

WPF

Introduction aux événements

INTRODUCTION

Définitions

On peut distinguer deux types de traitements :

- Le traitement par lot ou traitement différé ou batch (l'utilisateur lance le traitement et peut s'en aller).
- Le traitement conversationnel (sur grands systèmes on dit transactionnel) qui consiste en une suite d'échanges entre le programme et l'opérateur, à travers différents écrans (sous WINDOWS on dit fenêtre ou feuille).

WINDOWS est plutôt adapté à ce deuxième type de traitement et C# approprié à développer de telles applications.

Mieux, on peut dire que la conception en C# colle aux caractéristiques et modèles propres à WINDOWS.

Toute application WINDOWS est réductible à un ensemble de feuilles, au départ blanches, vides et sans limites.

Chaque feuille va ensuite être spécialisée et devenir une feuille de saisie, un document, une boîte de dialogue etc. *comportant un certain nombre d'objets* : zones de données, propositions de choix d'options ou d'actions.

Ces objets s'appellent *contrôles* (zones de texte, boutons, cases à cocher, options, listes déroulantes, barres d'outils etc.).

Les *actions de l'opérateur* (déplacement du curseur, de la souris, saisie ou modification ...) sur chacun de ces contrôles font partie du dialogue, correspondent à un *événement* et nécessitent le plus souvent un traitement instantané.

On parle de programmation *événementielle*.

A chaque événement correspond une séquence du programme, un sous-programme (une fonction), effectuant un traitement en fonction du contexte.

Un programme réalisé sous C# est donc morcelé en une multitude de sous-programmes, chacun réalisant le traitement d'un événement.

Les objets graphiques

On distingue :

Les feuilles (Form) ou fenêtres :

- Feuille principale
- Feuilles filles (une par document)
- Feuilles ou boîtes de dialogue.

Les contrôles :

- Etiquettes ou labels, pour dénommer des données
- Zones de texte, utiles pour contenir des données
- Boutons de commande
- Boutons d'option dits boutons radio
- Cases à cocher
- Conteneurs pour contenir d'autres contrôles
- Listes simples, déroulantes ou combinées (combobox)
- Barres de défilement

Tous les contrôles de l'espace de noms **System.Windows.Controls** héritent de la classe **System.Windows.FrameworkElement**; c'est la raison pour laquelle ils possèdent un grand nombre de caractéristiques communes.

Les propriétés

Chaque objet possède des **propriétés** qui définissent son identification, sa position, ses couleurs, son état, sa valeur, les possibilités de le modifier, ses liens avec d'autres objets, etc.

Quelques propriétés sont semblables pour la majorité des contrôles (par exemple : propriétés de forme, positionnements, couleurs, polices de caractères). Celles-ci sont le plus souvent fixées dès la conception et fréquemment de manière implicite.

Voici quelques exemples de propriétés :

| Affichage | Position |
|------------|----------|
| Visibility | Height |
| Opacity | Width |

D'autres propriétés sont le plus souvent utilisées. Nous allons en faire quelques commentaires.

Name : C'est le nom de la variable ou structure associée à l'objet. C'est avec ce nom que vous désignerez un contrôle pendant le traitement.

Text: C'est le contenu d'un champ texte ou d'un label.

isVisible : Certains contrôles peuvent très bien ne pas apparaître sur la feuille. Par exemple, un contrôle Timer qui n'a aucun intérêt visuel, ou un contrôle qui n'a momentanément pas de sens dans le contexte.

Cette propriété vaut True ou False.

isEnabled : Un contrôle, tout en étant visible peut ne pas être utilisable. Par exemple un bouton "Ajout" ou "Suppression" alors qu'il n'y a aucun élément à ajouter ou supprimer.

Dans ce cas, le contrôle est inaccessible par l'opérateur et apparaît en grisé.

Cette propriété vaut True ou False.

TabIndex : Définit l'ordre de passage d'un contrôle à un autre (à condition que sa propriété TabStop soit à true) avec les touches de tabulation.

isTabStop : Indique si le bouton peut recevoir le focus.

Les méthodes

Un objet possède également des **méthodes (des fonctions)**, séquences de traitement pré codées permettant de réaliser des actions

Certaines méthodes sont communes à tous les contrôles, comme la méthode **Focus** qui permettra de positionner le focus sur un contrôle particulier, comme la méthode **Clear** de la TextBox qui efface tout le texte du contrôle zone de texte.

Les évènements

Un objet réagit à certains **évènements** en exécutant une fonction (si on l'a programmée) dont le nom par défaut comporte le nom de l'objet et celui de l'évènement.

Comme avec les propriétés, on retrouve souvent les mêmes possibilités d'évènements pour les contrôles. :

C'est la base de la programmation événementielle.

Commentons les évènements les plus usités pour une zone de texte :

GotFocus :

Se produit lorsque le contrôle prend le *focus*.

TextChanged :

Se produit lorsque la valeur de la propriété Text a été modifiée.

Un seul caractère suffit. Permet de déclencher des contrôles très réactifs.

Beaucoup de traitements gagneront plutôt à être faits aux évènements Leave (On quitte le champ).

KeyDown, KeyUp

Ces évènements sont utiles à traiter particulièrement lorsque l'on veut gérer finement la saisie au clavier caractère par caractère.

KeyDown et KeyUp, utilisés pour toute touche, servent plus particulièrement à traiter les touches spéciales, touches de fonction ou combinaison avec *Ctrl*, *Maj* (ou Shift) et *Alt*.

Ces évènements sont exploitables à tout moment dans une feuille pour filtrer les raccourcis claviers (ou shortcuts), actions de l'opérateur alternatives à l'appui sur un bouton.

MouseDown, MouseUp, MouseMove, MouseEnter, MouseLeave:

On peut contrôler finement ce que fait la souris :

MouseDown : Un bouton de la souris est enfoncé ;

MouseUp : Un bouton de la souris est relâché ;

MouseMove : La souris se déplace au dessus du contrôle

MouseEnter : La souris entre dans la zone du contrôle

MouseLeave : La souris quitte la zone du contrôle

LostFocus:

Se produit lorsque le contrôle cesse d'être le contrôle actif du formulaire.

Déclenche les traitements lorsque l'on quitte un contrôle : après une saisie ou modification.

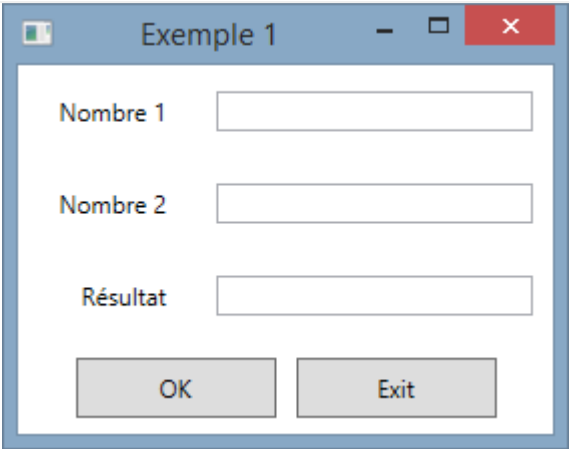
Click:

Utilisé pour les Boutons de commande et les boutons d'option et CheckBox

Drop, DragEnter, DragLeave, DragOver

Tous ces évènements sont relatifs à la technique de cliquer-glisser (Drag and Drop)

Un événement peut en cacher un autre

| | |
|---|---|
|  | <p>Le curseur étant dans la zone de saisie "Nombre1", nous pouvons aller dans Nombre2 de plusieurs manières :</p> <p>Click dans Nombre2 Tabulation ou ↓</p> <p>Nous devons donc envisager de traiter tous les cas possibles de passage de Nombre1 à Nombre2, en étant sûr de <i>traiter une fois et une seule fois</i> l'action de l'opérateur.</p> |
|---|---|

Du côté de Nombre1 nous devons envisager l'occurrence possible des événements :

LostFocus quand on quitte le champ,
ou KeyUp pour filtrer l'appui sur TAB

Pour Nombre2 :

GotFocus On arrive dans le champ (mais d'où vient-on?)
MouseEnter On y arrive avec la souris

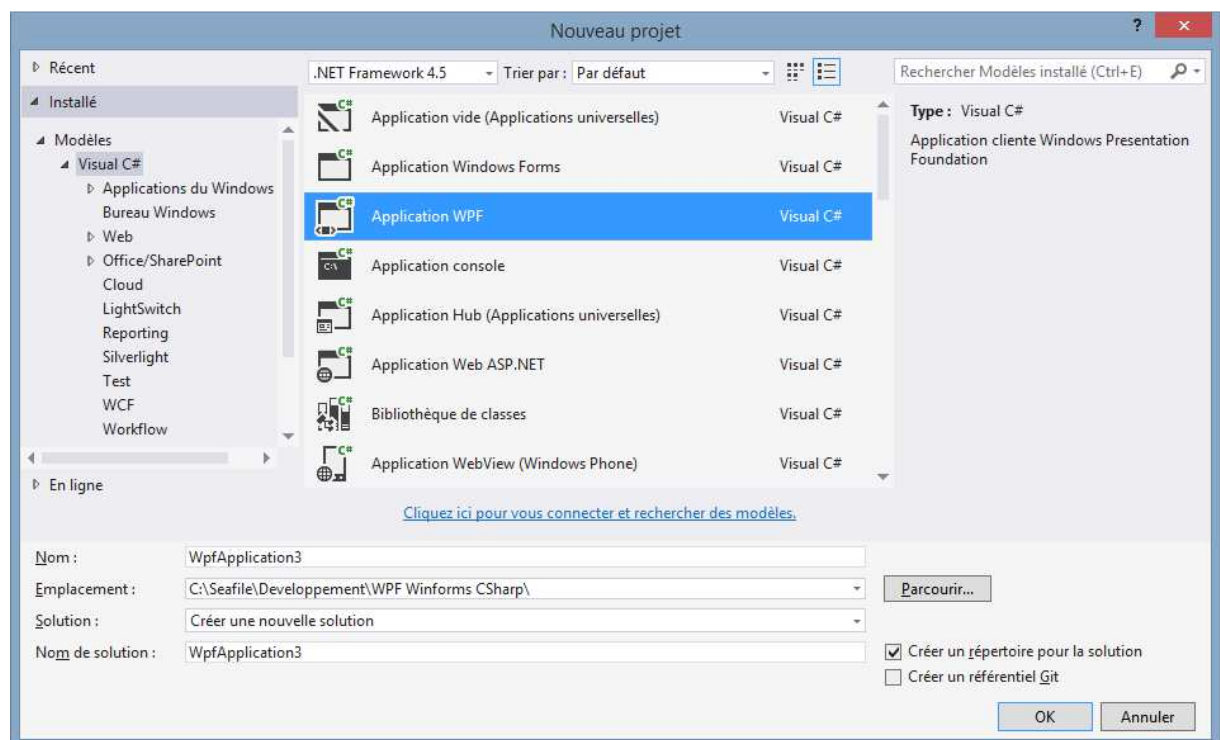
Suivant le traitement souhaité, celui-ci ne devant s'effectuer qu'une fois, il faudra choisir un événement approprié, unique et non ambigu.

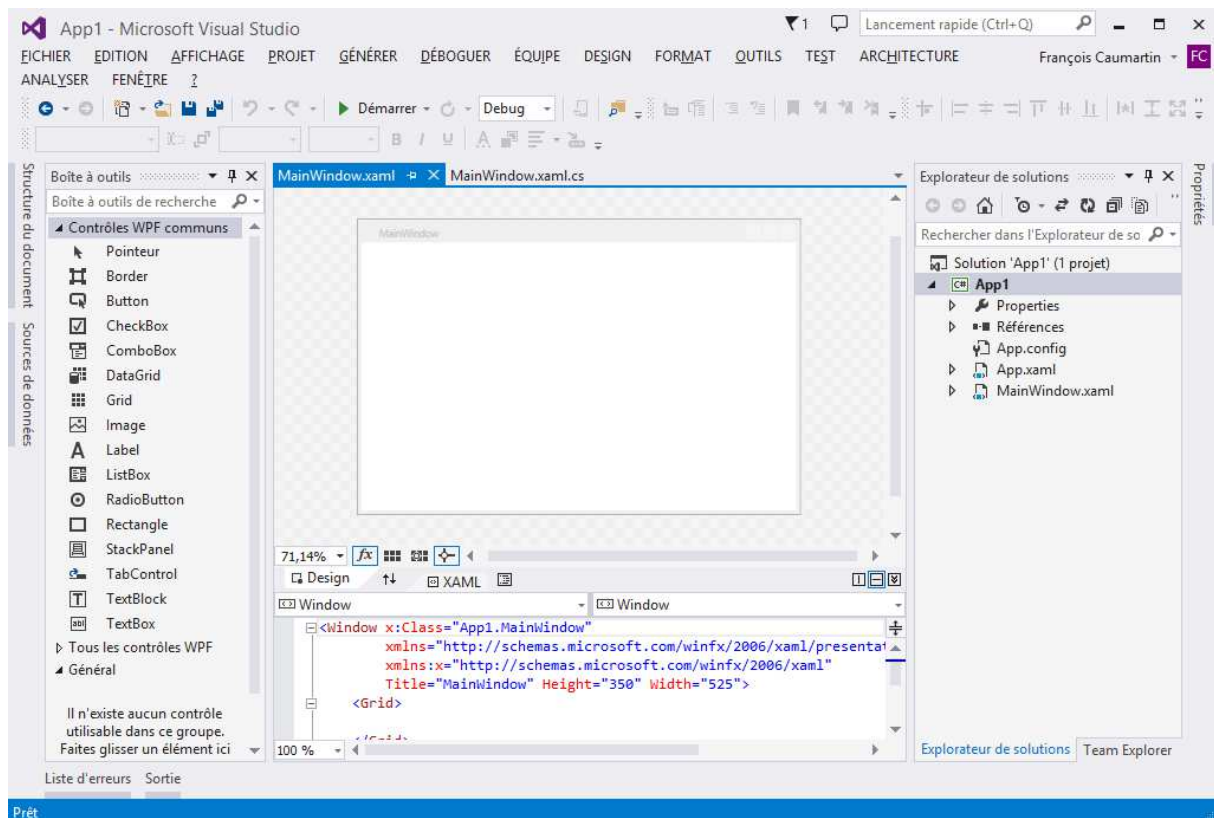
CREATION D'UNE APPLICATION WINDOWS

Mode d'emploi

Un projet est créé sous l'environnement Visual Studio qui se charge de créer les répertoires.

- a. créer un nouveau projet par le menu fichier, en choisissant le type "**Visual C#** " et le modèle "**Applications WPF**", et le nommer (ici WpfApplication1) App1
Choisir comme emplacement un répertoire de travail.





b. modifier le fichier frmTest.cs

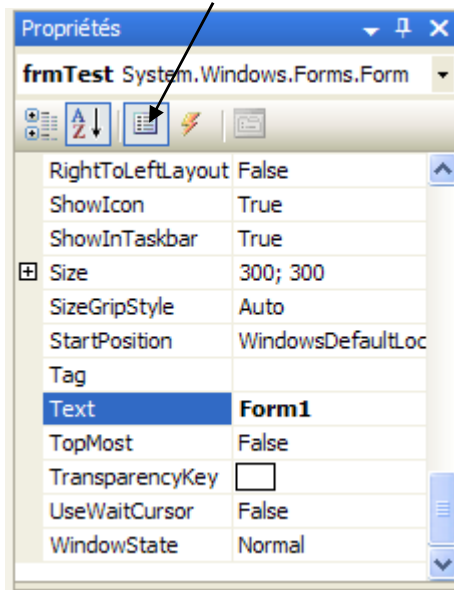
La propriété **Name** de la feuille, qui sert à adresser la feuille dans le code aura été modifiée à frmTest ; donner un titre à la feuille en modifiant la propriété **Text** dans la fenêtre des propriétés

Cette action sera à répéter systématiquement pour chaque feuille traitée dans une application.

Cette fenêtre comporte la liste des propriétés de l'objet sélectionné (ou de la feuille si aucune sélection n'est faite sur un contrôle).

S'il n'est pas intéressant d'agir à ce stade sur les propriétés de taille ou position du contrôle, on y définit en revanche son nom, sa valeur et son état initial et, peut-être, quelques autres valeurs ...

Classement par thèmes ou
Classement alphabétique
.... des propriétés ou des événements



La fenêtre des propriétés n'est renseignée que si la feuille est en mode design.

c. Dans le répertoire C:\InitC#\WinForms, on constate qu'un répertoire physique *App1* et différents fichiers sont créés:

- les fichiers qui décrivent le projet App1 (*.sln*, *.csproj*). , une solution pouvant contenir plusieurs projets
- le fichier qui contiendra le code C# (*frmTest.cs*) modifiable par le développeur
- et celui contenant le code C# généré par le concepteur Visual Studio (*frmTest.Designer.cs*)
- le fichier XML (*.resx*) où sont stockées les ressources -données, images,...- locales au formulaire frmTest.

Ces ressources peuvent être modifiées selon des critères de "culture" qui particularisent par exemple la langue, sous Visual Studio sans entraîner de recompilation de la page

Toute nouvelle feuille créée par la suite donnera toujours lieu à ces deux mêmes types de fichiers (ouvrir par l'icône "Afficher tous les fichiers" de l'explorateur de projet)

La fenêtre de code modifiable par le développeur peut être atteinte :

- en cliquant sur l'icône Afficher le code dans l'explorateur de solution
- en cliquant droit puis Afficher le code sur le fichier frmTest.cs dans l'explorateur de solution

La fenêtre de code générée par Visual Studio peut être atteinte par le menu contextuel du fichier Form1.Designer.cs puis **Afficher le code** dans l'explorateur de solution

On pourra choisir un élément de code particulier dans la liste déroulante de droite.


Les icônes affichées représentent chacun un élément différent, quelquefois précédé d'un icône de signalisation, indiquant leur accessibilité :

Quelques exemples ...

| <i>Icônes</i> | <i>Description</i> |
|---------------|---------------------|
| | Espace de Nom |
| | Classe |
| | Méthode ou fonction |
| | Propriétés |

| <i>Icônes</i> | <i>Description</i> |
|---------------|--------------------|
| | Champ ou variable |
| | Protégé |
| | Privé |
| | Publique |

- d. Sauvegarder et **surtout penser à générer le projet par le menu "générer"** - faute de quoi les fichiers **.cs** de codes sources des classes du projet ne sont pas compilés. Vérifier dans la fenêtre de sortie en bas qu'il n'y a pas d'erreurs de compilation.

- e. Tester l'application créée, soit en la lançant depuis l'EDI¹ (bouton  de la barre d'outils) soit depuis l'explorateur Windows par le fichier exécutable App1.exe qui a été généré sous le répertoire bin

Remarques:

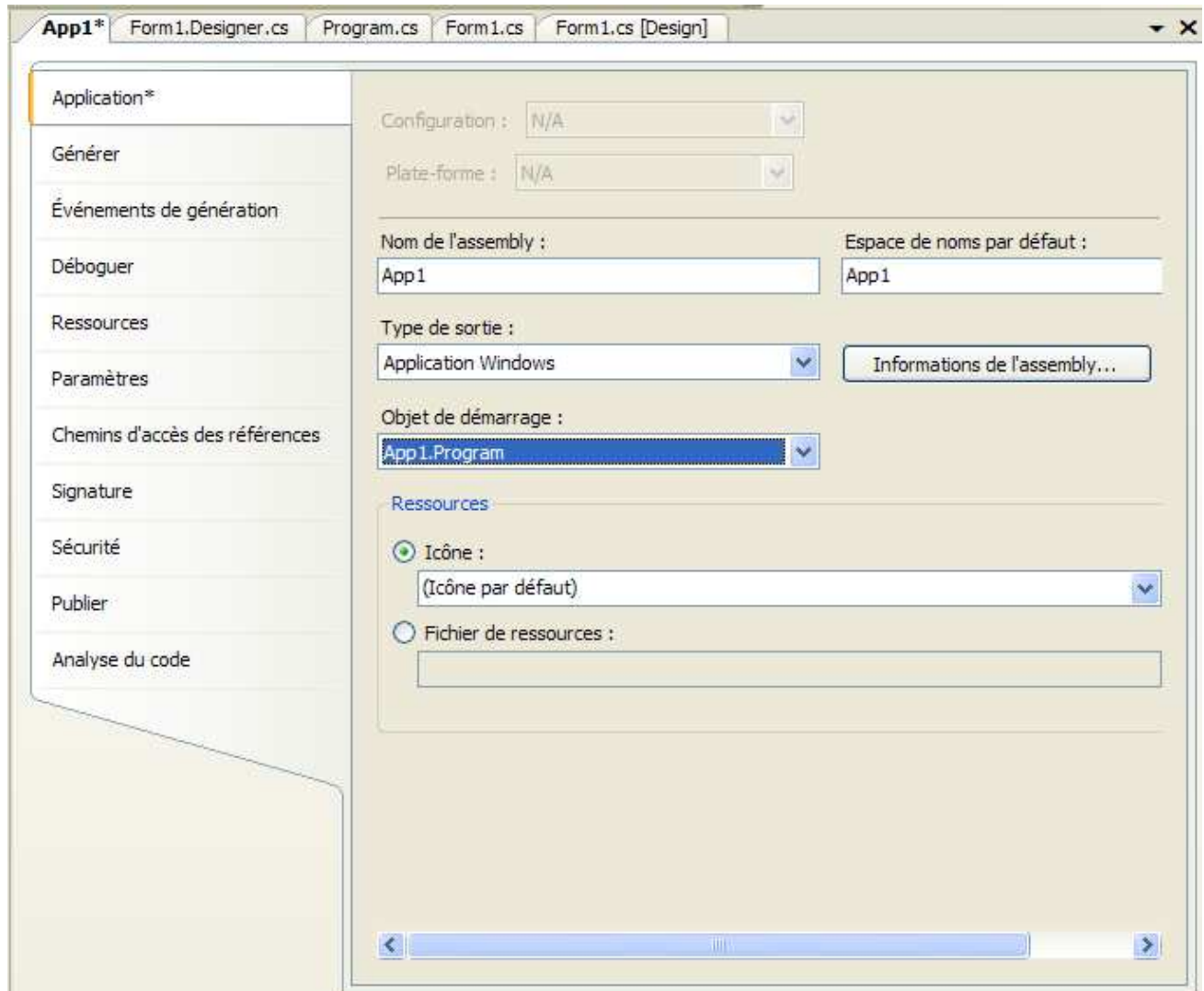
- la génération du projet (= la compilation de tous les sources) entraîne la sauvegarde des sources par défaut (paramétrable dans Outils / Options / Projets et Solutions / Générer et Exécuter)
- le lancement de l'application depuis l'EDI entraîne une génération préalable si cela le nécessite (fichiers sources modifiés récemment)
- la génération crée un fichier .exe rangé dans le répertoire bin/release s'il n'y a pas l'option de DEBUG ou dans le répertoire bin/debug dans le cas contraire (défaut)
- ces options DEBUG ou RELEASE du projet et le chemin de sortie de la génération se modifient depuis la fenêtre des propriétés du projet
- La version Debug génère des informations de débogage sous la forme de fichiers .pdb elle est utilisée pendant la phase de développement ; la version Release est destinée à être déployée : elle est entièrement optimisée et ne contient aucune information de débogage

¹ * Environnement de Développement Intégré

Caractéristiques du projet

Les ressources globales au projet sont stockées dans un fichier Resources.resx présent dans le répertoire Properties du projet, modifiable dans les propriétés du projet.

La fenêtre des propriétés du projet App1 permet d'ouvrir sa *page des propriétés* depuis sa dernière icône à droite.



On remarque dans les *propriétés de l'application*

1. on peut choisir l'objet de démarrage de l'application (ici la classe App1.Program). Nous verrons plus tard que si l'on peut choisir entre différentes feuilles au démarrage, celles ci devront alors posséder une méthode Main() de point d'entrée dans leur classe.
2. on peut donner un autre nom au fichier exécutable résultant de la génération en modifiant la nom de l'assembly. Nous verrons ce qu'est une assembly.

La classe de démarrage

La classe Program sert de point d'entrée à l'application (différence avec Visual Studio 2003)

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace App1
{
    static class Program
    {
        /// <summary>
        /// Point d'entrée principal de l'application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new frmTest());
        }
    }
}
```

La méthode Main appelle la méthode **Run** de la classe Application permet de lancer le traitement des messages Windows envoyés à l'application puis de créer le formulaire frmTest : le constructeur frmTest est alors exécuté avec la méthode InitializeComponent(), générée par l'éditeur (voir §3).

L'attribut **[STAThread]** (STA pour Single Threaded Apartment) signifie que seul le thread de la fenêtre pourra accéder aux composants de la fenêtre. Cet attribut est géré par la classe STAThreadAttribute qui dérive de la classe System.Attribute.

La méthode **EnableVisualStyles()** active les styles visuels pour l'application.

La méthode **SetCompatibleTextRenderingDefault()** active le rendu de texte par défaut pour les nouveaux contrôles.

Le code modifiable d'un formulaire

Une application Windows IHM est composée d'un ensemble de fenêtres qui se présentent à l'écran, soit directement au démarrage, soit à la demande de l'utilisateur.

Chaque fenêtre -ou feuille ou forme- est fractionnée sur 2 fichiers sources (en C# - extension .cs pour CSharp- dans notre cas)..

Chaque fichier source contient une section de la définition de classe, et toutes les parties sont combinées lorsque l'application est compilée.

La définition de classe est fractionnée grâce au modificateur de mot clé **partial**.

Code modifiable par le développeur :

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

namespace App1
{
    public partial class frmTest : Form
    {
        public frmTest()
        {
            InitializeComponent();
        }
    }
}
```

Les espaces de nom

Notre application avec toutes ses feuilles (donc classes) forme un espace de noms appelé App1 (du nom du projet par défaut).

Un **espace de noms** ("name space") est un conteneur de types que l'on peut utiliser dans le programme: des classes, des interfaces, des structures, des énumérations. Ces espaces sont organisés hiérarchiquement et fonctionnellement.

On constate les directives dans le source de la feuille frmTest.cs qui font référence aux espaces de noms:

System - Contient des classes fondamentales et des classes de base qui définissent les types de données référence et valeur (string, String, int ...), les événements et gestionnaires d'événements, les interfaces, les attributs et le traitement des exceptions courantes. Il contient aussi de nombreux espaces de noms de deuxième niveau.

System.Collections.Generic - Contient des interfaces et des classes qui définissent différentes collections d'objets, telles que des listes, des files d'attente, des tableaux de bits, des tables de hachage et des dictionnaires.

System.ComponentModel - Fournit des classes qui sont utilisées pour implémenter le comportement au moment de l'exécution et au moment du design des composants et des contrôles.

System.Data - permet d'accéder aux classes qui représentent l'architecture ADO.NET

System.Drawing - Permet d'accéder aux fonctionnalités graphiques de base de GDI+.

System.Text - Contient des classes représentant les codages de caractères ASCII, Unicode, UTF-7 et UTF-8.

System.Windows.Forms - Contient des classes permettant de créer des applications Windows qui profitent pleinement des fonctionnalités élaborées de l'interface utilisateur disponibles dans le système d'exploitation Microsoft Windows.

Les **directives using** du source dispensent de qualifier l'utilisation d'un type dans le code source si ce type fait partie d'un de ces espaces de noms.

Par exemple la directive **using System.ComponentModel;** permettra d'utiliser sa classe *Container* en la nommant simplement **Container** et non pas "

System.ComponentModel.Container ()".

Cependant la notation entièrement qualifiée reste utile dans le cas où deux espaces de noms présenteraient des classes de même nom.

La classe Form

La classe **Form** est utilisée pour créer des fenêtres standard (SDI, Single Document Interface), boîtes de dialogues ou multidocuments (MDI, Multiple Document Interface).

Ici la classe est dérivée en une classe **frmTest** qui permettra de personnaliser notre fenêtre.

La méthode **frmTest()** est le **constructeur de classe** ; elle appelle la méthode **InitializeComponent()** contenant le code d'initialisation du formulaire, généré par Visual Studio et généré dans le fichier frmTest.Designer.cs.

Cette méthode privée- sera enrichie au fur et à mesure que l'on rajoutera des composants à la feuille.

Au départ elle ne comporte que les initialisations de la feuille elle même:

C'est à partir de cette classe frmTest que sera créé en mémoire un objet feuille lors du démarrage de l'application (ou par la suite pour d'autres feuilles de l'application, à la demande de l'utilisateur).

On utilisera pour ce faire la classe **Application** dont le rôle est de fournir les méthodes statiques permettant de démarrer et arrêter une application de type Windows, et des propriétés statiques permettant d'obtenir des informations telles que path, versions

Ainsi l'instruction

```
Application.Run(new frmTest());
```

permet ici de lancer le traitement des messages Windows envoyés à l'application puis d'afficher une feuille à partir d'une instance en mémoire de la classe frmTest.

Le code généré par le concepteur

Le code généré par le concepteur se trouve dans le fichier frmTest.Designer.cs.

```
namespace App1
{
    partial class frmTest
    {
        /// <summary>
        /// Variable nécessaire au concepteur.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Nettoyage des ressources utilisées.
        /// </summary>
        /// <param name="disposing">true si les ressources managées
        /// doivent être supprimées ; sinon, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Code généré par le Concepteur Windows Form

        /// <summary>
        /// Méthode requise pour la prise en charge du concepteur - ne
        /// modifiez pas
        /// le contenu de cette méthode avec l'éditeur de code.
        /// </summary>
        private void InitializeComponent()
        {
            this.SuspendLayout();
            //
            // frmTest
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(292, 266);
            this.Name = "frmTest";
            this.Text = "Premier test";
            this.ResumeLayout(false);

        }

        #endregion
    }
}
```


Un champ de type **Container** est destiné à contenir d'autres composants (contrôles graphiques, ...).

La méthode **Dispose** sera exécutée à la fermeture du formulaire et permet de libérer les ressources.

Dans la méthode **Initializecomponent** sont codées toutes les descriptions des objets graphiques générées par le concepteur Visual studio.

On retrouve le titre de la fenêtre donné précédemment dans la fenêtre des propriétés (propriété **Text**), et le nom de la feuille modifié par sa propriété **Name**

Les méthodes **SuspendLayout** et **ResumeLayout** sont utilisées en tandem pour supprimer les événements *Layout* multiples lorsque vous ajustez plusieurs attributs du contrôle (leur place, leur taille, leur couleur, leur fonte, leur contenu pour les contrôles conteneur, etc ...).

Mise en oeuvre : ajout d'un bouton

Le choix du composant graphique se fait dans la boîte à outil de Visual Studio, en glissant le bouton de l'onglet **Windows Forms** sur la feuille.

Premier réflexe : Modifier les propriétés Name (btnQuitter)et Text (Quitter) du contrôle sur la feuille .

On s'aperçoit qu'une donnée membre btnQuitter de la classe des Button est déclarée dans la classe frmTest.

```
private System.Windows.Forms.Button btnQuitter;
```

et qu'il est pris en compte par la méthode **InitializeComponent()** vue précédemment

```
this.btnQuitter = new System.Windows.Forms.Button(); ❶
// ...
//
// btnQuitter
//
this.btnQuitter.Location = new System.Drawing.Point(104, 111);
this.btnQuitter.Name = "btnQuitter";
this.btnQuitter.Size = new System.Drawing.Size(75, 23);
this.btnQuitter.TabIndex = 0;
this.btnQuitter.Text = "Quitter";
this.btnQuitter.UseVisualStyleBackColor = true; ❷
```

❶ Instanciation du bouton

❷ Définition de ses propriétés

Les mécanismes événementiels

Les événements peuvent être déclenchés par l'action d'un utilisateur (via son clavier ou sa souris), par le programme lui-même (instructions C#) ou par le système (timers par exemple).

Le programmeur peut décider de gérer ou non un événement en lui associant du code spécifique, appelé gestionnaire de l'événement.

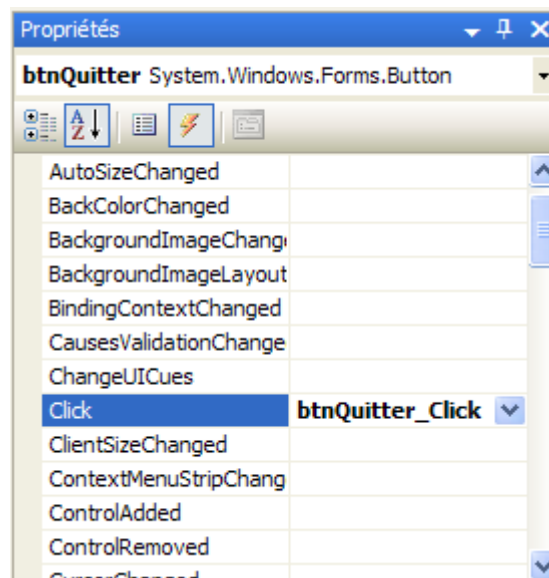
Il doit donc dans un premier temps écrire la méthode C# qui va traiter cet événement, puis dans un deuxième temps associer cette méthode à l'événement; l'événement peut d'ailleurs être associé à plusieurs méthodes qu'il activera toutes lorsqu'il claquera.

Il faut donc mettre en place un mécanisme qui va permettre de relier un objet "**abonné**" (notre page frmTest par exemple) à un objet "**annonceur**" (notre bouton btnQuitter par exemple) qui va déclencher un événement (clic par exemple). Lorsque l'événement va surgir chez l'annonceur, tous les abonnés seront prévenus et exécuteront le traitement qu'ils ont prévu pour cet événement.

Le .NET Framework utilise la technique du **délégué d'événement** qui connecte un événement avec son gestionnaire. On crée ce délégué de type **System.EventHandler** en lui donnant la référence à la méthode de traitement de l'abonné; ensuite le délégué est rajouté à l'événement de l'annonceur (se rappeler que les classes en C# présentent des membres de type événements).

Traiter le click du bouton

Le clic étant l'événement le plus courant du bouton, il suffit de double cliquer sur le bouton depuis la fenêtre de conception pour que le code soit généré **automatiquement** par Visual Studio, ou alors par le biais de la fenêtre de propriétés, double cliquer sur l'évènement choisi.



La feuille (l'abonné) prépare sa méthode privée de traitement. Microsoft conseille de normaliser les noms de gestionnaires de la sorte: *annonceur_événement* et oblige à récupérer les deux paramètres *sender* (qui référence à l'objet annonceur qui génère l'événement) et *e* (qui est une structure d'infos de l'événement, et dont la classe est EventArgs ou une de ses classes dérivées pour certains événements spécifiques...)

- Génération de la méthode d'événement dans frmTest.cs

```
public partial class frmTest : Form
{
    // ....
    private void btnQuitter_Click(object sender, EventArgs e)
    {

    }
}
```

- Génération du code qui crée un **délégué** d'événement (*new*) et rajoute (*+=*) au membre événement **Click** du bouton (l'annonceur) (codé dans la méthode **InitializeComponent** dans **frmTest.Designer.cs**)

```
this.btnQuitter.Click += new System.EventHandler(this.btnQuitter_Click);
```

Il ne reste au développeur qu'à coder le traitement à exécuter lorsque l'utilisateur clique sur ce bouton Quitter.

Première possibilité: la forme se ferme elle même par **this.Close()**. Comme c'est la seule feuille de l'application, l'application est donc arrêtée.

Deuxième possibilité: par la méthode statique **Application.Exit()**; de la classe Application, qui arrête le traitement des messages windows et ferme toutes les fenêtres de l'application.

On remarquera qu'un arrêt par la fermeture (Close) de la feuille rend la main au Main() de démarrage, lui donnant la possibilité d'enchaîner une autre instruction en sortie de la fonction Run (et pourquoi pas un autre Run de l'Application?)

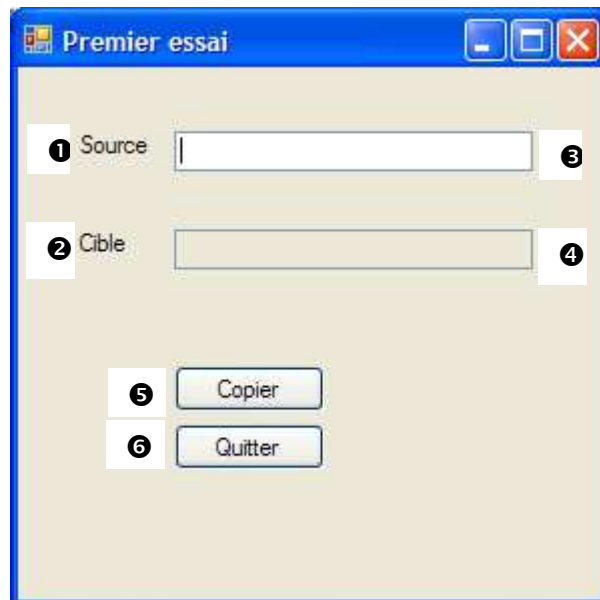
Exemple

Après avoir saisi du texte dans le contrôle source, en cliquant sur le bouton Copier, le texte saisi est ajouté au contrôle cible inaccessible en saisie.

Le contrôle source est alors automatiquement effacé.

Le focus se positionne sur le contrôle source.

Le bouton Quitter permet de terminer l'application.



Les contrôles graphiques et leurs propriétés modifiées :

- ❶ un contrôle Label
Modifier sa propriété **Text** en positionnant la valeur **Source**
- ❷ un contrôle Label
Modifier sa propriété **Text** en positionnant la valeur **Cible**
- ❸ un contrôle TextBox
Modifier sa propriété **Name** en positionnant la valeur **txtSource**
- ❹ un contrôle TextBox
Modifier sa propriété **Name** en positionnant la valeur **txtCible**
Positionner sa propriété **ReadOnly** à **true**
- ❺ un contrôle Button
Modifier sa propriété **Name** en positionnant la valeur **btnCopier**
Modifier sa propriété **Text** en positionnant la valeur **Copier**
- ❻ un contrôle Button
Modifier sa propriété **Name** en positionnant la valeur **btnQuitter**
Modifier sa propriété **Text** en positionnant la valeur **Quitter**

Le code associé :

L'évènement **Click** sur le bouton **Copier** provoque l'exécution de la fonction :

```
private void btnCopier_Click(object sender, EventArgs e)
{
    // recopie de la valeur saisie de la source vers la cible
    txtCible.Text = txtCible.Text + " " + txtSource.Text;
    // efface la source : on utilise la méthode Clear
    txtSource.Clear();
    // positionne le focus: on utilise la méthode Focus
    txtSource.Focus();
}
```

L'évènement **Click** sur le bouton **Quitter** provoque l'exécution de la fonction :

```
private void btnQuitter_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```