Prise en main de l'IDE Visual Studio

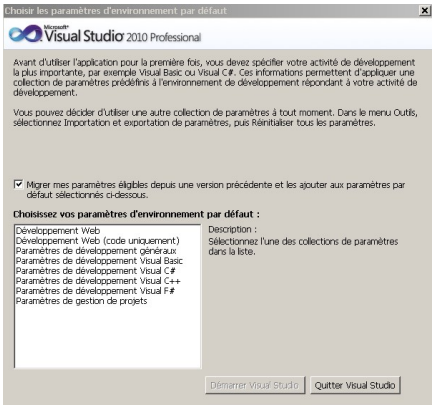
I. Introduction

Présentation

Un Environnement de Développement Intégré (EDI) ou IDE en anglais (Integrated Development Environment) est un logiciel regroupant au minimum les fonctionnalités suivantes : un éditeur de texte, un compilateur, des outils d'aide à la programmation et un débogueur. Un EDI fournit aussi des outils et des bibliothèques propres permettant de créer des interfaces graphiques IHM (Interface Homme-Machine en français) ou GUI (Graphical User Interface en anglais). Nous allons utiliser l'IDE Visual Studio en vous présentant successivement le menu, les différentes fenêtres de travail, la fenêtre d'édition, les barres d'outils et l'interface de débogage.

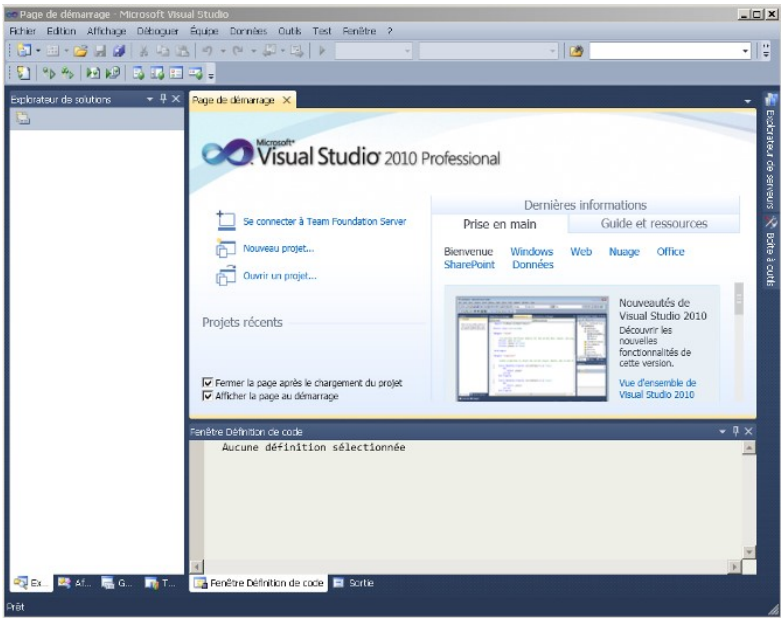
Lancez Visual C#

Le programme, si vous ne l'avez pas encore mis dans vos raccourcis du bureau peut se lancer à partir de Démarrer > Programmes > Microsoft Visual Studio> Microsoft Visual Studio. La fenêtre de lancement s'ouvre. Si vous ne vous êtes jamais connecté sur la station actuelle, le démarrage peut prendre du temps car il faut créer votre profil ou initialisé le premier lancement du logiciel. Après cela, vous arrivez à une fenêtre de sélection où vous choisissez de travailler en C#.

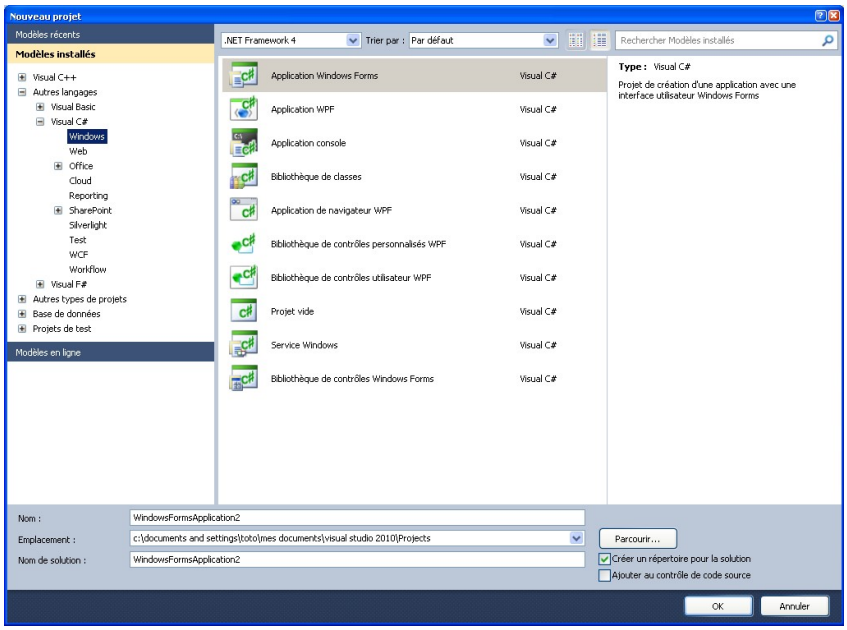


Créer son premier projet

Vous vous retrouvez face à une interface incluant diverses fenêtres, interface un peu déroutante au début :



Nous allons créer un premier projet succinct nous permettant de découvrir l'environnement. Pour cela, allez dans le menu en haut à gauche et choisissez : Fichier > Nouveau > Projet.



Il existe deux grands types d'application dans le monde informatique : les applications :

- Applications dites « Fenêtrées » constituées d'une interface avec des boutons, des cases à cocher, des listes déroulantes…
- Applications dites « Console » lancées dans une interface texte et n'utilisant pas la souris ni les graphismes.

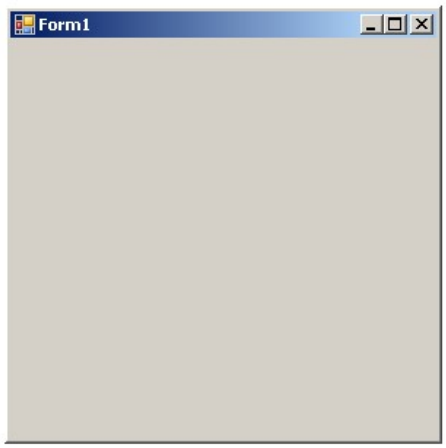
Les applications « fenêtrées » correspondent à ce que l'on rencontre tous les jours. Ces applications ne contiennent pas de fonction principale main() correspondant au démarrage du programme. Les points d'entrée de votre programme sont multiples et correspondent à des fonctions lancées suite à un

évènement utilisateur du type : click sur un bouton, déplacement de la fenêtre, déplacement de la souris.

A l'inverse, les applications « console » correspondent aux applications en mode texte (pensez au DOS, au terminal Linux). Elles n'ont pas de boutons ou d'autres interfaces visuelles, elles n'utilisent pas la souris et ne peuvent pas gérer de graphisme. Le programme a un unique point d'entrée représenté par la fonction main().

Sous Visual, on retrouve ces deux choix. Nous utiliserons les « Application Windows Forms » (Fenêtre = Form). Pour continuer, donnez un nom à votre projet et vérifiez que le répertoire de travail correspond à votre déport SVN.

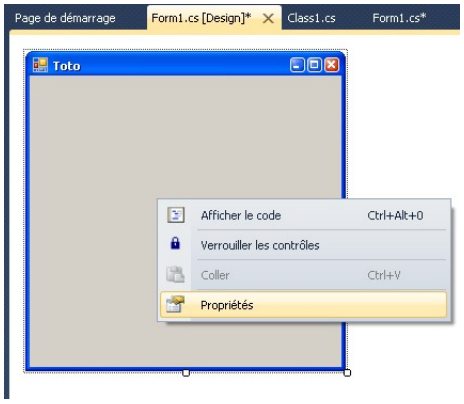
Une fois ces paramètres rentrés, cliquez sur OK. Visual va alors créer plusieurs fichiers servant de squelette à votre projet. Pour tester si tout s'est bien passé, appuyez sur F5, ce qui provoque la compilation et l'exécution du programme. Vous devriez voir apparaître la fenêtre suivante :

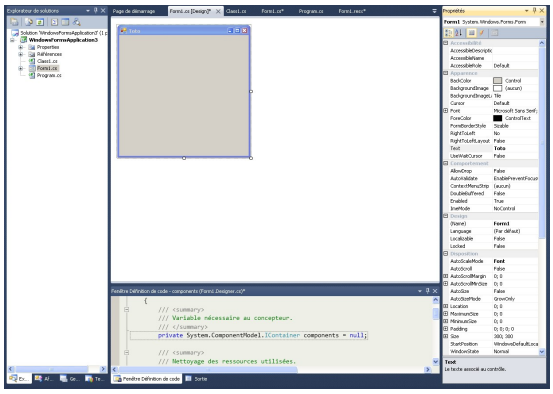


Un bug ? Un trou noir dans le système d'exploitation ? Non, une simple fenêtre vide qui attend vos modifications.

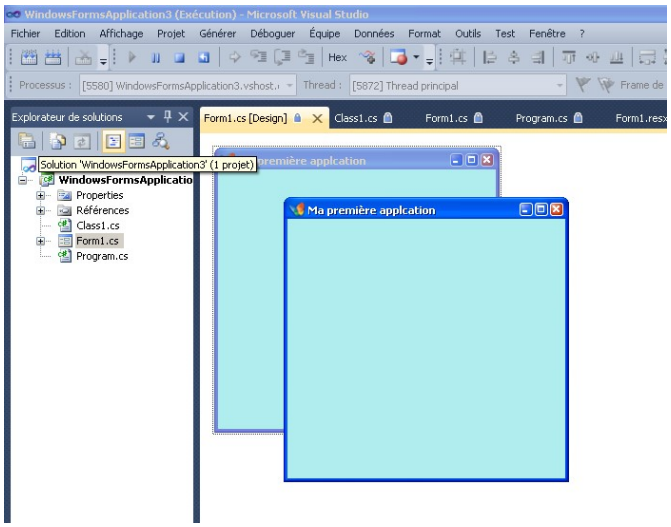
II. L'éditeur d'IHM

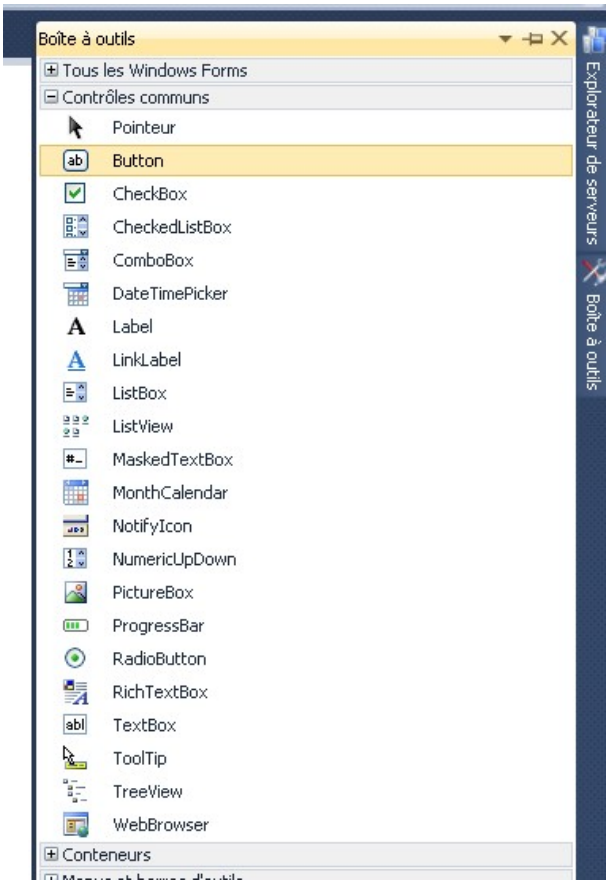
Vous allez réaliser votre premier programme. Dans l'explorateur de projet, double-cliquez sur « Form1.cs » pour ouvrir l'assistant d'édition de votre fenêtre. Nous allons d'abord modifier quelques propriétés basiques. Pour cela, faites un click droit sur la fenêtre et choisissez propriétés :





A droite de l'écran s'affiche la liste des propriétés de votre fenêtre. Vous pouvez changer le titre de votre fenêtre en modifiant le champ « Apparence > Text ». Vous pouvez changer la couleur de fond de votre fenêtre dans le champ « Apparence > BackColor ». Vous pouvez changer l'icône en modifiant « Style de la fenêtre > Icon », une recherche sur l'ordinateur vous donnera d'autres fichiers « .ico ». En appuyant sur F5, vous devriez voir la nouvelle fenêtre d'application relookée, voir exemple ci-dessous :



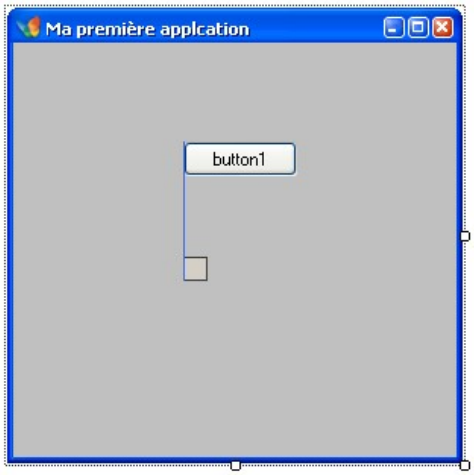


Une IHM (interface homme machine) est un terme générique servant à décrire toute interface visuelle interactive permettant de déclencher des évènements suite à des clicks et de choisir différentes options par des listes, menus, cases à cocher.

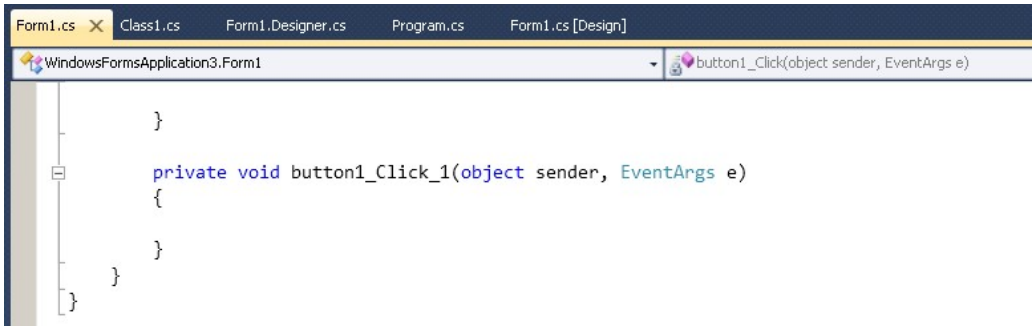
Boite à outils : Affichage > Boites à outils – CTRL ALT X

En activant la boite à outil, vous allez disposer d'un ensemble de composants (boutons, labels, barre de progression…) que vous allez glisser déposer dans votre maquette. Choisissez la section « Contrôles Communs », cliquez sur « Button », laissez appuyer, glissez vers votre maquette, la boite à outils peut alors disparaître, ça n'a pas d'importance, puis relâchez le bouton de la souris au milieu de votre maquette.

Recommencez cette manipulation en posant un Label sur votre maquette. Vous pouvez remarquer que l''assistant vous aide à positionner vos différents objets. En effet, si vous cherchez à vous positionner au même niveau que votre bouton juste en dessous, l'assistant va dessiner des traits d'alignement pour vous montrer si vous êtes au bon endroit. Vous pouvez ensuite cliquer sur le bouton et changez sa taille en utilisant les poignées. En rouvrant la fenêtre de propriétés, vous pouvez modifier les attributs de ce bouton. Par exemple, changez son contenu dans le champ « Apparence > Title », la police du texte dans « Apparence > Font ». En appuyant sur F5, vous devriez voir apparaître votre fenêtre avec ces nouveaux éléments :



Nous allons un peu programmer. Une facilité fournie par les environnements de développement moderne est l'assistance à la création de code. Ainsi, en double cliquant sur le bouton « Clique-moi » sur la maquette de la fenêtre, Visual crée et déclare automatiquement la fonction qui sera appelée suite à cet événement. Ensuite, il ouvre le fichier concerné dans l'éditeur et positionne le curseur à l'endroit précis où vous devez taper le code de cette fonction. Le gain en productivité et la facilité sont au rendez-vous. Vous devriez obtenir la vue suivante :



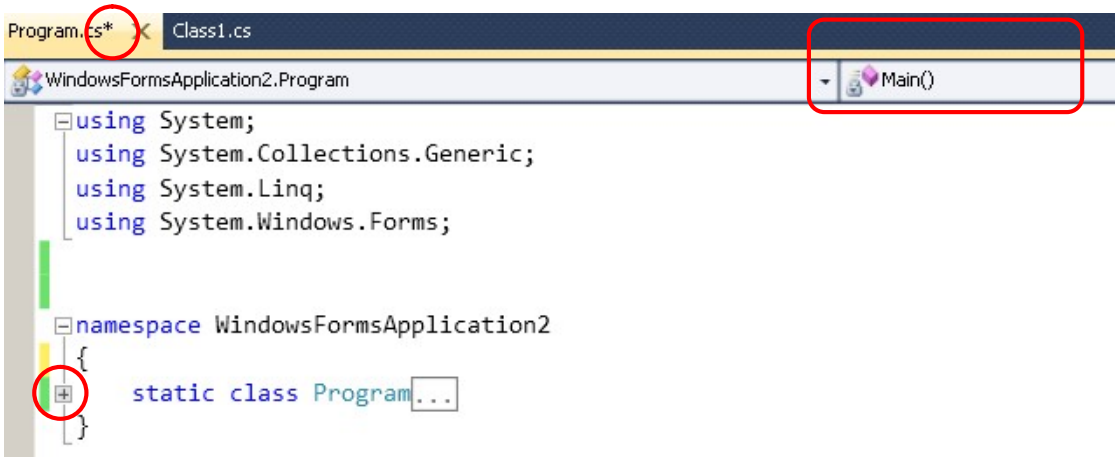
Vous allez taper le code suivant :

```
private void button1_Click_1(object sender, EventArgs e)
{
    label1.Text = "Aïe";
}
```

Le terme « label1 » désigne le nom du Label positionné sur la fenêtre dans l'assistant de conception. Si vous avez changé le nom de ce composant pour un autre, veuillez adapter le code en fonction. « .Text » permet d'accéder au champ « Apparence > Text » du bouton comme vous le faisiez précédemment dans la fenêtre de propriétés. Appuyez sur F5 et cliquez sur le bouton pour voir le résultat.

III. Fenêtre centrale d'édition

Nous allons nous pencher sur la fenêtre d'édition de code généralement positionnée au milieu de l'écran. Pour ouvrir d'autres fichiers, cliquez sur « Program.cs » et « class1.cs » dans l'explorateur de solutions. Vous pouvez passer d'un fichier à l'autre en cliquant sur chaque onglet.



L'étoile signale que des modifications ont été effectuées dans le fichier depuis sa dernière sauvegarde. Le raccourci CTRL-S permet de sauvegarder le document en cours d'édition et l'étoile disparaît alors. Le +/- à gauche permet de réduire le code de la fonction sur une seule ligne. En haut à droite, vous trouvez une liste contenant les fonctions du fichier, elle permet d'y accéder rapidement.

Revenez dans la fonction button1_Click_1 de tout à l'heure et tapez à l'intérieur « lab », une des fonctionnalités d'assistance au développement va alors s'activer et vous devriez voir ceci :

```
private void button1_Click_1(object sender, EventArgs e)
{
    label1.Text = "Aïe";
    lab
}
```

DataGridParentRowsLabelStyle
Label
label1
LabelEditEventArgs
LabelEditEventHandler
LinkLabel
LinkLabelLinkClickedEventArgs
LinkLabelLinkClickedEventHandler
NodeLabelEditEventArgs

Visual vous indique tous les termes dans le projet (classes, fonctions…) commençant par lab et bien sur « label1 » y figure en bonne place, à ce niveau un simple appui sur entrée va compléter automatiquement votre texte en « label1 ». Vous devriez obtenir ceci :

```
private void button1_Click_1(object sender, EventArgs e)
{
    label1.Text = "Aïe";
    label1
}
```

Appuyez ensuite sur « . » et vous allez voir apparaître l'ensemble des propriétés et méthodes associées à votre objet. Un simple appuie sur « te » nous amène au bon champ :

```
private void button1_Click_1(object sender, EventArgs e)
{
    label1.Text = "Aïe";
    label1.te
}
```

ResetText
Text — string
TextAlign
TextAlignChanged
TextChanged
UseCompatibleTextRendering

Vous pouvez remarquer qu'il fallait utiliser une majuscule pour accéder à la propriété « Text ». Remarquer à droite le terme « string » qui apparait 2 à 3 secondes après. Cela signifie que le champ « Text » est une variable de type « string » et qu'elle doit donc recevoir un contenu de ce type là sinon elle va lever une erreur d'exécution. Finissez votre ligne pour obtenir cela :

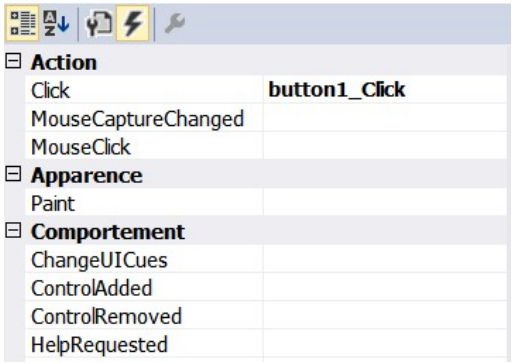
```
private void button1_Click_1(object sender, EventArgs e)
{
    label1.Text = "Aïe";
    label1.Text = "Ouille";
}
```

Le module gérant ce type de fonctionnalités s'appelle << Intellisense >> sous Visual. Il facilite la vie du programmeur et permet d'augmenter la productivité. On parle plus généralement d'autocomplétion.

IV. Associer évènement et fonction

Nous l'avons vu, double cliquez sur un composant déclenche la création de la fonction associée. De plus cette fonction est liée au gestionnaire d'évènements du composant (mise en place d'un écouteur) de manière automatique. Nous allons un peu regarder sous le capot.

Sélectionnez le composant bouton et ouvrez la fenêtre de propriétés. Cliquez sur l'icône éclair présent en haut de la fenêtre :



Une fois l'icône éclair actif, la liste des champs est remplacée par la liste des évènements du composant. Ici, il est indiqué que l'appui sur le bouton appelle la méthode button1_click.

Ouvrez le fichier Form1.Designer.cs (il est caché sous Form1.cs). Le contenu de ce fichier est généré dynamiquement en fonction des actions que vous effectuez dans l'éditeur d'IHM. Vous voyez le contenu suivant (en résumé) :

```
namespace WindowsFormsApplication1
{
    partial class Form1
    {
        ...

        Code généré par le Concepteur Windows Form

        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Button button2;
    }
}
```

Cliquez sur le + pour étendre le code généré automatiquement :

```
#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.button1 = new System.Windows.Forms.Button();
    this.label1 = new System.Windows.Forms.Label();
    this.button2 = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(124, 118);
    this.button1.Name = "button1";
    this.button1.Size = new System.Drawing.Size(144, 56);
    this.button1.TabIndex = 55;
    this.button1.Text = "Clique moi";
    this.button1.UseVisualStyleBackColor = true;
    this.button1.Click += new System.EventHandler(this.button1_Click);
```

Voici la fonction InitializeComponent appelée directement par le constructeur. Il est FORTEMENT déconseillé de modifier le code directement à cet endroit. En effet, si vous tapez du code qui contrarie le concepteur d'IHM, il peut planter et refuser d'ouvrir votre fenêtre pour modification. C'est embêtant et le seul recours et de revenir à une version antérieure de votre dépôt SVN. On peut cependant parfois supprimer brutalement une ligne qui pose problème au concepteur d'IHM pour se sortir d'une impasse.

Un rapide coup d'œil nous permet de comprendre ce que fait le concepteur d'IHM à votre place :

- `button1 = new Button()` : création du bouton par un new
- `button1.Location = new Point(124, 118)` : positionnement dans la fenêtre
- `button1.Size = new Size(144, 56);` dimension du bouton
- `button1.Text = "Clique moi";` texte affiché dans le bouton

Ces lignes se comprennent intuitivement. Vous pouvez constater que le code généré vous évite pas mal de code au clavier.

Il reste une ligne un peu complexe et hors programme que nous abordons par familiarité avec le Java (création du listener). En fait, le dispatcher de Button1 associé à l'évènement Clic enregistre un nouvel objet (new) appelant la fonction button1_Click. Pourquoi une syntaxe aussi complexe ?

Dispatcher

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

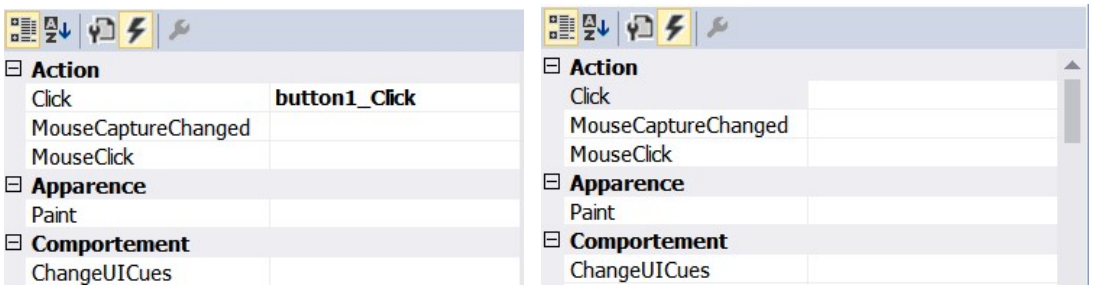
Techniquement, elle a plusieurs avantages. Le dispatcher `button1.Click` contient une liste des éléments à appeler : les écouteurs. Cette liste peut ainsi varier au cours du temps : on peut avoir plusieurs écouteurs associés à un évènement ou aucun. Le concepteur d'IHM lui ne gère que le cas : 1 évènement <-> 1 fonction qui concerne 95% des cas.

Prenons un plugin VLC qui permet l'affichage des sous-titres à partir d'un fichier texte. L'activation de ce plugin sans modifier vos fonctions existantes peut se

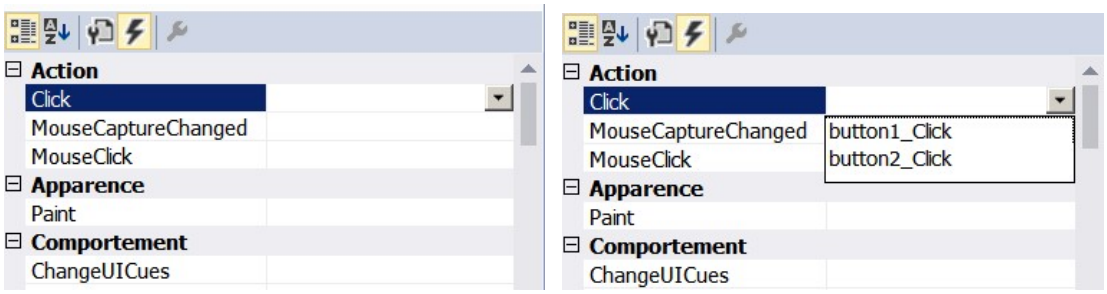
faire simplement par insertion d'écouteurs supplémentaires sur les boutons play/pause du lecteur vidéo.

Désinscrire une fonction à travers le concepteur d'IHM (écouteur)

Cliquez sur button1_Click, effacez le champ et ensuite appuyez sur entrée. Cela a pour effet de désassocier l'évènement clic et l'appel de la fonction :



Il possible d'associer une fonction déjà existante avec l'évènement. Pour cela cliquer dans la case à droite de Clic. Un icône défilement apparaît, cliquez alors dessus pour lister l'ensemble des fonctions pouvant convenir.



Sélectionnez une fonction et l'association est réactiver.

Associer une fonction à plusieurs composants

A noter qu'une même fonction peut être associée à plusieurs boutons. Cela semble étrange, mais pourtant pratique. Imaginez un projet « calculette », où l'appui sur les boutons 1,2,3…7,8,9 ajoute un chiffre à l'écran. L'action étant identique, une seule fonction peut suffir. En effet, l'écouteur récupère en paramètres :

```
button1_Click_1(object sender, EventArgs e)
```

1. Des informations sur le composant levant l'évènement : Object sender
2. Des informations sur l'évènement lui-même EventArgs e

Pour les informations de l'évènement, on peut par exemple avoir les coordonnées du clic souris. Pour le sender, son type est générique : object car il peut concerner un bouton, une combobox ou une listbox. Si vous savez que cet évènement est associé uniquement à des butons, vous pouvez caster sender en Button et obtenir toutes les informations associées – notamment le chiffre contenu dans sa propriété Text.

V. Structure d'un projet Visual Studio

Un nouveau projet comporte plusieurs fichiers de base. Voici comment ils se structurent. Dans **program.cs**, on peut trouver la ligne suivante :

```
static void Main()  {  ...  Application.Run(new Form1());  }
```

Il s'agit du point d'entrée du programme, le « main » habituel. Le new Form1() est l'appel du constructeur de la fenêtre de l'application. Le run met en place la boucle de gestion des messages clavier/souris… Ce fichier est généré automatique par Visual Studio, vous ne devez pas le modifier.

La fenêtre de l'application va être définie dans la classe Form1. Mais attention, contrairement à Java, cette classe va être décrite dans deux fichiers séparés :

Form1.Designer.cs : il contient les contrôles de l'IHM. Les lignes de code sont automatiquement générées en interaction avec le Designer. A l'intérieur, on trouve :

- Les attributs relatifs aux contrôles. Ainsi, si je crée un bouton, je vais trouver :

  ```
  private System.Windows.Forms.Button button1;
  ```

- La méthode InitializeComponent() qui instancie les différents contrôles :

  ```
  this.button1.Location = new System.Drawing.Point(82, 60);
  this.button1.Name = "button1";
  this.button1.Text = "Click me";
  ```

Form1.cs : il contient les méthodes de traitement des évènements. C'est principalement dans cette zone que vous codez. Les prototypes de fonctions sont construits par interaction avec le Designer. On y trouve la ligne suivante :

- public partial class Form1

  Elle indique grâce au mot clef « partial » qu'il s'agit de la suite de la déclaration de la classe Form1 initiée dans le fichier Form1.designer.cs

- Nous trouvons ensuite le constructeur

  ```
  public Form1()  {  InitializeComponent();  }
  ```

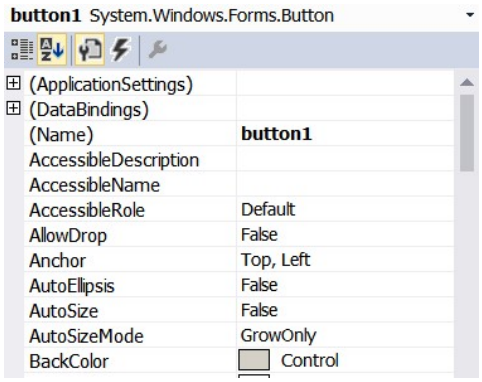
Attention, si vous faîtes des manipulations sur vos contrôles dans le constructeur, faites-le après l'appel de InitializeComponent, car avant, les contrôles ne sont pas encore créés !!

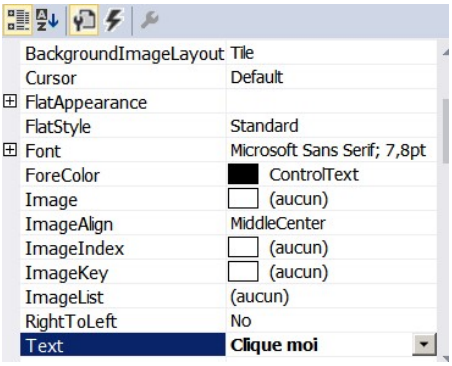
Ne faites pas l'erreur de croire que vous pouvez lancer des affichages à la fin de votre constructeur. En effet, votre fenêtre n'existe pas encore complètement ! Elle est créée en mémoire, mais elle n'est pas encore apparue à l'écran !

VI. Passez à la vitesse supérieure

Trouver le nom d'un champ

Dans la fenêtre des propriétés, vous avez deux présentations possibles :





 Liste des propriétés par classement alphabétique

 Liste des propriétés par classement thématique

Au début lorsque vous allez découvrir les composants de l'IHM, nous vous conseillons de commencer par la liste thématique car les noms des attributs sont triés par thème ce qui les rend plus facile à localiser. Une fois habitué aux principaux champs, il sera plus rapide de passer en mode alphabétique pour gagner du temps.

Modifier plusieurs champs de façon efficace

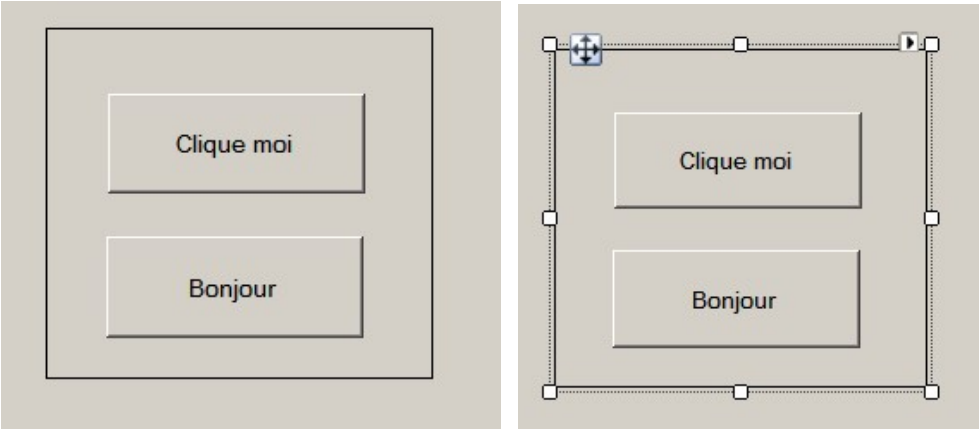
1- F4 ouvre la fenêtre des propriétés et sélectionne le dernier champ édité
2- Lorsque vous sélectionnez un bouton et modifiez son champ Text (+ENTER), si vous cliquez sur un autre composant et tapez au clavier, cela modifie directement son champ Text sans avoir à sélectionner ce champ dans la liste des propriétés. Cette méthode est vraie pour tous les champs.
3- Vous pouvez sélectionner plusieurs boutons et changer la police, le changement sera alors appliqué à l'ensemble des boutons.
4- F7 ou un double-clic vous fait passer du composant au code du clic.
5- SHIFT F7 vous fait passer du code de la fonction à son composant.

Positionnez et grouper des composants

Une fois sélectionnés, vous pouvez utiliser les flèches clavier pour déplacer un ou plusieurs composants.

Dans la boite des outils, sélectionnez le groupe : conteneur. Sélectionnez l'outil Panel. Une fois sélectionné, dessinez un rectangle autour de plusieurs composants. Ils sont maintenant un sous ensemble d'un panneau. Par défaut les

panneaux ont des bords invisibles, sélectionnez la propriété BorderStyle (Apparence) et choisissez un bord FixedSingle.



La sélection du Panel et son déplacement permet de déplacer l'ensemble des compostants qu'il embarque.

Une manipulation similaire est possible. Il faut sélectionner des composants, effectuer un couper, sélectionner le Panel et effectuer un coller. Les composants sélectionnés sont alors déployés à l'intérieur.

VII. Dessiner à l'écran

PictureBox et Bitmap

Créez un PictureBox sur votre fenêtre et donnez-lui une taille de 640x480. Modifiez sa propriété BorderStyle à FixedSingle pour qu'il soit entouré d'un cadre à l'écran. Dans le code, créez un objet Bitmap en utilisant la ligne suivante :

```
new Bitmap(640,480);
```

Il faut se demander où et quand l'objet Bitmap doit être créé. Techniquement vous avez deux choix : dans la fonction déclenchée par le double clic ou ... ailleurs. *Quel est le bon choix et pourquoi ?*

Associez ensuite ce bitmap à la PictureBox présente sur votre form en écrivant :

```
pictureBox1.Image = B;
```

Il faut bien distinguer ces deux objets. Un PictureBox est un composant de l'IHM positionné sur votre fenêtre et servant à afficher des contenus graphiques. Bitmap est un objet en mémoire servant à représenter une image. Pour qu'une image soit affichée à l'écran il faut donc utiliser ce système double : un objet qui stocke l'information couleur de l'image dans la mémoire et un objet qui l'affiche dans la fenêtre. Lancez votre programme, si l'association est faite correctement, vous devriez voir rien du tout ☺ car par défaut le bitmap est construit avec la couleur de fenêtre.

Un Bitmap est juste une représentation en mémoire de la couleur de chaque pixel de l'image, cette structure de données est très proche d'un tableau bidimensionnel. Cependant, elle ne permet pas de dessiner à l'intérieur.

VIII. Survivre dans la doc de .net

Afin de dessiner, nous allons utiliser un objet Graphics. En tapant "graphics C#" dans google. Vous arrivez directement sur la page d'aide de Microsoft concernant cette classe. Dispose-t-elle d'un constructeur ?

Une méthode statique, marquée d'un  dans la documentation, est une méthode qui existe même si aucun n'objet n'a été créée. Elle est rattachée à la classe et non à une instance. On y accède ainsi :

NomDeLaClasse.MethodeStatique(...)

Ces méthodes ont plusieurs intérêts dont celui de fournir des services.

Examinez la méthode FromImage de la classe Graphics, portez votre attention à son type de retour. Examinez son paramètre en entrée : un type Image. Dans google, tapez "Image C#". Vous obtenez ainsi les informations sur la classe Image et notamment sa hiérarchie d'héritage :

Nous trouvons la classe Image inscrite en noir. Au-dessus se trouvent ses classes parents et en en dessous ce qui nous intéresse plus précisément ici : ses classes enfants. Construisez ainsi votre objet Graphics.

Dans le code du bouton TEST, utilisez l'objet de type Graphics pour donner une couleur blanche à l'image affichée. Pour cela, utilisez sa méthode Clear. Vous remarquez dans la doc qu'elle accepte un paramètre Color. En allant examiner la classe Color, vous constatez que des propriétés statiques sont définies et qu'elles pourraient vous être utiles.

Lancez votre programme, cliquez sur le bouton. Normalement, rien ne s'affiche à l'écran. Cela est normal.

Nous avons dessiné dans l'objet Bitmap en mémoire mais à aucun moment, l'interface d'affichage, notre objet PictureBox, ne sait qu'un changement a été effectué. Il ne fait donc rien. Il faut donc lui demander de se mettre à jour. Cela sera fait par l'intermédiaire de sa méthode Invalidate(). Rajoutez la ligne suivante à la suite de votre commande Clear :

```
pictureBox1.Invalidate();
```

Relancez le programme vous devriez voir la différence. Pour être précis, Invalidate produit un rafraîchissement de l'écran asynchrone. En effet, il déclenche la demande de mise à jour, mais c'est le système d'exploitation qui décide au final quand effectuer cet affichage. S'il est très occupé, la demande attendra. C'est aussi un moyen d'optimiser l'affichage. En effet, si vous lancez 1000 commandes de dessin suivis d'un Invalidate, il serait maladroit d'effectuer 1000 mises à jour à l'écran. En général, le système optimise ces demandes, ainsi 1000 Invalidate lancés en moins de 1/1000s produiront un seul réaffichage.

IX. Fonctions Dessins et composants

Dessiner un segment

Dans la classe Graphics, recherchez la méthode qui dessine un segment. Elle va nécessiter d'autres objets pour être appliquée. Par défaut, on va tracer un trait de couleur rouge.

Pour dessiner ce segment, il va vous falloir deux points. Vous allez utiliser un composant de l'IHM appelé un TextBox. Cette boîte d'édition permet d'entrer du texte. Nous allons en utiliser 4 pour donner les coordonnées nécessaires. Vous pouvez récupérer le contenu de la TextBox en utilisant sa propriété Text.

Il va falloir ensuite convertir votre objet string en nombre. Pour cela, examinez la classe Int32 contenant des opérateurs de conversion intéressant. La gestion des exceptions est hors programme, considérez par défaut que l'utilisateur rentre des chiffres corrects.

Attention le centre du repère est en haut à gauche et l'axe des ordonnées est orienté vers le bas. Vos dessins seront ainsi inversés par rapport à la verticale.

Dessiner un Rectangle simple

Dans la classe Graphics, vous remarquez qu'il existe deux méthodes pour dessiner un rectangle : DrawRectangle pour un affichage simple et FillRectangle pour un rectangle plein. Vous remarquez aussi au passage, qu'il existe deux classes Rectangle et RectangleF de .net permettant de décrire un cadre. Evitez d'utiliser ces noms !

Vous pouvez afficher la couleur courante à travers un composant Label. Il sert habituellement à afficher du texte dans la fenêtre, mais en ne mettant aucun texte et en fixant sa couleur de fond, vous devriez faire apparaître un rectangle portant la couleur du crayon. Cette interface est proche de celle de Paint (petit utilitaire de dessin de windows).

Il nous faut maintenant choisir une couleur. Jour de chance, .net fournit une fenêtre complète pour la sélection d'une couleur par l'utilisateur. Examinez la classe ColorDialog, l'exemple en bas de page sera utile. Faites en sorte que cette fenêtre s'ouvre lorsque l'on double-clique sur le Label portant la couleur.

Créez un bouton pour lancer l'affichage d'un rectangle à partir des informations rentrées. Pour plus de clarté, vous pouvez faire un clear de la zone d'affichage pour la nettoyer des essais précédents.

Dessiner un Rectangle plein

Il nous faut maintenant une couleur de fond. Reprenez l'approche précédente pour sélectionner la couleur de fond.

Plutôt que de créer des boutons : rectangle simple - rectangle plein - ellipse simple - ellipse pleine... nous allons mettre en place un RadioButton permettant de choisir entre l'option plein ou vide. Pour activer un Radiobutton, par défaut, modifiez sa propriété Checked. L'intérêt des RadioButtons est qu'un unique bouton peut être sélectionné. Ainsi, lorsque vous cliquez sur un RadioButton pour l'activer, automatiquement, le RadioButton précédemment sélectionné est désactivé.

Remarque : pour ceux qui se demandent comment gérer plusieurs groupes de RadioButtons, il faut effectivement faire en sorte qu'ils deviennent indépendants. Pour cela, il faut choisir dans la boite à outils un Containeur appelé Panel. Ce genre de container permet de séparer des zones d'interaction dans la fenêtre de l'écran. Leur utilisation est simple. Si vous créez un composant à l'intérieur, ce composant appartient au Panel. Ainsi lorsque vous déplacez ce Panel, les composants à l'intérieur restent attachés. Vous pouvez faire aussi un couper-coller pour transférer plusieurs composants dans un Panel.

Afficher du texte

Trouvez dans Graphics, la méthode permettant d'afficher du texte, il en existe plusieurs avec des approches différentes.

Pour sélectionner la police, une classe est fournie par .net : FontDialog.

Afficher une image

Créez un nouveau bouton dans l'application et repérez un fichier image stocké sur le disque.

Examinez à nouveau la classe Bitmap, trouvez un constructeur permettant de créer un Bitmap à partir d'un fichier image sur le disque.

Dessinez l'image choisie à la position définie par l'utilisateur. Pour éviter tout problème de taille, faites en sorte qu'elle ait une largeur de 200 et un ratio préservé pour la hauteur. Recherchez dans Graphics une fonction permettant de dessiner un Bitmap. Attention, rappelez-vous que cette classe possède une superclasse abstraite... cela n'est pas dit par hasard.

Pour sélectionner dynamiquement le fichier, nous allons utiliser la classe OpenFileDialog. Faites en sorte que l'utilisateur puisse filtrer les fichiers de type JPG ou BMP et que JPG soit le type par défaut.

Une fois le fichier sélectionné, examinez ensuite les propriétés d'OpenFileDialog pour récupérer le chemin choisi ainsi que le nom du fichier.

Dessiner une ellipse

Nous allons uniquement tester le dessin d'une ellipse simple, cependant, nous allons faire varier l'épaisseur du trait.

Pour cela, nous allons utiliser un composant NumericUpDown. Certaines de ses méthodes permettent de fixer la valeur minimale et maximale autorisée, l'incrément entre chaque valeur et la valeur numérique contenu dans le composant au lancement. Retrouvez chacune de ces méthodes.

Sauvegarder l'image courante

La classe Bitmap intègre une fonction de sauvegarde prenant en paramètre le nom du fichier de sortie. L'image finale sera sauvegardée ici uniquement sous forme Bitmap.

De manière symétrique à OpenFileDialog, il existe une classe SaveFileDialog.

Pour changer, plutôt que de créer un bouton pour lancer l'action, nous allons passer par un menu. Dans la boîte à outils, choisissez le composant MenuStrip et glissez le sur votre fenêtre. Cliquez dans la case vide pour créer le menu Fichier. Cliquez dans les cases apparaissant pour créer le sous-menu Sauvegarder. Double-cliquez sur le menu Sauvegarder pour créer une fonction répondant à la sélection de ce sous-menu.

X. Gestion de liste

Utilisez un composant TabControl pour créer des onglets. Ce contrôle doit recouvrir quasiment la taille de la fenêtre. Il va donc recouvrir l'interface de votre Paint. Pas de soucis, les éléments peuvent être transférés à l'intérieur très facilement. Sélectionnez l'ensemble des contrôles de votre projet Paint, effectuez un couper, créez le composant Tabcontrol et effectuez un coller.

Les différents onglets ne génèrent pas des parties différentes dans le code ou ne créent pas des fichiers supplémentaires. Les onglets sont comme les Panels : une simple présentation de la form actuelle. Ainsi, l'ensemble des contrôles créés sur les différents onglets sont présents dans la classe Form1. Si vous créez un bouton sur chaque onglet, ils vont s'appeler button1 button2 button3... exactement comme s'ils avaient été créés sans que les onglets existent.

Gestion d'une liste

Créez une ListBox dans ce deuxième onglet. Créez un nouveau bouton qui lit le contenu d'une TextBox et l'insère dans votre ListBox. Une ListBox permet d'afficher plusieurs éléments, elle gère une liste d'entrées que vous pouvez manipuler. Ainsi votre objet ListBox ne possède pas directement de méthode "Insérer". Il faudra utiliser une approche indirecte de la forme MaListBox.MaList.Insérer(...). Il faut trouver dans ListBox la propriété correspondante. Cette propriété vous donnera aussi le type de liste qui gère les entrées d'une ListBox. En lisant les informations sur cette classe, vous saurez les manipulations que l'on peut faire.

Créez une ComboBox. L'intérêt d'une ComboBox est de fournir à l'utilisateur une liste de choix, ainsi pour le genre d'une personne on peut lui proposer "Madame,

Monsieur, Mademoiselle...". Faîtes en sorte que votre ComboBox contienne quelques prénoms usuels et que l'un d'entre eux soit sélectionné par défaut.

Maintenant, lorsque vous cliquez sur le bouton insertion, on doit ajouter dans la ListBox l'entrée suivante : Prénom choisi _ numéro de création _ info supplémentaire contenue dans la TextBox.

Gestion des sélections

Dans les propriétés de la ListBox, activez la sélection multiple en modifiant le paramètre SelectionMode. Maintenant en cliquant avec la souris, vous pouvez sélectionner différents éléments. Les éléments sélectionnés apparaissent en bleuté.

Créez une deuxième ListBox sur la droite de la première. Entre les deux, créez un bouton contenant le symbole ">". En cliquant sur ce bouton, vous devez transférer les éléments sélectionnés dans la ListBox de gauche dans celle de droite. Faîtes attention, l'information indiquant les éléments sélectionnés ne se trouve pas dans la liste des éléments ! Il faudra la rechercher dans les propriétés du composant ListBox.

Créez un bouton RAZ, qui fait repasser l'ensemble des éléments de la ListBox de droite dans la ListBox de gauche.

Créez un bouton Remove qui supprime l'ensemble des éléments sélectionnés de la ListBox de gauche.

XI. Indicateurs et Jauges

Créez un troisième onglet en effectuant un clic droit sur les onglets actuels dans le designer.

Créez une TrackBar. Ce composant est similaire à un NumericUpDown. Donnez-lui comme plage de 0 à 100% comme s'il s'agissait d'un bouton pour régler le volume. Faites en sorte que sa graduation soit de 10 en 10. Lorsque vous faîtes varier le curseur de la TrackBar, faîtes en sorte qu'un label affiche la valeur sélectionnée.

Créez une ProgressBar servant à afficher une jauge allant de 0% à 100%. Lorsque vous faîtes varier le curseur de la TrackBar, faîtes en sorte que votre jauge varie en fonction.

Créez un composant HScrollBar correspondant à une barre de défilement horizontale. Son comportement est identique à un NumericUpDown.

Créez une deuxième ProgressBar servant à afficher une jauge allant de 0% à 100% en rapport avec la position du composant HScrollBar. Faîtes en sorte que sa couleur soit verte.

XII. Travail à rendre

En deux semaines, l'ensemble de votre répertoire SVN sera téléchargé. L'absence du travail sur le dépôt SVN sera sanctionné par une note nulle. A la racine de votre répertoire SVN doit se trouver un répertoire contenant votre projet visual studio. Le projet doit pouvoir être recompilé totalement.