

CHAPITRE 1

PRÉSENTATION DE .NET ET DE VISUAL STUDIO

1.1 FRAMEWORK .NET

L'arrivée du *framework* .NET amène une révolution chez Microsoft au début des années 2000.

Il se définit comme une **volonté de permettre d'une part une unification de la famille des langages de Microsoft** (par une interprétation machine commune de tous les langages .NET) et, par le fait même, **une réutilisation optimale des classes/bibliothèques/composants développés, le tout axé autour du thème de l'interopérabilité** (en fait, tant au niveau des O/S différents que par une rétro-interopérabilité COM/.NET).

1.1.1 Présentation générale de la philosophie .NET

Les caractéristiques principales et les objectifs visés par le *framework* et la philosophie .NET sont les suivantes :

- ***Une interopérabilité compréhensive avec le code existant.*** Dans cette optique, les composants COM développés peuvent interagir et interopérer avec les composants .NET.
 - Ainsi, ce qui a été développé peut toujours être récupéré et utilisé, à condition d'utiliser la couche d'interopérabilité entre les types de données .COM et .NET.
- ***Une intégration des langages .NET totale et complète.*** .NET supporte les héritages, les manipulations d'exceptions et le débogage de code inter-langagiers.
- ***Un moteur d'exécution commun à tous les langages .NET.*** Ce moteur permet, entre autres, de comprendre et interpréter les types de données entre chacun des langages de programmations .NET.
- ***Une bibliothèque de classes de base.*** L'introduction d'un modèle d'objets consistant, simple et utilisé par tous les langages .NET (et les autres qui le

comprennent) améliorer grandement la compréhensibilité inter-langagière et l'interopérabilité en général.

- ***Un modèle de déploiement hautement simplifié.*** Sous .NET, il n'est plus nécessaire d'enregistrer un composant binaire dans les registres du système. Depuis, .NET permet l'existence (non conflictuelle) de plusieurs versions de fichiers d'extension DLL (*Dynamic Linked Library*) sur une même machine.

On peut donc percevoir à travers ces caractéristiques une volonté réelle d'unification et d'interopérabilité entre les langages, ce qui a pour conséquences directes de faciliter le développement de programmes (que ce soit des composants ou des exécutifs) et la compréhension et la réutilisation de composants .COM ou .NET.

Plus techniquement, ces caractéristiques sont réalisées grâce à l'action de trois entités inter-reliées :

- CLR (*Common Language Runtime*);
- CTS (*Common Type System*);
- CLS (*Common Language Specification*).

De la perspective d'un programmeur, il faut voir la plate-forme .NET en tant qu'environnement d'exécution et que bibliothèque étendue de classes de base.

1.1.2 CLR, CIL et Jitter

La couche assurant l'exécution est formellement appelée la **couche CLR** (*Common Language Runtime*). Ce composant constitue une implémentation du **standard CLI** (*Common Language Infrastructure*) de Microsoft.

Le rôle général de CLR est de **définir et gérer l'environnement d'exécution des programmes**.

De façon plus spécifique, CLR s'occupe de **localiser, charger et gérer les types .NET pour nous**.

Ses autres tâches consistent à veiller à certains détails de bas niveaux, tels que :

- La **gestion de la mémoire** (dont un ramasse-miettes);
- La **création de domaines d'application**, de **threads** et des piles d'**objets**;

- L'exécution de différentes vérifications de **sécurité**.

Le code interprété afin de mener à bien ses rôles **n'est pas directement celui du langage de programmation** (c'est-à-dire Visual Basic, C#, C++, F#), **mais plutôt celui d'un langage intermédiaire commun** à tous et compilé à partir de ces derniers, nommé **CIL** (*Common Intermediate Language*).

Ce langage intermédiaire ressemble à un langage de bas niveau (assembleur) à l'intérieur duquel il n'y a aucune instruction relative au système d'exploitation ou au matériel, toujours dans le but de la volonté d'indépendance de plate-forme souhaitée par Microsoft.

Il ne faut pas mélanger CIL et CLI. D'un point de vue du CLI, on dira la chose suivante :

Tout langage CLI est un langage de programmation qui est utilisé pour réaliser des composants ou des programmes conformément aux spécifications définies par CLI, dans le but de pouvoir être compilé dans le langage intermédiaire CIL.

Il existe une multitude de langages CLI au-delà de ceux inclus par Microsoft dans ses suites de développement (C#/F#/C++/VB .NET), comme notamment L# (Lisp), P# (Prolog), IronPython et Eiffel.

Puisque ces langages ne contiennent aucune instruction spécifiques ni à la plate-forme, ni au matériel de la machine, la conséquence est que ces instructions ne peuvent pas être directement interprétées par le CPU (l'unité centrale de traitement) :

- Ce rôle est assuré par un **compilateur spécialisé au contexte d'exécution** nommé **JIT** (Just-In-Time), qu'on appelle souvent un **Jitter**, qui compile les instructions à la volée (donc **dynamiquement** à l'exécution).

On peut donc voir le **CLR** comme une **machine virtuelle** nécessaire à .NET, un peu dans le genre de celle de Java, mais contrairement à ce dernier, .NET permet au développeur de construire des applications en utilisant le langage de son choix.

CLR utilise et est basé sur les deux autres entités nommées précédemment :

- CTS (*Common Type System*);
 - CLS (*Common Language Specification*).
-

1.1.3 CTS et CLS, BCL

La spécification de CTS sert à **décrire tous les types possibles de données et toutes les constructions de programmations supportés par l'exécution** (qui elle est assurée par CLR).

Elle **spécifie la façon dont ces entités peuvent interagir avec les autres et comment elles sont représentées en méta-données .NET**.

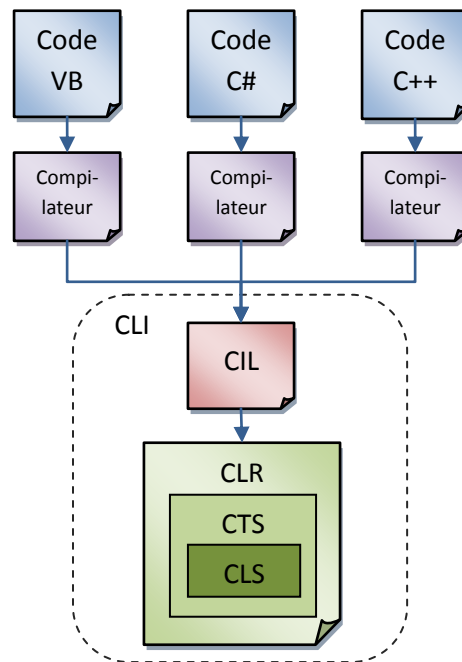
Cette résolution vers un type CTS unique peu importe le langage .NET utilisé est définie dans l'assembly nommé *mscorlib.dll*.

Le tableau suivant montre comment les types de base sont exprimés en différents langages .NET :

Type de donnée CTS	Mot clé en VB .NET	Mot clé en C#	Mot clé en C++/CLI
System.Byte	Byte	byte	unsigned char
System.SByte	SByte	sbyte	signed char
System.Int16	Short	short	short
System.Int32	Integer	int	int ou long
System.Int64	Long	long	_int64
System.UInt16	UShort	ushort	unsigned short
System.UInt32	UInteger	uint	unsigned int ou unsigned long
System.UInt64	ULong	ulong	unsigned _int64
System.Single	Single	float	Float
System.Double	Double	double	Double
System.Object	Object	object	Object^
System.Char	Char	char	wchar_t
System.String	String	string	String^
System.Decimal	Decimal	decimal	Decimal

Type de donnée CTS	Mot clé en VB .NET	Mot clé en C#	Mot clé en C++/CLI
System.Boolean	Boolean	bool	Bool

À ce point, il reste à décrire le CLS. Cette spécification définit un sous-ensemble des types spécifiés par CTS que tous les langages .NET peuvent comprendre et interpréter. C'est donc un sous-ensemble direct à CTS.



1.1.4 BCL

Enfin, outre les spécifications CLR et CTS/CLS, la plate-forme .NET offre une **bibliothèque de classes de base** nommée **BCL** (*Base Class Library*) qui est disponible à tous les langages de programmation .NET.

Cette bibliothèque encapsule les nombreuses primitives comme :

- Les threads;
 - Les entrées et sorties vers les fichiers (I/O) (`System.IO`);
 - Le rendu graphique (`System.Drawing`);
 - L'interaction avec les composants matériels externes.
-

De plus, elle définit des types pour faciliter :

- L'accès aux bases de données (`System.Data`);
- La manipulation de documents XML (`System.XML`);
- La sécurité (`System.Security`);
- Les interfaces de programmation applicatives (API) de bureau, de consoles, à distances ou pour le Web (`System.Web`).

La taille et les fonctionnalités inhérentes à cette bibliothèque sont comparables aux librairies fournis avec le SDK de Java.

1.1.5 Assemblies .NET

Les fichiers binaires exécutables (les *assemblies*) produits par un compilateur .NET possèdent des extensions de fichiers de type *.dll/*.exe et possèdent les caractéristiques suivantes :

- 1) Ils **ne sont pas enregistrés dans le registre système** (plusieurs versions d'un même fichier binaire peuvent coexister, même référés au sein d'une même application cliente);
- 2) Les fichiers binaires .NET **ne contiennent pas d'instructions spécifiques à la plate-forme** (comme nous l'avons dit auparavant, ils ont plutôt du code intermédiaire CIL);
- 3) En plus des instructions CIL, **le code assembleur .NET** (ou) **contient des méta-données** complètes décrivant l'ensemble **de tous les types** (classes, structures, énumérations, et ainsi de suite) et **tous les membres** de ces types (propriétés, méthodes, événements, etc.) **définis dans le fichier binaire compilé** :
 - Ces méta-données sont notamment utilisées par l'environnement d'exécution .NET, ainsi que par de nombreux outils de développement.
 - En autres, ce sont ces méta-données qui permettent l'*IntelliSense* lorsque nous écrivons du code.
 - Par ailleurs, elles servent à des outils de parcours d'objets, de débogage et de compilation.

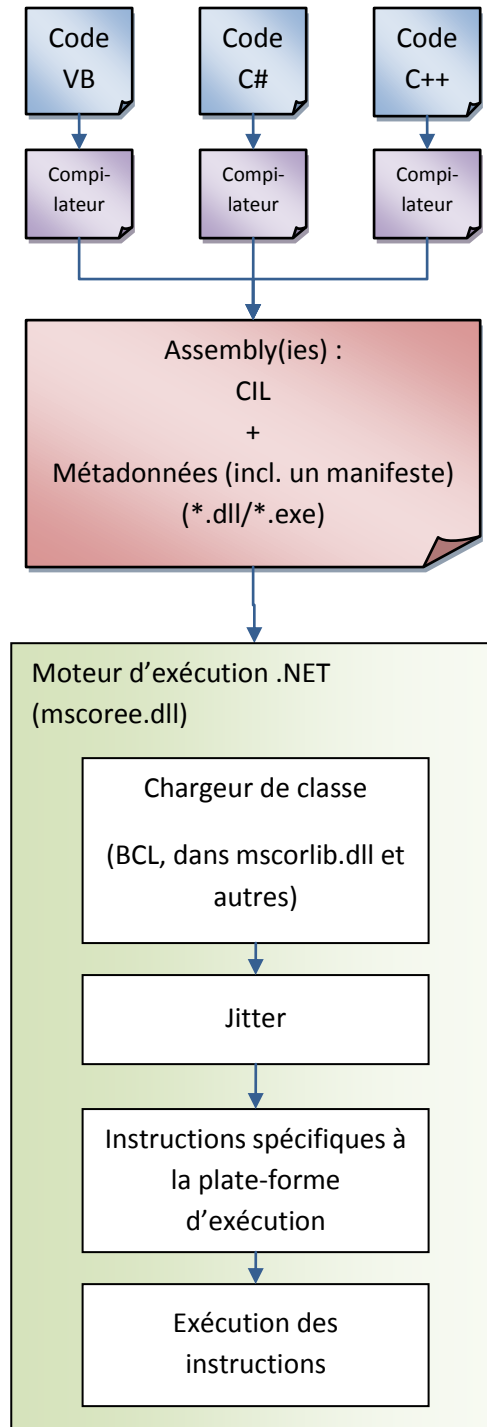
- Enfin, elles sont une partie intégrée de certaines technologies .NET, comme :
 - La réflexion (la faculté d'un langage à être son propre langage);
 - La sérialisation d'objets;
 - Certains services Web basés sur XML.
- 4) Enfin, en plus des méta-données « standards », les assemblies **contiennent également une description d'eux-mêmes qui se concrétisent sous un manifeste**, qui contiennent notamment les informations suivantes :
 - Le nom de *l'assembly*;
 - La version actuelle de *l'assembly*;
 - L'information culturelle (langue);
 - La liste des *assemblies* externes (et leurs versions) référencés nécessaires à l'exécution du fichier binaire.

1.1.6 En résumé!

Ainsi, en résumé, chaque langage conforme à l'infrastructure CLI possède un compilateur qui lui est propre dont le rôle est d'assurer une compilation uniforme en un ou plusieurs *assemblies* (un DLL ou un EXE) qui contien(nen)t et le code du programme en langage intermédiaire CIL et des métadonnées qui informent sur tous les types définis et ses membres dans les fichiers binaires, ainsi que des informations sur lui-même à l'intérieur d'un manifeste.

Ce ou ces *assemblies* pourront alors être interprétés puis exécutés sur une plate-forme donnée moyennant un Jitter approprié.

Le schéma à la page suivante résume cela en image.



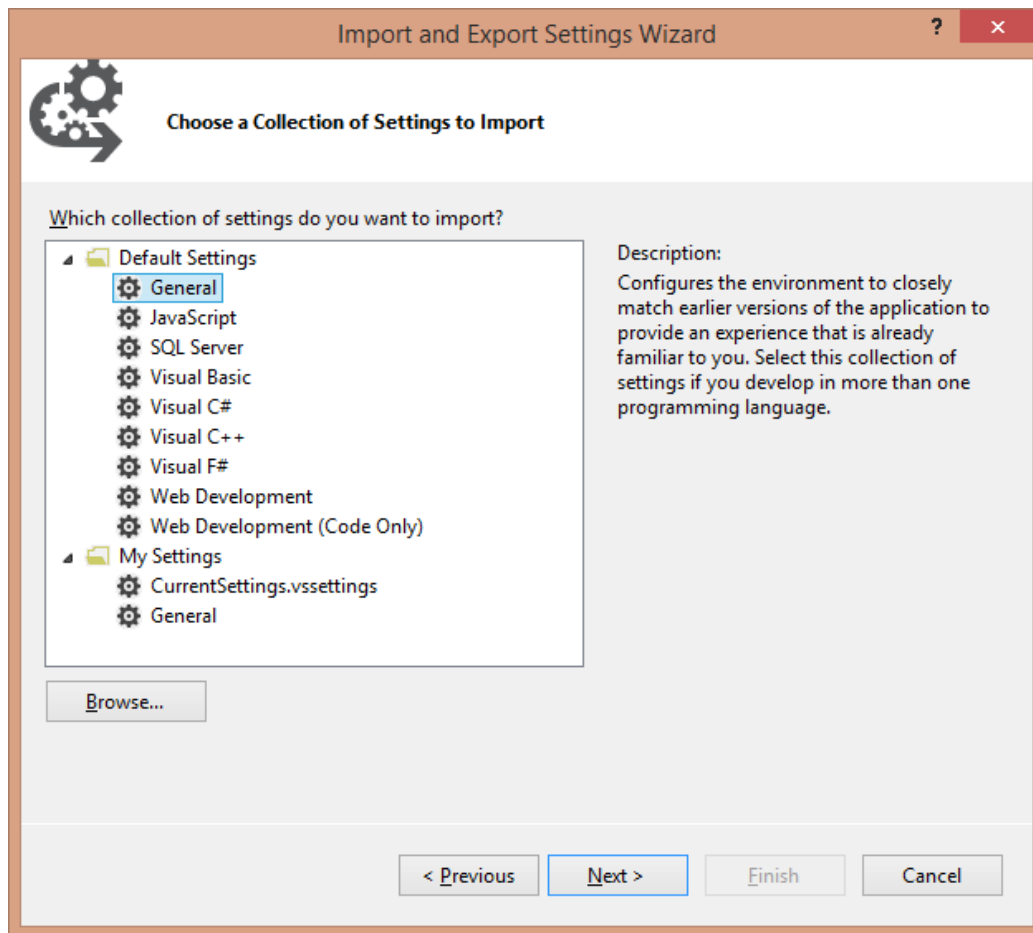
1.2 VISUAL STUDIO 2013

À travers ce chapitre, nous présenterons un tour d'horizon de la plate-forme de développement logiciel Microsoft Visual Studio .NET 2013..

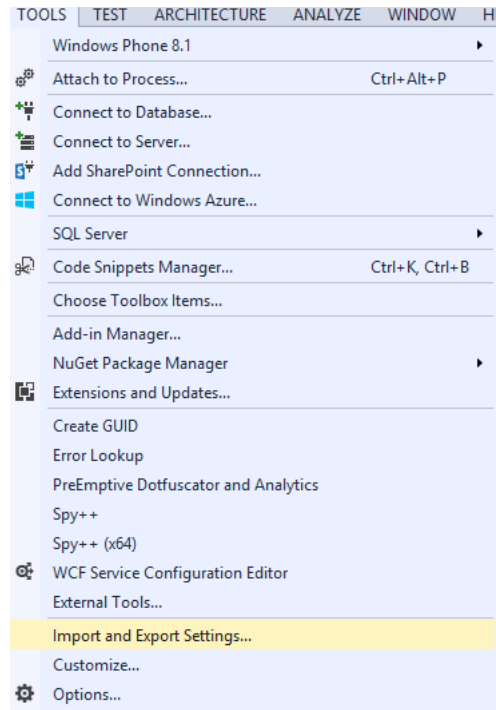
Tout d'abord, nous verrons succinctement la mouture Express de Visual C# 2013 (c'est-à-dire les différences par rapport à la version complète de Visual Studio. Puis, nous explorerons les différents menus, interfaces et fenêtres d'intérêt de Visual Studio .NET, de façon à se familiariser à son environnement de développement. Nous serons ainsi prêts à créer et gérer des solutions et des projets.

1.2.1 Démarrage

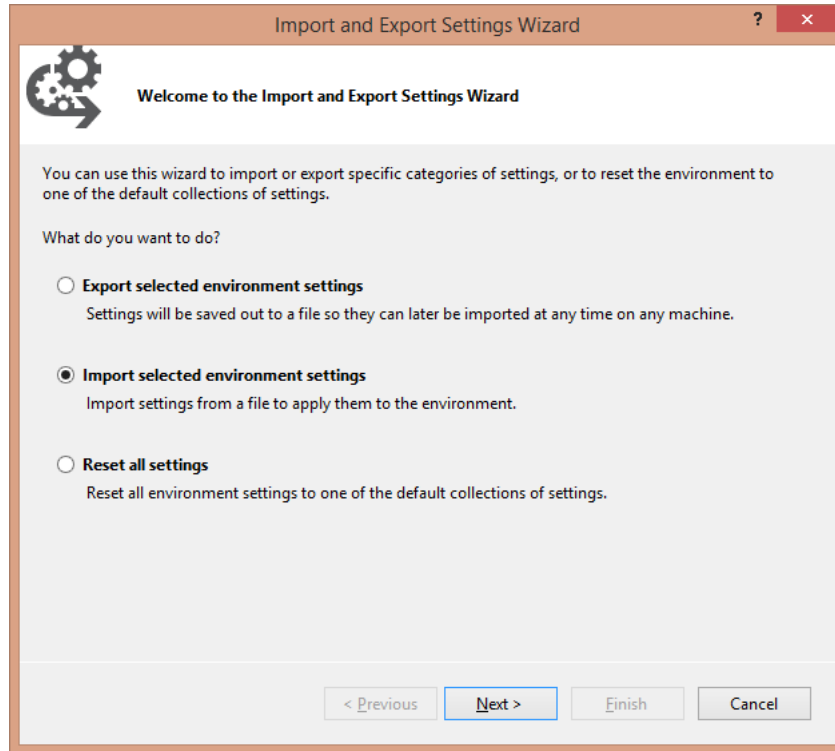
Lors du premier démarrage, on vous demandera choisir parmi certains environnements de travail prédéfini. Comme nous travaillerons surtout avec C#, vous pourriez choisir les paramètres de développement « Visual C# », mais par « défaut », vous pouvez toujours utiliser les paramètres de développement « généraux ».



Vous pourrez toujours changer de paramètres prédéfinis par la suite, en vous rendant au menu « Outil », puis « Importation et exportation de paramètres.



Cela vous mènera à une interface à partir de laquelle vous pouvez soit sauvegarder vos paramètres de votre environnement de travail comme nouveau « modèle », soit en importer un ou réinitialiser l'environnement avant que vous modifiez les paramètres.



1.2.2 Création d'une solution et d'un projet

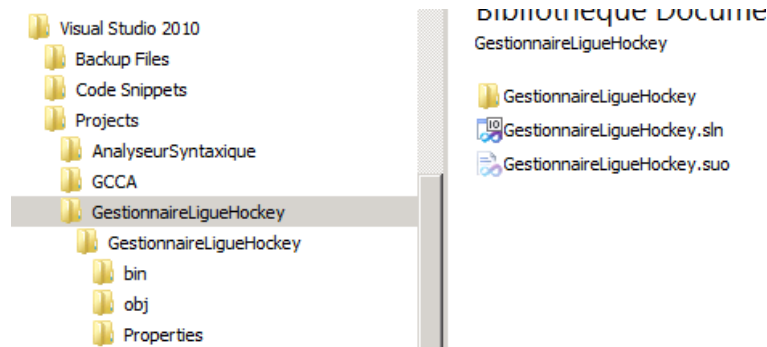
En Visual Studio, une solution est un ensemble (un conteneur) de fichiers de code et d'autres ressources qui sont utilisées afin de construire une application. Les fichiers de code sont regroupés à l'intérieur d'un projet, qui constitue l'entité compilable sous différents types de composants (des applications Windows ou de consoles en tant qu'exécutables .exe, des .dll, etc.). En temps normal, une solution possède donc au moins un projet.

Plus précisément, les solutions gèrent la façon dont Visual Studio configure, génère et déploie les projets qu'elles contiennent. Une application complexe peut même contenir plusieurs solutions. Elles permettent ainsi de :

- Travailler sur plusieurs projets au sein de la même instance d'exécution de l'IDE Visual Studio .NET.
- Travailler sur les éléments de la solution selon des paramètres et des options qui s'appliquent à un ensemble de projets définis.
- Utiliser l'explorateur de solutions afin d'offrir un soutien au développement et au déploiement de l'application.

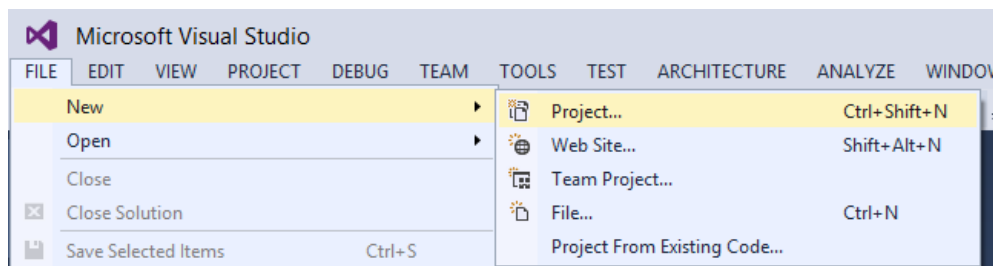
Le fichier de solution est enregistré dans deux (2) fichiers : .sln et .suo (qui est un fichier caché) :

- **Le fichier de définition de solution (.sln)** contient des métadonnées qui indiquent les projets associés à la solution, les éléments disponibles au niveau solution et qui ne sont associés à aucun projet particulier, puis les configurations de génération de solution qui définissent les configurations de projet à appliquer.
- **Le fichier d'options utilisateurs de solution (.suo)** contient des métadonnées concernant la configuration des différents éléments de personnalisation de l'IDE.

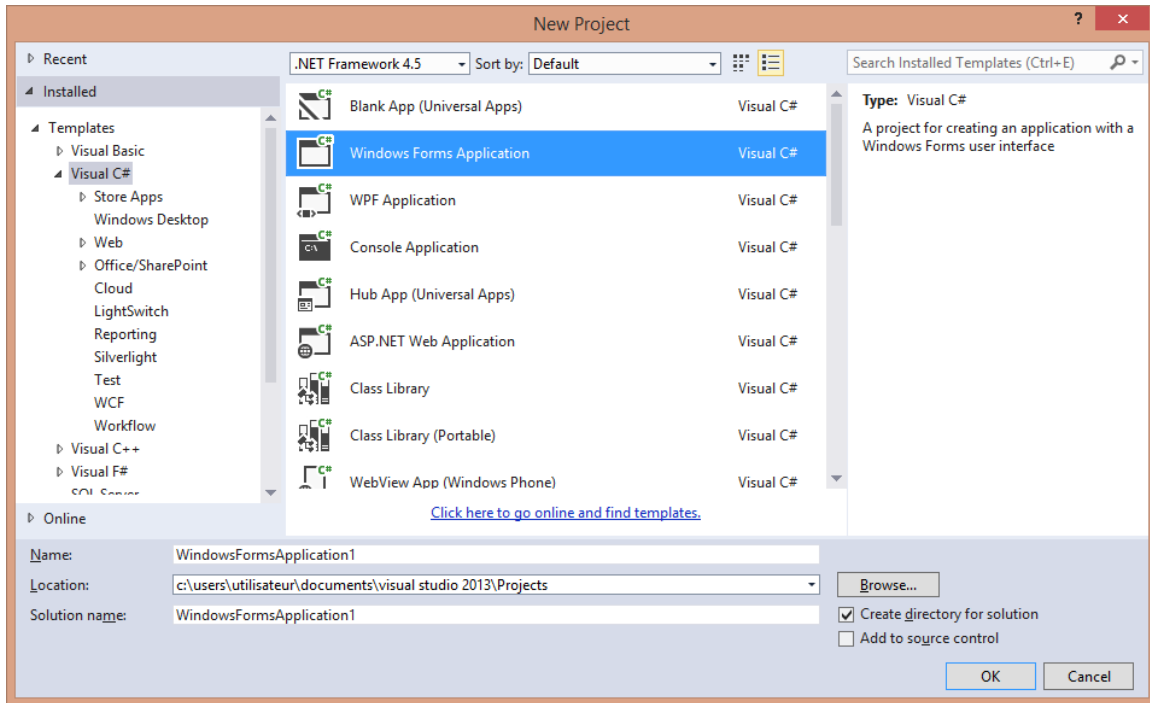


Ainsi, dans un contexte de développement en équipe, le fichier de définition de solution (.sln) pourrait être partagé entre tous les développeurs, mais chacun d'entre eux aurait son propre fichier d'options utilisateurs (.suo) selon ses préférences personnelles.

Afin de créer un projet (et une solution par le fait même), dirigez-vous au menu *Fichier > Nouveau > Projet...*, ou appuyez sur *Ctrl+Shift+N*.



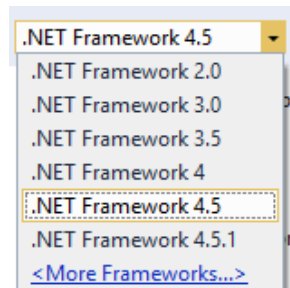
Une fois que cela est fait, une boîte de dialogue nommée « Nouveau Projet » apparaîtra.



Vous avez maintenant l'opportunité de choisir le modèle de projet qui sied à vos ambitions. Ces types de projet sont classés par langage de programmation. Comme vous pouvez le constater, les modèles sont très nombreux. Le choix d'un modèle implique de déterminer les éléments, le code et la configuration par défaut que le projet contiendra.

Dans le cadre de ce cours, nous nous intéresserons surtout aux modèles listés à *Visual C#* > *Windows Desktop*. Cette sous-catégorie contient les modèles relatifs à la création d'interfaces utilisateurs de types consoles, Windows Forms ou WPF et au support de ces dernières (comme les bibliothèques de classes, par exemple), ou encore des applications de services WCF. Évidemment, il ne s'agit que d'un projet quelconque dans votre solution : rien n'empêche d'ajouter par la suite un autre projet à la solution dont le modèle et/ou le langage .NET utilisé (et le type de sortie lors de la compilation du projet) est différent.

Dans la fenêtre de choix de modèle, en haut à droite, vous pouvez voir une liste de choix ayant l'allure d'un bouton où nous pouvons lire, par défaut, « .NET Framework 4.5 ». En cliquant sur cette liste, vous verrez les choix qui suivent :



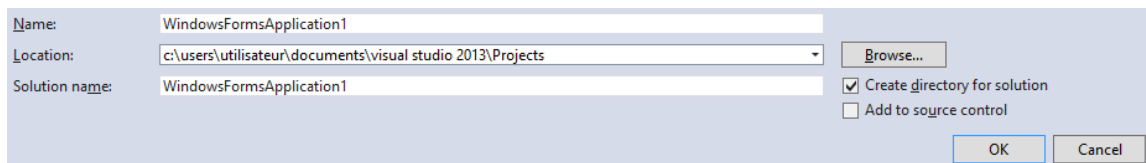
En fait, il s'agit simplement de la version du framework .NET que vous désirez utiliser pour coder et construire votre programme. Ainsi, si vous choisissez « .NET Framework 2.0 », vous n'aurez accès qu'aux types de projet supportés par .NET 2.0 et le programme sera compilé de façon à ce que toutes les machines capables d'interpréter les binaires de .NET 2.0 puissent le faire.

Voici une brève description des modèles potentiellement utiles à l'intérieur du présent cours :

- **Projet vide** : un projet vide ne contient qu'une solution avec un projet à l'intérieur il n'y a aucun composant et aucune référence inclus.
- **Application console** : ce type de projet est configuré afin de déployer une application fonctionnant en invite de commandes (à la manière de MS-DOS).
- **Application Windows Forms** : les projets de type Windows Forms sont ceux qui sont classiquement utilisés pour la création d'interfaces utilisateurs graphiques Windows.
- **Application WPF** : depuis le *framework* 3.5, Microsoft offre une toute nouvelle approche pour créer des interfaces utilisateurs graphiques par le biais d'applications *Windows Presentation Foundation* et c'est ce que permet la création d'un projet de ce type. Nous verrons plus tard cette session une introduction en la matière.
- **Bibliothèque de classes** : ce projet permet de créer une bibliothèque de classes d'extension .dll. Nous en ferons dans un chapitre ultérieur.

Lorsque vous désirez créer une solution, vous devrez choisir un quelconque modèle pour votre projet. Une fois le modèle choisi, vous devez spécifier un endroit où enregistrer la solution sur un disque donné :

- Le champ « *Nom* » vous invite à écrire le nom du projet;
- Le champ « *Nom de la solution* » est explicitement celui de la solution en tant que tel (*vous l'aurez compris : le nom de la solution peut être différent de celui du projet*).



The screenshot shows the 'Add New Project' dialog in Visual Studio. The 'Name' field is 'WindowsFormsApplication1'. The 'Location' field is 'c:\users\utilisateur\documents\visual studio 2013\Projects'. The 'Solution name' field is 'WindowsFormsApplication1'. There is a 'Browse...' button next to the 'Location' field. Two checkboxes are present: 'Create directory for solution' (checked) and 'Add to source control' (unchecked). 'OK' and 'Cancel' buttons are at the bottom right.

Avec Visual Studio.NET, vous développez des projets.

Un **projet** contient :

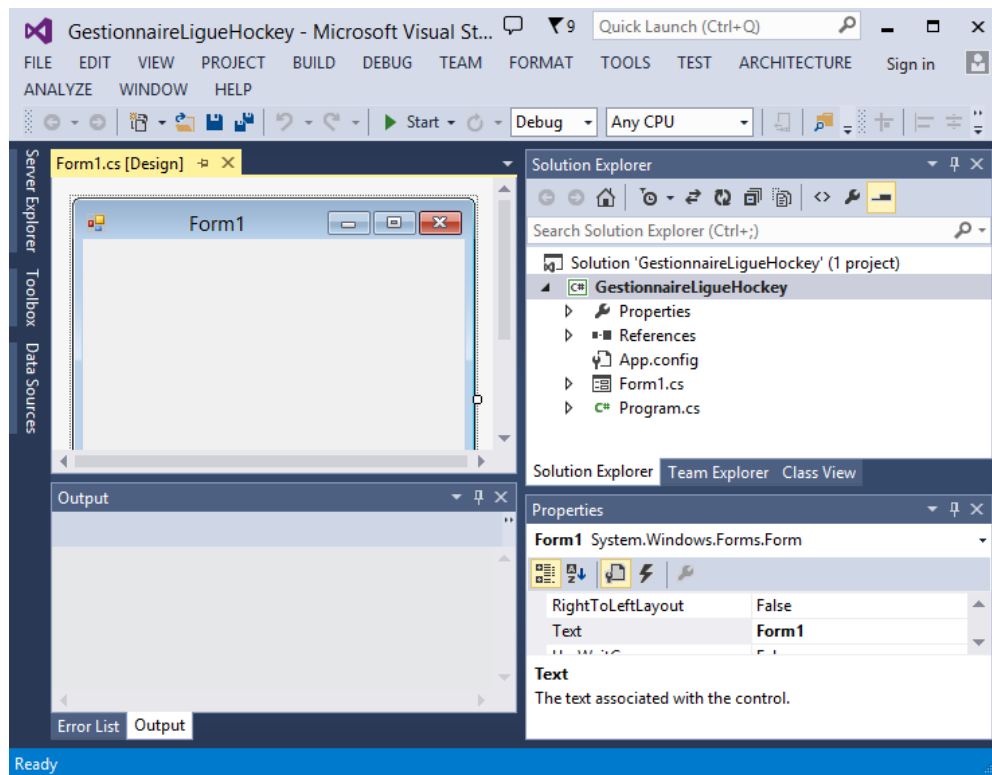
- Tous les fichiers de codes source;
- Tous les icones et les images, ou autres fichiers de données;
- Les configurations de compilation;

Bref, tout ce qui est nécessaire de façon à pouvoir compiler en un programme exécutable (assembly .exe ou .dll).

Les projets sont contenus dans une **solution** (qui peut contenir un ou plusieurs projets).

Une fois ces informations entrées, il ne reste plus qu'à cliquer sur le bouton « *OK* » afin de créer officiellement le projet et la solution conséquente.

En choisissant de créer une application Windows Forms dont le projet et la solution sont nommés « *GestionnaireLigueHockey* », vous devriez voir à peu près ceci :



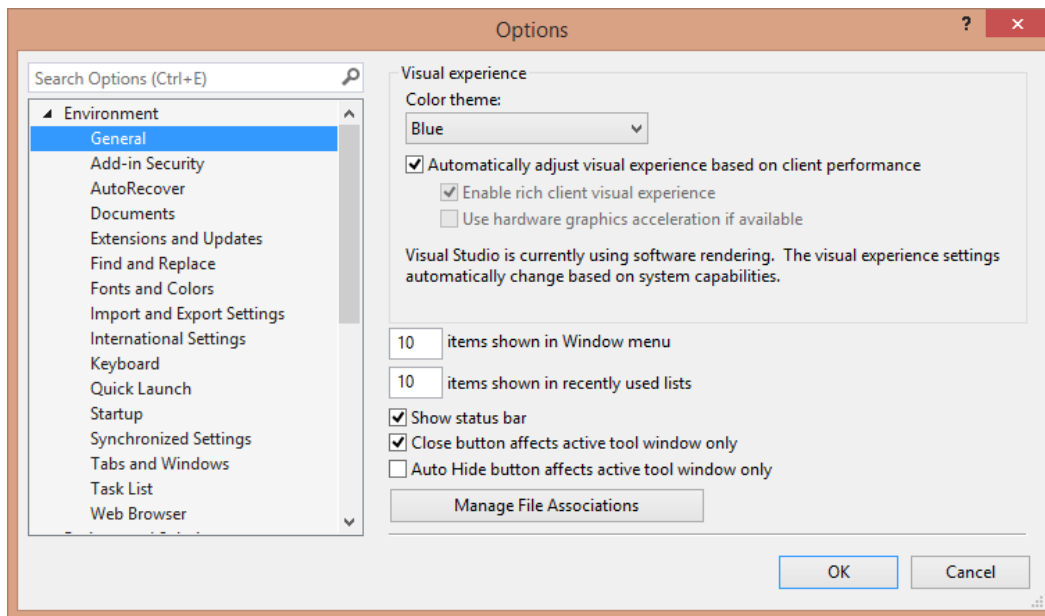
Les prochaines sous-sections serviront à décrire les différents outils qui apparaissent dans cette interface. Avant d'y passer, sachez que vous pouvez en tout temps sauvegarder les

modifications apportées à un élément de la solution via le menu *Fichier > Enregistrer [Nom de l'élément sélectionné]*, ou *Ctrl+S*, ou encore sauvegarder l'ensemble des éléments modifiés en allant au menu *Fichier > Enregistrer tout*, ou *Ctrl+Shift+S*.

1.2.3 Options

La boîte de dialogue *Options* vous permet de configurer l'environnement de développement intégré (IDE) selon vos besoins. Elle vous permet de définir des emplacements d'enregistrement par défaut pour différents éléments de la solution, de modifier la présentation le comportement par défaut des fenêtres, et beaucoup plus.

Afin d'atteindre la boîte de dialogue des options, rendez-vous au menu *Outils > Options*.



Vous retrouverez dans l'onglet de gauche une arborescence des catégories d'options. En cliquant sur l'une d'entre elles, les options spécifiques à la catégorie s'afficheront dans l'onglet de droite (*notez que cliquer sur un nœud racine comme « Environnement » affiche les mêmes options que cliquer sur la sous-catégorie « Général »*).

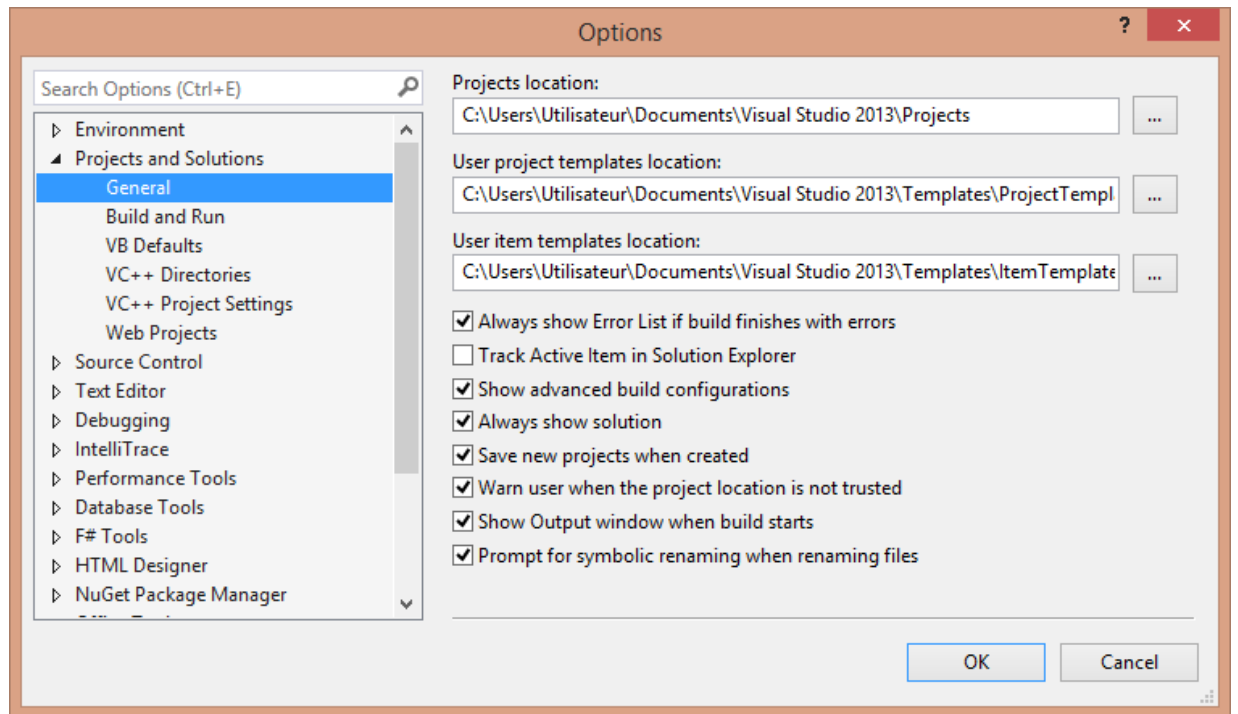
Bien entendu, nous ne verrons pas chacune des options disponibles – il y en a près d'une centaine – mais voici en somme ce que permet de configurer chacune des **principales** catégories d'options :

- **Environnement** : les options offertes permettent de définir comment certains éléments de l'environnement de développement intégré (IDE) s'affichent et se comportent. Les différentes sous-catégories (pages) sont les suivantes (les pages jugées plus importantes apparaissent en caractère gras) :

Page	Description des options
Général	Personnaliser la présentation et le comportement des fenêtres, des menus et d'autres éléments de l'environnement de développement intégré (IDE).
Clavier	Configurer le schéma de configuration du clavier selon le langage de programmation et créer de nouveaux raccourcis pour les commandes fréquemment utilisées.
Démarrage	Spécifier le contenu ou l'interface utilisateur affiché lors du démarrage de Visual Studio.
Documents	Définir les paramètres de gestion et d'affichage de documents, y compris du comportement des fichiers divers.
Importation et de exportation paramètres	Modifier les préférences d'enregistrement des profils.
Liste des tâches	Définir des options pour les jetons de commentaires et les tâches (« TODO », « UNDONE », ...) qui seront recherchées pour constituer les listes de tâches.
Navigateur Web	Modifier les pages « Accueil » et « Recherche », changer d'éditeur de code source système et configurer des options d'Internet Explorer.
Paramètres internationaux	Sélectionner une langue par défaut.
Polices et couleurs	Définir la police et les couleurs dans certains éditeurs (notamment l'éditeur de code), boîtes de dialogue, fenêtres d'outils et autres éléments de l'interface utilisateur.

Page	Description des options
Récupération automatique	Modifier les paramètres par défaut d'enregistrement et de restauration automatiques des fichiers (enregistrement des fichiers tous les x minutes et conservation de la copie de récupération pour y jours)
Sécurité des compléments / macros	Spécifier les paramètres de sécurité et les chemins d'accès pour les compléments et les macros.

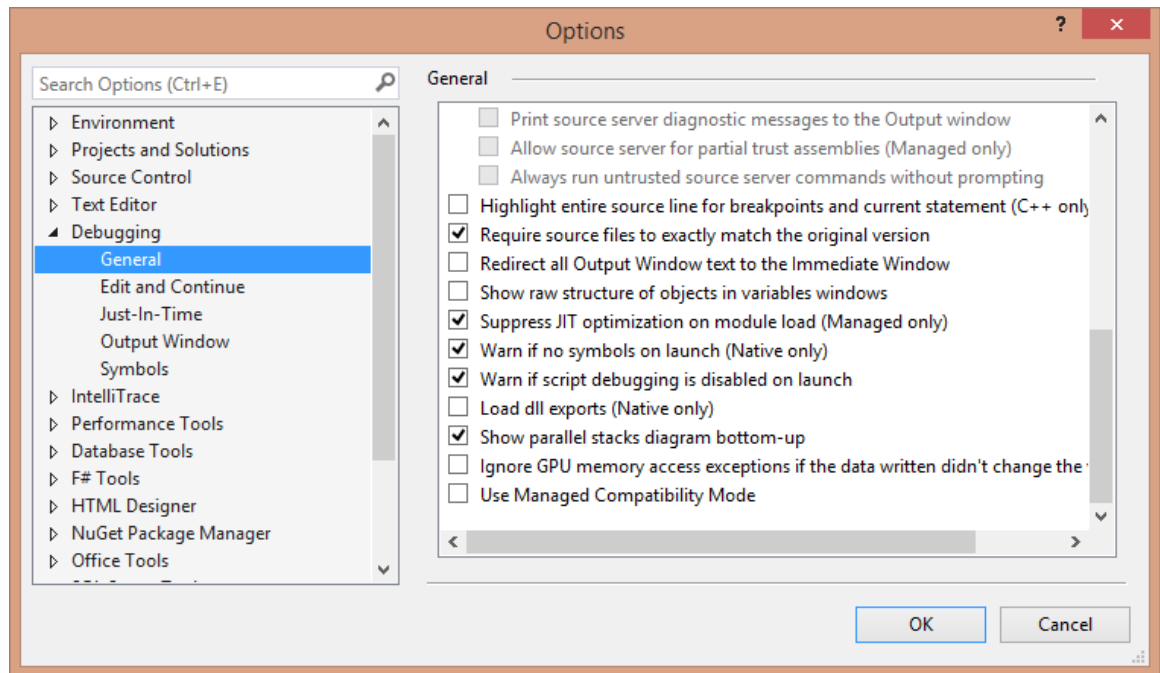
- **Projets et solutions** : les options disponibles permettent de définir les options par défaut pour développer et générer vos projets et solutions dans l'environnement de développement intégré (IDE). Ces options sont plutôt importantes. Voici ce que contient sommairement chacune des pages :



Page	Description des options
Général	Définir le chemin d'accès par défaut pour les dossiers du projet <i>Visual Studio</i> et déterminer le comportement par défaut de certains éléments comme la fenêtre des sorties, des listes des tâches et de l'explorateur de solutions à mesure que les projets sont développés et générés.
Générer et exécuter	Déterminer certains comportements avant la génération d'une solution (enregistrer toutes les modifications, enregistrer les modifications des documents ouverts uniquement, inviter à enregistrer les modifications des documents ouverts, ou ne pas enregistrer les modifications des documents ouverts) et quelques comportements par défaut à l'exécution (générer uniquement des projets de démarrage et des dépendances à l'exécution et choisir si on doit générer automatiquement, sur demande ou non une solution à l'exécution si un projet est obsolète.
Valeurs par défaut VB	<p>Spécifier les paramètres par défaut pour les options de projet Visual Basic. Lorsqu'un projet est créé, les instructions d'options spécifiées sont ajoutées à l'entête du projet dans l'éditeur de code et celles-ci s'appliquent à tous les projets Visual Basic. Ces options sont au nombre de trois (3) :</p> <ul style="list-style-type: none"> - <i>Option Explicit</i> : configure le compilateur de façon à ce qu'une déclaration explicite des variables soit requise par défaut. - <i>Option Strict</i> : configure le compilateur de manière à ce qu'une conversion restrictive explicite soit requise (à « <i>On</i> », p. ex., aucune conversion implicite d'un <code>String</code> vers un <code>Double</code> ne serait alors possible). - <i>Option Compare</i> : configure la méthode de comparaison de chaînes à utiliser par le compilateur (respect ou non de la casse). Nous reviendrons plus spécifiquement sur cette option au prochain chapitre lorsque nous traiterons de la manipulation des chaînes de caractères.

Vous avez également des sous onglets qui présentent des options spécifiques à la paramétrisation de projets C++ et aux projets web par rapport à IIS (*Internet Information Services*) pour des projets en ASP .NET, par exemple.

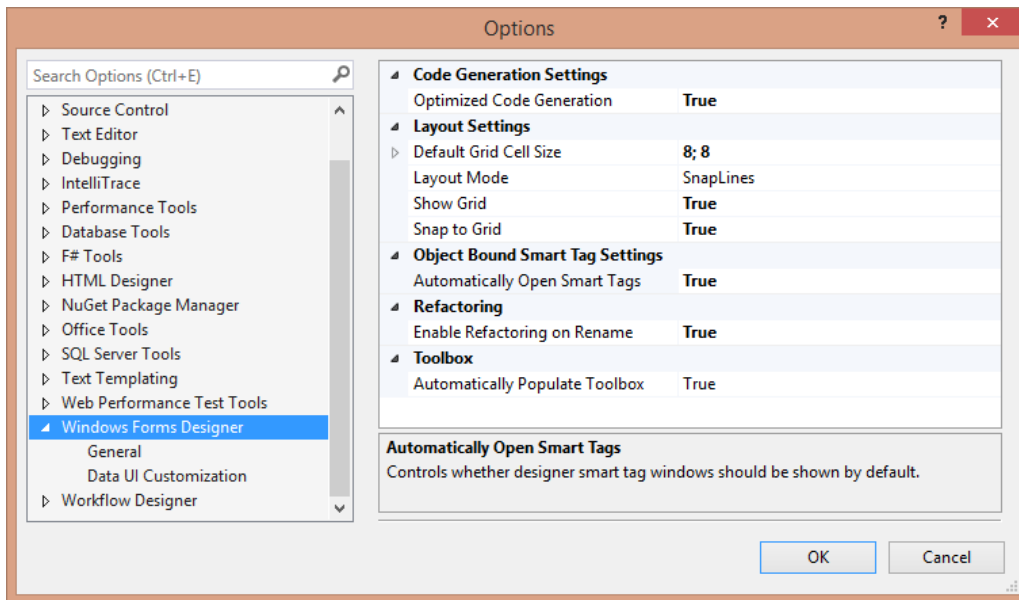
- **Débogage** : les options présentes permettent de personnaliser la configuration de l'outil de débogage. Les principales options d'intérêts de débogage se retrouvent dans les deux premières pages :



Page	Description des options
Général	Déterminer les options relatives aux points d'arrêt et aux comportements à adopter si un point d'arrêt est atteint ou si une exception est interceptée par le <i>Common Language Runtime</i> (CLR).
Modifier & Continuer	Activer ou non la possibilité de pouvoir modifier le code source lorsque le programme est en mode arrêt et d'appliquer les modifications sans avoir à mettre fin à la session de débogage ni à générer de nouveau le programme en question (cela devrait être activé par défaut).

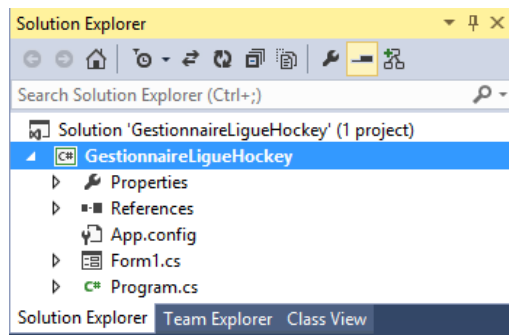
Dans l'onglet « Fenêtre sortie », vous pourrez aussi activer ou désactiver certaines options relatives aux informations devant apparaître en sortie pour vous aider à déboguer l'application.

- **Concepteur Windows Forms** : les options offertes permettent de changer les paramètres par défaut des grilles d'éditions d'interfaces utilisateurs et d'autres fonctionnalités des concepteurs visuels de Visual Studio.



1.2.4 Explorateur de solutions

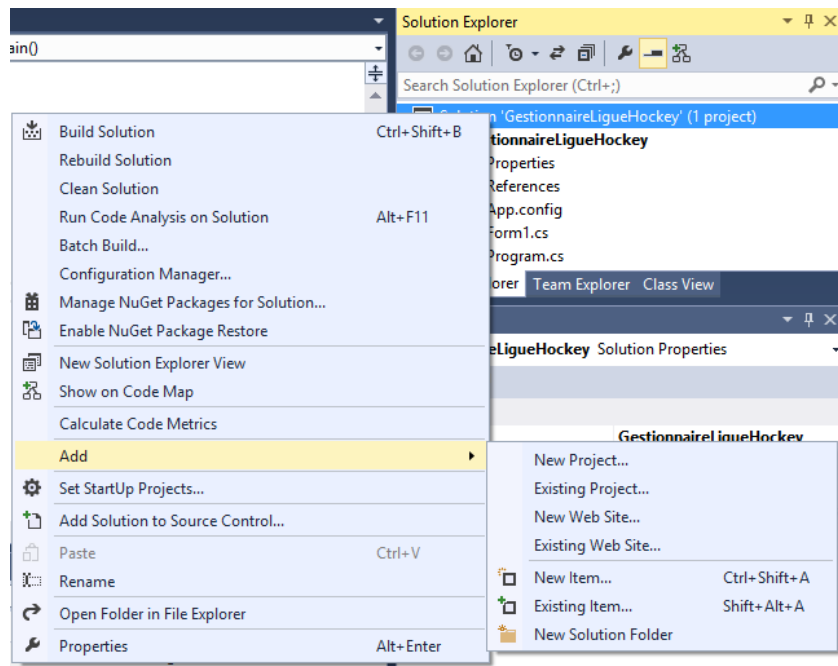
L'explorateur de solutions vous offre une vue organisée de vos projets et de leurs fichiers, ainsi qu'un accès aux commandes qui s'y rapportent. Une barre d'outils associée à cette fenêtre présente les commandes couramment utilisées pour l'élément en cours dans la liste. Il devrait être présent par défaut lorsque vous créez un projet. Si jamais elle n'apparaît pas, vous pouvez le retrouver en allant au menu *Affichage > Explorateur de solutions*, ou *Ctrl+Alt+L*.



Comme nous pouvons le constater, l'explorateur de solutions est constitué d'une arborescence logique où la racine est la solution en tant que telle et où chacun de ses nœuds est un projet de la solution. Puis, chaque projet contient ses éléments, comme les fichiers de code, les références et les paramètres, par exemple.

1.2.4.1 GESTION D'UNE SOLUTION

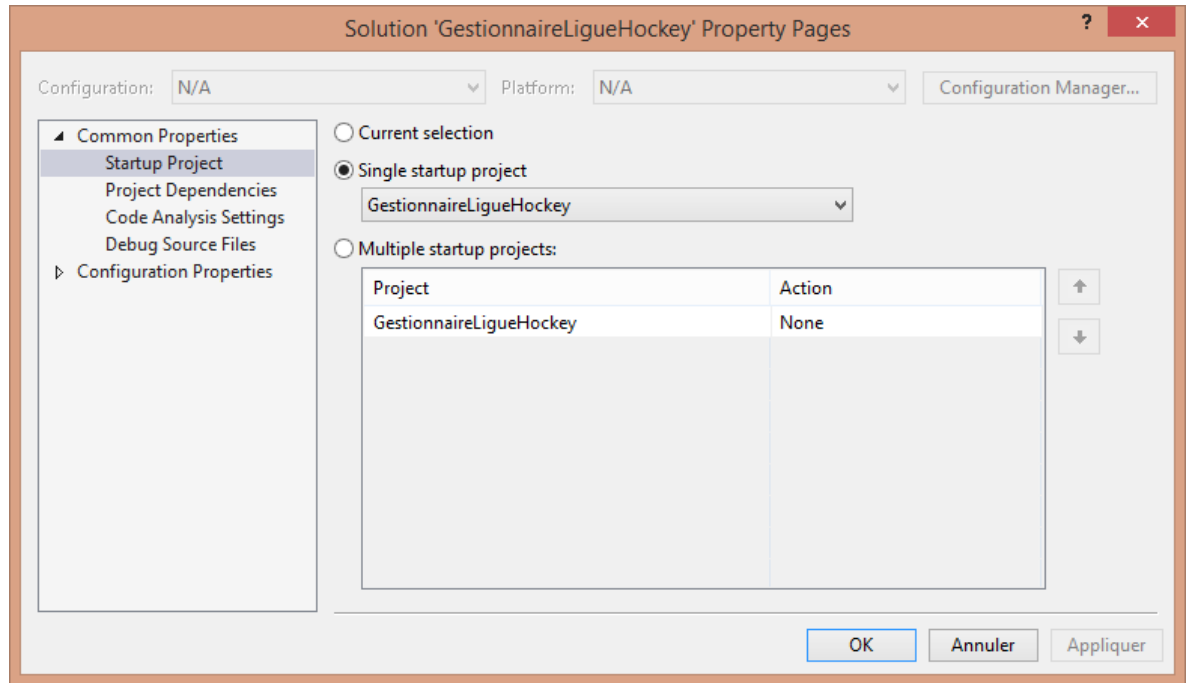
En cliquant sur le bouton droit de la souris sur la solution dans l'arborescence de l'explorateur de solutions, vous devriez avoir le menu contextuel suivant :



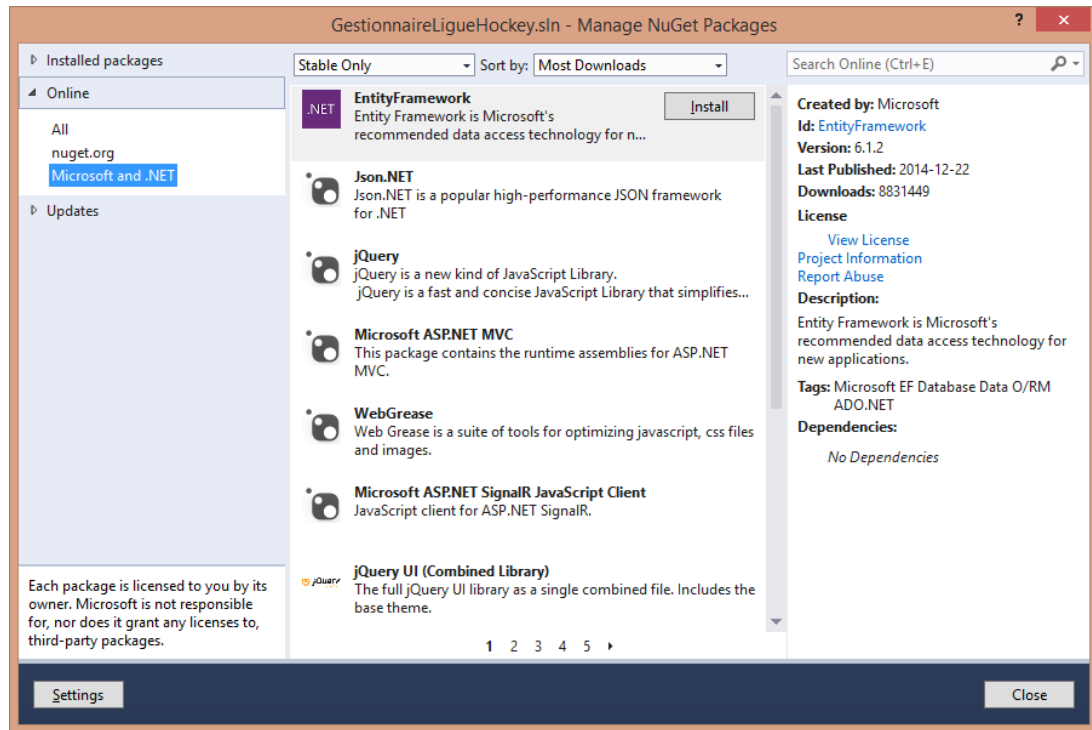
Les premières options vous permettent de compiler ou de configurer la solution. Grossièrement :

- **Générer la solution** : compile et construit les fichiers binaires de sortie selon la configuration de la solution et des projets associés.
- **Nettoyer la solution** : supprime tout fichier intermédiaire ou de sortie, ne laissant que les fichiers du projet et des composants à partir desquels de nouvelles instances des fichiers intermédiaires ou de sortie peuvent ensuite être générées.
- **Regénérer la solution** : nettoie tout d'abord la solution avant de la générer.
- **Définir des projets de démarrage** : sélectionne quels projets d'une solution seront les premiers à être générés, débogués et déployés dans Visual Studio, puis exécutés en premier lieu lors du déploiement de la solution. Cela vous amènera à une boîte de dialogue dans laquelle vous pourrez définir un seul projet de

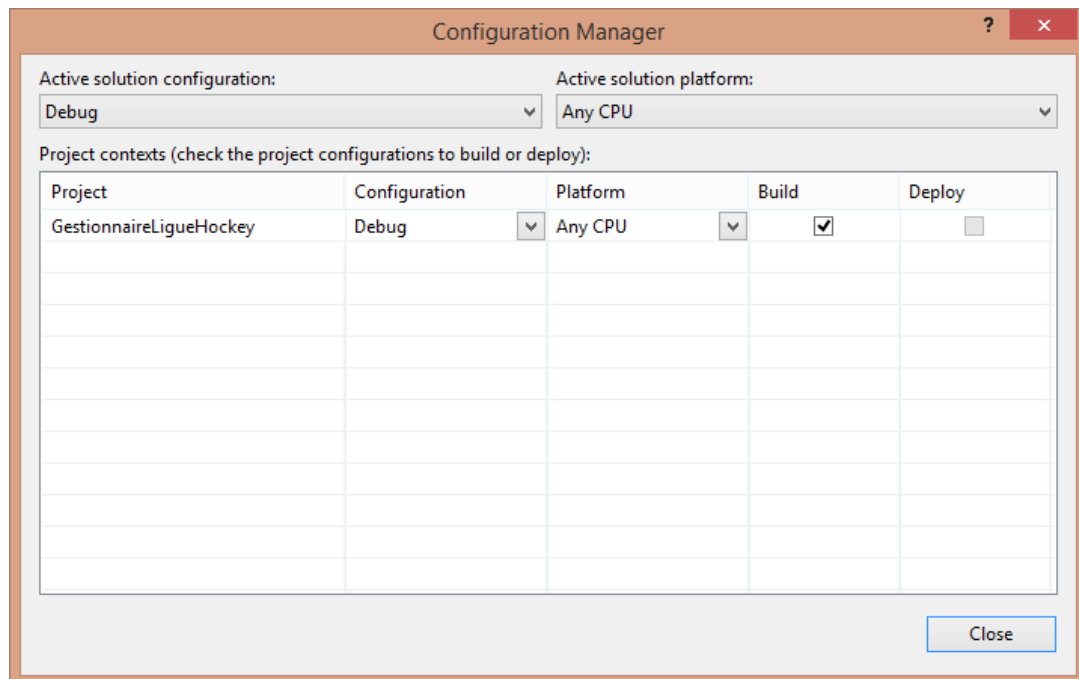
démarrage parmi tous les projets de la solution, ou plusieurs projets de démarrage :



- **Ajouter** : permet d'ajouter des éléments à la solution, tels une nouvelle solution ou un nouveau projet dans la solution sélectionnée vide ou à partir d'un projet préexistant.
- **Renommer** : renomme la solution.
- **Gérer les packages NuGet** : NuGet a été introduit dans Visual Studio 2010 et constitue une extension de Visual Studio. Les packages Nuget sont en quelque sorte des bibliothèques importables/intégrables/utilisables en quelques clics à partir de certaines sources (Microsoft ou sources libres (nuget.org)). Par exemple, pour travailler avec Entity Framework, vous devriez installer le NuPack adéquat à partir de cette interface.



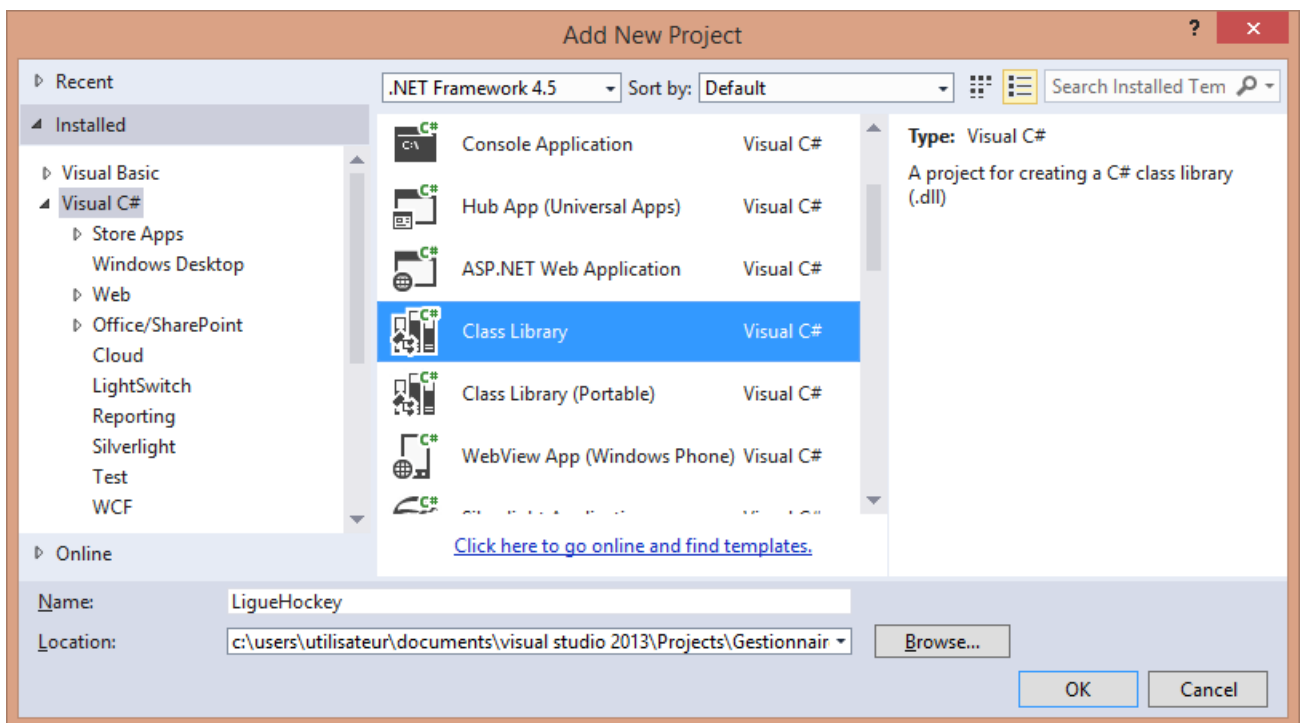
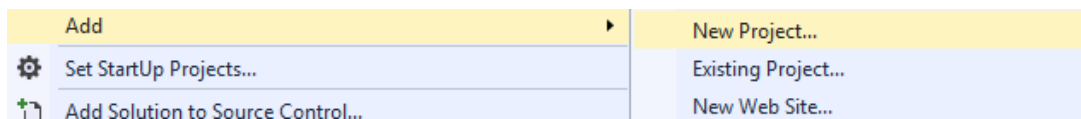
- **Gestionnaire de configuration** : permet de spécifier la configuration active pour la solution de façon générale, mais également pour chacun des projets de la solution, si chacun d'entre eux doit être généré ou non et dans quelle configuration (Debug, Release, ou une configuration personnalisée) :



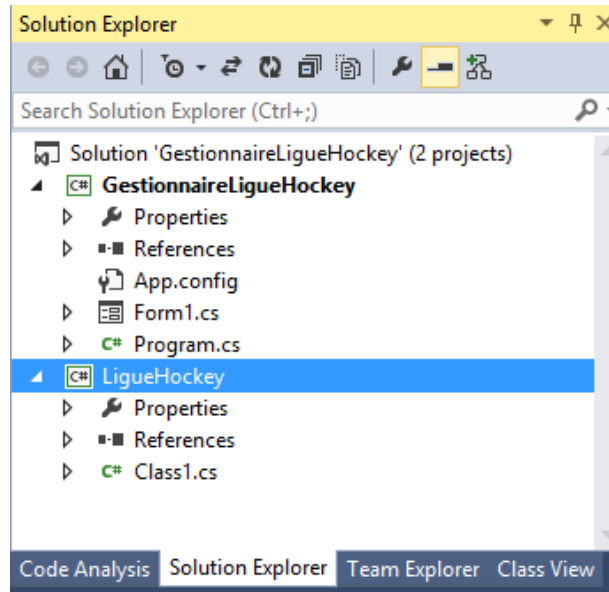
- **Propriétés** : amène à une boîte de dialogue qui comprend les fenêtres de gestion de configuration et de définition de projets de démarrage, plus la configuration des dépendances au projet.

Exemple

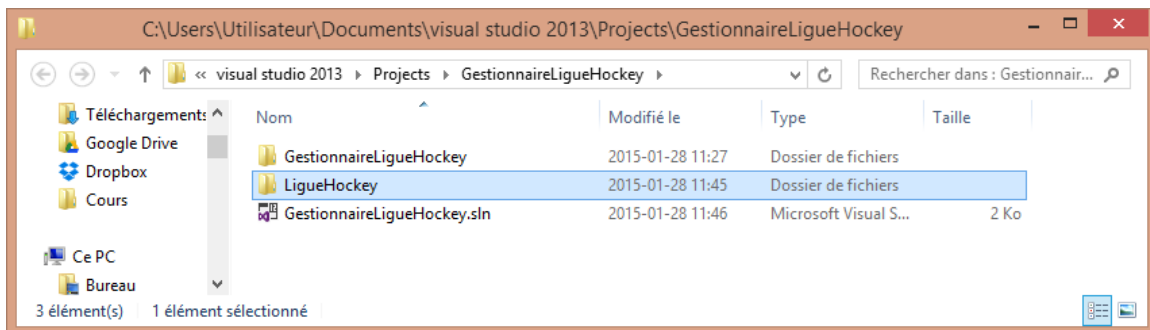
Supposons que vous avez créé une solution contenant un projet de type Windows Forms en C# dont la solution et le projet portent le nom « *GestionnaireLigueHockey* ». Pour ajouter un nouveau projet – par exemple une bibliothèque – il faudrait alors faire un clic droit sur la solution et se rendre à *Ajouter > Nouveau projet...*, puis sélectionner « Bibliothèque de classes » dans l'onglet Visual C#. Enfin, donnez « *LigueHockey* » comme nom du projet.



Cela vous ajoutera un projet supplémentaire dans l'explorateur de solution :



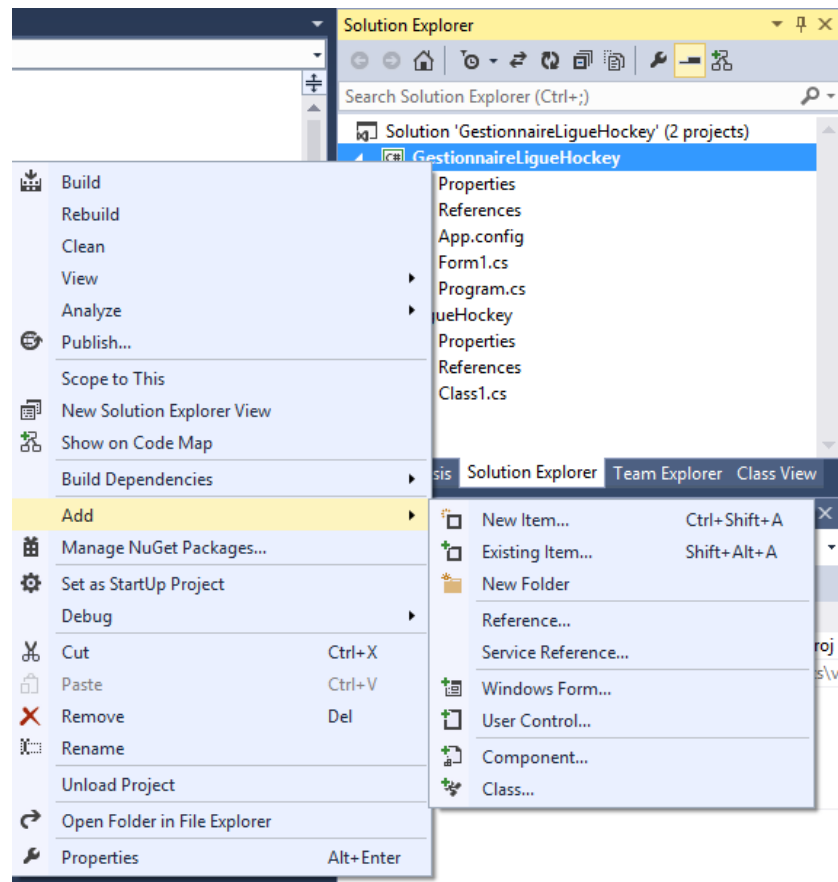
Vous aurez également un répertoire supplémentaire dans votre dossier de solution portant le nom du nouveau projet (vous pouvez atteindre directement le dossier de la solution en utilisant l'option « *Afficher le dossier dans l'explorateur de fichiers* » du menu contextuel).



1.2.4.2 GESTION D'UN PROJET

L'explorateur de solutions vous permet également de gérer chacun des projets en modifiant ses propriétés ou en lui ajoutant des dossiers (espaces de noms), des éléments comme des classes ou des modules ou des références à des composants .NET ou .COM.

Sélectionnez un projet et cliquez sur le bouton droit de la souris afin de faire apparaître un menu contextuel :

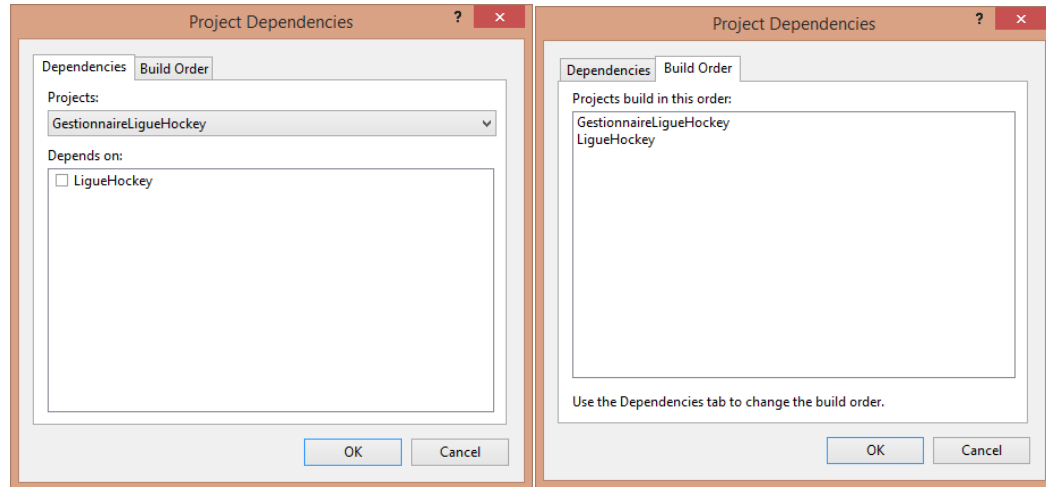


Ce menu offre les options suivantes :

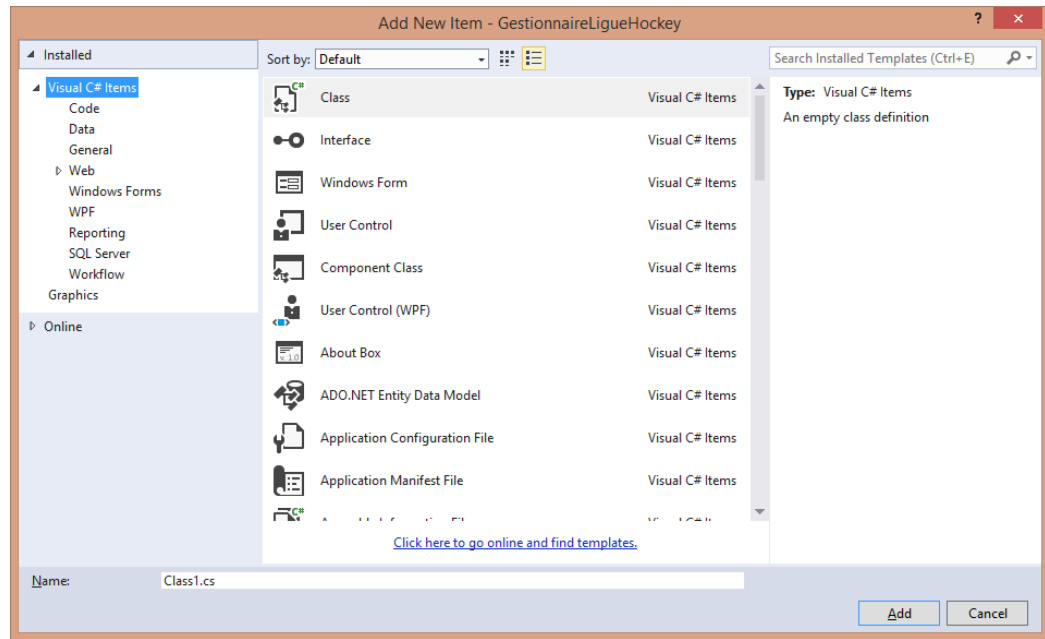
- **Générer, régénérer ou nettoyer le projet** : compile ou nettoie le projet et ses dépendances.
- **Publier** : publie le fichier binaire et ses dépendances sur un disque local, un espace partagé, un site Web ou un serveur FTP.
- **Analyse > Calculer la métrique du code** : des métriques utilisés pour obtenir des indices de maintenabilité, de complexité/couplage au niveau des classes et du nombre de lignes de code des membres peuvent être calculés.

Code Metrics Results					
Filter: None					
Min: Max:					
Hierarchy	Maintainability In...	Cyclomatic Comp...	Depth of Inheritan...	Class Coupling	Lines of Code
GestionnaireLigueHockey (Debug)	80	6	7	8	12
GestionnaireLigueHockey	80	6	7	8	12
Form1	79	5	7	5	9
Dispose(bool) : void	80	3		3	3
Form1()	82	1		2	3
InitializeComponent() : void	80	1		4	3
Program	81	1	1	3	3
Main() : void	81	1		3	3

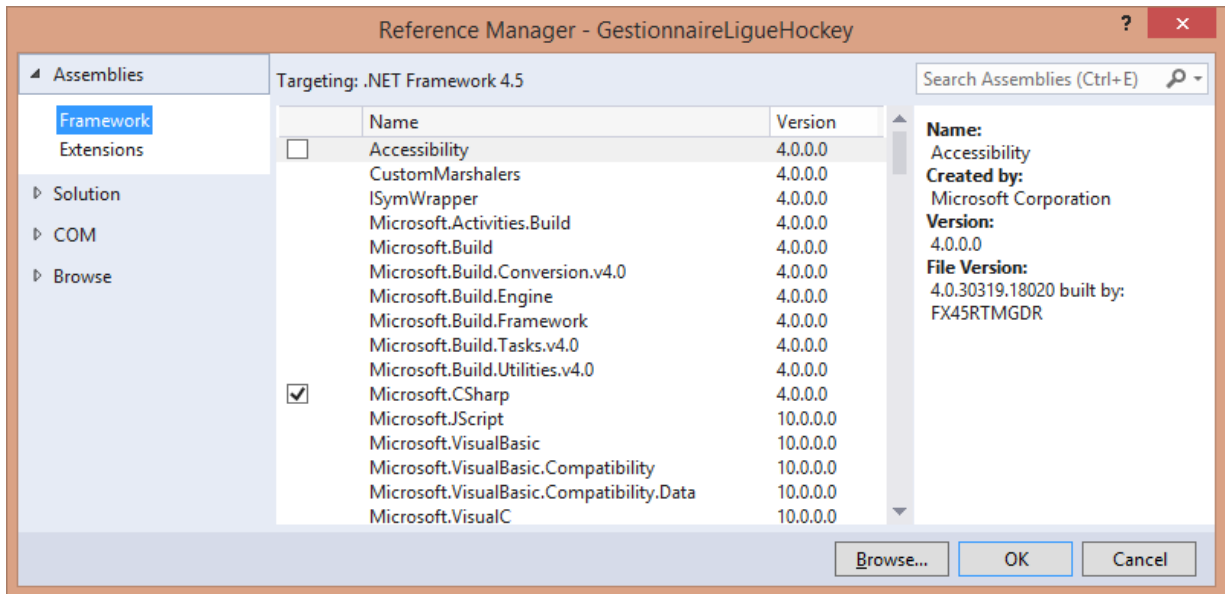
- **Dépendances de génération > Dépendances du projet ...:** on peut à cet endroit indiquer certains liens de dépendances entre différents projets d'une même solution, par exemple, afin de modifier l'ordre de la génération des *assemblies*.



- **Couper, copier, supprimer ou renommer :** édite le projet.
- **Définir en tant que projet de démarrage :** définit le projet sélectionné comme étant celui qui sera utilisé comme projet de démarrage unique.
- **Afficher le diagramme de classes :** charge dans un onglet (et s'il n'existait pas auparavant, l'ajoute au projet) un diagramme de classes qui permet de visualiser les éléments d'un projet, leurs contenus et leurs relations (dépendances). Il s'agit d'une alternative de développement intéressante pour définir rapidement les squelettes des éléments (définir les classes, les propriétés, les méthodes, les variables d'instances, etc.) que nous n'aborderons pas spécifiquement dans le cadre de ce cours, mais sachez tout de même que vous pouvez l'utiliser.
- **Ajouter :** permet d'ajouter des éléments au projet sélectionné :
 - **Nouvel élément :** ouvre une boîte de dialogue qui vous permet de sélectionner un type d'élément à ajouter à votre projet :



- **Élément existant** : ouvre une boîte de dialogue d'ouverture de fichier et vous invite à sélectionner un fichier (normalement un fichier d'extension .cs, ou un fichier média, par exemple) contenant l'élément que vous désirez ajouter à la solution.
- **Nouveau dossier** : crée un nouveau dossier en tant que conteneur logique d'éléments pour votre projet. En Visual C#, les éléments créés dans ces dossiers appartiennent par défaut à l'espace de noms racines et celui du dossier. Si vous créez un dossier nommé « Échanges », alors toute classe créée dans ce dossier aura donc par défaut comme espace de nom « GestionnaireLigueHockey.Échanges ».
- **Formulaire Windows, Contrôle utilisateur, Composant ou Classe** : ouvre la boîte de dialogue d'ajout de nouvel élément en présélectionnant le type d'élément en question. Le but est de vous permettre d'ajouter plus rapidement les éléments les plus communs.
- **Ajouter une référence** : permet d'ajouter des références au projet via une boîte de dialogue. Cette dernière peut nécessiter quelques instants pour s'afficher.



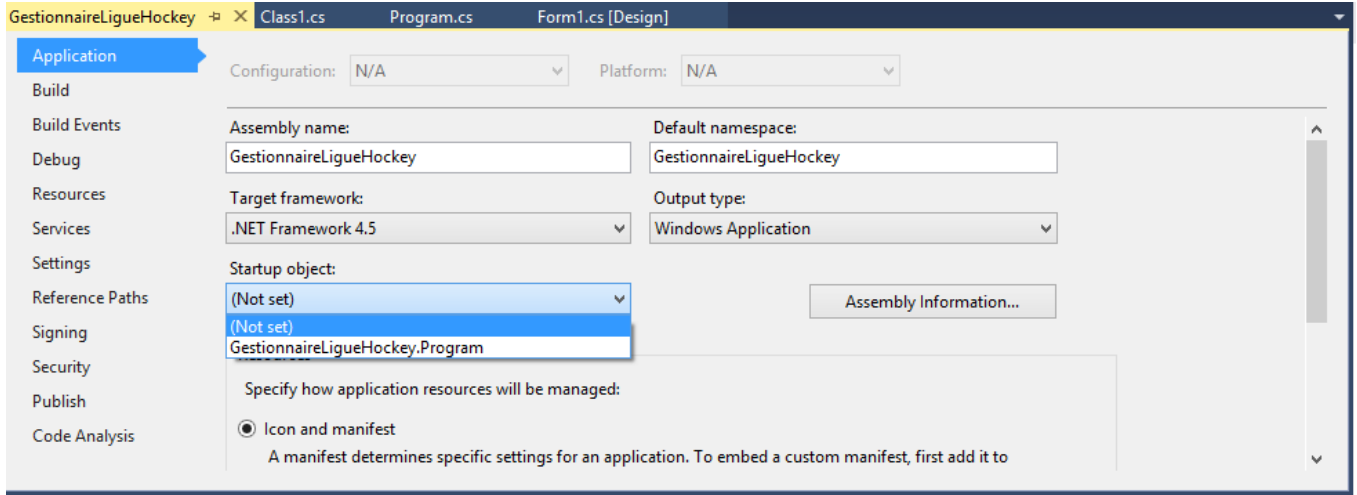
Comme vous pouvez le constater, il y a plusieurs onglets :

- **Assemblies > Framework** : répertorie tous les composants du *framework* .NET pouvant être référencés.
- **COM** : répertorie tous les composants COM pouvant être référencés. Un lot d'applications en offre, comme Microsoft lui-même (p. ex. avec Office, sur lequel nous reviendrons dans un chapitre ultérieur), Adobe pour Acrobat et autres.
- **Solution > Projets** : répertorie tous les composants réutilisables créés à partir de projets locaux
- **Parcourir** : permet de naviguer à la recherche d'un composant dans le système de fichiers.

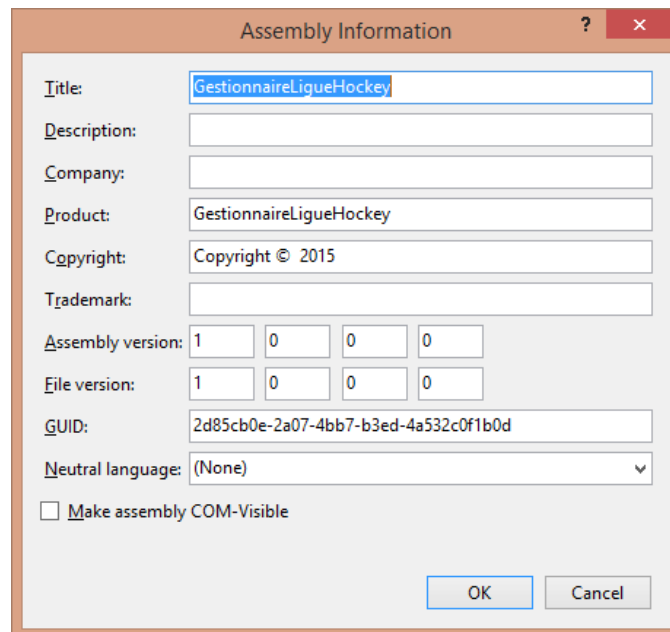
Une fois ajoutée au projet, vous pourrez utiliser les éléments que la référence contient pour programmer.

- **Déboguer** : permet de lancer le projet en mode débogage.
- **Propriétés** : ouvre une boîte de dialogue où vous pourrez définir des propriétés particulière pour votre projet. Vous y trouverez beaucoup d'éléments en commun avec les propriétés ou la configuration de la solution, mais celles-ci sont spécifiques au projet sélectionné. C'est un peu la tour de contrôle du projet. Il y a beaucoup d'onglets et d'options. Nous décrirons ici les onglets principaux.

- **Application** : les options de cet onglet vous permettent de définir les propriétés de l'application générée. Entre autres, vous pouvez spécifier l'espace de noms racine du projet, le type d'application générée (application Windows Forms, application console, bibliothèque de classes (.dll) ou service Windows), l'élément de démarrage (le point d'entrée) de l'application (dans le cas d'une application Windows Forms, on spécifie un formulaire de démarrage; dans le cas d'une bibliothèque de classes, il n'y en a pas), un icône, et des informations sur l'*assembly*.



Si vous cliquez sur le bouton « Information de l'*assembly*... », la boîte de dialogue suivante apparaîtra :

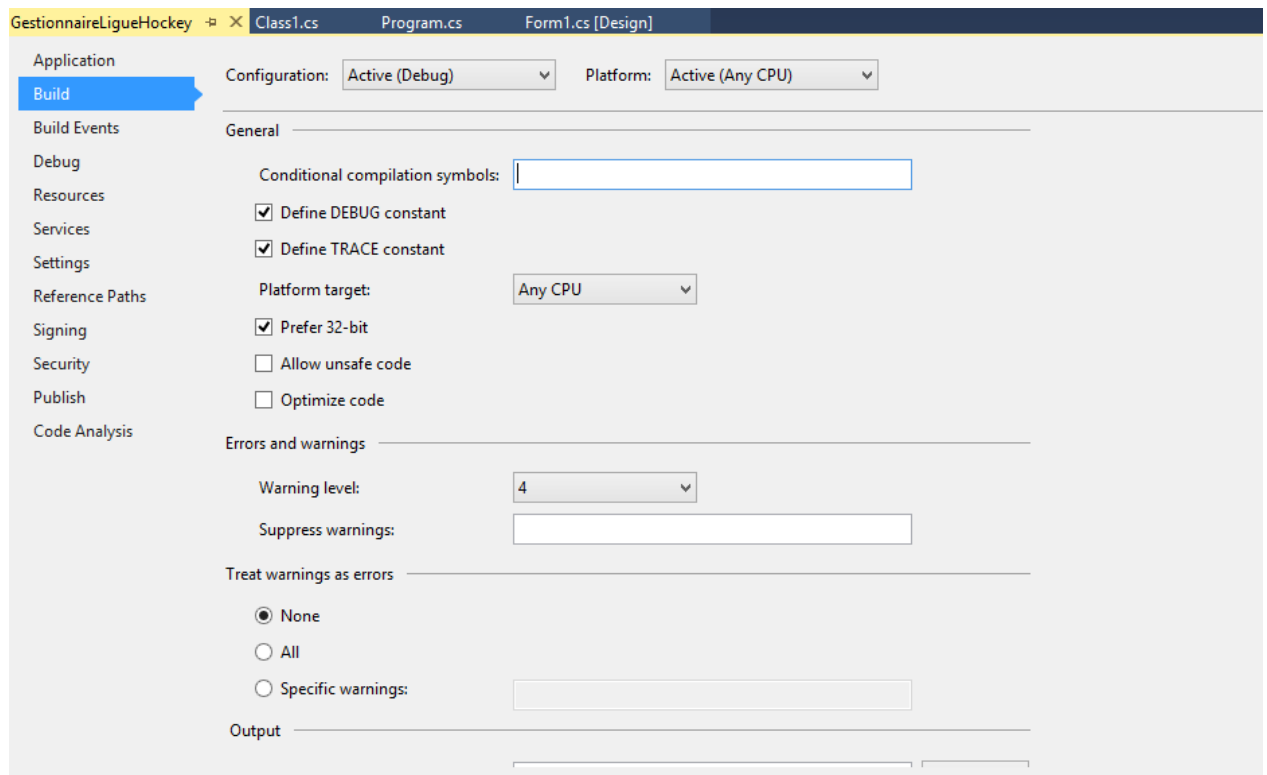


Comme vous le voyez, vous pouvez changer les propriétés de l'*assembly* qui seront inhérentes au fichier binaire généré.

Notez que le GUID est un identificateur à toute fin pratique unique qui est généré automatiquement pour un *assembly* donné, qui peut s'avérer nécessaire dans certain contexte pour travailler, par exemple, avec une version particulière d'un même *assembly*.

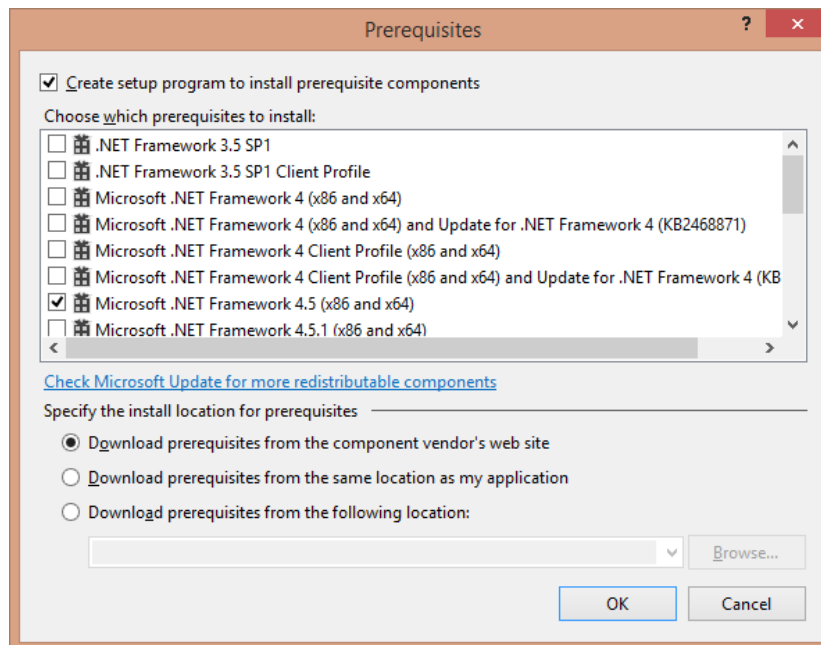
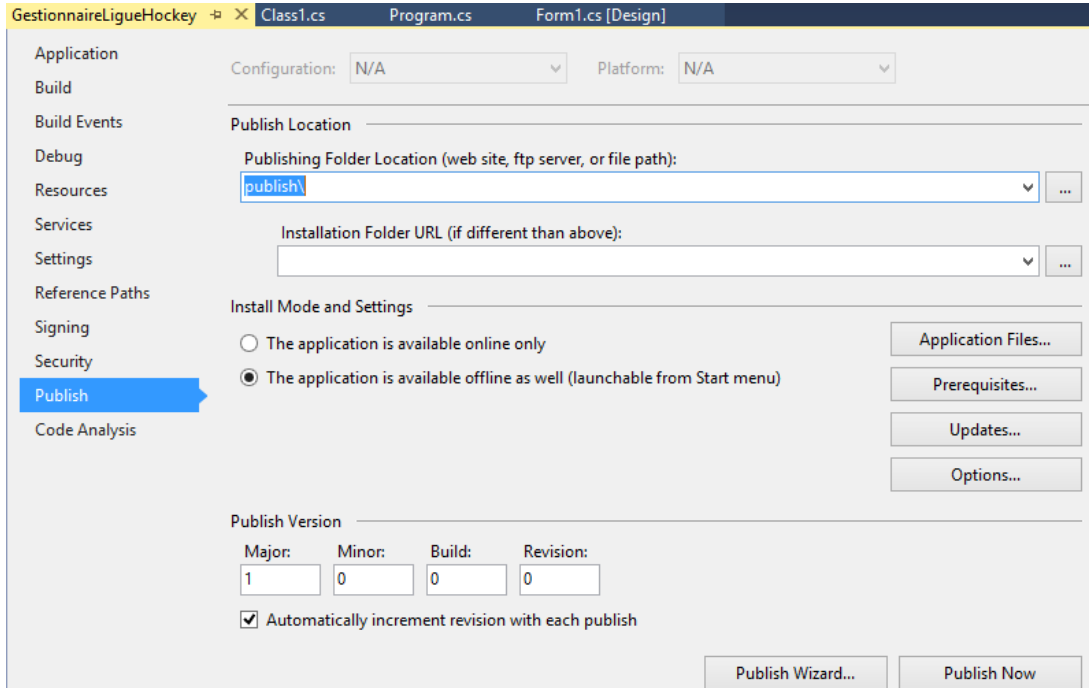
L'option « Rendre l'assembly visible par COM » sert à déterminer si les types décrits dans l'assembly sont disponibles pour un composant COM, dans un objectif de rétrocompatibilité inversée .NET-COM.

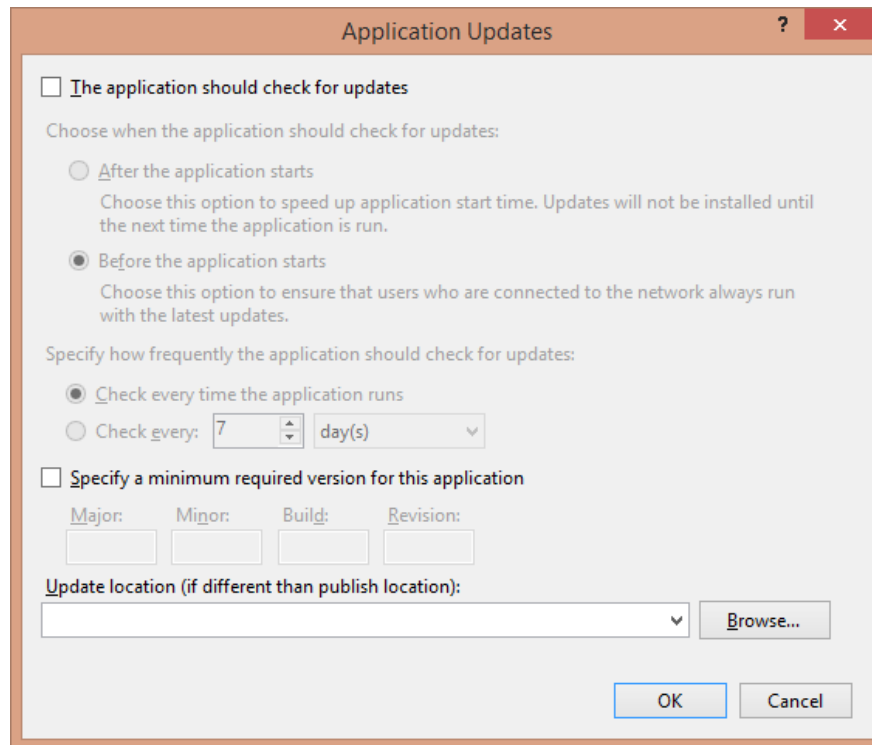
- **Générer** : vous pouvez y spécifier un chemin de sortie pour la génération des fichiers de compilation, la configuration d'exécution active, les options de compilations vues auparavant et la configuration de certains éléments de compilations (en indiquant si vous désirez qu'à la rencontre d'une situation donnée, cela soit considéré comme étant une erreur ou non, ou encore si un avertissement doit être indiqué dans la console d'erreurs. Vous pouvez également déterminer le chemin de sortie de ou des *assemblies* générés.



- **Publier** : cet onglet vous permet de configurer les options relatives à la publication de l'application. Vous pouvez entre autres spécifier un chemin

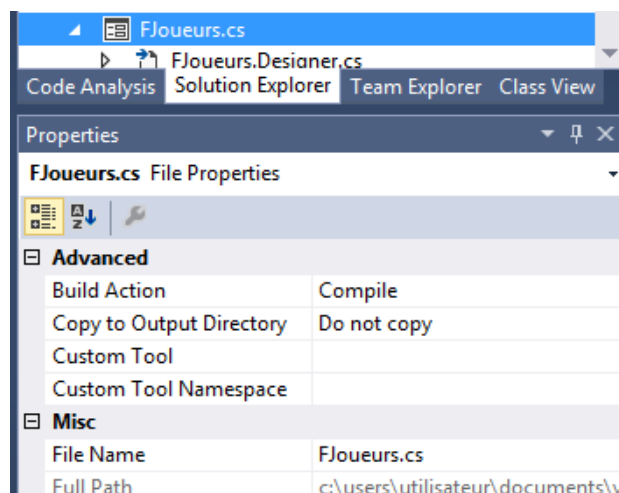
local ou distant pour la publication de l'application, spécifier la version de l'application et d'autres paramètres concernant les paramètres pour la mise à jour de l'application, les fichiers publiés et les pré-requis inclus ou non avec la publication.





Cela complète le tour d’horizon de l’explorateur de solutions et les boîtes de dialogues principales que nous pouvons atteindre via ce dernier, afin de permettre une gestion des solutions, des projets, de leurs éléments et leurs références et des options de compilation, de débogage, de génération et de publication.

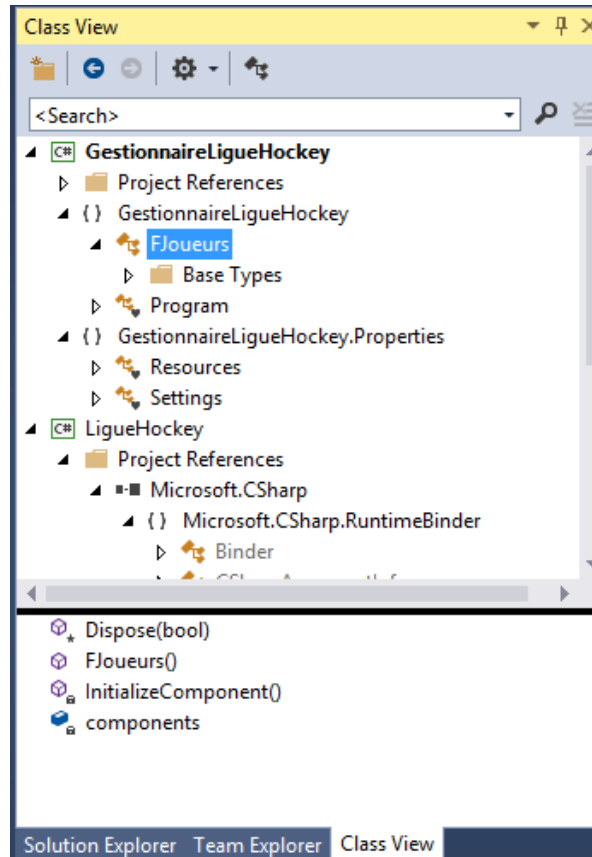
Enfin, notez qu’en cliquant sur un élément dans l’explorateur de solutions, il y a quelques propriétés qui s’afficheront dans la boîte des propriétés et que vous avez la possibilité de modifier. Par exemple, en cliquant sur un formulaire Windows, vous pourriez avoir ceci :



Nous reviendrons plus en détails sur cette boîte de propriété dans une prochaine section.

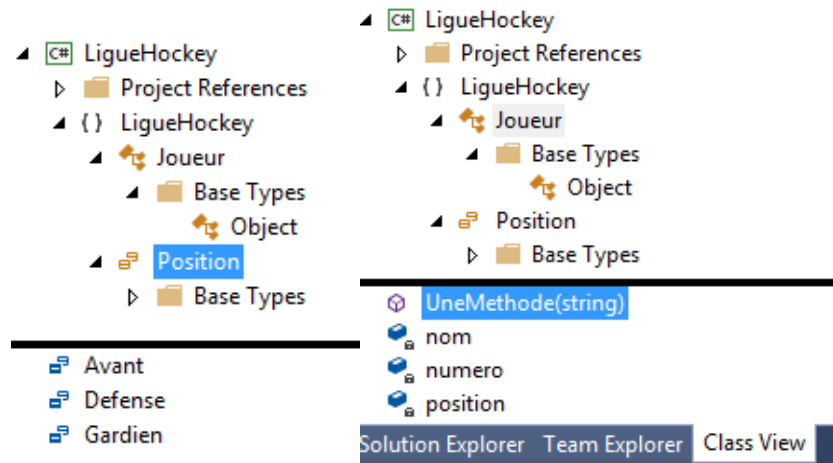
1.2.5 Affichage de classes

Dans la même boîte contenant l'explorateur de solution, vous avez également accès à un second onglet : l'affichage de classes.

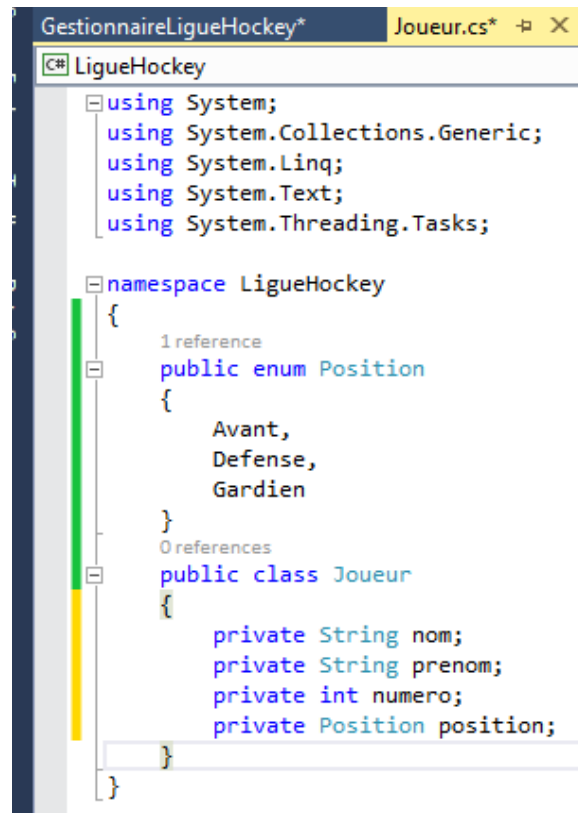


Cette vue présente chaque projet de la solution avec d'une part les références et, d'autre part, les objets du projet (les classes, les énumérations, les structures, les interfaces, les modules, ...) de façon hiérarchique selon les espaces de noms auxquels ils appartiennent.

De plus, si vous cliquez sur un objet donné, vous verrez dans le volet du bas la liste des membres de ce dernier. Dans le cas d'une classe, les propriétés, les variables d'instances et les méthodes seront listées et vous pourrez voir la signature complète de celles-ci (incl. les paramètres). Dans le cas d'une énumération, vous verrez les membres (éléments/valeurs) de celle-ci.

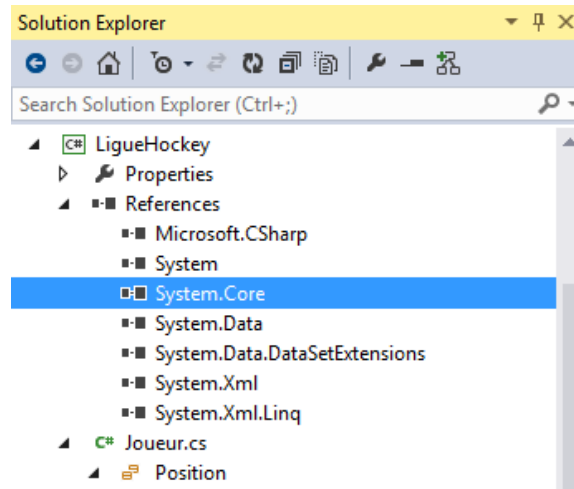


En double-cliquant sur un élément de l'arborescence, un onglet s'ouvrira et le curseur pointera (et sélectionnera) l'élément en question :

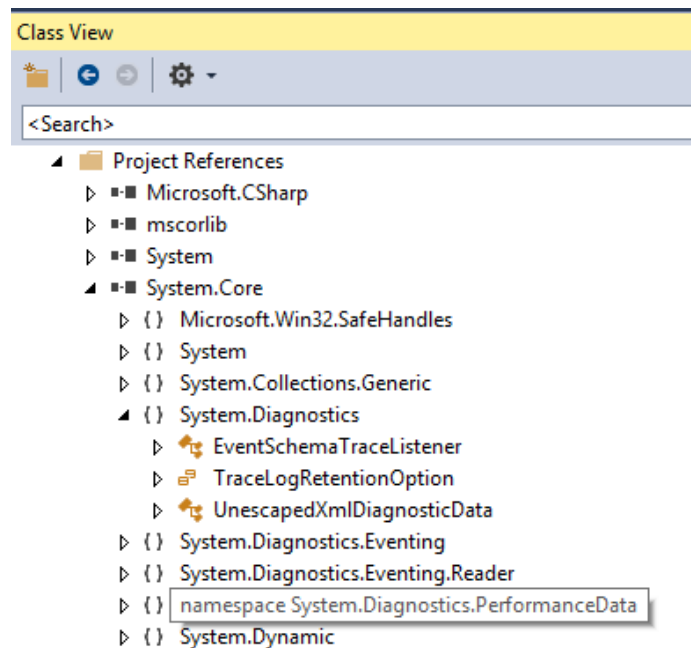


C'est donc particulièrement pratique pour atteindre rapidement des membres d'éléments qui ne sont pas déjà ouverts dans un onglet, et c'est encore plus utile si l'élément en question possède beaucoup d'instructions, puisqu'on peut se diriger directement vers une méthode donnée, par exemple.

Par rapport aux références, nous pouvons également développer la hiérarchie entre les espaces de noms jusqu'aux membres de ceux-ci.



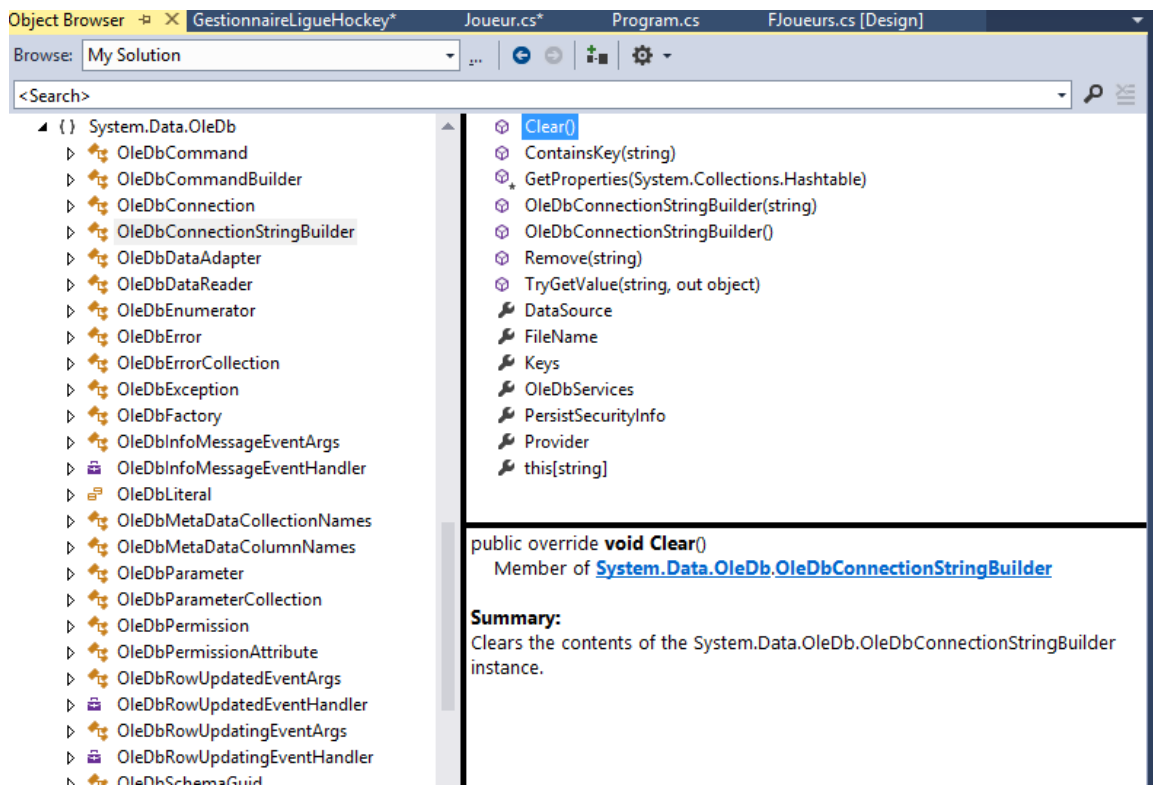
Si vous double-cliquez sur un espace de noms dans les références, un onglet sera créé, mais plutôt qu'une page de code, c'est l'explorateur d'objets qui sera ouvert à l'élément sélectionné :



1.2.5.1 EXPLORATEUR D'OBJETS

L'explorateur d'objets permet de sélectionner et examiner les éléments pouvant être utilisés dans vos projets. Vous pouvez le trouver au menu *Affichage > Explorateur d'objets*. Il est composé de trois (3) onglets :

- Le volet **Objets**, à gauche, présente les objets, de façon très similaire à l’affichage de classes.
- Le volet **Membres**, au coin supérieur droit, énumère les membres de l’élément sélectionné, au même titre que le volet inférieur de l’affichage de classes.
- Le volet **Description**, au coin inférieur droit, constitue la principale différence (outre la taille des interfaces respectives) avec l’affichage de classes et il permet d’avoir une description complète de l’objet (selon des métadonnées définies par le développeur) et ses relations avec les autres objets. Il y a des hyperliens directement vers ces derniers afin de faciliter l’exploration.



En somme, vous pourriez considérer l’affichage de classes comme étant une version miniature et plus complète de l’explorateur d’objets.

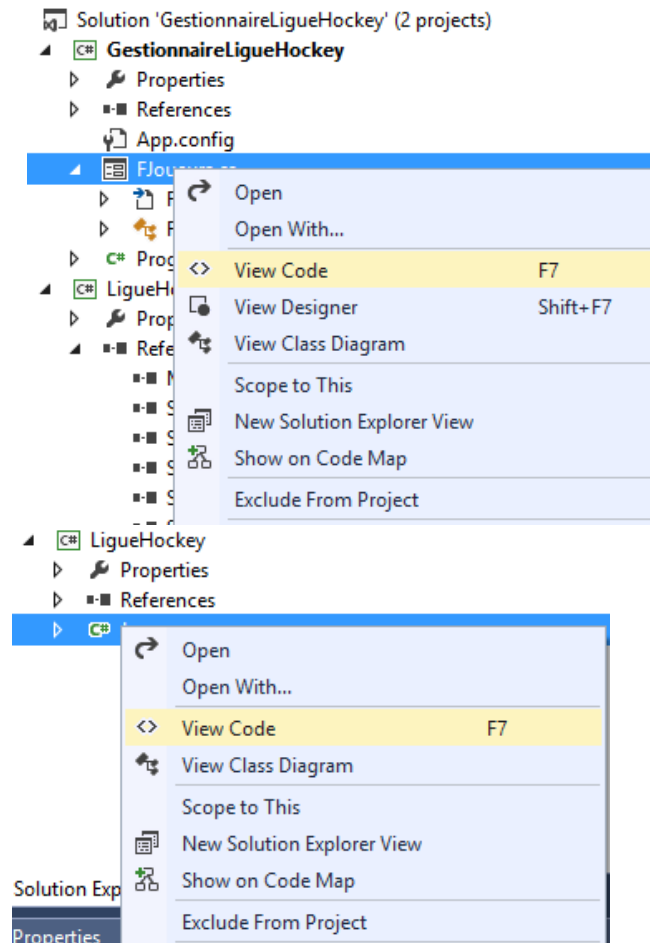
1.2.6 Programmation

La région principale de l’interface de développement intégré (IDE) est normalement réservée au volet pouvant contenir des onglets qui peuvent servir à l’affichage de certaines fenêtres d’options, d’outils (p. ex. l’explorateur d’objets) ou de présentation (p. ex. la page de démarrage), mais surtout pour la programmation des éléments de vos projets. Selon le type de projet dans lequel vous êtes (application Console ou

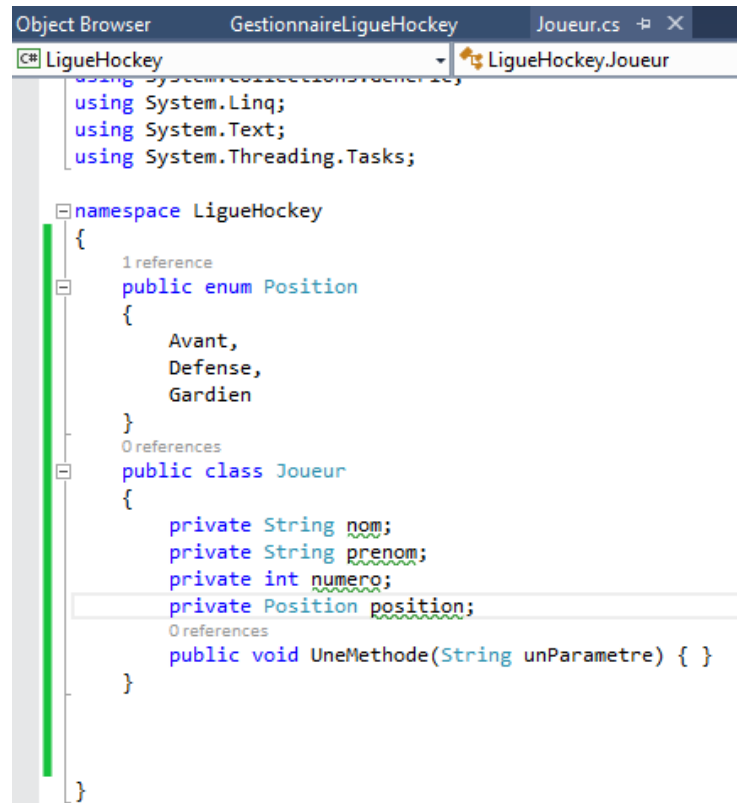
bibliothèque de classes versus application *Windows Forms*, par exemple), vous pourrez basculer entre des modes différents : le mode Code et le mode Design.

1.2.6.1 MODE CODE

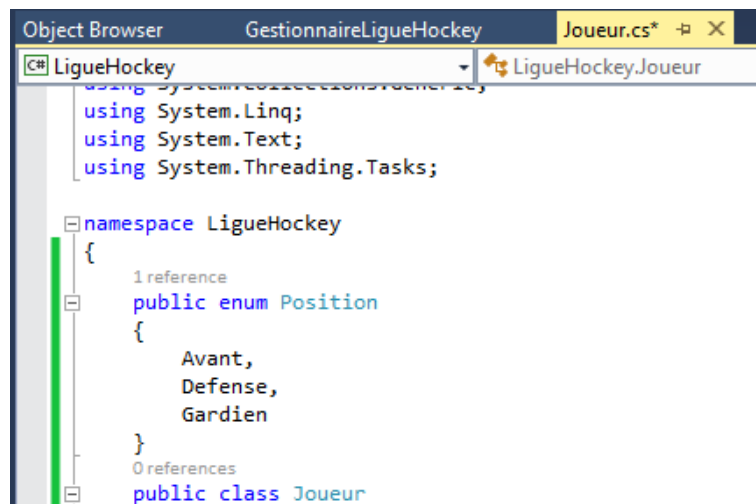
Pour voir le code d'un élément de programmation de votre projet, cliquez sur le bouton droit de la souris sur un fichier `.cs` afin de faire apparaître le menu contextuel et choisissez «*Afficher le code*». La figure de droite montre un menu contextuel légèrement différent dans le cas où le fichier de programmation est un formulaire *Windows Forms*.



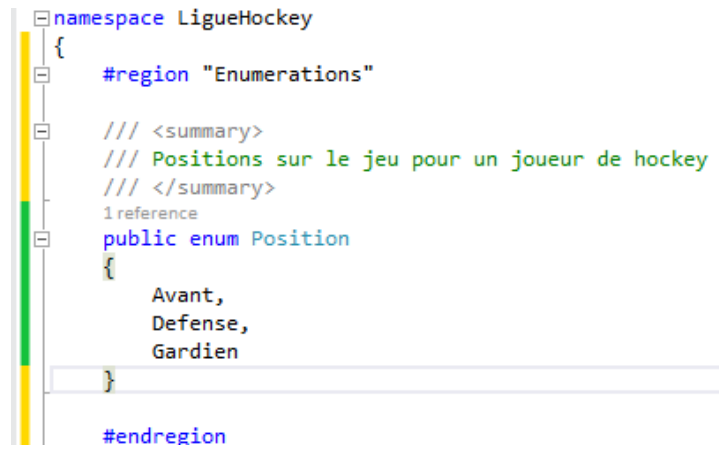
S'il n'y avait pas déjà un onglet pour cet élément, il y en aura un qui sera créé, portant le nom du fichier. Sinon, le focus sera remis sur l'onglet concerné.



Visual Studio vous aide de différentes façons afin de vous donner des repères visuels, de vous rendre à certains endroits précis dans le code et de vous permettre de programmer plus efficacement et rapidement. Remarquez également que s'il y a des modifications non enregistrées dans l'élément, un astérisque (*) apparaît à droite du titre de l'onglet.



Concernant les repères visuels, vous aurez tôt fait de remarquer que le code apparaît en différentes couleurs. À moins que vous n'ayez été changé certaines options d'affichage, voici à quoi correspondent ces couleurs :



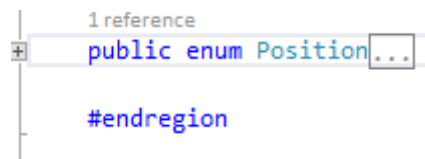
```
namespace LigueHockey
{
    #region "Enumerations"

    /// <summary>
    /// Positions sur le jeu pour un joueur de hockey
    /// </summary>
    1reference
    public enum Position
    {
        Avant,
        Defense,
        Gardien
    }

    #endregion
}
```

- **Vert** : les commentaires.
- **Gris** : les balises de métadonnées (voir la section du chapitre 3 sur les commentaires de documentation).
- **Bleu** : les mots clés du langage.
- **Turquoise** : les noms des propriétés.
- **Rouge** : les littéraux de type chaînes de caractères.
- **Noir** : tout le reste (le nom des membres (classes, méthodes, variables, ...), les opérateurs et les nombres).

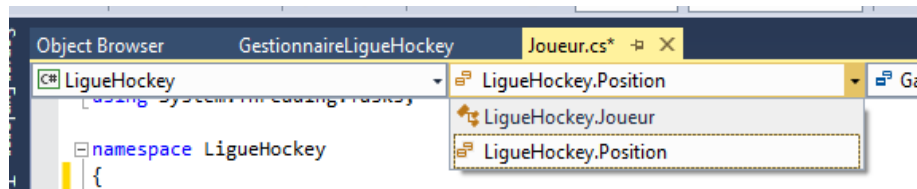
Vous remarquerez également en marge gauche une ligne verticale avec des petits carrés affichant le signe « - » vis-à-vis chaque membre. Si vous cliquez sur ce dernier, vous pourrez cacher le code de ce membre (ainsi que le code de tous les sous-membres, s'il y a lieu) et il ne restera plus que la signature du membre qui sera visible. Le signe sera alors transformé en un « + », ce qui vous donnera la possibilité de réafficher le code que contient le membre.



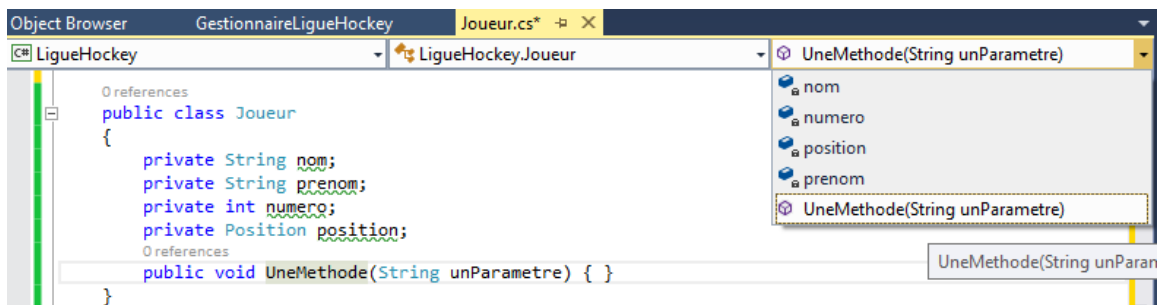
```
1reference
+ public enum Position...
    #endregion
```

Une autre caractéristique appréciable (particulièrement lorsqu'un élément contient beaucoup de lignes de code) est la possibilité de se rendre rapidement à des membres

particuliers. Cette fonction est assurée par des listes déroulantes situées tout au haut de l'onglet. En C#, la première liste déroulante développe sur l'ensemble des structures, classes et énumérations décrites dans le fichier. Le choix d'un élément de la liste amènera le curseur de la souris à l'endroit où l'élément est défini dans le code. Le deuxième impact est que les éléments de la seconde liste d'options (à droite) seront modifiés pour contenir l'ensemble des membres de la classe/structure/énumération activement sélectionnée.

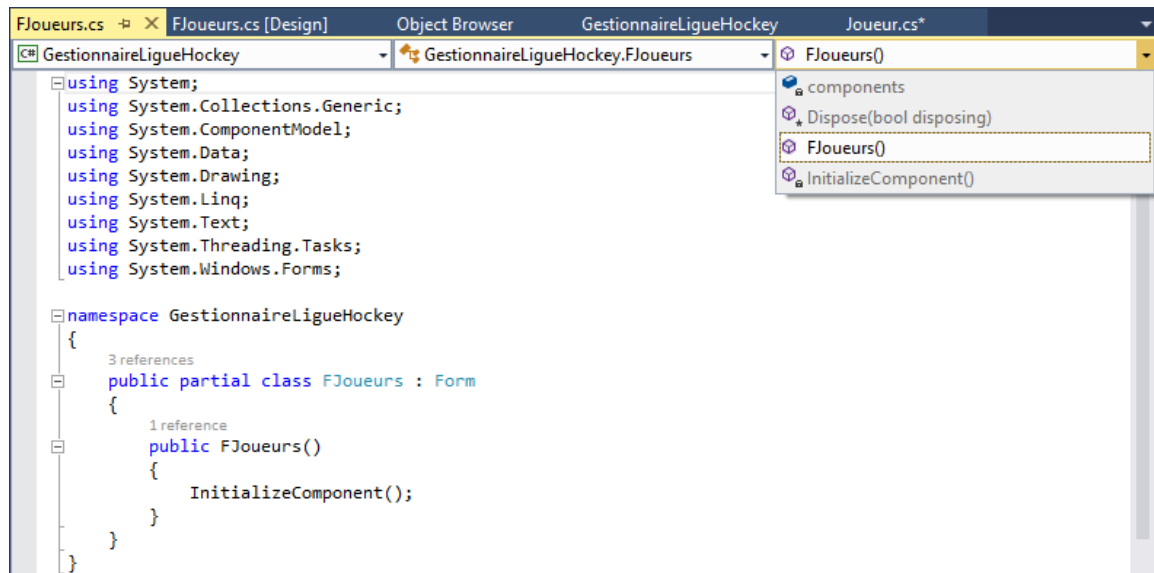


Comme mentionné, la seconde liste déroulante sert à indiquer un point plus précis où se rendre à l'intérieur du membre sélectionné dans la liste de gauche. Vous y retrouverez donc des membres de plus bas niveaux, comme les méthodes et les propriétés, par exemple.



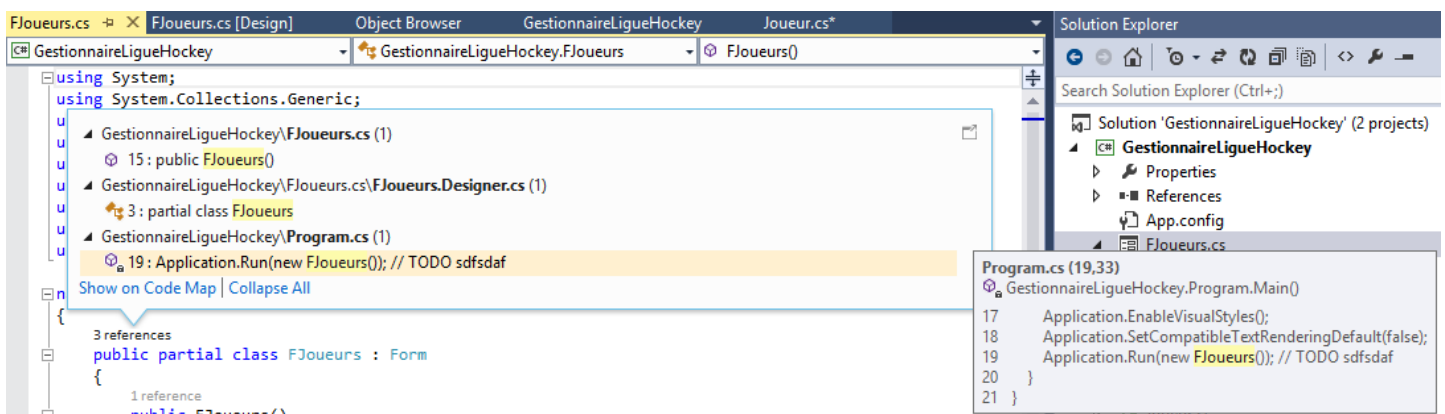
Ainsi, si vous sélectionnez « Joueur » dans la liste de gauche, puis « *UneMethode()* » dans la liste de droite, vous serez amené directement à cette propriété dans le code.

Si vous êtes dans le code d'un formulaire *Windows Forms*, les choix dans les listes déroulantes pourraient différer quelque peu selon le langage ou les configurations que vous utilisez (VB .NET versus C#, par exemple). En C#, ce sera identique aux classes normales, c'est-à-dire que vous aurez accès aux classes et *al.* dans la liste de gauche et aux membres (méthodes et *al.*) dans la liste de droite. :



Vous remarquerez que **tous** les membres possibles pour la classe sélectionnée font partis de la liste déroulante, y compris ceux qui ne sont pas directement définis dans le fichier de code actif (p. ex. `Dispose()` et `InitializeComponent()`) dans la liste déroulante précédente. tout au haut de la liste dans l'exemple précédent). En fait, les membres en gris pâle sont les membres qui appartiennent à la même classe mais définis ailleurs (dans un autre fichier code). La raison pour cela est que nous sommes dans une **classe partielle**. Nous reviendrons sur ce concept dans un futur chapitre.

Enfin, une nouvelle caractéristique a été introduite dans la dernière version de Visual Studio : vous avez peut-être remarqué qu'au-dessus de chaque membre dans le code apparaît une mention « *x references* ». En cliquant sur une des mentions, vous pourrez avoir une liste de toutes les instructions code qui font références au membre en question et accéder à ceux-ci en un clin d'œil.

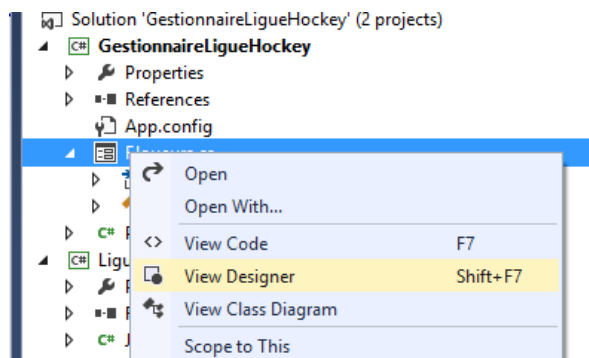


C'est d'autant pratique pour rapidement se rendre à certains membres, à observer le degré de dépendance/de sollicitation d'un membre et aussi de s'assurer qu'un membre est utilisé quelque part dans le code.

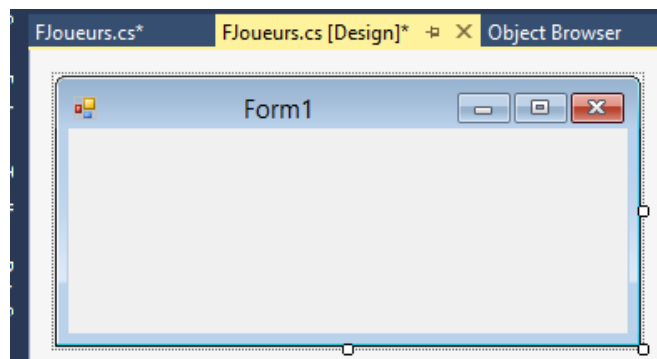
1.2.6.2 MODE DESIGN

Le second mode de programmation pour vos fichiers de codes implémentant les *Windows Forms* est le mode Design. Il permet de visualiser ce dont à quoi aura l'air vos fenêtres à l'exécution, à placer et déplace les différents contrôles qu'elles contiennent, puis à modifier les propriétés de contenus et d'affichage. Ici, nous n'aborderons pas les différents détails concernant les contrôles ni leurs propriétés, que vous devriez manipuler lors du cours INF1003 sur les interfaces utilisateur. Nous nous contenterons de voir les éléments généraux.

Cliquez sur un élément *Windows Forms* du projet avec le bouton droit de la souris, puis, dans le menu contextuel, choisissez « Conception de vues ».



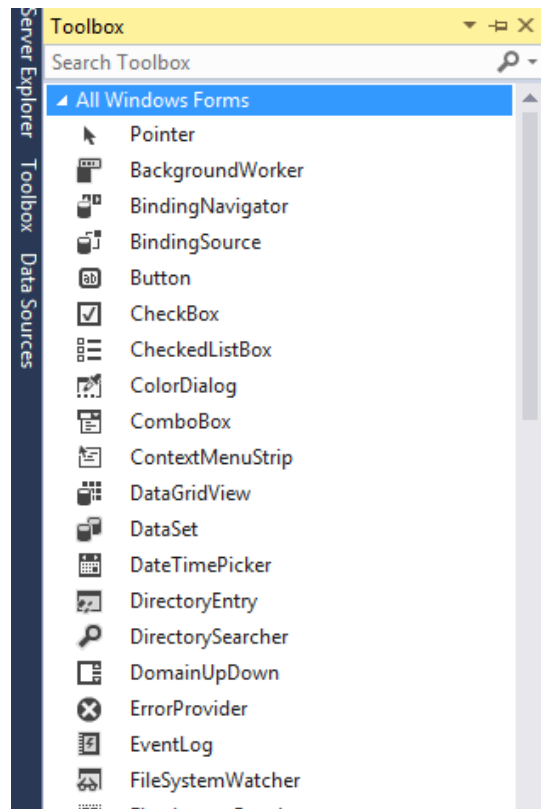
Un onglet s'ouvrira, portant encore une fois le nom du fichier Visual Basic, mais cette fois-ci avec [Design] à la fin. La feuille de l'onglet contiendra une prévisualisation de la fenêtre.



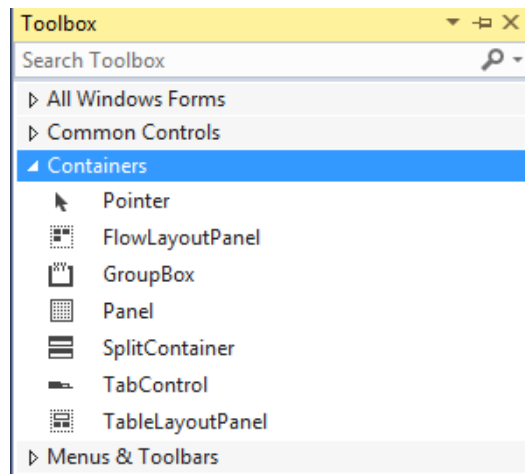
Évidemment, initialement, il n'y aura qu'un formulaire sans aucun contrôle utilisateur à l'intérieur. Ce sera à vous d'en ajouter et d'en modifier les propriétés.

Boîte à outils

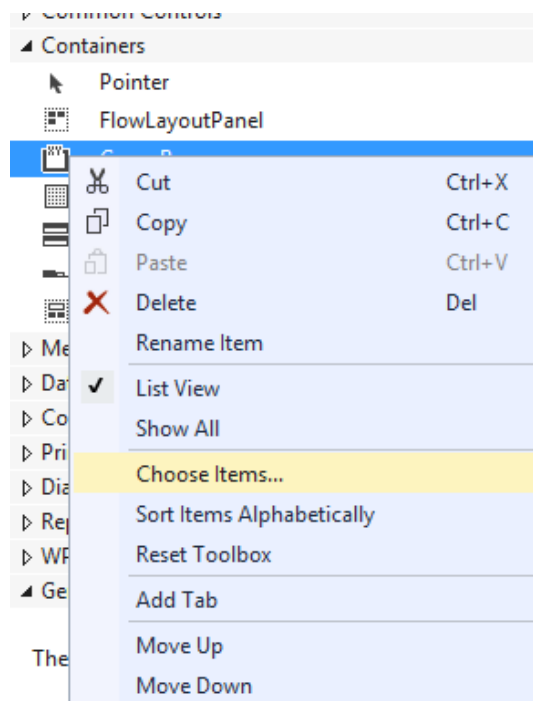
Pour ajouter des contrôles, repérez tout à gauche dans la figure précédente un icône d'outils à côté duquel il est écrit à la verticale « Boîte d'outils » (selon votre configuration, il pourrait ne pas y être : dans ce cas, allez au menu *Affichage > Boîte à outils*). Lorsque vous cliquerez sur ce dernier, une boîte contenant bon nombres d'éléments classés dans différentes catégories de contrôles fera son apparition.



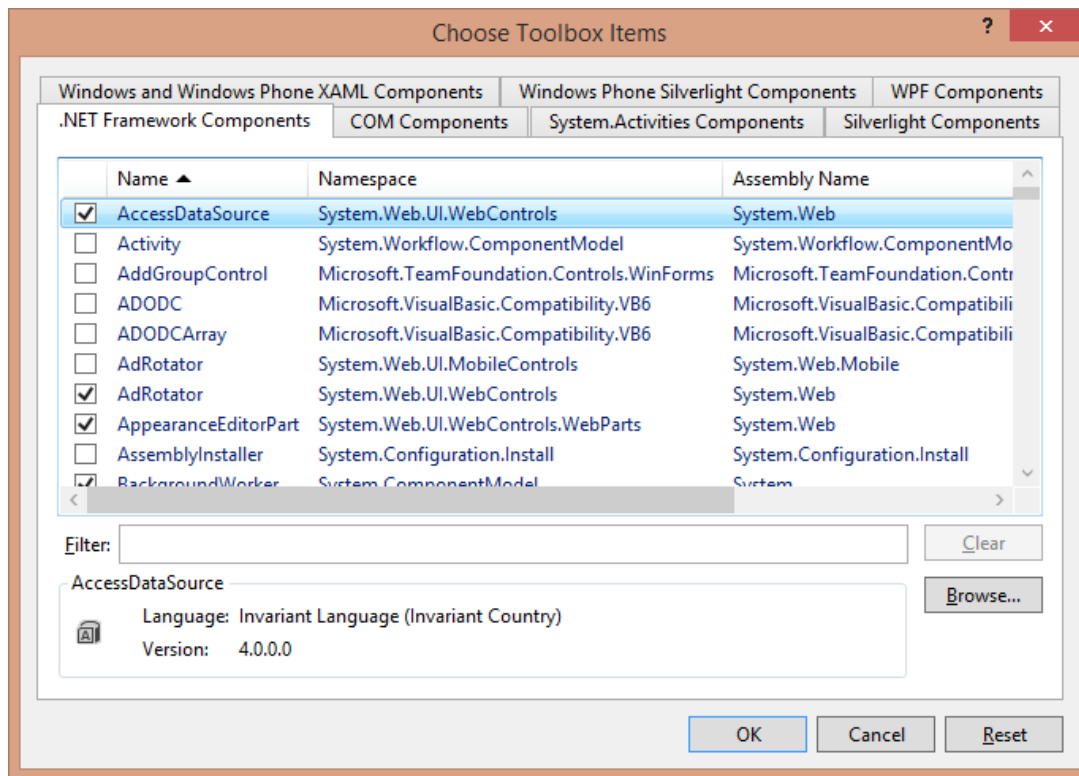
La boîte à outils est fondamentale pour le développement rapide d'une application utilisant des interfaces graphiques utilisateurs, qu'elle soit de type *Windows Forms* ou *Windows Presentation Foundation*. Elle fournit des contrôles qui peuvent être directement glissés vers un formulaire. Chaque item contrôle peut se retrouver dans un ou plusieurs onglets (catégories). Vous pouvez bien sûr développer (montrer) ou réduire (cacher) les contrôles qu'ils contiennent en cliquant sur le + ou le – dans l'étiquette d'entête.



Vous pouvez également éditer les objets contenus dans chaque catégorie. Il suffit de cliquer sur le bouton droit de la souris et choisir les options conséquentes dans le menu contextuel.

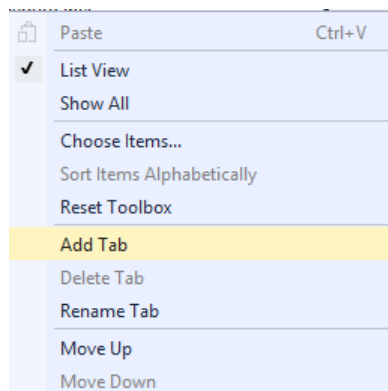


Ainsi, il est possible de trier les contrôles en ordre alphabétique, les changer de positions les uns par rapport aux autres avec « Monter » ou « Descendre » (ou par du tenir et glisser et relâcher), les renommer ou les supprimer. Vous pouvez aussi modifier la liste des contrôles en cliquant sur « *Choisir les éléments...* ». Une boîte de dialogue apparaîtra alors (cela peut prendre un bon moment selon votre machine).

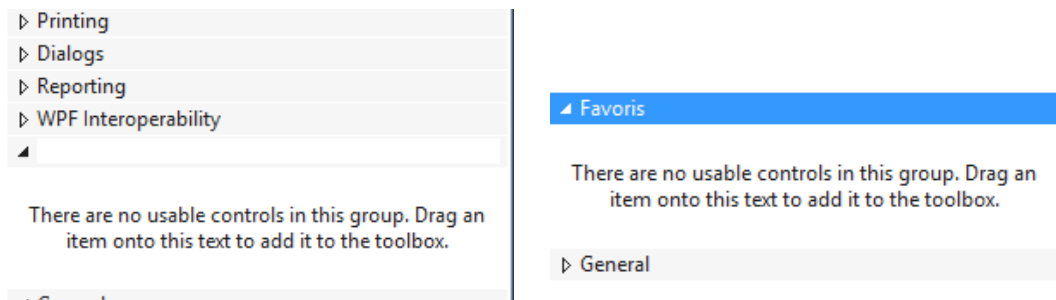


Comme vous le voyez, les contrôles (composants) sont séparés en différents onglets selon s'il s'agit de contrôles provenant du *framework* .NET, des composants COM, WPF ou autres. Les contrôles qui font déjà partis des contrôles de la boîte à outils sont sélectionnés. Il suffit alors de défiler et en choisir d'autres en les cochant ou, à l'inverse, de les enlever de la boîte à outils en les décochant.

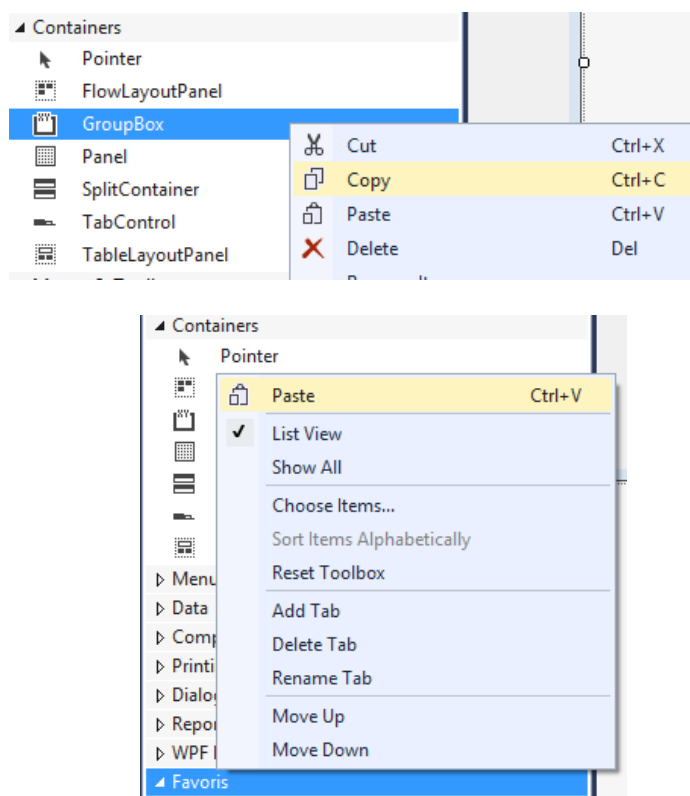
Vous pouvez aussi créer votre propre onglet en choisissant dans le menu contextuel précédent l'option « Ajouter un onglet ».



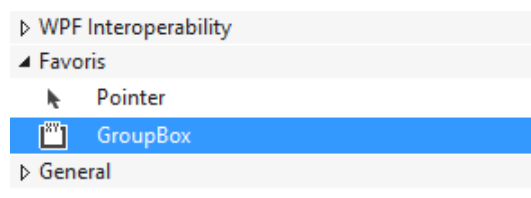
Ce dernier s'ajoutera à la fin des onglets, avant « Général ». Vous pourrez alors lui donner un nom (p. ex. « Favoris »).



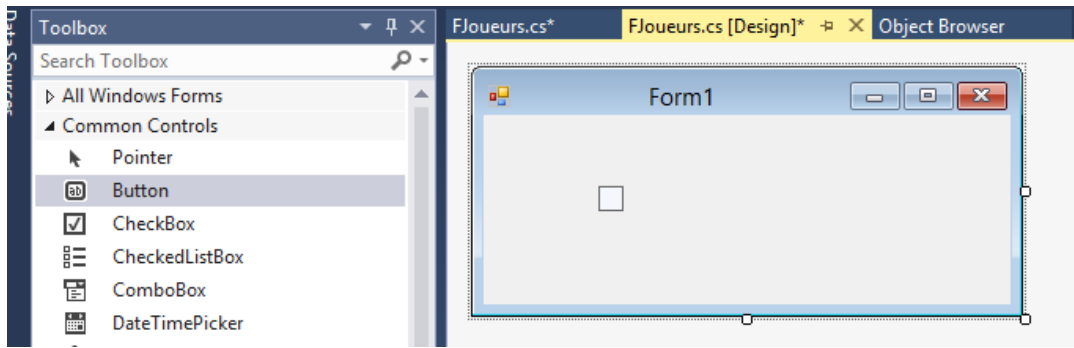
Vous pouvez lui ajouter des contrôles en glissant des contrôles provenant d'autres onglets, en les sélectionnant avec la boîte de dialogue précédemment vue ou en effectuant un copier/coller à partir d'un contrôle d'un autre onglet.



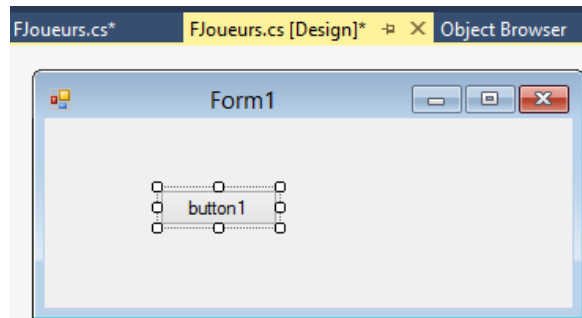
Le nouveau contrôle apparaîtra alors dans l'onglet « Favoris ».



Pour ajouter des contrôles à la fenêtre, il suffit généralement de glisser le contrôle à l'intérieur de cette dernière.

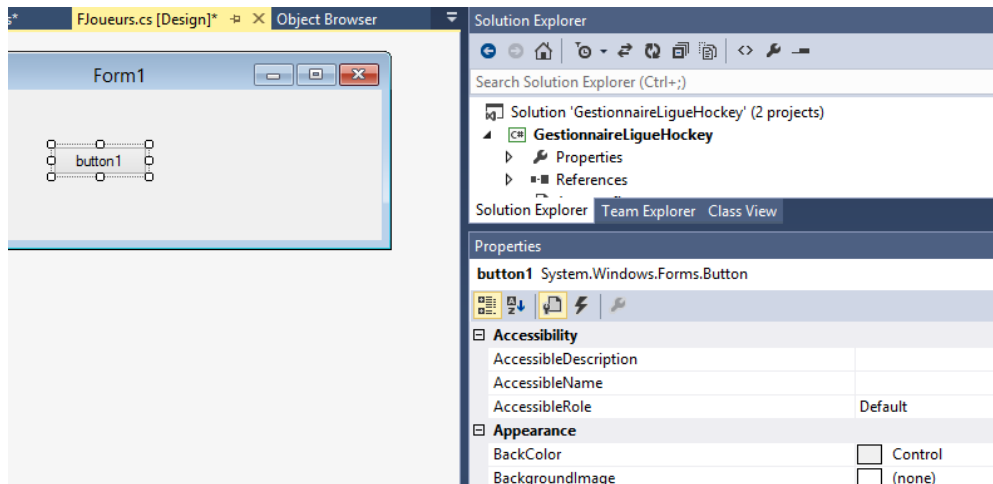


Le contrôle sera alors automatiquement créé et il sera sélectionné.



Fenêtre Propriétés

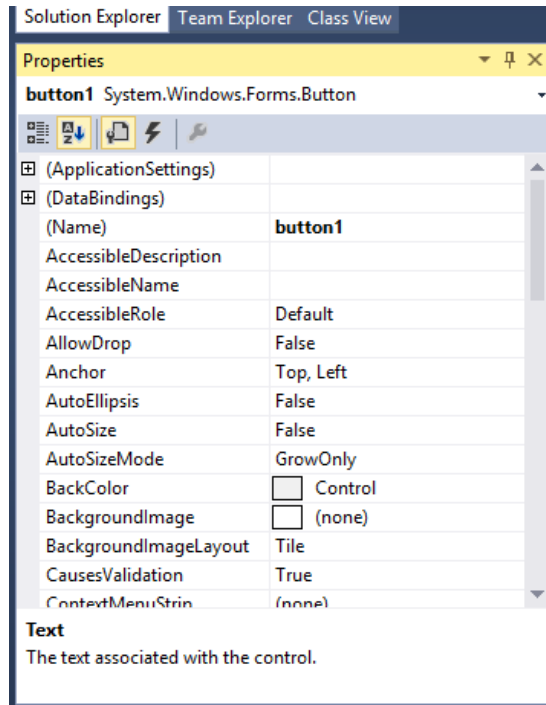
Pour éditer le contrôle, il faudra vous servir de la fenêtre **Propriétés**. Celle-ci apparaît normalement à droite dans Visual Studio (mais comme toutes les boîtes et fenêtres, elle est déplaçable selon vos goûts). Vous pouvez la faire apparaître en allant au menu *Affichage > Fenêtre Propriétés*, ou de façon très rapide en appuyant sur **F4**.



Nous ne donnerons pas le détail des propriétés des contrôles, bien évidemment. Cependant, vous devriez déjà savoir que chaque contrôle possède un ensemble de propriétés différentes, dont certaines sont communes, selon les contrôles sur lesquels ils

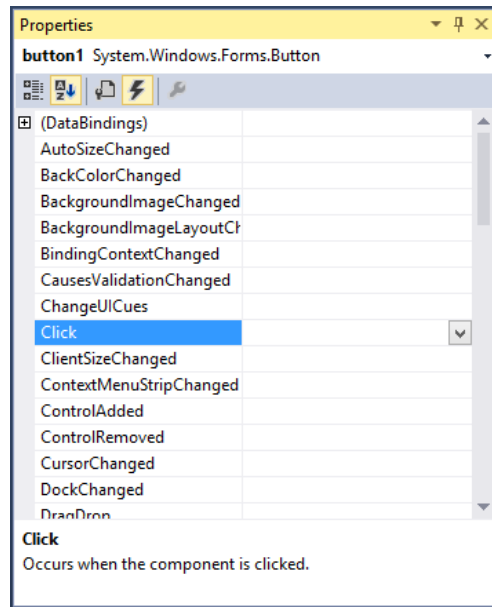
sont basés. Ces propriétés sont, par défaut, triées par catégories de propriétés (p. ex. pour le contrôle de type bouton : accessibilité, apparence, comportement, données, design, focus, ...).

Les propriétés peuvent aussi être triées en ordre alphabétique en cliquant sur l'icône avec les lettres A et Z, tel qu'illustré ci-contre :



Dans ce cas, les catégories de propriétés n'apparaissent plus. Remarquez également que la valeur d'une propriété est définie dans la cellule adjacente à celle-ci. Pour certaines propriétés, comme le nom du contrôle ou le texte à afficher à l'écran, vous pourrez taper les valeurs au clavier. Pour d'autres propriétés, comme les couleurs de texte ou de fond, vous aurez un choix à sélectionner parmi une liste déroulante.

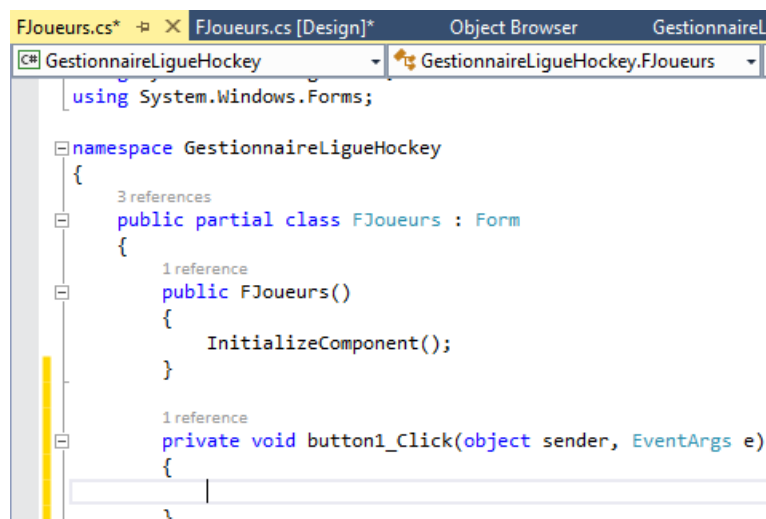
En cliquant sur le bouton représenté par un éclair (⚡), vous pourrez afficher les événements associables au contrôle.



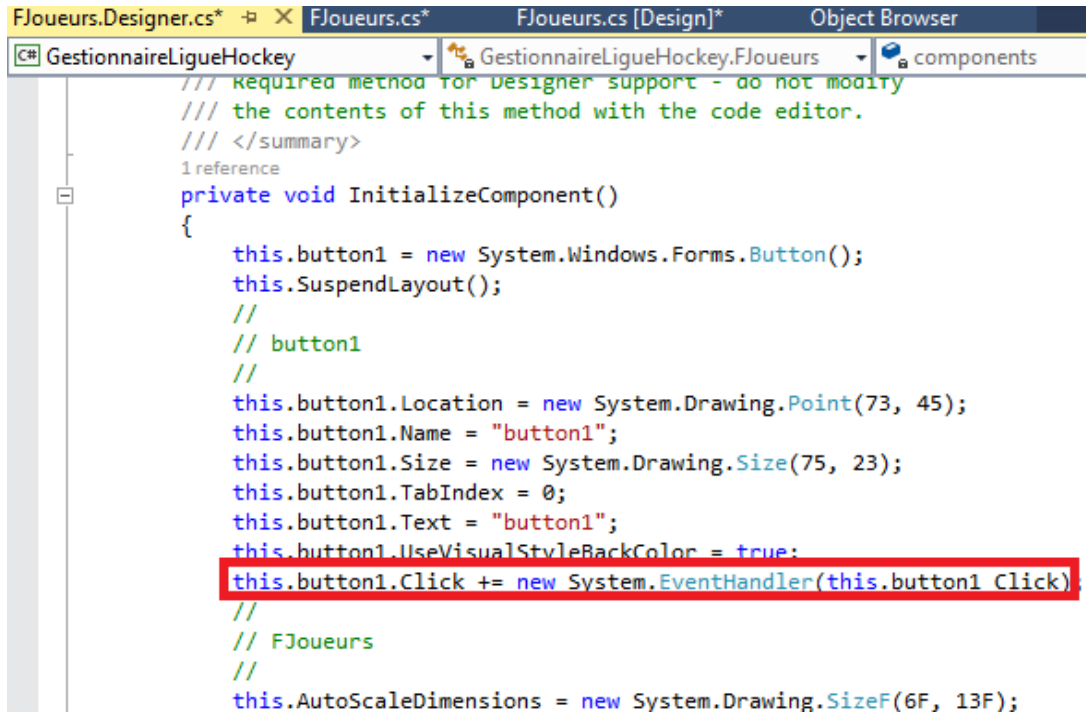
Les **événements** représentent des instructions dont l'exécution peut être déclenchée par des actions de l'utilisateur directes (clic ou double-clic sur un bouton, sélectionner un élément dans une liste déroulante) ou indirectes (chargement d'un formulaire, changement d'index dans une liste déroulante). *Nous en reparlerons!*

Visual Studio vous facilite grandement la tâche en créant et gérant à votre place les événements prédéfinis pour les contrôles de vos interfaces.

Dans la figure précédente, vous remarquerez qu'il n'y a aucune valeur pour les événements du bouton. Pour ajouter un événement, il suffit de double-cliquer sur la cellule vide vis-à-vis celui que nous désirons créer. Le code reliant l'action à une méthode particulière sera alors généré et vous serez placé à l'intérieur de celle-ci, prêt à entrer les instructions à exécuter au cas où l'événement se produit.

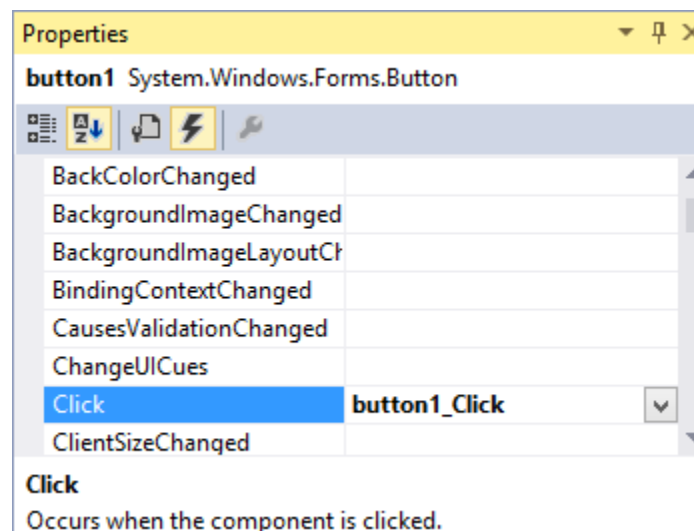


Dans le fichier *FJoueur.Designer.cs* (où la classe de base est définie), du code est également généré automatiquement pour lier la méthode en tant qu'événement pour le bouton :

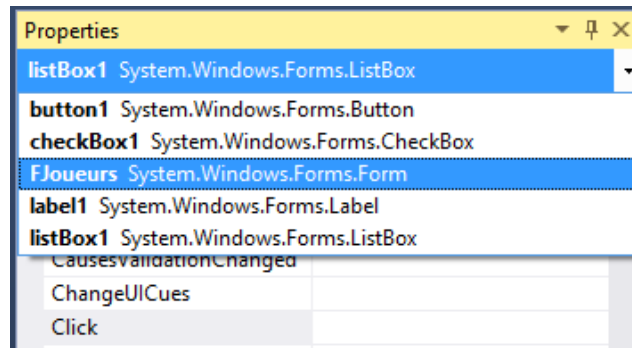


```
1 reference
private void InitializeComponent()
{
    this.button1 = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(73, 45);
    this.button1.Name = "button1";
    this.button1.Size = new System.Drawing.Size(75, 23);
    this.button1.TabIndex = 0;
    this.button1.Text = "button1";
    this.button1.UseVisualStyleBackColor = true;
    this.button1.Click += new System.EventHandler(this.button1_Click);
    //
    // FJoueurs
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
}
```

Si vous retournez voir en mode Design dans la fenêtre *Propriétés* et dans les événements, vous devriez voir le nom de la méthode appelée dans le cas d'une action donnée.



Enfin, vous avez la possibilité de passer d'un contrôle à un autre en utilisant la liste déroulante tout en haut de la fenêtre *Propriétés*. Vos contrôles y seront triés en ordre alphabétique selon leur nom.



1.2.7 Débogage

Cette section présente un condensé sur le débogage d'une application. Aussi, nous ne présenterons que les mécanismes de base, soit la fenêtre de liste d'erreurs, le démarrage en mode Débogage, les points d'arrêts et les pas à pas, puis Pour de plus amples détails sur tous les mécanismes, n'hésitez pas à consulter MSDN.

1.2.7.1 ERREURS ET AVERTISSEMENTS

Avant même de pouvoir exécuter votre application, vous devez vous assurer que celle-ci compile. Visual Studio vous indique à même le code lorsque vous avez des erreurs syntaxiques et de logique (pour ces dernières, celles qui sont détectables) de programmation ou encore les avertissements, i.e. du code qui à priori n'empêchera pas la compilation de l'application, mais qui pourrait potentiellement provoquer la levée d'exceptions (i.e. plantage du programme) lors de l'exécution :

- Les erreurs de syntaxe sont soulignées en rouge (normalement) :
- Les avertissements sont soulignés en vert (normalement):

```
1 reference
private void btnCreer_Click(object sender, EventArgs e)
{
    int x = "Bonjour";
    string y = x + 2;
}

0 references
public String FaireX()
{
    String uneString;
}
```

En laissant le curseur de la souris vis-à-vis l'erreur ou l'avertissement, un étiquette d'aide apparaîtra afin de vous en indiquer la nature.

```

1 reference
private void btnCreer_Click(object sender, EventArgs e)
{
    int x = "Bonjour";
    string y
}

0 references
public String FaireX()
{
    String uneString;
}

0 references
public String FaireX()
{
    String uneString;
}

```

class System.String
Represents text as a series of Unicode characters.

Error:
Cannot implicitly convert type 'string' to 'int'

The variable 'uneString' is declared but never used

Il est possible que la présence d'erreurs fasse en sorte que les avertissements ne soient pas indiqués. De toute façon, mieux vaut s'occuper en priorités des erreurs.

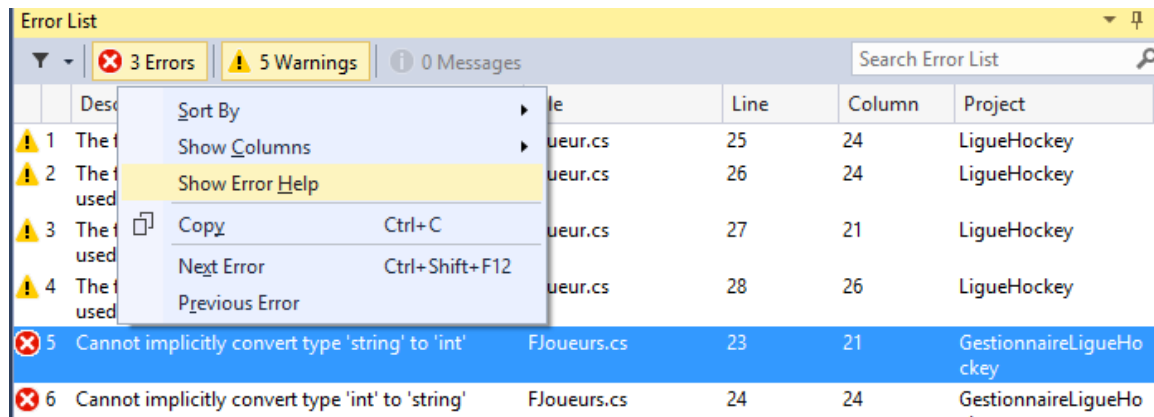
Les erreurs et les avertissements sont également listés dans une fenêtre normalement située en bas dans Visual Studio. Si elle n'apparaît pas, allez au menu *Affichage > Liste d'erreurs* (ou Ctrl + E).

Error List					
<div> <div>3 Errors</div> <div>5 Warnings</div> <div>0 Messages</div> </div> <div>Search Error List</div>					
	Description	File	Line	Column	Project
1	The field 'LigueHockey.Joueur.nom' is never used	Joueur.cs	25	24	LigueHockey
2	The field 'LigueHockey.Joueur.prenom' is never used	Joueur.cs	26	24	LigueHockey
3	The field 'LigueHockey.Joueur.numero' is never used	Joueur.cs	27	21	LigueHockey
4	The field 'LigueHockey.Joueur.position' is never used	Joueur.cs	28	26	LigueHockey
5	Cannot implicitly convert type 'string' to 'int'	FJoueurs.cs	23	21	GestionnaireLigueHockey
6	Cannot implicitly convert type 'int' to 'string'	FJoueurs.cs	24	24	GestionnaireLigueHockey
7	The variable 'uneString' is declared but never used	FJoueurs.cs	29	20	GestionnaireLigueHockey

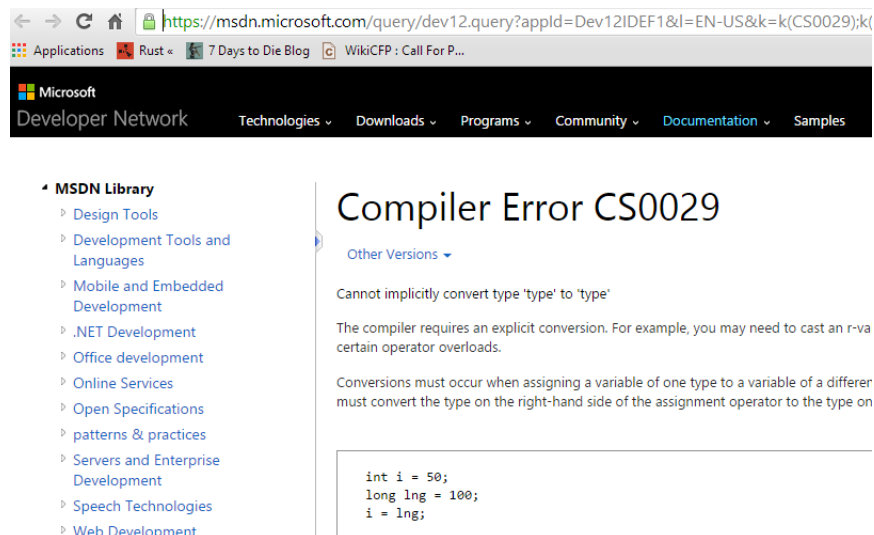
Les erreurs et les avertissements sont classés à part. Vous n'avez qu'à cliquer sur l'un ou l'autre des boutons afin d'en afficher la liste. Vous trouverez alors une description de l'erreur, dans quel fichier et projet elle est située et à quel endroit exactement (ligne et colonne). Vous pouvez trier en ordre croissant ou décroissant pour n'importe quelle des informations en cliquant sur l'entête de la colonne correspondant.

Par ailleurs, en double-cliquant sur l'élément, vous serez transporté à l'endroit dans le fichier en question.

Aussi, si vous cliquez sur le bouton droit de la souris vis-à-vis une erreur, vous pourrez choisir « Afficher de l'aide sur l'erreur ».

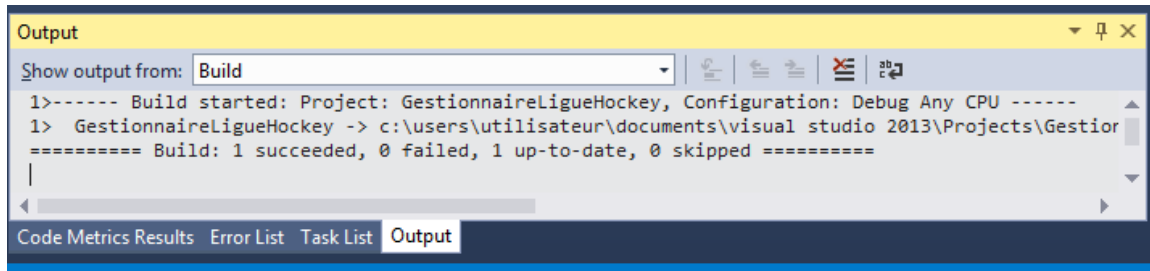


Selon vos options de configuration d'aide (et si MSDN est installé localement sur votre machine), cela vous amènera vers une page d'aide en ligne ou locale sur l'erreur en question.



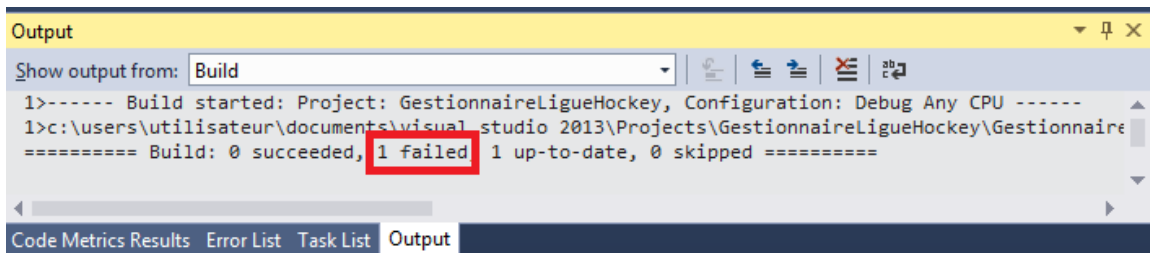
1.2.7.2 GÉNÉRATION ET EXÉCUTION

Pour compiler le code et générer le ou les *assemblies*, il ne doit plus y avoir d'erreurs (par contre, les avertissements sont acceptés, à moins d'avoir configuré la solution autrement). Pour ce faire, allez au menu *Générer* > *Générer la solution* (**Ctrl + Shift + B**). Dans la fenêtre de sortie (si elle n'apparaît pas, rendez-vous au menu *Affichage* > *Sortie* (**Ctrl + Alt + O**)). Si tout se passe bien, vous devriez avoir les sorties suivantes :



« *1 a réussi, 0 a échoué, 1 est à jour...* » (l'important est de n'avoir aucun échec) vous indique qu'un projet a été généré avec succès.

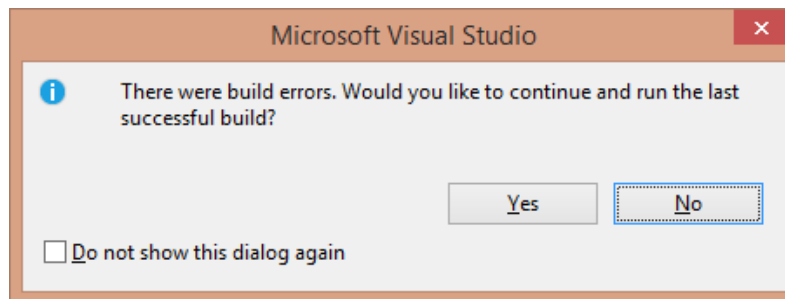
Par contre, si vous avez ceci :



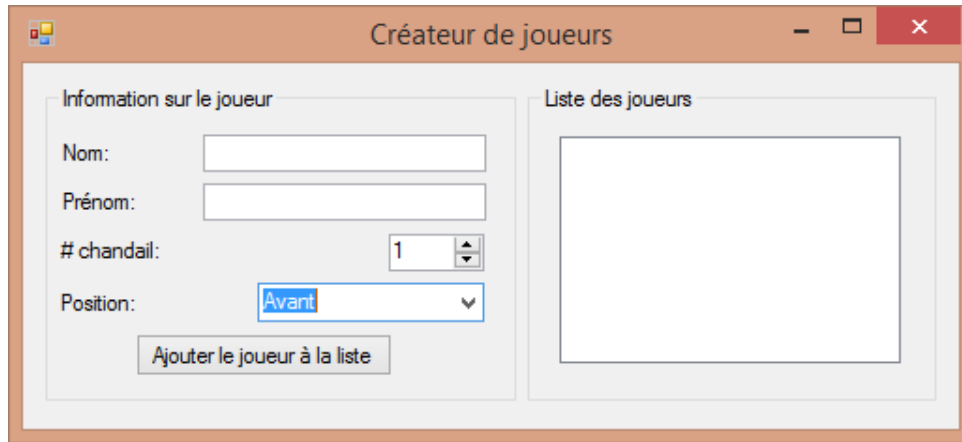
Cela est signe que vous avez encore des erreurs à corriger et que le fichier binaire n'a pas pu être compilé.

Une fois votre projet généré, vous pouvez exécuter votre application en mode débogage en appuyant sur **F5** ou en allant au menu *Déboguer* > *Démarrer le débogage*. Normalement, dans la configuration par défaut, si jamais le projet a subi des modifications depuis la dernière compilation, une régénération aura lieu.

En cas d'échec de génération, vous verrez apparaître la boîte de dialogue suivante :



Elle vous indique que vous ne pouvez exécuter l'application selon votre projet actuel, mais qu'il est tout de même possible de procéder avec la dernière génération qui a réussi. Si la génération a fonctionné, ou si vous cliquez sur le bouton « *Oui* » dans la boîte de dialogue précédente, l'application s'exécutera. Dans le cadre d'une application *Windows Forms*, vous devriez voir apparaître une fenêtre un peu comme ceci :



Remarquez dans la barre d'outils de Visual Studio que certains boutons sont maintenant disponibles (activés).

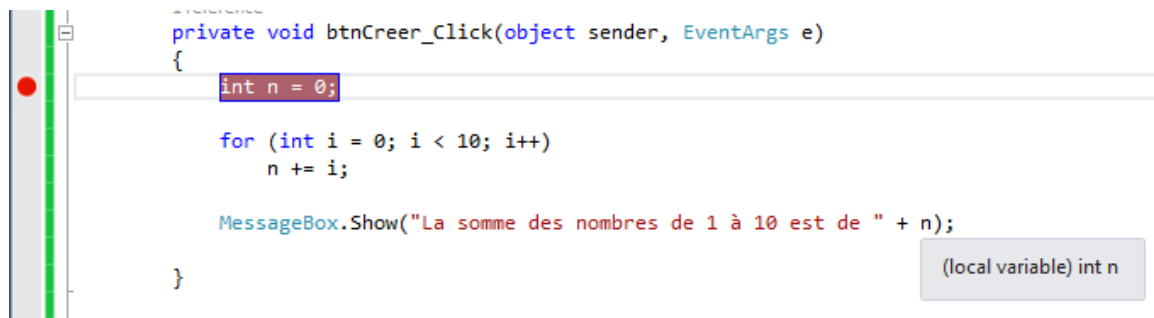


Ces boutons vous permettent, dans l'ordre, de mettre l'application sur pause (après quoi, ce dernier sera désactivé et c'est le bouton « Jouer » qui pourra être cliqué afin de poursuivre l'exécution de l'application), de forcer l'arrêt de la session de débogage (et ainsi fermer l'application) et, enfin, de redémarrer l'application.

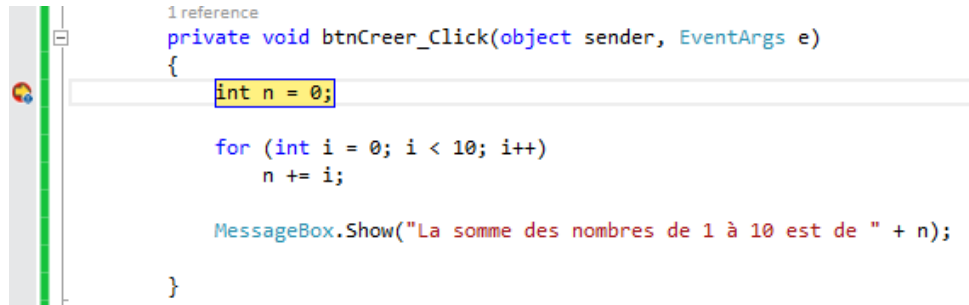
1.2.7.3 POINTS D'ARRÊT

En tant que programmeur, vous voudrez souvent être aux premières loges et observer ce qui se passe plus précisément lors de l'exécution de l'application. Entre autres, vous pouvez observer et suivre l'ordre d'exécution des instructions et les valeurs des variables.

Plus encore, vous pouvez ajouter à des endroits précis (lignes) dans le code des points d'arrêts. Pour cela, il suffit de cliquer dans la marge vis-à-vis l'instruction souhaitée. Un bouton rouge devrait apparaître (pour l'enlever, il suffit de re cliquer sur celui-ci).



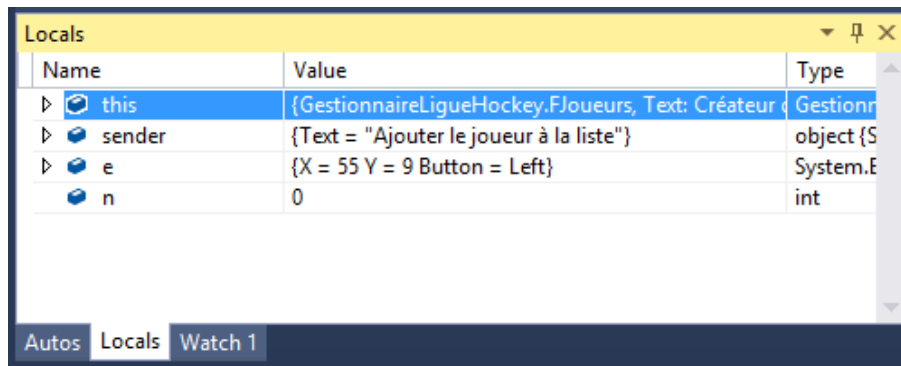
Une fois un point d'arrêt atteint, l'application sera mise sur pause et vous serez dirigé directement à ce point dans le fichier concerné.



La prochaine instruction à être exécutée (et non la dernière instruction exécutée) apparaîtra avec un soulignage jaune et un curseur en forme de flèche en marge. En l'occurrence, l'instruction marquée par le point d'arrêt sera la première à être exécutée.

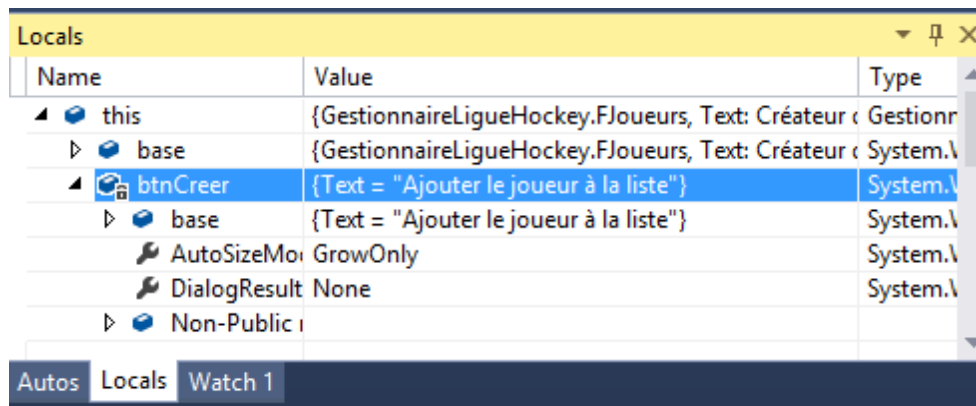
1.2.7.4 VARIABLES LOCALES

L'application étant en pause, vous devriez être en mesure de pouvoir observer les valeurs des structures et des variables dans une fenêtre en bas de l'écran (normalement, il s'agit d'une fenêtre qui remplace celle de la liste d'erreurs). Si elle n'y est pas, vous pouvez la récupérer en vous rendant au menu *Déboguer > Fenêtres > Locales*.



Dans cette fenêtre, vous pouvez voir la liste des objets/variables ainsi que leur valeur et type dans les colonnes suivantes.

Vous pouvez également développer les sous-membres des objets ou des structures, par exemple. Il suffit de cliquer sur le signe « + ».



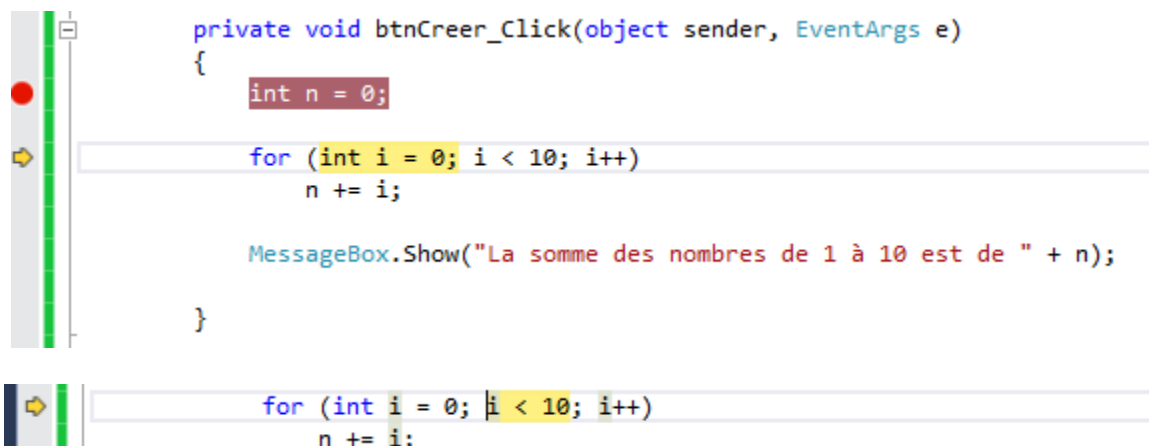
1.2.7.5 PAS À PAS

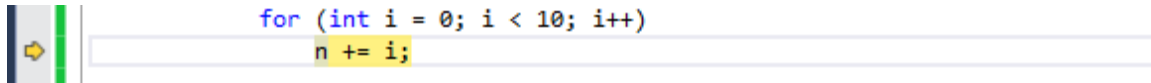
Une fois un point d'arrêt atteint, le but sera de suivre le cours d'exécution de l'application. Pour cela, vous devrez vous servir de certaines fonctionnalités qui apparaissent dans la barre d'outils à côté de « Pauser » et « Jouer ».



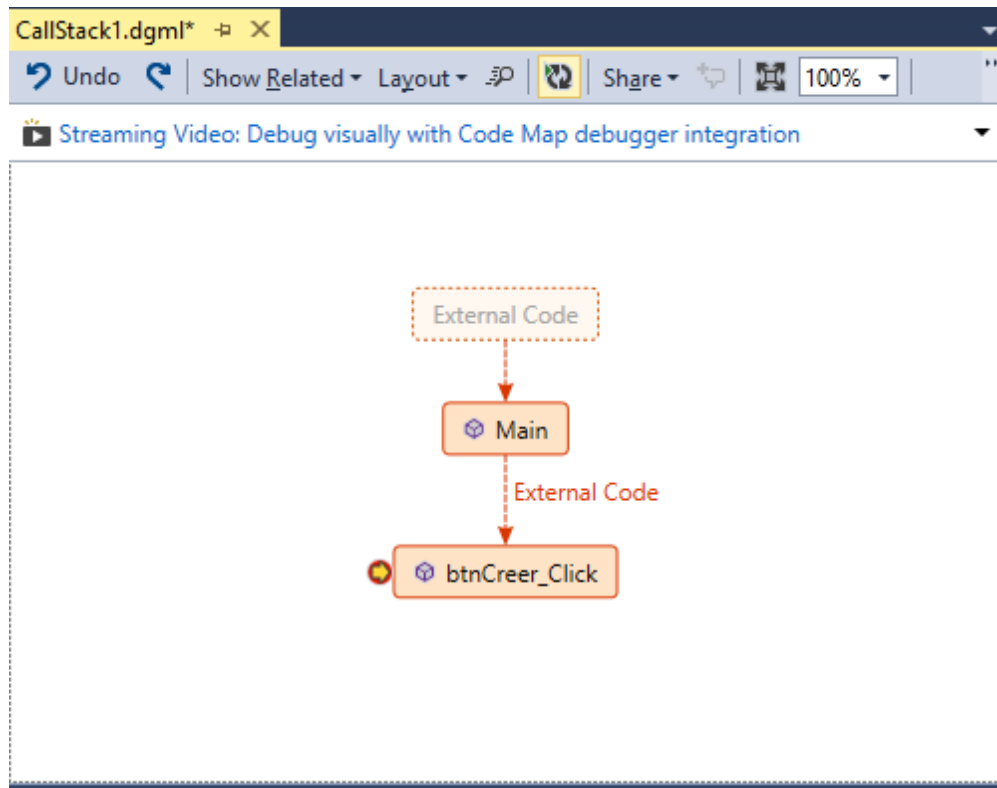
Sans prendre en considération la flèche de gauche, les trois autres boutons sont, de gauche à droite, les suivants :

- **Pas à pas détaillé (F11)** : exécute l'instruction soulignée en jaune et pointée par le curseur. Si l'instruction comporte un ou plusieurs appels à des méthodes, le curseur entre dans la méthode et le pas à pas s'y poursuit. Si l'instruction à exécuter est la fin d'une méthode (}), la prochaine instruction du pas à pas est celle de la méthode appelante ou, si c'est la dernière instruction du flot d'exécution, le programme se termine. Dans le cas d'événements dans les applications *Windows Forms*, une fois la méthode contenant les instructions de l'action terminée, l'exécution normale de l'application se produit.





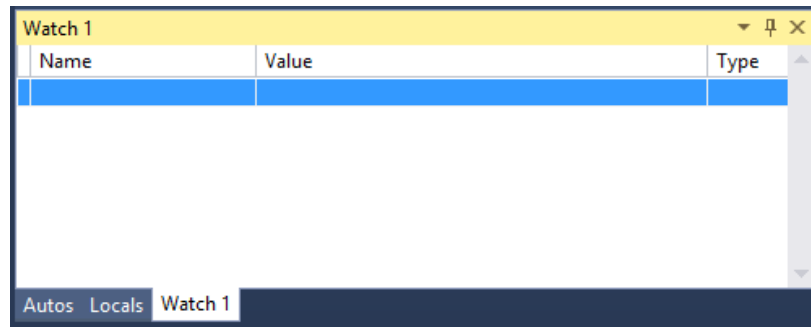
- **Pas à pas principal** (F10) :
 - Si l'instruction contient une ou plusieurs méthodes à appeler, le pas à pas les exécute sans s'y arrêter à l'intérieur.
 - Si l'instruction ne possède pas d'appel, le comportement est le même que pour un pas à pas détaillé.
- **Pas à pas sortant** (Shift + F11) : exécute le reste de la méthode courante et retourne à la méthode appelante. Si vous avez d'autres points d'arrêt définis, au prochain point, vous retournerez au pas à pas.
- **Carte de code** : nouvelle option qui permet de visualiser où se situe le point d'arrêt de façon schématique par rapport aux différents appels de méthodes/membres (autrement la pile d'appels) jusqu'à l'atteinte du point d'arrêt :



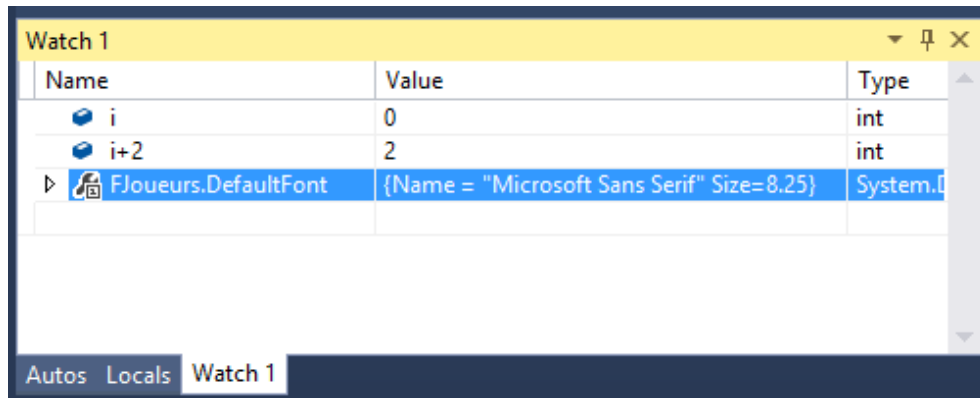
Vous pouvez donc suivre les instructions une à une ou sauter ou quitter certains appels à des méthodes tout en observant les (l'évolution des) valeurs des variables.

1.2.7.6 ESPIONNAGE

Enfin, au-delà de l'observation de l'ensemble des variables du contexte d'exécution dans lequel vous vous trouvez, vous pouvez faire de l'espionnage de variables ou d'expressions avec les espions. L'onglet apparaît normalement à la suite de « *Variables locales* ». Sinon, lorsque vous avez atteint un point d'arrêt, retrouvez-la en allant au menu *Déboguer > Fenêtres > Espion > Espion 1 à 4*. La fenêtre suivante apparaîtra :



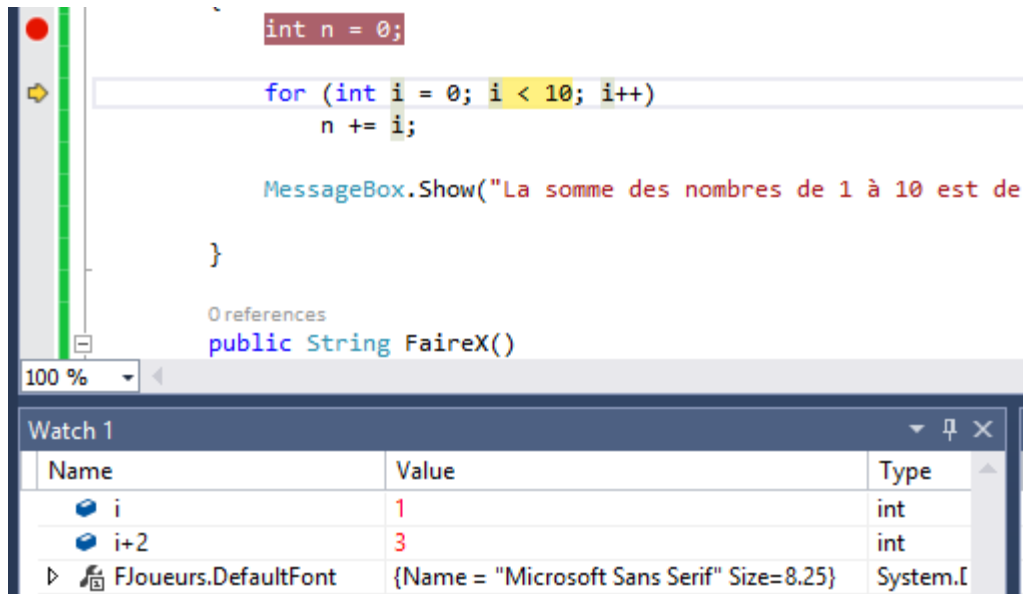
Il suffit alors de taper au clavier sur une ligne à la colonne « Nom » le nom d'une variable à surveiller ou d'une expression quelconque, comme ceci :



L'IntelliSense est intégrée aussi lorsque vient le temps de définir vos espions.

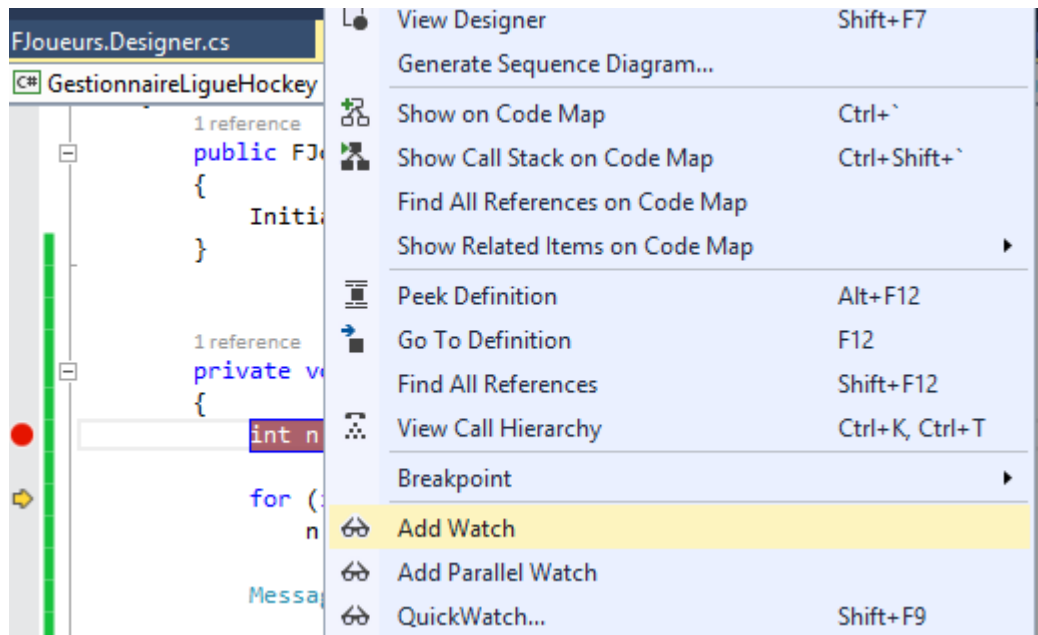
Si le dernier pas d'exécution a modifié des valeurs des variables locales ou des espions, elles s'afficheront en rouge.

Vous pouvez également ajouter des variables ou des membres de la fenêtre *Variables locales* dans les fenêtres *Espion*. Il suffit de cliquer sur le bouton droit de la souris vis-à-vis un élément quelconque et de choisir l'option « Ajouter un espion ».



L'espionnage est donc une façon fortement pratique pour suivre seulement certains éléments et même calculer des valeurs via des expressions qui les utilisent, sans être encombré par l'ensemble d'entre eux, comme c'est le cas dans la fenêtre *Variables locales*.

Enfin, vous avez accès aux fonctions précédentes directement en passant par le menu contextuel (par un clic droit vis-à-vis le membre ou la variable dans le code) :

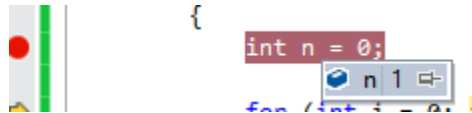


Watch 1		
Name	Value	Type
i	2	int
i+2	4	int
FJoueurs.DefaultFont	{Name = "Microsoft Sans Serif" Size=8.25}	System
n	1	int

1.2.7.7 INFORMATIONS EXPRESS

Enfin, vous pouvez également observer certaines informations « express » directement dans le code pendant les pas à pas, sans avoir recours à l'utilisation des variables locales ou aux espions.

Vous pouvez placer le curseur de la souris vis-à-vis le nom d'une variable et sa valeur devrait apparaître sous la forme d'un commentaire (*tooltip*) :



S'il s'agit d'un objet (ou une instance de n'importe quel type complexe), vous pourrez développer dans le commentaire les différents membres et observer les valeurs.