex = 1  
82|      Me.Version.Text = "Version {0}..{1:00}"
```



Le code généré ici est exactement celui que nous devrions développer à la main si nous voulions faire l'équivalent de ce que fait le générateur du Concepteur de formulaire. Attention, il n'est pas interdit d'à aller y jeter un coup d'œil, mais ne vous lancez pas à le modifier si vous ne savez pas ce que vous faites, sauf pour corriger des éventuelles erreurs de compilation liées à des destructions intempestives de contrôles par exemple.



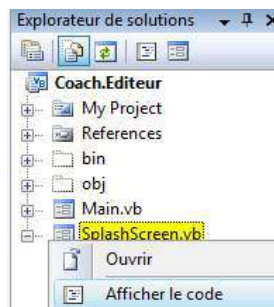
Pour éviter de nous embrouiller, Visual Studio nous demande de coder le comportement du formulaire dans un autre fichier de code séparé de celui généré par le Concepteur de formulaire. Où se trouve cet autre fichier ?

Il s'agit du fichier **SplashScreen.vb**. Pourtant, nous avons vu que lorsque l'on double

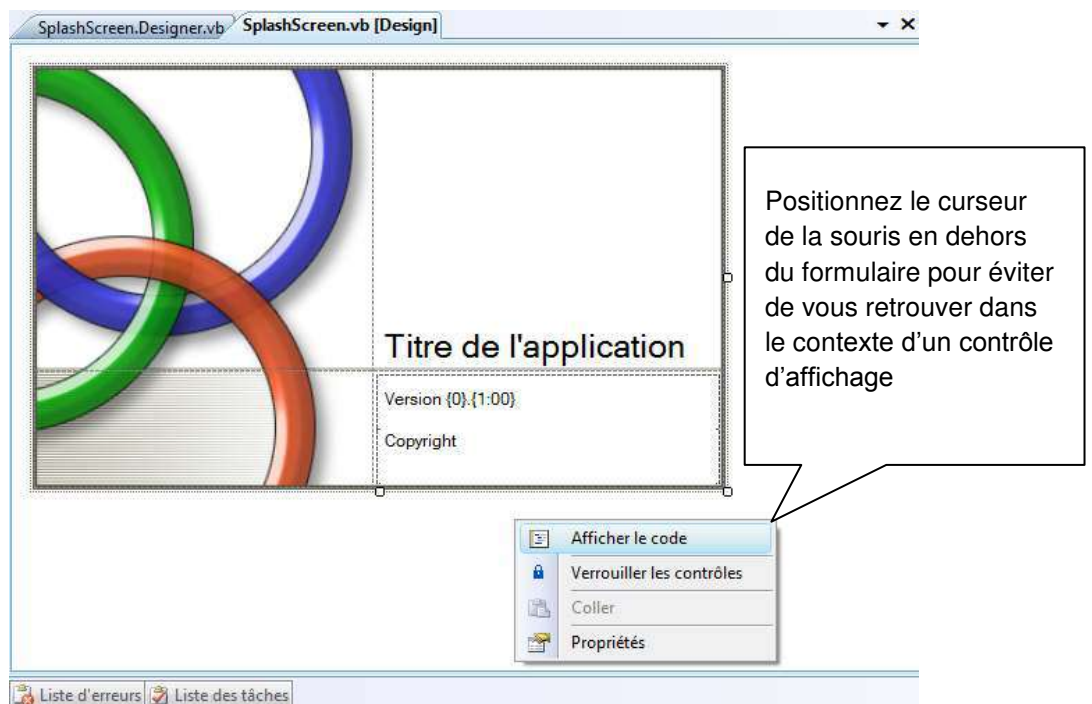
clique sur ce fichier, Visual Studio nous affiche par défaut la représentation visuelle du formulaire. En fait, c'est l'option par défaut parce qu'en général il faut bien commencer par dessiner le formulaire avant d'écrire son comportement 😊.



Pour voir le code du formulaire, vous pouvez suivre plusieurs chemins :

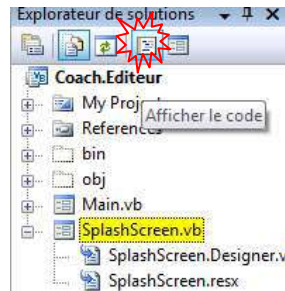
- Faites un clic droit sur le fichier **SplashScreen.vb** dans l'**Explorateur de solutions** > **Afficher le code**.



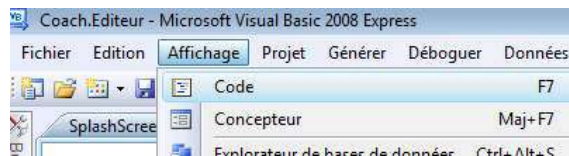
- Faites un clic droit n'importe où sur le Concepteur de formulaire dans l'onglet **SplashScreen.vb [Design]**, puis > **Afficher le code**.



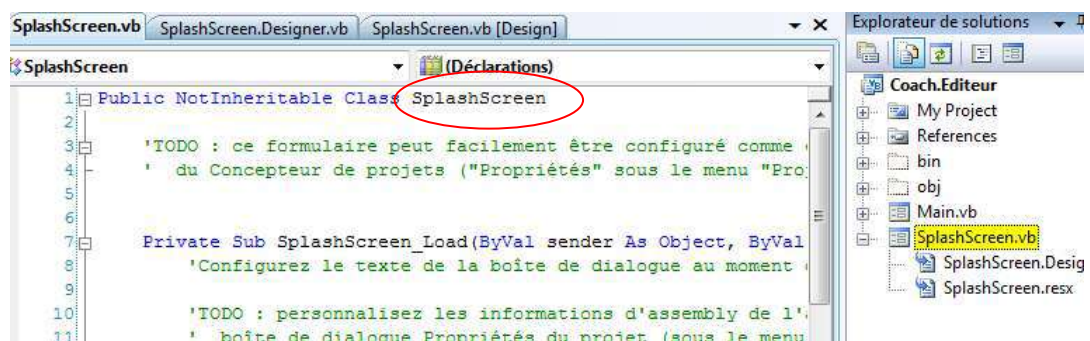
- Sélectionnez le fichier **SplashScreen.vb** dans l'**Explorateur de solutions** puis cliquez l'icône  (Afficher le code) de la barre d'outils. Inversement il suffit de cliquer sur l'icône  (Afficher le concepteur de vues) pour afficher la représentation graphique du formulaire.



- Ou encore Visual Studio vous propose d'afficher le fichier sélectionné par son menu **Affichage > Code**.



Dans ce fichier de code, vous retrouvez la définition de la classe du même nom que dans le fichier **SplashScreen.Designer.vb**.





Comment fait le compilateur pour s'y retrouver dans la définition de la classe puisqu'elle est éclatée dans plusieurs fichiers distincts ?

Il s'agit là d'une propriété du langage Visual Basic proprement dit, qui autorise la définition d'une classe découpée en plusieurs fichiers physiques. Pour indiquer au compilateur qu'il va devoir recoller les morceaux et fusionner l'ensemble des déclarations pour n'en faire qu'une, on ajoute à la directive de classe le mot clé **Partial** :

```
1 <Global>.Microsoft.VisualBasic.CompilerServices.DesignerGe
2 Partial Class SplashScreen
```

Dans le second fichier, le mot **Partial** est omis mais si l'on voulait être rigoureux il devrait apparaître. VB n'autorise d'ailleurs pas plus d'une déclaration à omettre le mot clé.

```
1 Public NotInheritable Class SplashScreen
```

La classe **SplashScreen** est une classe dite **partielle**, c'est-à-dire que la définition de la classe est divisée en plusieurs déclarations.



Pour tout savoir sur les classes partielles en langage VB, cliquez sur :

<http://msdn.microsoft.com/fr-fr/library/yfzd5350.aspx>

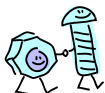
2. Observez le code du comportement du formulaire SplashScreen : il consiste à récupérer les informations de l'assembly pour les afficher dans les zones de l'écran de démarrage correspondant :
 - Affichez le code du fichier **SplashScreen.vb**.
 - Retrouvez la procédure **SplashScreen_Load** qui est exécutée au moment de l'affichage de l'écran de démarrage.
 - Recherchez par exemple les lignes de code qui récupère le titre de l'application :

```
13: 'Titre de l'application
14: If My.Application.Info.Title <> "" Then
15:     ApplicationTitle.Text = My.Application.Info.Title
16: Else
17:     'Si le titre de l'application est absent, utilisez le nom de l'application
18:     ApplicationTitle.Text = System.IO.Path.GetFileNameWithoutExtension _
19:         (My.Application.Info.AssemblyName)
20: End If
```



Que signifie le mot **My** écrit en bleu ?

Comme le Framework .NET contient une multitude de classes et qu'il n'est parfois par simple de s'y retrouver, Visual Basic se propose de vous fournir des raccourcis vers quelques fonctionnalités les plus couramment utilisées du Framework. Comment ? Grâce à un espace de nom qui lui est propre appelé **My**, qui contient un certain nombre de classes organisées hiérarchiquement vous donnant un accès rapide aux informations concernant l'application et son environnement d'exécution. Vous y trouvez par exemple les classes **My.Application**, **My.Computer**, **My.User** ou encore **My.Resources** qui donnent un accès aux ressources de l'application.



D'un point de vue programmation objet, les classes de l'espace de nom **My** sont vraiment très faciles à utiliser car elles n'ont pas besoin d'être instanciées. En quelque sorte, elles vous fournissent des objets immédiatement opérationnels sur lesquels vous pouvez directement travailler. Nous reparlerons de cette notion « d'instanciation » lorsque nous aborderons les principes objet.



Ainsi pour connaître le titre de votre application, il suffit de récupérer la valeur de la propriété **Title** de l'objet **My.Application.Info** dans l'espace de noms **My**. Si celui-ci n'est pas renseigné, le code recherche le nom du fichier de sortie de l'application (auquel on soustrait l'extension) via la propriété **AssemblyName** de l'objet **My.Application.Info**.



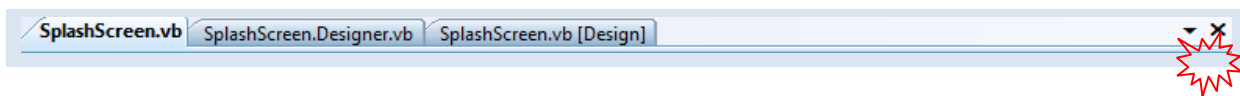
Pour tout savoir sur le développement avec **My** :

<http://msdn.microsoft.com/fr-fr/library/5btzf5yk.aspx>

Pour en savoir plus spécifiquement sur l'objet **My.Application.Info** :

<http://msdn.microsoft.com/fr-fr/library/0f1ec0yf.aspx>

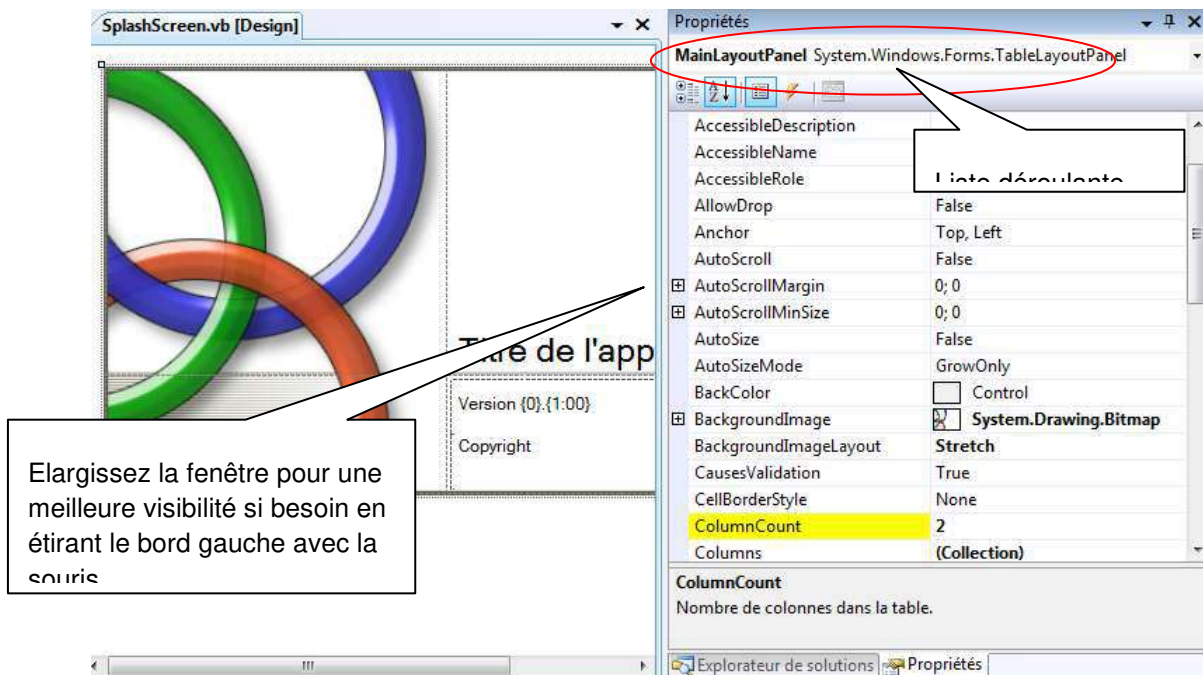
- Fermez le fichier **SplashScreen.vb** en sélectionnant le menu **Fichier** > **Fermer** ou en cliquant la croix à droite de la fenêtre ouverte sur la surface de travail.



D'une manière générale, c'est une bonne pratique de fermer au fur et à mesure les fichiers ouverts sur la surface de travail sur lesquels vous ne travaillez plus.

3. Pour terminer, personnalisez l'image de l'écran de démarrage :





- Affichez le Conceptionneur de formulaire en double cliquant sur le fichier **SplashScreen.vb** dans l'**Explorateur de solutions**.
- Cliquez sur l'image à gauche de l'écran de démarrage pour la sélectionner.
- Faites un clic droit sur la zone > **Propriétés** ou appuyez la touche **F4** pour faire apparaître la fenêtre de propriétés de l'élément d'interface que vous avez sélectionné.
- Dans la fenêtre **Propriétés** qui apparaît sur la droite, vérifiez tout d'abord que vous êtes bien sur le bon objet, dont le nom et le type s'affiche dans la liste déroulante en haut.




Dans le cas d'un formulaire complexe contenant de nombreux éléments d'interface, cette liste déroulante est très utile pour sélectionner directement le bon élément sans passer par le Concepteur de formulaire. D'une manière générale, c'est une bonne pratique que de vérifier le nom du contrôle d'affichage dont vous modifiez les propriétés.



Notez que la fenêtre **Propriétés** est dotée d'une barre d'outils dont les boutons sont les suivants :

- Le bouton  affiche les propriétés en les triant par catégorie ;
- Le bouton  affiche les propriétés en les triant par nom ;
- Le bouton  affiche les propriétés de l'objet sélectionné ;
- Le bouton  affiche les événements de l'objet sélectionné – cette vue sert à ajouter facilement des méthodes de réponses aux événements (nous reviendrons sur ce point plus tard) ;

- Le bouton  affiche une page de propriétés complémentaires de l'objet (si toutefois il en existe une bien sûr).

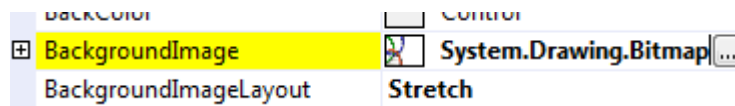



Dans la fenêtre de **Propriétés**, les propriétés affichées en **gras** sont celles dont la valeur a été modifiée par rapport à la valeur fournie par défaut par l'objet d'interface. Du coup, pour chacune de ces propriétés dont la valeur indiquée est différente de la valeur par défaut, le Concepteur de formulaire génère une (ou plusieurs) ligne(s) dans le fichier **SplashScreen.Designer.vb** pour coder le paramétrage correspondant. Merci Visual Studio !

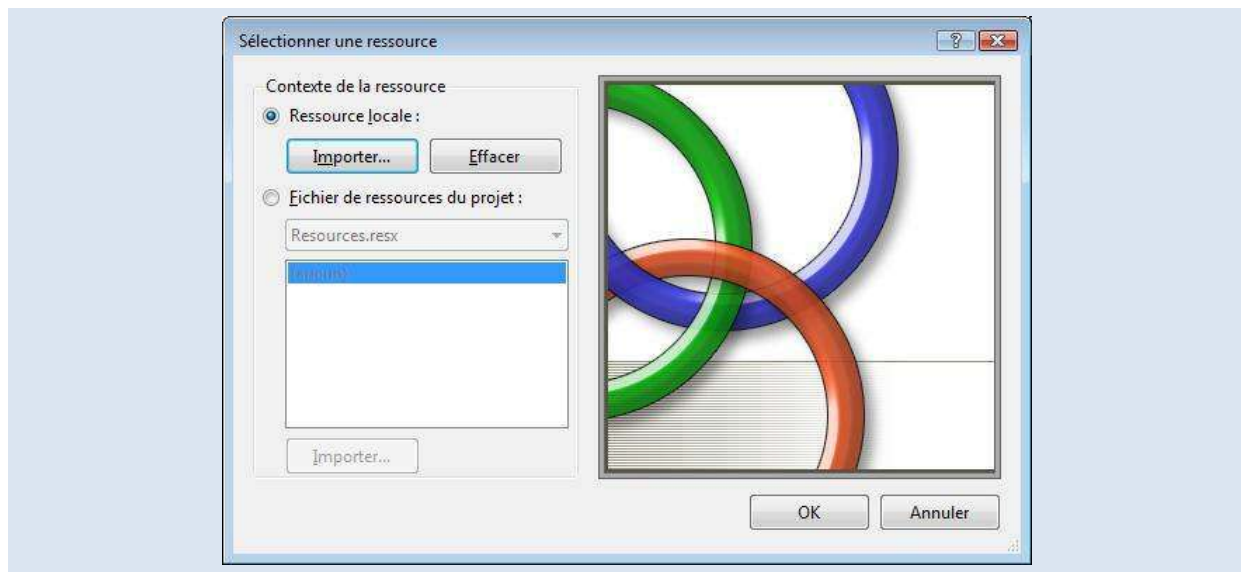


Nous, ce qui nous intéresse est de changer l'image de fond de la zone d'affichage. Pour cela, vous disposez de la propriété **BackgroundImage** qui apparait en gras puisqu'elle a été renseignée avec l'image que vous voyez sur l'écran de démarrage.

- Sélectionnez la ligne de la propriété **BackgroundImage**.

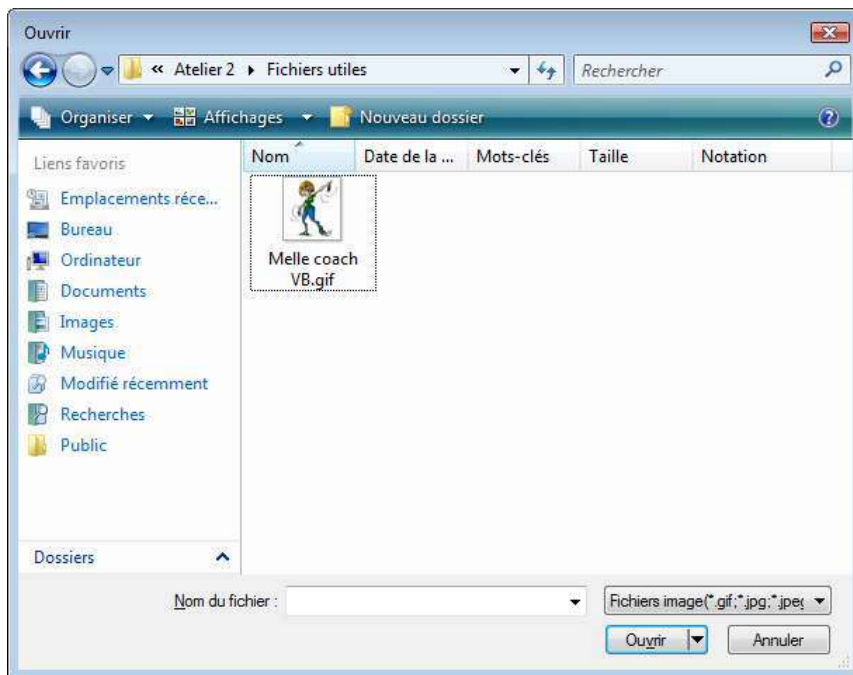


- Sélectionnez le bouton  à droite de la zone de saisie de la valeur de la propriété.

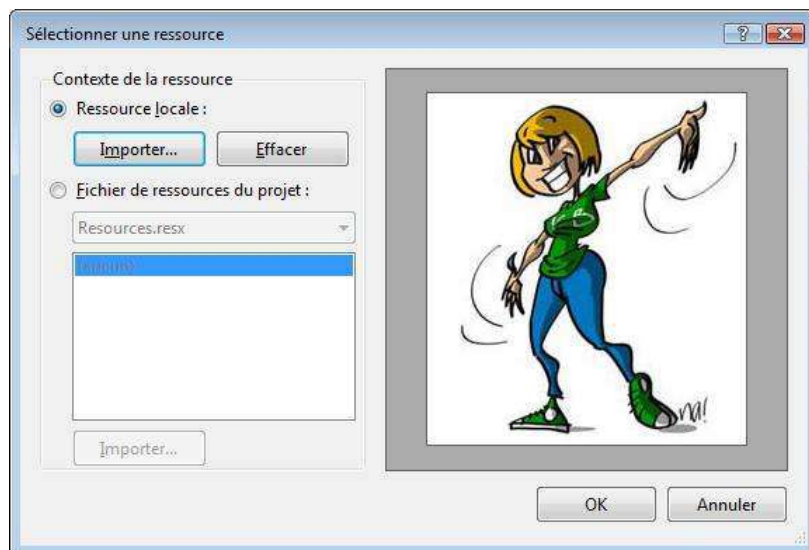


Visual Studio nous assiste dans la sélection de l'image qui se comporte comme une ressource embarquée du projet et est exposée au moment de l'exécution en tant qu'objet **System.Drawing.Bitmap**.

- Cliquez le bouton **Importer....**
- Retrouvez le fichier **Melle coach VB.gif** représentant Melle coach VB dans le dossier **..\Atelier 2\Fichiers utiles** fourni avec l'atelier.

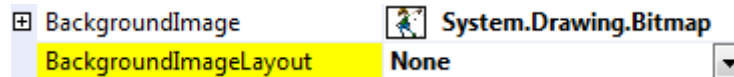


- Cliquez **Ouvrir**.

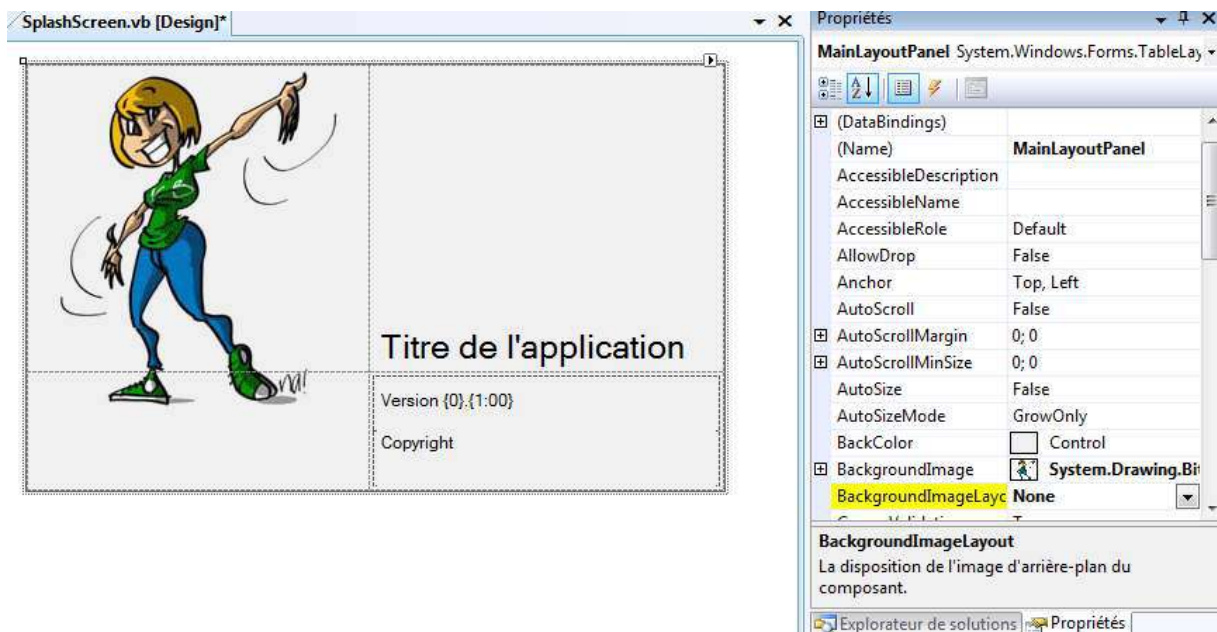


- Cliquez **OK**.

- Configurez la propriété **BackgroundImageLayout** à la valeur **None** pour supprimer l'ajustement (stretch) de l'image sur l'entière surface du contrôle d'affichage.



- Enregistrez vos modifications. Vous devez obtenir :



4. Exécutez l'application pour tester l'écran de démarrage :

- Lancez l'application (touche F5).
- L'écran de démarrage suivant s'affiche pendant quelques secondes avant le formulaire principal de l'application :



CONTROLLER L’AFFICHAGE ET L’ARRÊT DE L’APPLICATION



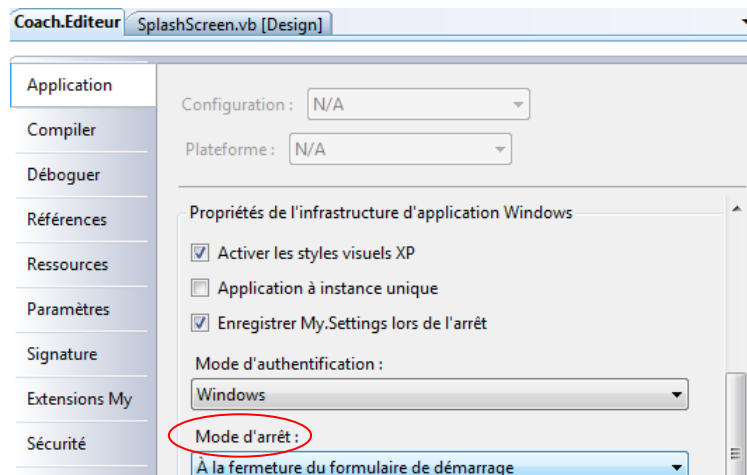
Maintenant que vous savez contrôler le démarrage de l’application, la question est : quand l’application s’arrête-t-elle ?


Comme précédemment, deux possibilités s’offrent à vous : soit vous laissez faire Visual Studio en lui donnant un minimum d’information pour qu’il gère les choses à votre convenance ; soit vous contrôlez par code l’arrêt de l’application.

Dans cet exercice, nous allons voir comment contrôler de manière déclarative l’arrêt du programme.

Déroulement de l’exercice :

1. Configurez Visual Studio pour que l’application s’arrête lorsque l’utilisateur décide de fermer le formulaire principal de l’application :
 - Editez le Concepteur de projets en cliquant sur **My Project** dans l’**Explorateur de solutions**.
 - Dans l’onglet **Application**, vérifiez que l’option **A la fermeture du formulaire de démarrage** est sélectionnée dans la liste de la propriété **Mode d’arrêt**.



Voilà pourquoi l'application se fermait précédemment lorsque vous cliquez l'icône  en haut à droite du formulaire **Main**.

Une seconde valeur **A la fermeture du dernier formulaire** vous est proposée dans cette liste pour le cas où l'application comprendrait plusieurs formulaires. Dans ce cas, il serait certainement plus logique d'interrompre le programme lorsque tous les formulaires sont fermés.



Nous verrons dans la suite de cet atelier comment contrôler l'arrêt de l'application par code. Nous pourrions ainsi proposer à l'utilisateur de quitter l'application de deux manières :

- en cliquant l'option **Fichier > Quitter** du menu principal de l'application
- ou en utilisant le menu contextuel associé à l'icône de notification que nous allons programmer dans la barre de notification d'état de Windows.

Mais pour mettre au point ces deux scénarios, il faut que nous enrichissions le formulaire avec ce qu'on appelle des **contrôles Windows Forms**. C'est l'objet de l'exercice suivant.

TRAVAILLER A BASE DE CONTROLES (COMPOSANTS)

De la même manière que le Framework .NET fournit toute une palette de classes pour nous aider à programmer plus vite le code de l'application, il nous fournit une boîte à outils de contrôles d'affichage pour nous aider à dessiner plus vite l'interface d'un formulaire.



Qu'est ce qu'un *contrôle Windows Forms* ?

Ce n'est ni plus ni moins qu'une classe du Framework .NET ayant une représentation graphique, que l'on peut donc ajouter au design d'un formulaire.

Par exemple : un contrôle de type **TextBox** est un objet basé sur la classe [System.Windows.Forms.TextBox](#) du Framework .NET représentant une zone de texte sur l'écran.

Peut-être avez-vous déjà entendu parler également de *composant Windows Forms* ?

Ce sont des objets similaires aux contrôles Windows Forms à ceci près qu'ils n'ont pas d'équivalence graphique, mais sont néanmoins très utiles au moment du design d'un formulaire.

Par exemple : un composant de type **BindingSource** s'occupe de gérer la source de données liée à un formulaire et s'appuie sur la classe [System.Windows.Forms.BindingSource](#) du Framework .NET.



Pour avoir une vue d'ensemble des contrôles et composants Windows Forms, rendez-vous à l'adresse suivante :

<http://msdn.microsoft.com/fr-fr/library/ettb6e2a.aspx>



Enfin, sachez que vous pouvez bien évidemment développer vos propres contrôles personnalisés dans l'objectif de les partager ou de les réutiliser d'une application à une autre. Pour explorer ce sujet, rendez-vous sur le lien : <http://msdn.microsoft.com/fr-fr/library/6hws6h2t.aspx>

Dans cet exercice, vous allez apprendre à :

- Utiliser les contrôles Windows Form MenuStrip et NotifyIcon,
- Utiliser le composant Windows Form ContextMenuStrip,
- Développer un gestionnaire d'évènement,
- Définir et utiliser une ressource liée au projet.

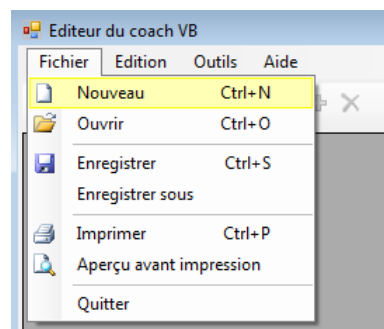
Objectif

Dans cet exercice, nous vous proposons de dessiner le formulaire principal de l'application en utilisant quelques contrôles standards du Framework .NET.

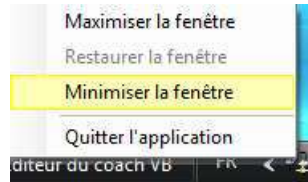
Contexte fonctionnel

L'objectif de cet exercice est de modifier le titre et les dimensions de la fenêtre principale de l'application et d'ajouter à celle-ci deux types de menus :

- Une barre de menu standard qui s'affiche sous le titre de la fenêtre :



- Un menu contextuel, qui va s'afficher quand l'utilisateur fera un clic droit sur une icône s'affichant dans la zone de notification (en bas à droite de l'écran) :



CONFIGURER LES CARACTERISTIQUES DE LA FENETRE PRINCIPALE



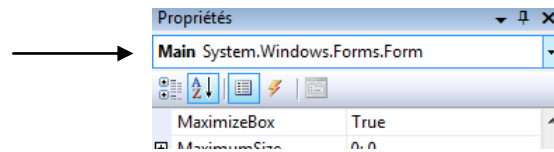
Le formulaire principal que nous avons appelé **Main** dans notre application **Coach.Editeur** est en réalité l'objet conteneur global dans lequel nous allons positionner tous les contrôles d'affichage qui vont servir à dessiner l'écran.

Au même titre qu'un contrôle, un formulaire est donc un objet possédant son propre ensemble de propriétés, de méthodes et d'évènements.

Dans cet exercice, nous allons modifier deux propriétés du formulaire que sont la taille et le titre de la fenêtre.

Déroulement de l'exercice :

1. Affichez la fenêtre de propriétés du formulaire :
 - Affichez le fichier **Main.vb** dans le Concepteur de formulaire.
 - Faites un clic droit n'importe où sur le formulaire > **Propriétés** ou appuyez la touche **F4**.
 - Dans la fenêtre **Propriétés** qui apparaît sur la droite, vérifiez dans la liste d'objets que vous êtes sur l'objet **Main** de type **System.Windows.Forms.Form**.

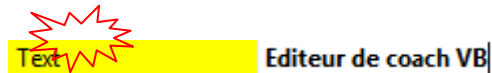


1. Modifiez le texte dans la barre de titre du formulaire :

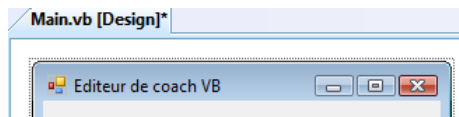
- Dans la liste des propriétés de l'objet **Main**, sélectionnez la propriété **Text**.
- Tapez par exemple la valeur suivante : **Editeur de coach VB**.



Pour saisir la valeur d'une propriété, il suffit de cliquer sur l'intitulé de la propriété et vous pouvez démarrer aussitôt la saisie. En effet, pour gagner du temps, il n'est pas nécessaire de positionner le curseur dans la zone de texte de saisie de la valeur.



- Validez par la touche **Entrée** ou cliquez ailleurs dans Visual Studio. Notez que le titre du formulaire reproduit immédiatement le nouvel intitulé.




Nous allons retrouver cette propriété **Text** au niveau de nombreux contrôles. Elle référence toujours ce qui est affiché par le contrôle à l'écran, que ce soit le texte d'un libellé, le titre d'un bouton etc...




Vous pouvez éditer le fichier **Main.Designer.vb** à partir de l'Explorateur de solutions pour observer la nouvelle ligne de code générée par le Concepteur de formulaire suite à votre modification.

```
21 'Elle peut être modifiée à l'aide du Concepteur Windows
22 'Ne la modifiez pas à l'aide de l'éditeur de code.
23 <System.Diagnostics.DebuggerStepThrough()>
24 Private Sub InitializeComponent()
25     Me.SuspendLayout()
26     '
27     'Main
28     '
29     Me.AutoScaleDimensions = New System.Drawing.SizeF(6
30     Me.AutoScaleMode = System.Windows.Forms.AutoScaleMo
31     Me.ClientSize = New System.Drawing.Size(284, 264)
32     Me.Name = "Main"
33     Me.Text = "Editeur de coach VB"
34     Me.ResumeLayout(False)
```

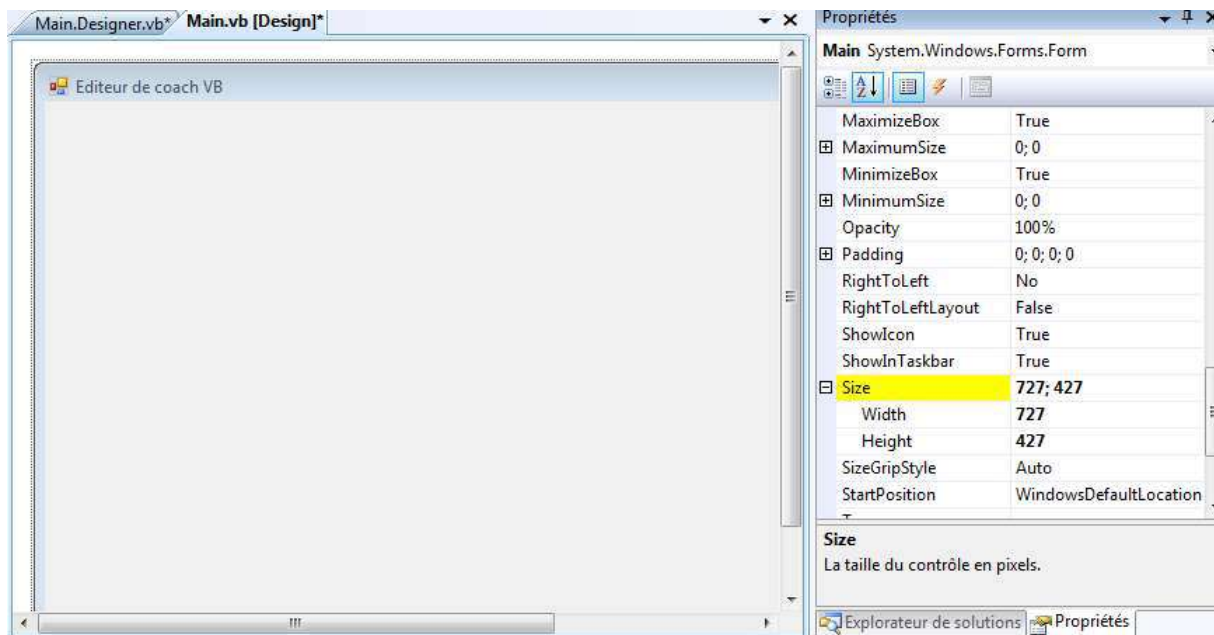
2. Modifiez la taille du formulaire :

- Toujours dans la fenêtre **Propriétés** de l'objet **Main**, sélectionnez la propriété **Size** (dimension).
- Ouvrez-la en cliquant sur , et indiquez la valeur **727** pour **Width** (largeur) et **427** pour **Height** (hauteur).

 Size	727; 427
Width	727
Height	427

Valeurs en pixels

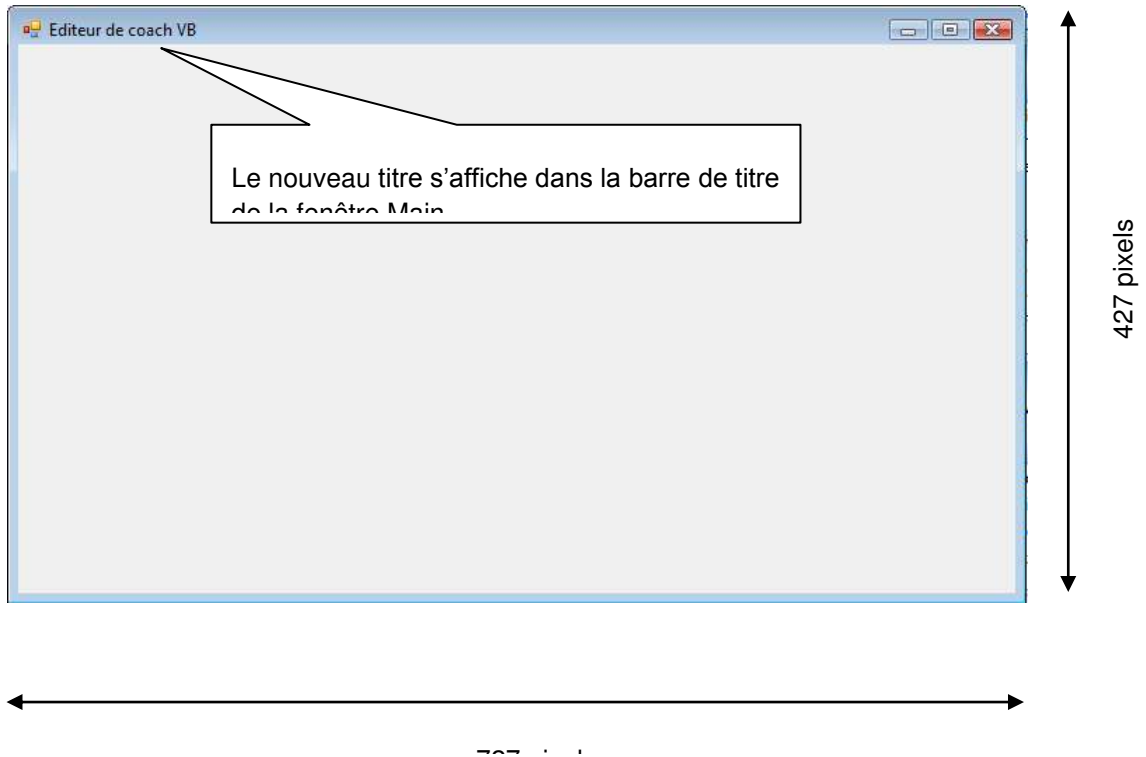
- Le Concepteur de formulaire réajuste automatiquement les dimensions du formulaire en conséquence :



- Enregistrez vos changements.

3. Exécutez l'application pour tester le formulaire :

- Lancez l'application (touche F5).
- Le formulaire s'affiche après l'écran de démarrage :



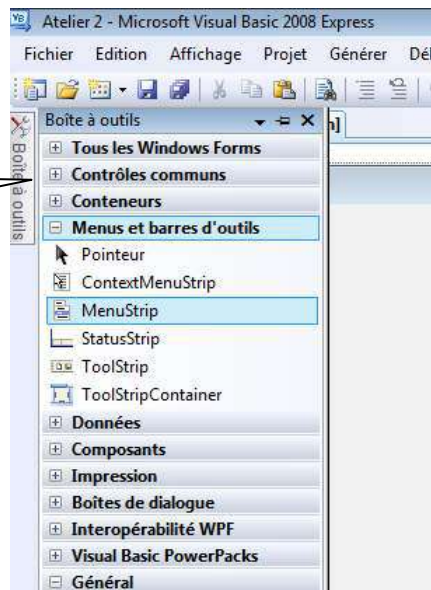
CONSTRUIRE LE MENU DE L'APPLICATION

Dans cet exercice nous allons ajouter notre premier contrôle au formulaire.



Tous les contrôles Windows Forms sont disponibles dans la **Boîte à outils** de Visual Studio qui s'affiche en standard sur la gauche de votre écran. Si elle n'est pas visible, vous pouvez l'afficher en cliquant le menu **Affichage > Boîte à outils** ou avec la combinaison de touches **Ctrl+Alt+X**.

Tous les contrôles et composants sont classés par catégorie.

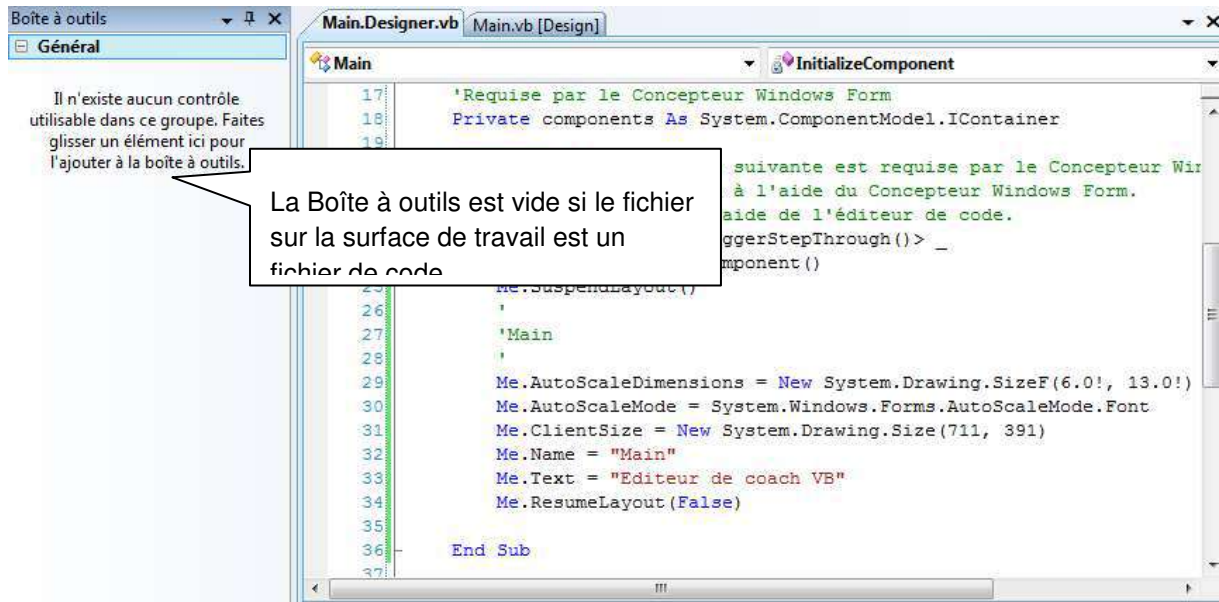


Déroulement de l'exercice :

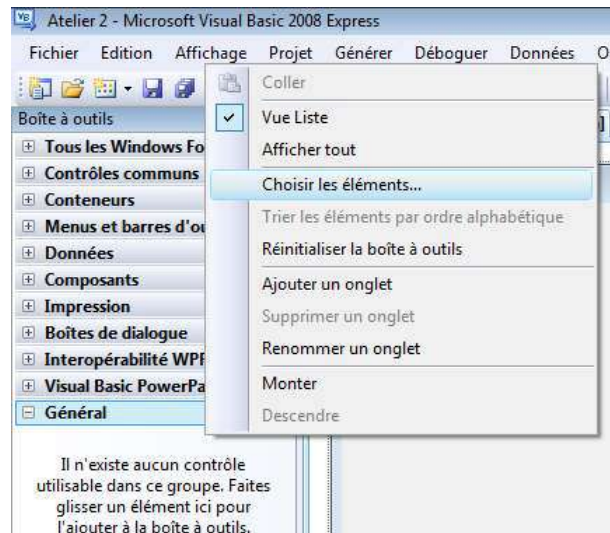
1. Dessinez une barre de menu sur le formulaire **Main** à l'aide d'un contrôle **MenuStrip** :
 - Commencez par afficher le formulaire **Main** en mode Design.



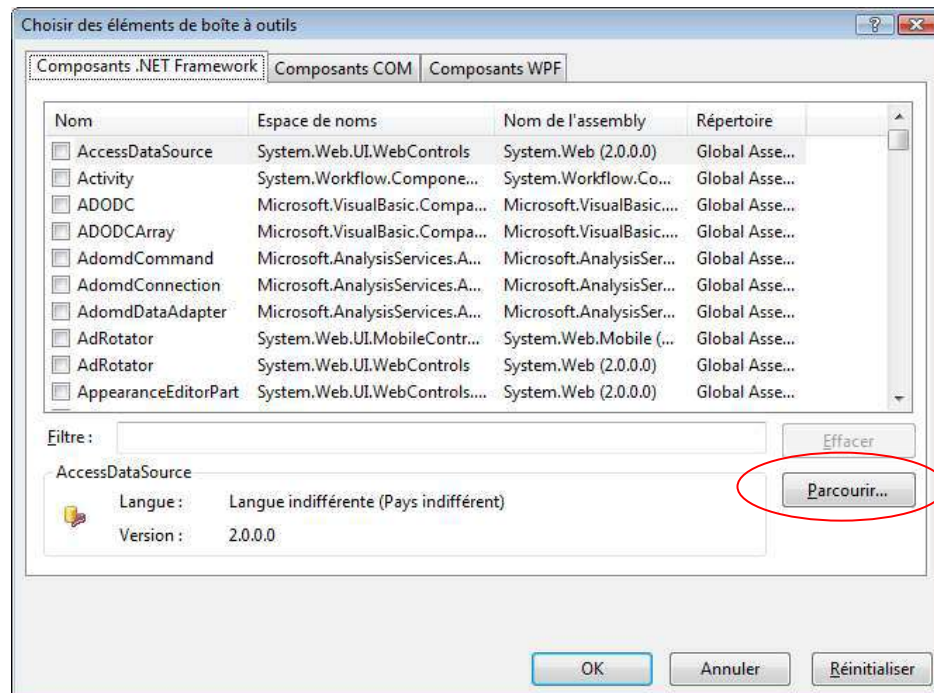
La **Boîte à outils** affiche uniquement les composants qui sont disponibles compte tenu du fichier en cours dans la surface de travail. Par exemple, si la page active est une page de code, la **Boîte à outils** est vide. Voilà pourquoi il est important de commencer par afficher le Conceptionneur de formulaire de la fenêtre sur laquelle vous voulez travailler.



Si vous ne trouvez pas le contrôle qui vous intéresse dans la liste de la boîte à outils, vous pouvez l'ajouter (ou à l'inverse l'en retirer) en faisant un **clic-droit** dans une rubrique quelconque de la **Boîte à outils > Choisir les éléments...** :





La boîte de dialogue **Choisir des éléments de boîte à outils** vous aide à sélectionner les composants ou contrôles dont vous avez besoin. La boîte peut prendre un certain temps à s'afficher du fait de la multitude de contrôles à charger dans la liste.

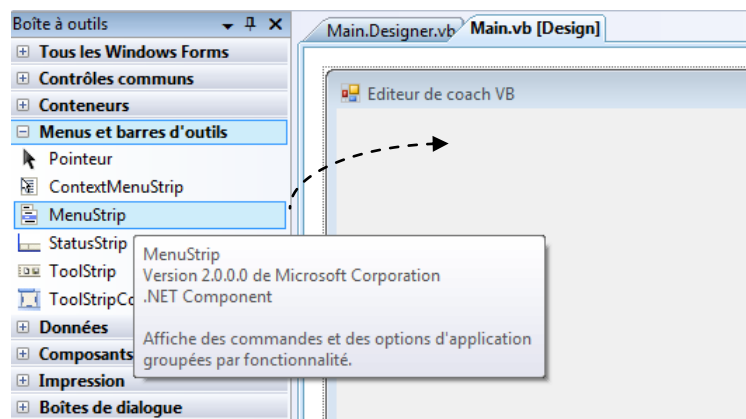


Le bouton **Parcourir** permet de sélectionner directement un assembly (.dll) qui contiendrait les contrôles voulus, à partir d'un emplacement sur disque.



Vous pouvez ajouter des composants issus d'une bibliothèque de contrôles que vous auriez vous-même développer, de nombreuses sociétés tierces à Microsoft, ou de sites communautaires comme <http://windowsclient.net/>. Sur la page d'accueil de ce dernier, sélectionnez la galerie de contrôles **Control Gallery** pour consulter la liste de contrôles et d'exemples téléchargeables.

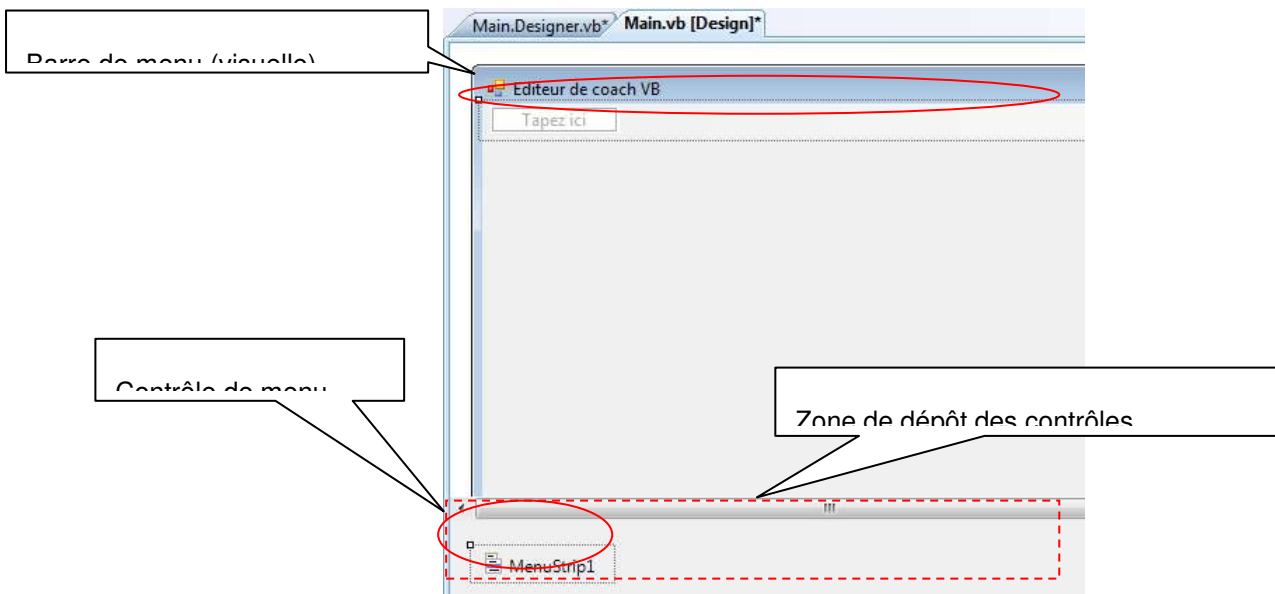
- Ouvrez la Boîte à outils de Visual Studio et fixez-la en cliquant sur l'icône  (sauf si elle est déjà figée et que l'icône affiche au contraire la punaise verticale ).
- Faites un glisser déplacer de la catégorie **Menus et barres d'outils** du contrôle **MenuStrip** n'importe où sur la surface du formulaire **Main**.



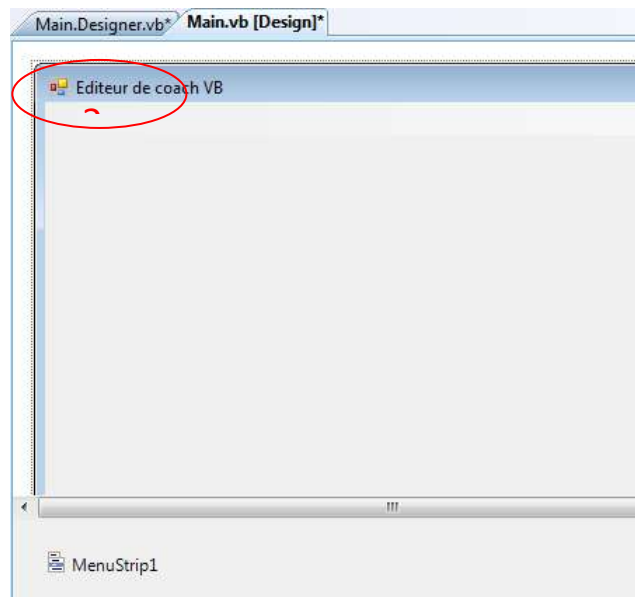


Deux nouveaux éléments apparaissent sur le formulaire :


- Une barre de menu vide sous la barre de titre de la fenêtre,
- Et un composant nommé **menuStrip1** dans une nouvelle zone au bas de la surface de travail.

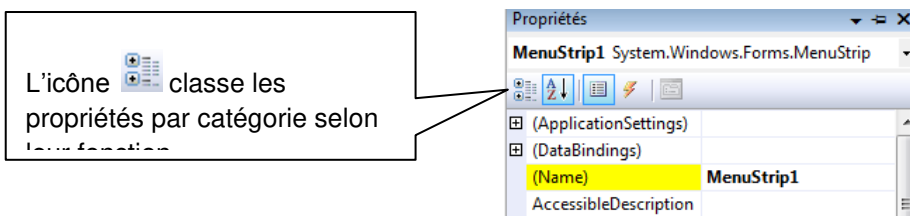


Si vous cliquez maintenant n'importe où sur le formulaire, la barre de menu sous la barre de titre disparaît.



Ceci s'explique par le fait que le menu ne comporte pour l'instant aucune option donc Visual Studio ne sait pas comment le dessiner. D'où l'intérêt de la zone de dépôt au bas du formulaire, qui permet de retrouver quoiqu'il arrive le contrôle, de façon notamment à travailler sur les propriétés de celui-ci.

- Sélectionnez le contrôle **menuStrip1** dans la zone de dépôt. La ligne de menu doit réapparaître en dessous de la barre de titre du formulaire.
- Faites un **clic-droit** sur le contrôle **menuStrip1** > **Propriétés** ou appuyer sur **F4**.
- Dans la fenêtre de propriétés du contrôle, retrouvez le nom du contrôle donné par la propriété **(Name)**. Cette propriété apparaît parmi les premières si vous êtes en classement alphabétique (bouton  de la barre d'outils des propriétés) :





Le nom des contrôles est très important puisqu'il sert à identifier l'objet correspondant dans le code. Soyez donc vigilant quant à la façon dont vous nommez vos contrôles et composants pour les retrouver facilement au moment du codage. Une bonne pratique consiste à établir un plan de nommage précis que vous pouvez suivre dès que vous avez à définir un nom de variable.

- Modifiez le nom du contrôle comme suit :

(Name) `mainMenuStrip`

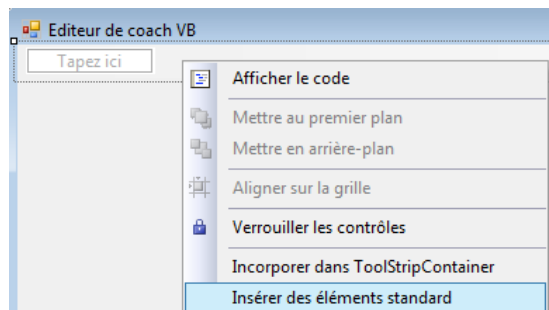


De quoi avez-vous besoin dans la barre de menu ?

L'application que nous développons va servir assez classiquement à manipuler des données. Nous avons donc besoin d'un menu **Fichier** pour manipuler des fichiers de données, d'un menu **Edition** pour les actions standards de copier/coller, d'un menu d'**Aide** etc... Ce ne sont là ni plus ni moins que les menus habituels que vous rencontrez dans toute application Windows. C'est pourquoi le Concepteur de formulaire se propose de vous donner un petit coup de pouce pour dessiner la barre de menu avec les éléments standards d'une application professionnelle.

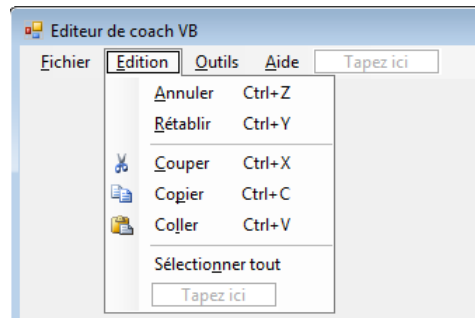
2. Ajoutez les éléments de menu standard de Windows sur la barre de menu `mainMenuStrip` :

- Dans le Concepteur de formulaire, sélectionnez le contrôle `mainMenuStrip` pour faire apparaître la barre de menu sous le titre de la fenêtre.
- Faites un clic droit sur la barre de menu > **Insérer des éléments standard**.





Vous obtenez :



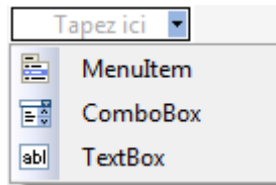
C'est carrément magique non ☺ ?

Par contre, la magie s'arrête au design car bien évidemment Visual Studio vous laisse écrire le code qui doit s'exécuter sur le clic de chacune des options de menu. Mais c'est toujours ça de pris et comme on dit, ce qui est fait n'est plus à faire...



Pour ajouter de nouvelles options de menu, il suffit de cliquer sur à la suite des autres options de menu ou de sous-menu.

Comme vous pouvez le constater avec les éléments standards ajoutés par Visual Studio, le contrôle `MenuStrip` permet de construire des menus dynamiques très riches contenant des images, des informations de raccourcis clavier, des barres d'espacement etc... Si vous cliquez par exemple la flèche ▼ qui apparaît à droite de la zone lorsque vous la sélectionnez, une liste déroulante montre qu'il est possible de créer une zone de saisie (basée sur le contrôle standard `TextBox`) ou une liste de valeurs (basée sur le contrôle standard `ComboBox`) en tant qu'option de menu. L'option `MenuItem` crée une option de menu classique matérialisée par un libellé.



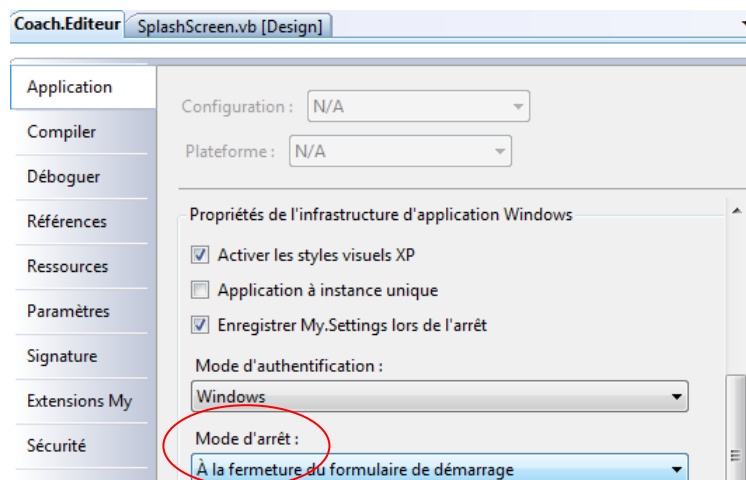
- Avant de poursuivre, enregistrez toutes vos modifications.

CODER LA FERMETURE DE L'APPLICATION

Maintenant que la barre de menu de la fenêtre principale de l'application est dessinée, il faut coder les instructions en réponse au clic de l'utilisateur sur toutes les options de celle-ci.



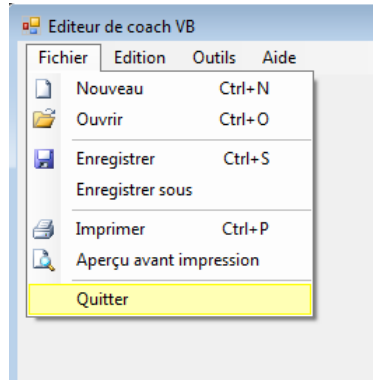
Vous vous souvenez que nous avons configuré précédemment le mode d'arrêt de l'application de manière déclarative dans le Concepteur de projets pour que l'application s'arrête à la fermeture du formulaire de démarrage.



Coder la fermeture de l'application revient donc à coder la fermeture du formulaire **Main**.

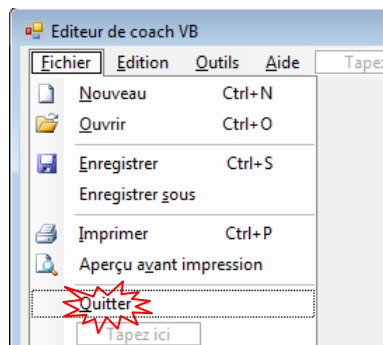
Contexte fonctionnel

Dans cet exercice, nous allons écrire le code de fermeture de l'application associé au clic sur l'option de menu **Fichier > Quitter** de la barre de menu du formulaire **Main**.



Déroulement de l'exercice :

1. Générer une procédure associée au clic sur l'option de menu Quitter :
 - Dans le **Concepteur de formulaire**, sélectionner le menu **Fichier** de la barre de menu du contrôle **mainMenuStrip** puis double cliquez sur l'option **Quitter**.





Visual Studio vous bascule automatiquement dans le fichier de code **Main.vb** et crée une procédure **QuitteurToolStripMenuItem_Click** dans laquelle vous pouvez programmer le code de fermeture du formulaire :

Le design du formulaire est accessible par l'onglet correspondant resté ouvert

```
1 Public Class Main
2
3 Private Sub QuitteurToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ToolStripMenuItem1.Click
4
5 End Sub
6 End Class
7
```



Cette procédure s'appelle un *gestionnaire d'évènement* (*event handler* en anglais) parce qu'elle est exécutée automatiquement lorsqu'un évènement est déclenché.

Pourquoi écrire des gestionnaires d'évènement ?

Le principe de la programmation des Windows Forms repose sur l'approche dite *évènementielle* c'est-à-dire que le code de l'application s'exécute dans un ordre non déterminé à l'avance (par opposition à un code qui s'exécuterait en suivant une séquence prédéfinie), en réponse à une action de l'utilisateur, telle que le clic de la souris ou l'utilisation des touches du clavier sur l'interface de l'application. En somme, si l'utilisateur ne clique jamais sur l'option de menu **Quitteur**, le code de cette procédure destinée à fermer le formulaire ne s'exécutera jamais.

Pour toute action de l'utilisateur, le runtime d'exécution déclenche un **évènement** caractérisant cette action. Il s'offre même le luxe de déclencher d'autres évènements pas nécessairement en rapport avec une action de l'utilisateur, en fait chaque fois qu'il considère qu'il pourrait être opportun de brancher un traitement pour l'application. Par exemple, au chargement du formulaire, se produit l'évènement **Load** pour vous prévenir que c'est le moment idéal pour initialiser certaines données ou contrôles.

Du coup, coder l'application revient à écrire des gestionnaires d'évènement pour s'abonner (comme on s'abonne à un flux RSS) aux évènements percutants pour la logique de votre application et à y brancher le code de traitement approprié.



Comment écrire un gestionnaire d'évènement en réponse à un évènement ? Et oui ! Comment le système sait-il que la procédure **QuitteToolStripMenuItem_Click** est associée au clic sur le bouton **Quitte** du menu de l'application ?



Rappelez vous il y a deux approches possibles, déclarative et par code. En fait, dans notre cas c'est le Concepteur de formulaire qui a automatiquement paramétré la procédure pour nous, et ce en utilisant la méthode déclarative.



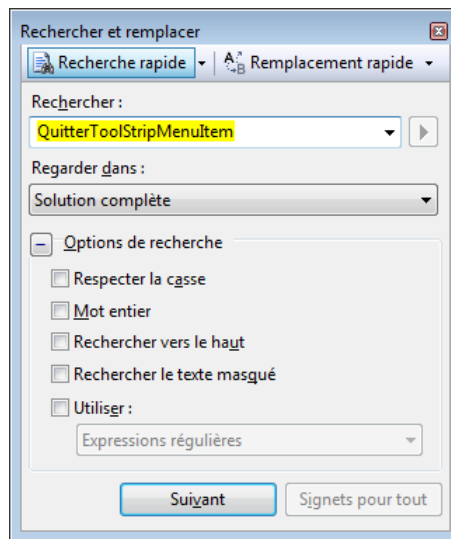
Regardons ensemble ce qui a été généré.

Contrairement à ce qu'on pourrait croire, le nom *QuitteToolStripMenuItem_Click* de la procédure n'est pas en cause !

Pour comprendre comment ça marche, il faut d'abord retrouver la déclaration de l'option de menu **Quitte** dans le fichier **Main.Designer.vb**.

2. Retrouvez la déclaration de l'objet associé à l'élément de menu Quitte :

- Ouvrez le fichier **Main.Designer.vb** à partir de l'**Explorateur de solutions**.
- Dans le menu de Visual Studio, cliquez sur **Edition > Recherche rapide** et rentrez le nom de la variable **QuitteToolStripMenuItem** dont on souhaite retrouver la définition.



- Cliquez sur **Suivant** jusqu'à retrouver la ligne suivante :

```
289| Friend WithEvents toolStripSeparator2 As System.Windows.Forms.ToolStripSeparator
290| Friend WithEvents QuitterToolStripMenuItem As System.Windows.Forms.ToolStripMenuItem
291| Friend WithEvents EditionToolStripMenuItem As System.Windows.Forms.ToolStripMenuItem
```



Notez tout d'abord le type de la variable **System.Forms.ToolStripMenuItem**. Il va nous servir dans quelques instants.



La clause **WithEvents** est le premier élément nécessaire à la construction d'un gestionnaire d'évènement. En définissant la variable **QuitterToolStripMenuItem** à l'aide de **WithEvents**, cela signifie que vous autoriser l'écriture de procédures en réponse aux évènements de cette variable.



Notez que la définition de **QuitterToolStripMenuItem** a été générée automatiquement lorsque vous avez dessiné le menu sur le formulaire (via l'option d'insertion des éléments standards). Elle correspond à la définition de la variable associée à l'option de menu dans votre programme.

Si vous cliquez plusieurs fois sur le bouton **Suivant** de la fenêtre de recherche, vous constaterez qu'il y a également d'autres références à cette même variable, par exemple pour reproduire dans le code, l'intitulé (**Text**) de l'option de menu, son positionnement (**Size**) et son identifiant (**Name**).

```
171 | QuitterToolStripMenuItem  
172 |  
173 | Me.QuitterToolStripMenuItem.Name = "QuitterToolStripMenuItem"  
174 | Me.QuitterToolStripMenuItem.Size = New System.Drawing.Size(205, 22)  
175 | Me.QuitterToolStripMenuItem.Text = "&Quitter"
```

Le caractère & devant la lettre Q définit un raccourci clavier pour ce menu

- Fermez le fichier **Main.Designer.vb**.



Le second élément pour écrire un gestionnaire d'évènement se trouve au niveau de la définition de la procédure elle-même. Il s'agit du mot clé **Handles** qui fonctionne de paire avec la clause **WithEvents**.

3. Observez la signature du gestionnaire d'évènement généré dans le fichier de code :

- Revenez sur la définition de la procédure **QuitterToolStripMenuItem_Click** dans le fichier **Main.vb**.

Code VB

```
Public Class Main
```

```
Private Sub QuitterToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles QuitterToolStripMenuItem.Click
```

```
End Sub
```

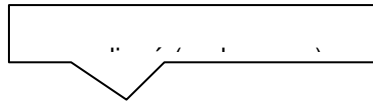
```
End Class
```



Le mot clé **Handles** est le second élément nécessaire à la construction d'un gestionnaire d'évènement. Pensez à la littérature anglaise qui parle d'event *handler* là où nous parlons de gestionnaire d'évènement. Il signifie que la procédure **QuitterToolStripMenuItem** gère (*handles*) l'évènement **Click** de la variable **QuitierToolStripMenuItem**.

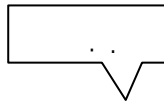
Attention une fois encore à la terminologie utilisée !

- Visual Studio nomme la procédure par défaut avec le format :



<nom de la variable>_<nom de l'évènement>

- Alors que le nom de l'évènement est noté :



<nom de la variable>.<nom de l'évènement>

En fait le nom de la procédure importe peu. Vous pourriez par exemple remplacer sans hésiter *QuitierToolStripMenuItem_Click* par *Quitier*.

En revanche, le nom de l'évènement auquel vous l'accrochez est très important, de même que la signature de la procédure.

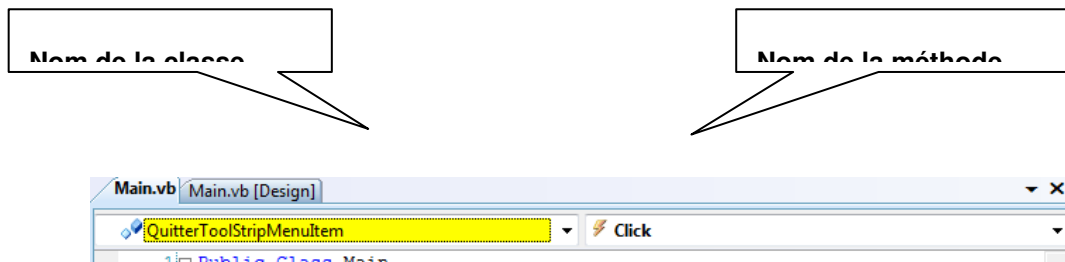


Lorsque vous double cliquez sur un objet en mode Design, Visual Studio génère automatiquement la procédure de réponse à *l'évènement par défaut* associé à un objet, ici l'évènement **Click** sur un objet de type **System.Forms.ToolStripMenuItem**. Mais ce type d'objet dispose de bien d'autres évènements qui vous permettent d'interagir sur son fonctionnement.

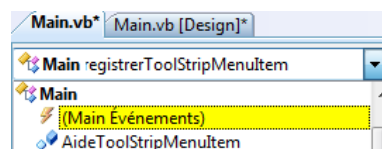
Où trouver le nom de l'évènement auquel s'abonner ?

En haut de l'éditeur de code de Visual Studio, vous trouverez une barre contenant deux listes déroulantes. Elle donne une liste des évènements disponibles pour chaque objet du formulaire.

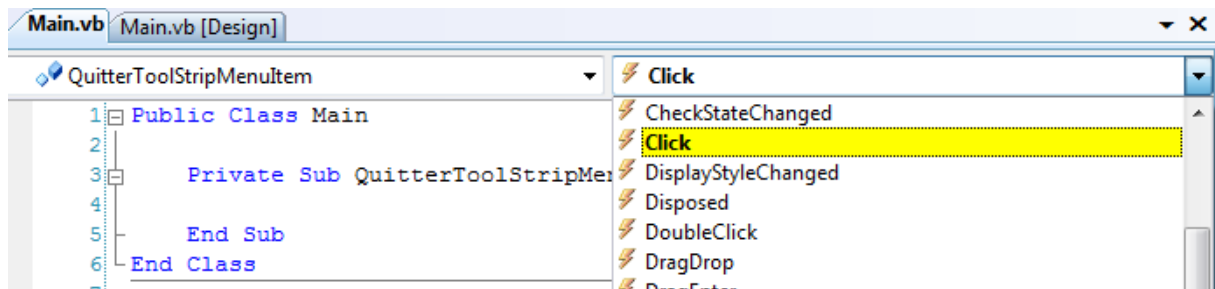
- Dans la liste de gauche **Nom de la classe**, sélectionnez l'objet **ToolStripMenuItem** sur lequel porte l'évènement attendu.



Tous les autres éléments de la liste sont des objets qui font partie de votre formulaire, pour la plupart générés par l'action d'ajout des éléments standards au menu **mainMenuStrip**. Le premier élément de la liste, **(Main Évènements)** correspond à l'objet formulaire lui-même pour lequel le Framework .NET fournit toute une batterie d'évènements caractérisant le cycle de vie du formulaire.



- Dans la liste de droite **Nom de la méthode**, sélectionnez l'évènement auquel vous voulez vous abonner pour l'objet sélectionné.



L'évènement **Click** apparaît en caractères gras car Visual Studio détecte que vous avez déjà un gestionnaire pour cet évènement. Si vous cliquez sur cette option, Visual Studio amène le curseur à l'emplacement du gestionnaire dans le code.

Tous les autres évènements ne sont pas en gras car aucun gestionnaire n'existe encore pour ceux-ci dans l'application. Si vous cliquez sur un évènement quelconque, Visual Studio génère la procédure de réponse à l'évènement et l'ajoute à votre code avec la signature adéquate ☺.



Au fait, qu'est ce qu'on entend par *signature* et pourquoi est-elle importante pour définir un gestionnaire d'évènement ?

La signature d'une procédure est sa ligne de déclaration comprenant :

- Le type de la méthode (procédure sans valeur de retour ou fonction avec une valeur de retour),
- La définition de tous les paramètres de la méthode avec leur type de données,
- Et le type de données de la valeur de retour de la méthode s'il y en a une (ce qui n'est pas le cas dans une procédure de réponse à un évènement).

Pour les gestionnaires d'évènement, c'est très facile, car la signature du gestionnaire

d'évènement est souvent la suivante :

Code VB

```
Private Sub NomDuGestionnaire(ByVal sender As System.Object, _  
                               ByVal e As System.EventArgs) _  
    Handles NomDeLEvenement  
  
End Sub
```

où :

- **Sender** de type **System.Object**, est l'élément à l'origine du déclenchement de l'évènement (dans notre cas, ce serait donc l'objet **QuitterToolStripMenuItem**),
- Et où **e** sert à transmettre toutes informations utiles au gestionnaire pour l'aider à répondre à l'évènement. En fait, le type **System.EventArgs** représente un jeu d'information vide. Lorsqu'un évènement transmet des informations au gestionnaire, la signature de l'évènement est légèrement différente et **e** est d'un autre type.

Par exemple, l'évènement **DragDrop** de l'objet **QuitterToolStripMenuItem** envoie des informations de positionnement très utile pour gérer le glisser déplacer de l'élément. La signature d'un gestionnaire pour cet évènement fait donc appel à un argument **e** de type plus complexe : **System.DragEventArgs**.

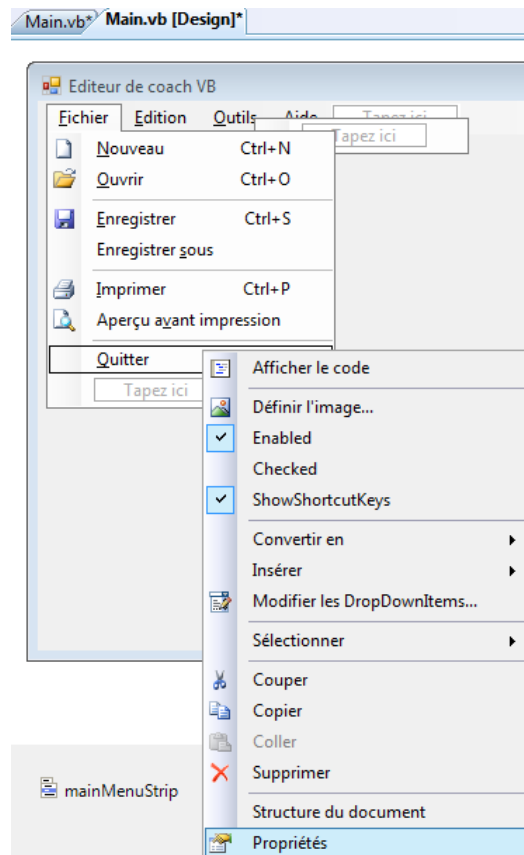
```
7 Private Sub QuitterToolStripMenuItem_DragDrop(ByVal sender As Object, _  
8                                             ByVal e As System.Windows.Forms.DragEventArgs) _  
9                                             Handles QuitterToolStripMenuItem.DragDrop  
10  
11 End Sub
```


La signature d'une procédure en réponse à un évènement ne peut donc pas être inventée. Lorsque le système exécute les procédures en réponse à un évènement, il ne peut invoquer que des méthodes respectant rigoureusement la signature attendue. C'est sa manière de communiquer avec vous pour vous transmettre des informations pertinentes sur ce qui se passe dans l'application.

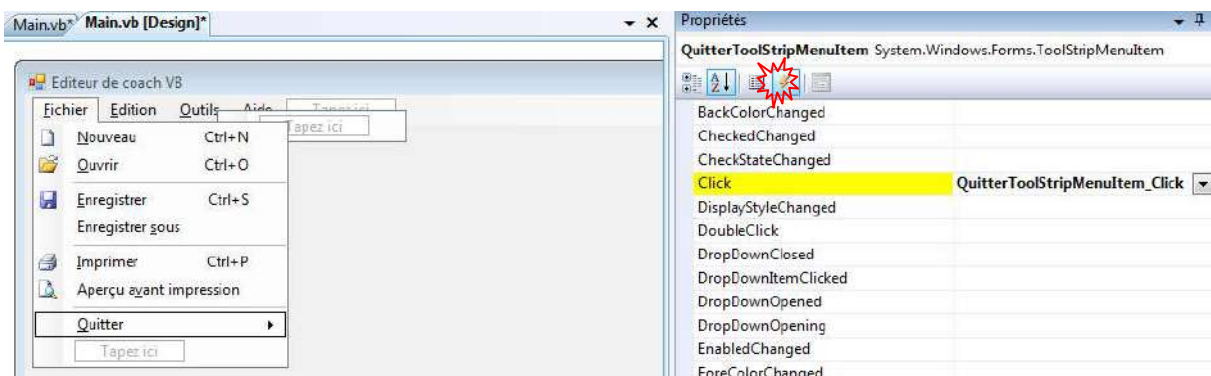


Vous retrouvez également tous les évènements disponibles sur un objet à partir de la fenêtre de Propriétés de l'objet en mode Design.


- Afficher le Concepteur de formulaire du fichier **Main.vb**.
- Sélectionnez le menu **Quitter** puis faites un clic droit > **Propriétés** pour afficher la fenêtre de propriétés pour cet objet.



- Dans la fenêtre **Propriétés**, cliquez le bouton  de la barre d'outils pour afficher l'ensemble des événements correspondant. Vous devez retrouver le nom de la procédure **quitterToolStripMenuItem_Click** en face de l'événement **Click** indiquant que l'évènement possède un gestionnaire d'évènement :





Il suffit de cliquer sur le bouton  de la barre d'outils de la fenêtre **Propriétés** pour revenir à la liste des propriétés de l'objet.



Pour générer automatiquement un gestionnaire en réponse à un évènement, repérez l'évènement dans la liste puis double cliquez dans la zone de texte à droite du nom de l'évènement. Visual Studio bascule dans la fenêtre de code et génère une procédure sur la base du format de nom que nous avons étudié précédemment.



Pour tout savoir sur les évènements et gestionnaires d'évènements :

[http://msdn.microsoft.com/fr-fr/library/2z7x8ys3\(VS.80\).aspx](http://msdn.microsoft.com/fr-fr/library/2z7x8ys3(VS.80).aspx)

Pour en savoir plus sur les clauses **WithEvents** et **Handles** :

[http://msdn.microsoft.com/fr-fr/library/stf7ebaz\(VS.80\).aspx](http://msdn.microsoft.com/fr-fr/library/stf7ebaz(VS.80).aspx)

Et enfin, si vous voulez voir comment cela se passe avec l'approche par code, plus souple et dynamique :

[http://msdn.microsoft.com/fr-fr/library/6yyk8z93\(VS.80\).aspx](http://msdn.microsoft.com/fr-fr/library/6yyk8z93(VS.80).aspx)

4. Codez maintenant la fermeture du formulaire :

- Revenez sur la définition de la procédure **QuitterToolStripMenuItem_Click** dans le fichier **Main.vb**.
- Ajoutez le code suivant :

Code VB

```
Public Class Main
```

```
Private Sub QuitterToolStripMenuItem_Click(ByVal sender As System.Object, _  
                                           ByVal e As System.EventArgs) _  
    Handles QuitterToolStripMenuItem.Click  
    'Fermeture du formulaire  
    Me.Close()  
  
End Sub  
End Class
```



Que signifie le mot **Me** ?

Le mot clé **Me** référence l'objet dans lequel le code s'exécute au passage de la ligne courante. Dans notre cas, nous développons la classe **Main**. **Me** permet donc de retrouver l'instance de notre classe **Main** qui va être créée au moment de l'exécution du programme.



Voici un lien très intéressant pour ne pas confondre **Me** avec d'autres mots clé qui lui ressemblent et qui pourraient donc porter à confusion :

<http://msdn.microsoft.com/fr-fr/library/20fy88e0.aspx>



Et que signifie **Close()** ?

Il s'agit d'une méthode de l'objet référencé par **Me**, c'est-à-dire par l'instance en cours de notre classe **Main**. Et vous vous en doutez, c'est une méthode dont l'action est bien sûr de *fermer* (*close*) le formulaire.

Mais notre classe **Main** n'a pas de méthode appelée **Close** ☹...Alors d'où sort-elle ?

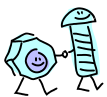
```
1 Public Class Main
2
3     Private Sub QuitterToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As EventArgs)
4         'Fermeture du formulaire
5         Me.Close()
6     End Sub
7 End Class
```



Vous vous rappelez que **Main** est une classe partielle (mot clé *Partial*) c'est-à-dire qu'elle est définie en plusieurs morceaux répartis dans plusieurs fichiers. Peut-être que la méthode que nous cherchons est définie dans l'autre fichier, **Main.Designer.vb** ?

- A partir de l'**Explorateur de solutions**, ouvrez à nouveau le fichier **Main.Designer.vb**.
- Notez la définition de la classe **Main** tout en haut du fichier.

```
1 <Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
2 Partial Class Main
3     Inherits System.Windows.Forms.Form
```



Si vous cherchez la méthode **Close** dans le fichier **Main.Designer.vb**, vous ne la trouverez pas, et pour cause puisque ce fichier ne fait que traduire ce que vous dessinez avec Visual Studio dans le formulaire et nous n'avons rien fait en rapport avec la fermeture du formulaire.

En revanche, la déclaration de la classe en haut du fichier est un peu plus complète que celle que nous avons dans le fichier **Main.vb**. Elle montre notamment que la classe **Main** *hérite* (Inherits) d'une classe du Framework .NET appelée **Form** dans l'espace de nom **System.Windows.Forms**.

Que signifie *l'héritage* ?

L'héritage est une association entre deux classes qui assure l'utilisation par une classe des fonctionnalités déjà définies dans l'autre classe, histoire de ne pas tout réécrire. Nous reviendrons sur ce concept de base de la programmation orienté objet plus tard dans ce tutorial.

Dans notre cas, retenons que notre classe **Main hérite de la classe Form** fournie par le Framework .NET de façon à hériter de toutes les caractéristiques et du comportement standard d'un formulaire Windows Forms. Sans cet héritage, vous devriez construire le formulaire de A à Z en créant des propriétés pour définir sa taille, son titre etc... et en décrivant son comportement tel que l'ouverture ou la fermeture du formulaire.

Grâce au Framework .NET, nous pouvons donc utiliser une méthode **Close** qui vient de la classe **System.Windows.Form.Form** dans laquelle est codé l'ordre de fermeture du formulaire.



N'oubliez pas que la **Library** de MSDN vous permet de retrouver toutes les caractéristiques de n'importe quelle classe du Framework .NET. Par exemple, retrouvez la définition de la classe **Form** sur :

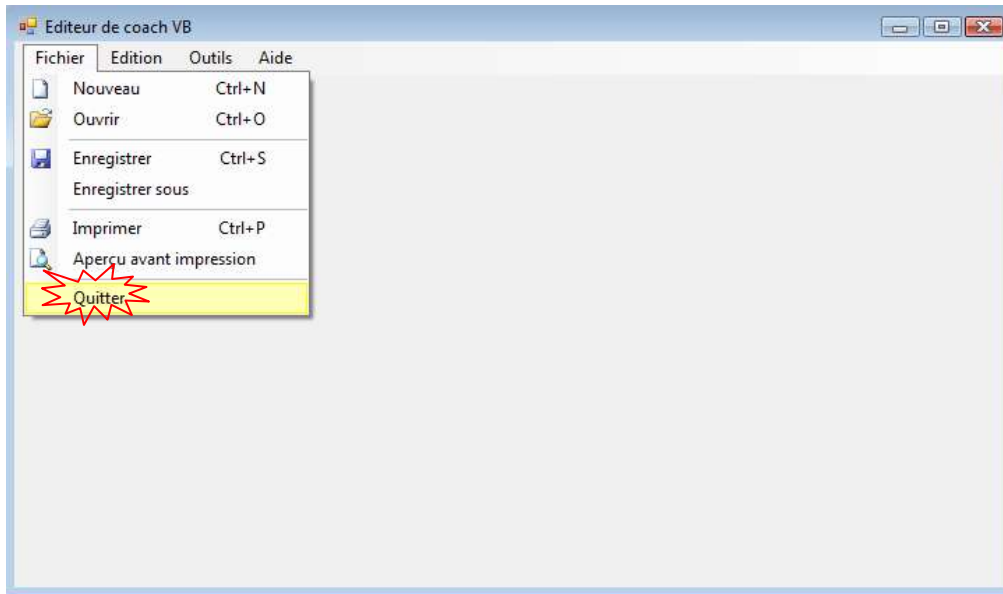
<http://msdn.microsoft.com/fr-fr/library/system.windows.forms.form.aspx>

Consulter la rubrique **Voir aussi** au bas de la page pour aller sur le lien listant tous les membres de la classe à savoir ses propriétés, ses méthodes et ses évènements. C'est là que vous trouverez la définition de la méthode **Close**.

☐ **Voir aussi**
Concepts
[Applications d'interface multidocument \(MDI, Multiple Document Interface\)](#)
Référence
~~[membres Form](#)~~
~~[System.Windows.Forms, espace de noms](#)~~

5. Et si vous testiez votre première ligne de code ?

- Enregistrez tous vos changements.
- Lancez l'exécution de l'application (F5).
- Cliquez le menu **Fichier > Quitter**. Le formulaire doit se fermer et dans la foulée, l'exécution de l'application doit s'arrêter.



Bravo ! Vous commencez à avoir une application qui roule ! Elle démarre avec un écran de démarrage et s'arrête proprement avec les menus de la barre de menu.

AFFICHER L'APPLICATION DANS LA ZONE DE NOTIFICATION

Dans cet exercice, vous allez maintenant manipuler un *composant* Windows Form. Il s'agit du composant **NotifyIcon** qui affiche une icône dans la zone de notification d'état de la barre des tâches de Windows. C'est un composant très utile par exemple pour contrôler des applications qui s'exécutent en arrière-plan et qui n'ont pas d'interface utilisateur.

Contexte fonctionnel

Dans cet exercice, nous allons rajouter une icône dans la zone de notification d'état de la barre des tâches de Windows en bas à droite du bureau, qui atteste que notre application est en cours d'exécution. Lorsque l'utilisateur clique sur l'icône, un menu contextuel lui propose des options standards lui permettant de quitter l'application et de redimensionner la fenêtre principale.

Le procédé s'articule en quatre étapes :

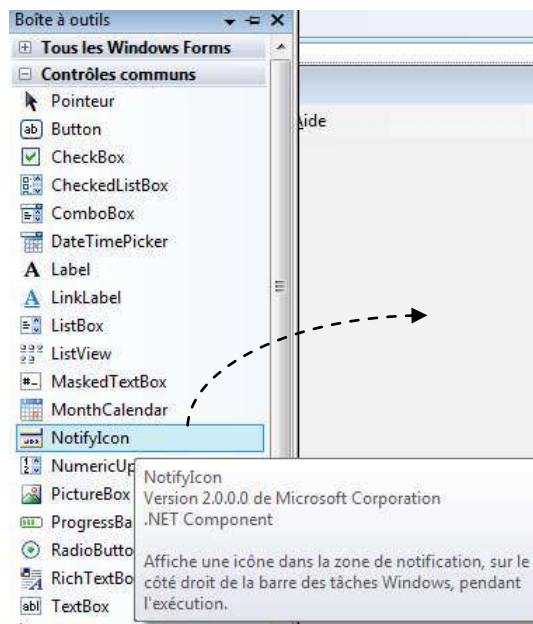
- Ajout d'une icône de notification au programme, qui s'affiche dans la zone de notification de Windows au moment de l'exécution.

- Ajout d'un menu contextuel au programme.
- Association du menu contextuel au clic sur l'icône de notification.
- Codage de l'option Quitter du menu contextuel pour fermer le formulaire.


Déroulement de l'exercice :

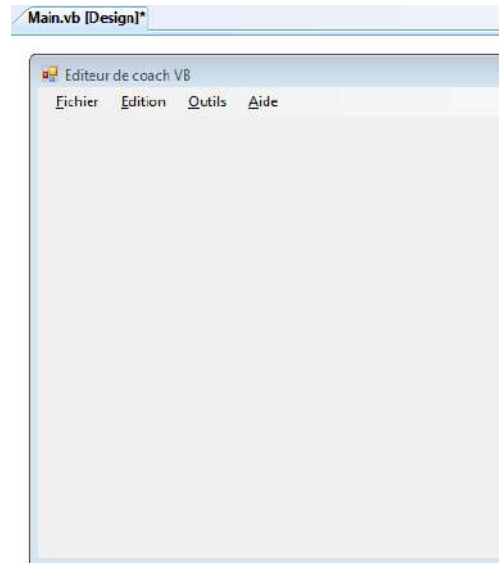
1. *Etape 1* : ajout d'une icône de notification à l'application :

- Ouvrez le formulaire **Main.vb** en mode Design en double-cliquant sur le fichier **Main.vb** dans l'**Explorateur de solutions**.
- Faites un glisser-déplacer de la Boîte à outils, rubrique **Contrôles communs** > du composant **NotifyIcon** n'importe où sur la surface du formulaire.




Comme il s'agit d'un composant, c'est-à-dire qu'il n'a pas d'équivalence graphique directement sur le formulaire, Visual Studio nous l'affiche directement dans la zone de dépôt de contrôles en dessous du formulaire, sous la forme d'un objet nommé **NotifyIcon1**.

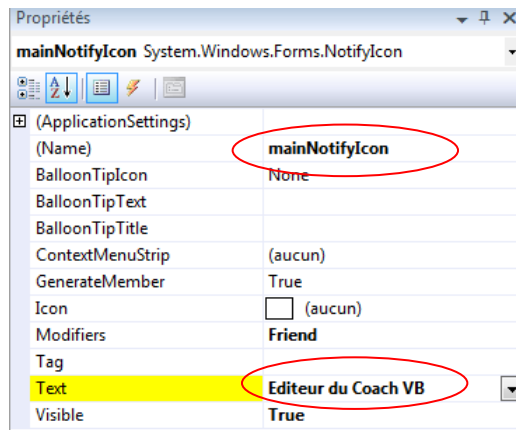
Au travers d'une balise active (smart tag)  qui apparaît en haut à droite du contrôle lorsque celui-ci est sélectionné, Visual Studio nous propose des raccourcis vers les actions de configuration type pour paramétrer le plus rapidement possible ce contrôle. En l'occurrence ici, il faut principalement lui associer une icône pour l'afficher dans la zone de notification d'état de la barre des tâches de Windows. Mais une fois n'est pas coutume, nous n'allons pas utiliser cette approche déclarative et opter pour une configuration de l'icône de notification par code en utilisant les ressources du projet ☺.



Pour tout savoir sur le composant Windows Forms **NotifyIcon** :

[http://msdn.microsoft.com/fr-fr/library/7yyz6s5c\(VS.80\).aspx](http://msdn.microsoft.com/fr-fr/library/7yyz6s5c(VS.80).aspx)

- Afficher les propriétés du composant par un clic droit > **Propriétés**.
- Modifiez le nom du composant par **mainNotifyIcon** en changeant la valeur de la propriété **(Name)**. Cette propriété apparaît parmi les premières si vous êtes en classement alphabétique (bouton  de la barre d'outils de la fenêtre Propriétés).
- Modifiez également la propriété **Text** avec la valeur **Editeur du Coach VB**. Ce texte apparaîtra en aide rapide (« tooltip ») lorsque le pointeur de souris sera au dessus de l'icône.

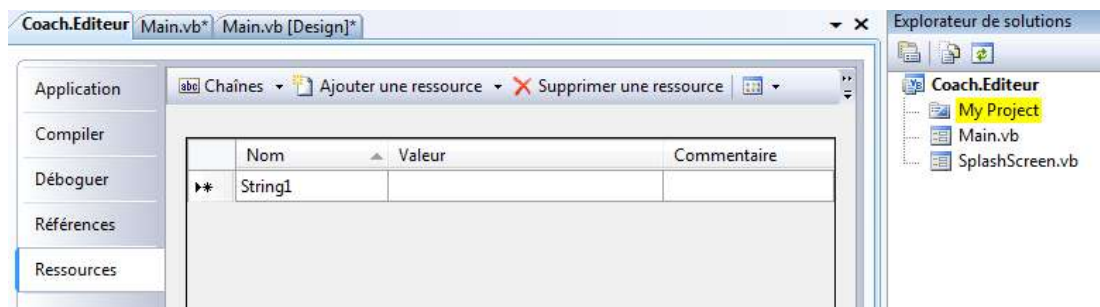


Notez au passage dans cette fenêtre **Propriétés** que la propriété **Icon** est celle qui va nous permettre d'associer une icône au composant, celle-là même qui apparaîtra dans la zone de notification.

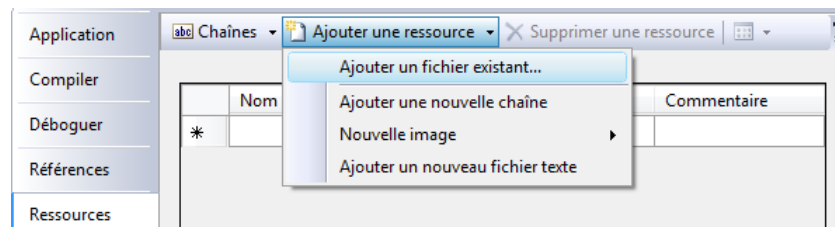
Comment configurer l'icône par code ?

Les images, les icônes ou encore les fichiers audio sont ce qu'on appelle des *ressources* du programme. Nous allons récupérer une icône et la lier au projet en utilisant le **Concepteur de ressources** du projet puis nous paramètrons la propriété **Icon** du contrôle **mainNotifyIcon** avec la ressource ajoutée.

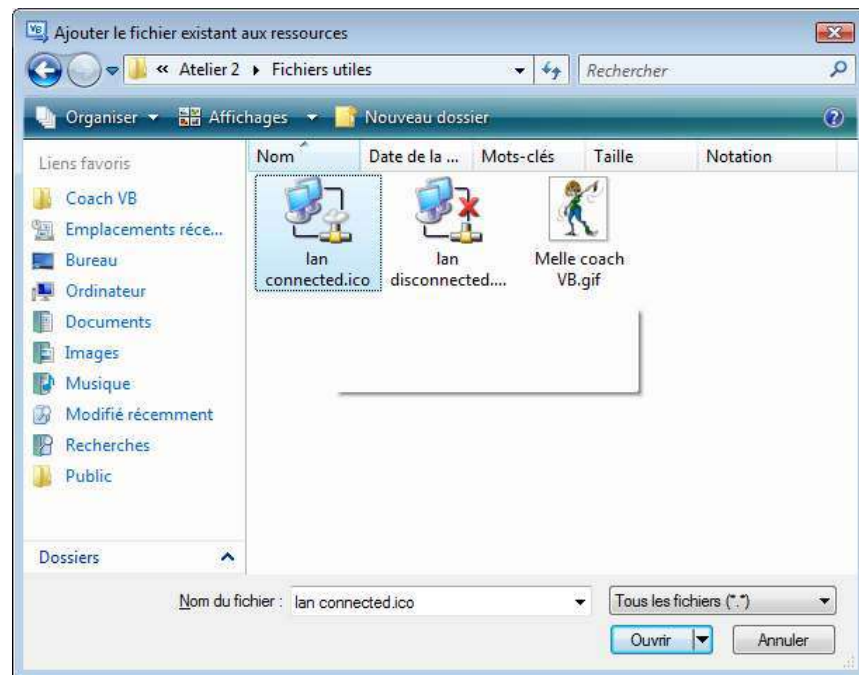
2. Configurez l'icône du composant de notification avec un fichier icône fourni avec le code de l'atelier :
 - A partir de l'**Explorateur de solutions**, affichez le Concepteur de projets en cliquant sur **My Project**.
 - Sélectionnez l'onglet **Ressources**.





- Sur l'écran de gestion des ressources, cliquez le menu **Ajouter une ressource > ajouter un fichier existant...** :

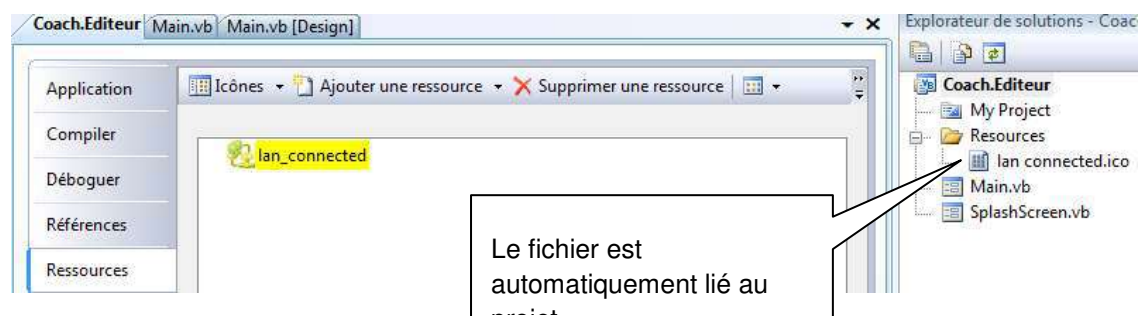


- Dans la boîte de dialogue **Ajouter le fichier existant aux ressources**, naviguez jusqu'au répertoire **..\Atelier 2\Fichiers utiles** fourni avec cet atelier.
- Sélectionnez le fichier **lan_connected.ico** puis cliquez le bouton **Ouvrir**.



Nous avons choisi à titre d'exemple une icône représentant une connexion LAN . On pourrait en effet imaginer que votre application ait besoin à un moment ou un autre d'être connectée à Internet ou sur le réseau de l'entreprise. Dans ce cas, il pourrait être intéressant de détecter par programmation l'état de connexion de la machine de l'utilisateur et de modifier l'icône affichée par le composant **mainNotifylcon** en conséquence, afin de donner à l'utilisateur une indication sur la connectivité de l'application en cours. L'icône  correspondrait par exemple à l'état déconnecté.

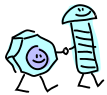
Nous n'allons charger qu'une seule icône dans le projet à ce stade du tutorial.



- Fermez le Concepteur de ressources du projet.
- Enregistrez vos modifications sur le projet.



Il faut maintenant charger la propriété **Icon** du composant **mainNotifyIcon** avec la ressource que nous venons d'ajouter au programme.



Où doit-on brancher le code de configuration de la propriété **Icon** du composant ?

La question serait plutôt *quand* puisque nous développons avec une approche événementielle. L'idéal serait d'agir au tout début de la vie du formulaire pour que l'icône apparaisse dans la zone de notification de Windows dès le démarrage de l'exécution.

Quel est le tout premier événement disponible sur un objet ?

Il s'agit de son **constructeur**. Le constructeur est une méthode membre d'une classe qui est appelée par le système au moment de la *construction* de l'objet. On dit aussi que le nouvel objet est *instancié*. Du coup, cette méthode sert à créer convenablement une instance d'objet pour la classe, en le configurant dans un état valide.

Un constructeur en VB est une méthode publique nommée **New** sans aucune valeur de retour.

- Ouvrez le fichier de code **Main.vb**.
- Entrez directement sous la définition de la classe **Main** la ligne suivante :

Code VB

```
Public Class Main
```

Public Sub New

```
Private Sub QuitterToolStripMenuItem_Click(ByVal sender As System.Object, _  
                                           ByVal e As System.EventArgs) _
```

```
    Handles QuitterToolStripMenuItem.Click
```

```
    ...
```

```
End Sub
```

```
End Class
```

- Validez par **Entrée**. Visual Studio rajoute la suite de la procédure pour vous ☺ !

Code VB

```
Public Class Main
```

```
    Public Sub New()
```

```
        ' Cet appel est requis par le Concepteur Windows Form.
```

```
        InitializeComponent()
```

```
        ' Ajoutez une initialisation quelconque après l'appel
```

```
        ' InitializeComponent().
```

```
    End Sub
```



```
Private Sub QuitterToolStripMenuItem_Click(ByVal sender As System.Object, _  
                                           ByVal e As System.EventArgs) _  
    Handles QuitterToolStripMenuItem.Click  
    ...  
End Sub  
End Class
```



Notez entre autres l'appel à la méthode **InitializeComponent** que vous trouverez dans le fichier **Main.Designer.vb**. Il s'agit de la procédure qui initialise tous les éléments de votre interface et qui est donc appelée systématiquement à la construction du formulaire. D'habitude, on ne s'en préoccupe guère parce que Visual Studio créait le formulaire pour nous. Mais comme nous sommes en train de définir un constructeur spécifique pour le formulaire **Main**, c'est à nous maintenant d'appeler cette méthode !

- Dans le constructeur de la classe, ajoutez maintenant le code pour initialiser l'icône du contrôle **mainNotifyIcon** avec celui chargé dans les ressources du projet.

Code VB

```
Public Class Main  
  
    Public Sub New()  
  
        ' Cet appel est requis par le Concepteur Windows Form.  
        InitializeComponent()  
    End Sub  
End Class
```

```
' Ajoutez une initialisation quelconque après l'appel  
' InitializeComponent().  
  
mainNotifyIcon.Icon = My.Resources.lan_connected  
  
End Sub  
  
...  
End Class
```



Vous vous souvenez de l'objet **My** ?

C'est un espace de noms qui contient des classes donnant des raccourcis vers les fonctionnalités les plus couramment utilisées du Framework. **My.Resources** donne un accès simple et rapide aux ressources de l'application.



N'oubliez pas d'utiliser à fond l'aide de l'**IntelliSense** pour éviter les erreurs. Par exemple, à chaque fois que vous tapez un point, l'IntelliSense se manifeste et vous guide dans le choix des éléments possibles compte tenu du contexte.

```
7 | ' Ajoutez une initialisation quelconque après l'appel  
8 | mainNotifyIcon.Icon = My.Resources.|  
9 | d Sub  
10| ivate Sub QuitterToolStripMenuItem  
11| 'Fermeture du formulaire  
12| Me.Close()  
13| d Sub  
14| ...
```

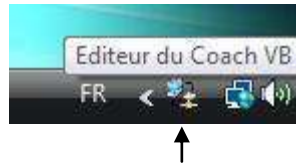
The screenshot shows the IntelliSense dropdown menu for the code line `mainNotifyIcon.Icon = My.Resources.|`. The dropdown lists several options: Culture, lan_connected (highlighted in yellow), and ResourceManager. Below the list are two tabs: 'Commun' and 'Tous'. A tooltip on the right side of the dropdown shows the details for the selected 'lan_connected' resource, including 'System.Object', 'ReadOnly Property lan_connect', and 'Friend ReadOnly Property lan_connect'.

Vous devez voir votre icône en tant que ressource.

3. Testez le fonctionnement de l'icône de notification de l'application :

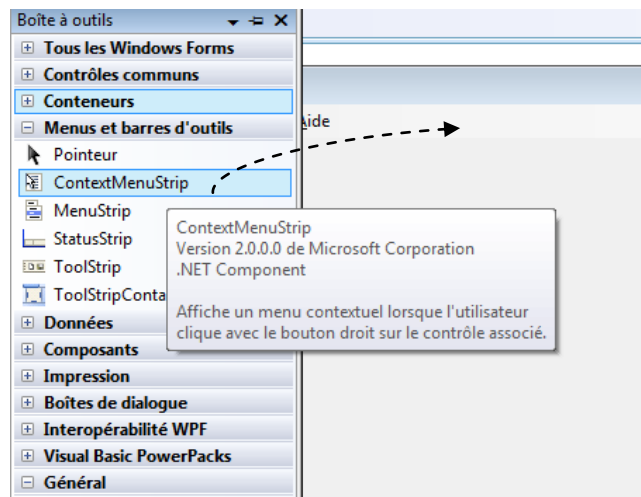
- Enregistrez tous les changements.

- Lancez l'exécution de l'application (F5) pour tester le fonctionnement de l'icône de notification.
- Vérifiez la présence de l'icône de connexion dans la zone de notification d'état de la barre des tâches de Windows.
- Arrêtez le curseur de la souris sur l'icône pour valider le texte d'aide (tooltip).



4. Etape 2 : ajout d'un menu contextuel au programme :

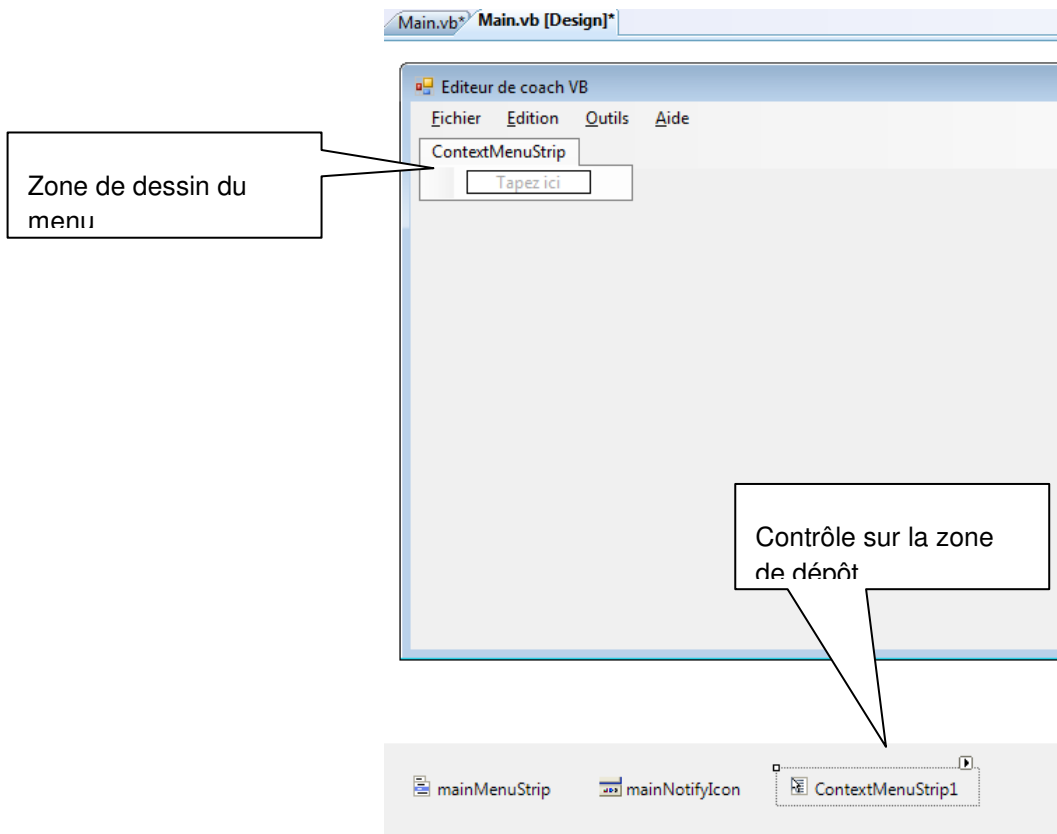
- Revenez sur le formulaire **Main.vb** en mode Design.
- Faites un glisser déplacer de la **Boîte à outils** > rubrique **Menus et barre d'outils** > du contrôle **ContextMenuStrip** n'importe où sur la surface du formulaire :



Deux nouveaux éléments apparaissent sur le formulaire :

- Une barre de menu vide sous la barre de titre de la fenêtre,

- Et un composant nommé **contextMenuStrip1** dans la zone de dépôt de contrôles.



Vous observez un comportement rigoureusement identique à celui du contrôle **MenuStrip**, à ceci près qu'à l'exécution un menu contextuel de type **ContextMenuStrip** n'apparaît que dans le *contexte* pour lequel il est défini (d'où son nom). Donc le positionnement du menu juste en dessous du menu principal du formulaire ne fait pas foi. Mais il faut bien que Visual Studio vous l'affiche quelque part proprement pour vous permettre de le construire ☺.

Dans notre cas, nous allons l'associer au composant **mainNotifyIcon** pour qu'il apparaisse sur le clic droit de l'icône dans la zone de notification d'état de la barre des tâches de Windows.

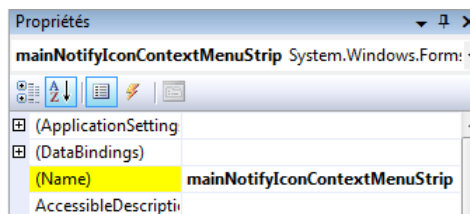


Pour tout savoir sur le contrôle Windows Forms **ContextMenuStrip** :

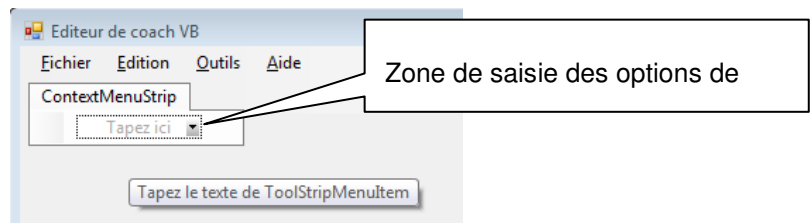
[http://msdn.microsoft.com/fr-fr/library/ms229641\(VS.80\).aspx](http://msdn.microsoft.com/fr-fr/library/ms229641(VS.80).aspx)

5. Configurez les options du menu contextuel :

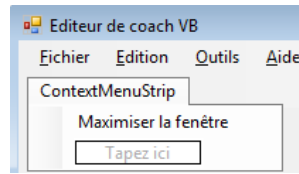
- Sélectionnez le contrôle **contextMenuStrip1** dans la zone de dépôt de contrôles et faites un clic droit > **Propriétés** pour faire apparaître sa fenêtre de propriétés.
- Dans les propriétés du contrôle, changez son nom par **mainNotifyIconContextMenuStrip** en changeant la valeur de la propriété **(Name)**.



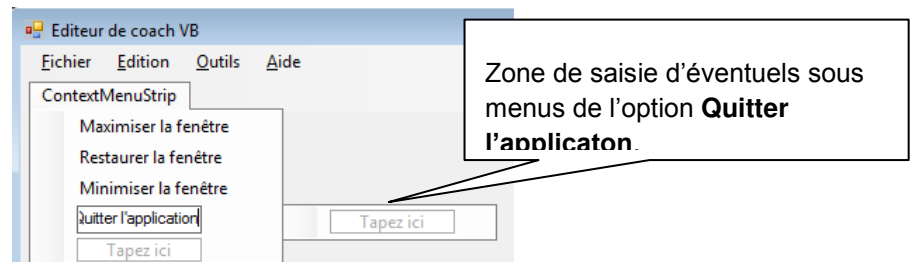
- Sur la zone de dessin du menu contextuel, cliquez sur **Tapez ici** pour saisir une première option de menu :



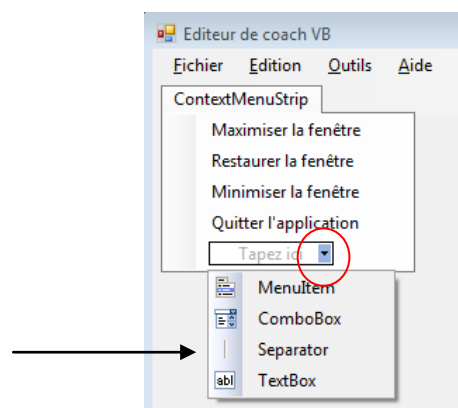
- Saisissez **Maximiser la fenêtre** puis validez par **Entrée** :



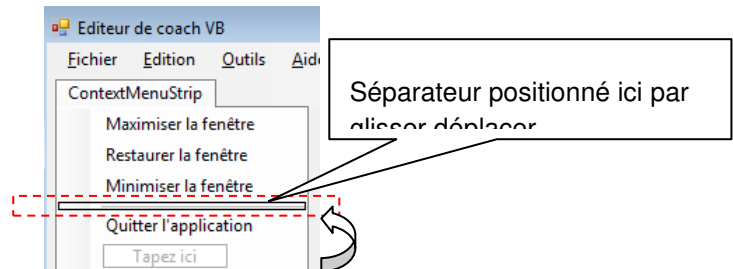
- Recommencez l'opération avec les options **Restaurer la fenêtre**, **Minimiser la fenêtre**, et **Quitter l'application** :



- Cliquez sur la flèche ▼ qui apparaît à droite de la zone (lorsque vous la sélectionnez) en dessous de l'option **Quitter l'application**. Dans la liste déroulante, sélectionnez **Separator** pour insérer une ligne de séparation dans la liste des menus :



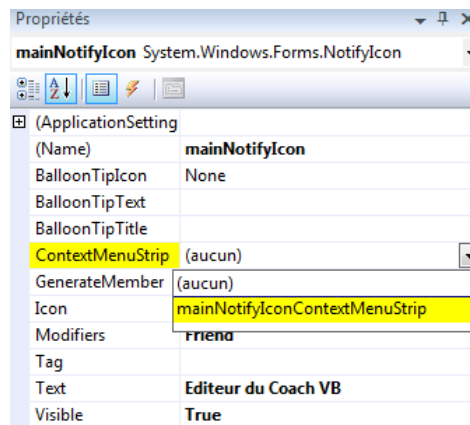
- Faites un **glisser-déplacer** de la ligne de séparation afin de la positionner avant le menu **Quitter l'application** :



- Enregistrez vos changements.

6. *Etape 3* : association du menu contextuel au clic sur l'icône de notification :

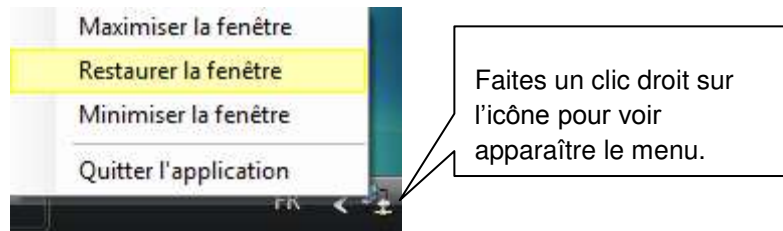
- Sélectionnez le contrôle **mainNotifyIcon** dans la zone de dépôt de contrôles, puis faites un clic droit > **Propriétés** ;
- Dans les propriétés du contrôle, changez la valeur de **ContextMenuStrip** en utilisant la liste déroulante proposée, et sélectionnez **mainNotifyIconContextMenuStrip** :



7. Testez le fonctionnement de l'icône de notification de l'application :

- Enregistrez tous les changements.

- Lancez l'exécution de l'application (F5).
- Vérifiez la présence de l'icône de connexion dans la zone de notification d'état de la barre des tâches de Windows.
- Faites un clic droit sur l'icône pour valider le déclenchement du menu contextuel :

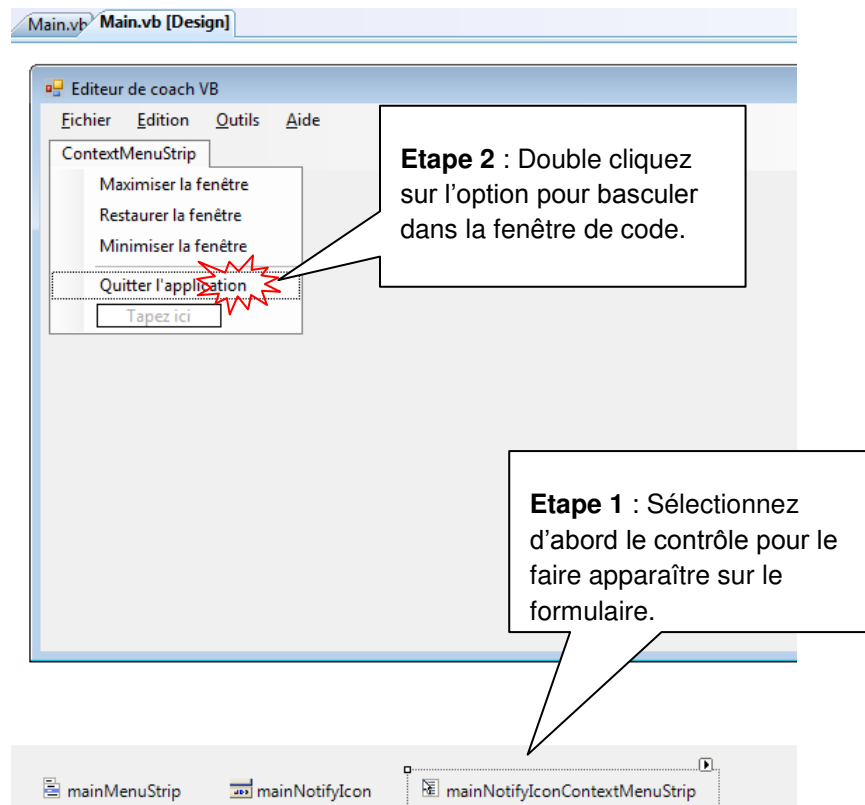


L'étape suivante consiste à coder maintenant les actions associées au clic de l'utilisateur sur les différentes options du menu contextuel. Dans cet atelier, nous allons nous concentrer sur l'option **Quitter l'application** qui permet à l'utilisateur d'arrêter l'exécution du programme.

Le procédé de codage d'une option de menu contextuel est rigoureusement le même que celui que nous avons suivi précédemment pour coder l'option **Quitter** du menu principal du formulaire **Main**.

8. *Etape 4* : codage de l'option Quitter l'application :

- Sélectionnez le contrôle **mainNotifyIconContextMenuStrip** dans la zone de dépôt de contrôles afin de le faire apparaître sur le formulaire.
- Double cliquez sur l'option du menu contextuel **Quitter l'application** pour générer la procédure de réponse à l'évènement **Click** sur l'option de menu :



Visual Studio bascule dans la fenêtre de code et génère à la suite des membres de la classe **Main**, une nouvelle méthode nommée **QuitterLapplicationToolStripMenuItem_Click**.

```

15 Private Sub QuitterLapplicationToolStripMenuItem_Click(ByVal sender As System.Object, _
16                                     ByVal e As System.EventArgs) _
17                                     Handles QuitterLapplicationToolStripMenuItem.Click
18
19 End Sub
20 End Class

```

Nom de l'évènement associé au gestionnaire d'évènement.

- Ajoutez le code de fermeture du formulaire :

Code VB

```
Private Sub QuitterLapplicationToolStripMenuItem_Click( _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles QuitterLapplicationToolStripMenuItem.Click  
  
    'Fermeture du formulaire  
  
    Me.Close()  
  
End Sub
```



Ca ne vous rappelle rien ce code ? C'est évidemment rigoureusement le même que celui que nous avons programmé dans la procédure associée au clic de l'option **Quitter** du menu principal du formulaire.

Ce qui nous amène à la question suivante :

Est-ce qu'on ne pourrait pas récupérer le code du gestionnaire d'évènement **QuitterToolStripMenuItem_Click** pour l'appliquer à l'évènement **Click** sur l'option **Quitter l'application** du menu contextuel ?

La réponse est oui, vous vous en doutez. Mais comment ?

C'est le mot clé **Handles** qui détermine sur quel évènement s'abonne le gestionnaire d'évènement correspondant. Alors pourquoi ne pas ajouter un nouvel évènement au gestionnaire **QuitterToolStripMenuItem_Click** que nous avons écrit pour l'option **Quitter** du menu principal, pour qu'il prenne en charge aussi le clic de l'utilisateur sur l'option **Quitter l'application** du menu contextuel de l'icône de notification. Il suffit de le brancher également sur l'évènement **Click** de l'objet **QuitterLapplicationToolStripMenuItem**.

- Reprenez le gestionnaire d'évènement **QuitterToolStripMenuItem_Click** et ajoutez-lui le second évènement séparé par une virgule.
- Supprimez la méthode **QuitterLapplicationToolStripMenuItem_Click** qui ne sert plus à rien.

```
10 Private Sub QuitterToolStripMenuItem_Click(ByVal sender As System.Object, _  
11                                     ByVal e As System.EventArgs) _  
12                                     Handles QuitterToolStripMenuItem.Click, QuitterLapplicationToolStripMenuItem.Click  
13     'Fermeture du formulaire  
14     Me.Close()  
15 End Sub  
16  
17 Private Sub QuitterLapplicationToolStripMenuItem_Click(ByVal sender As System.Object, _  
18                                     ByVal e As System.EventArgs) _  
19                                     Handles QuitterLapplicationToolStripMenuItem.Click  
20  
21 End Sub
```



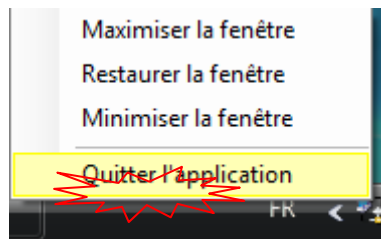
Une bonne pratique serait de renommer le gestionnaire d'évènement avec un nom plus générique qui refléterait son action.

- Renommez par exemple la méthode **QuitterToolStripMenuItem_Click** en **FormClose**.

```
1 Public Class Main
2     Public Sub New()
3
4         ' Cet appel est requis par le Concepteur Windows Form.
5         InitializeComponent()
6
7         ' Ajoutez une initialisation quelconque après l'appel InitializeComponent().
8         mainNotifyIcon.Icon = My.Resources.lan_connected
9     End Sub
10
11     Private Sub FormClose(ByVal sender As System.Object, _
12                           ByVal e As System.EventArgs) _
13         Handles QuitterToolStripMenuItem.Click, QuitterLapplicationToolStripMenuItem.Click
14         'Fermeture du formulaire
15         Me.Close()
16     End Sub
17
18 End Class
```

9. Testez le clic sur l'option de menu contextuel associé à l'icône de notification :

- Enregistrez tous les changements.
- Lancez l'exécution de l'application (F5).
- Faites un clic droit sur l'icône pour faire apparaître le menu contextuel.
- Vérifiez que l'application s'arrête proprement en cliquant l'option **Quitter l'application** du menu contextuel.



- Relancez l'exécution de l'application (F5).
- Vérifiez la fermeture du formulaire sur un clic de l'option **Quitter** du menu principal.



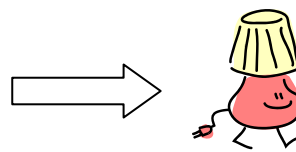
En conclusion, avec les clauses **WithEvents** et **Handles**, il est possible d'autoriser un gestionnaire d'évènements à gérer un ou plusieurs types d'évènements.

C'est un peu comme dans une maison, souvent dans une même pièce vous avez plusieurs boutons qui provoquent l'allumage d'un même jeu de plafonnier.

Types d'évènement



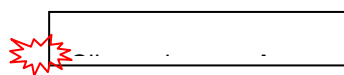
Gestionnaire d'évènement



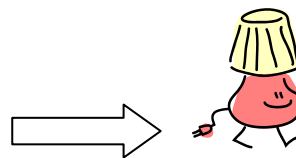
A l'inverse, un ou plusieurs gestionnaires d'évènements peuvent être configurés pour gérer un même type d'évènements. C'est-à-dire que le clic d'un bouton de la pièce peut allumer le plafonnier de la pièce et celui également du hall vois



Types d'évènement



Gestionnaires d'évènement



Bravo ! Vous avez développé votre première application Visual Basic de type Windows. Dans les prochains ateliers, nous allons poursuivre sa construction tout en explorant tous les principes de programmation et caractéristiques liés au langage Visual Basic.

POUR ALLER PLUS LOIN...

Il existe d'autres alternatives aux différents procédés illustrés dans cet atelier. Par exemple, une autre alternative pour exécuter et fermer une application consiste à utiliser l'objet **Application** du Framework .NET et ses méthodes **Run** et **Exit** :



Pour Application.Run() :

[http://msdn.microsoft.com/fr-fr/library/system.windows.forms.application.run\(VS.80\).aspx](http://msdn.microsoft.com/fr-fr/library/system.windows.forms.application.run(VS.80).aspx)

Pour Application.Exit() :

[http://msdn.microsoft.com/fr-fr/library/system.windows.forms.application.exit\(VS.80\).aspx](http://msdn.microsoft.com/fr-fr/library/system.windows.forms.application.exit(VS.80).aspx)

Mais attention, si deux procédés peuvent paraître identiques du point de vue de l'utilisateur, ils peuvent suivre un cheminement très différent d'un point de vue cycle d'exécution dans l'application. Par exemple, l'utilisation de la méthode **Close** d'un formulaire déroule le cycle de fermeture complet du dit formulaire c'est-à-dire déclenche une succession d'évènements caractéristiques de la fermeture d'une fenêtre Windows Form. En revanche, l'utilisation de la méthode **Exit** de l'objet **Application** a pour effet de fermer tous les formulaires ouverts dans l'application mais sans déclencher le cycle complet de fermeture de chacun d'eux.

Tout cela pour dire que le Framework recèle de nombreuses possibilités adaptées à chaque scénario. Aidez-vous de MSDN pour bien comprendre le domaine d'application de chacun des éléments que vous mettez en œuvre.