Cours C#

02.04.2003

Introduction

Tout le code de cette documentation a été fait avec le bloc-note et avec le Framework .NET.
Pour pouvoir utilisé cette documentation vous devez être capable d'installer le Framework .NET.

Cette documentation fonctionne comme ceci :
- Tous les mots en bleu/italique sont des variables et peuvent avoir des noms différents.
- Tous les mots en vert/italique sont des noms se rapportant au C++.
- Tous les mots en violet/italique sont des noms se rapportant au C#.

Qu'est-ce que le C# ?

C# doit beaucoup au vaste héritage de C++. Les développeur C++ et Java ne se sentiront pas dépaysé. Langage moderne, intuitif, orienté objet, C# comporte des améliorations substantielles, par exemple un système de types regroupé ou le code "unsafe" pour une maîtrise maximale. Le langage est également enrichi de nouvelles constructions que la majorité des développeurs comprendront aisément.

Les Caractéristiques du C# par rapport au C++

- orientation objet prononcé tout doit être incorporé dans les classes.
- libération automatique des objets.
- disparition des pointeurs.
- remplacement des pointeurs par des références.
- disparition du passage d'argument par adresse au profit du passage par référence.
- nouvelles manipulations des tableaux.
- passage de tableaux en arguments.
- nouvelles manières d'écrire les boucles.
- disparition de l'héritage multiple mais possibilité d'implémenter plusieurs interfaces dans une classe.

Syntaxe principal

Syntaxe :

```
using System;

class SyntaxePrincipal
{
        public static void Main()
        {
        }
}
```

Explications :

La première ligne (*using System*) signale que l'on fera appel à des fonctions de l'architecture .NET regroupées dans un espace de noms (namespace) appelé *System*, ce qui serait égal en C++ à un *#include*.
Ensuite vient la partie principale du langage C#, avec la classe (*SyntaxePrincipal*) et notre méthode principale (*Main()*).
Le nom de la classe peut avoir n'importe quel nom (test, salut1, premier_programme), mais doit commencer par une lettre et n'ose pas avoir d'espace. (1test, premier programme) ? ces deux noms vont créer une erreur à la compilation.

Compilation :

Pour compiler le programme, allez dans l'invite de commande et tapez : csc *NomDuFichier*.cs
Remarquez qu'un fichier C# prend l'extension *.cs*, donc écrivez votre programme dans le bloc-note et enregistrez-le avec l'extension *.cs.*

Premier programme

Nous allons commencer par un petit programme nous permettant d'afficher un message.

Syntaxe :

```
using System;

class PremierProgramme
{
        public static void Main()
        {
                // Ceci est un commentaire
                Console.Out.WriteLine("Hello World");
        }
}
```

Explications :

Nous avons un commentaire (qui se crée avec 2 slaches (*//*)), et la fonction permettant d'afficher quelque chose à l'écran (*Console.Out.WriteLine()*). Et vous voilà vous êtes capable d'afficher quelque chose à l'écran.

Mots réservés du langage c#

Ces mots n'osent pas être utilisés comme variable, car comme leur nom l'indique ce sont des mots réservés.

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
in	int	interface	internal
is	lock	long	namespace
new	null	object	operator
out	override	params	private
protected	public	readonly	ref
return	sbyte	sealed	short
sizeof	stack	alloc	static
string	struct	switch	this
throw	true	try	typeof
uint	ulong	unchecked	unsafe
ushort	using	vitual	void
while			

Types de données

Les types de données sont utilisés pour « identifier » les variables.

Type de données	Description	Exemple d'utilisation
int	Nombres entier	int count ; count = 42 ;
long	Nombres entier (plage plus étendu)	long count ; count = 42 ;
float	Nombres à virgule flottante	float count; count = 3.14F ;
double	Nombres à virgule flottante en double précision	double count ; count = 0.42 ;
decimal	Valeurs monétaires	decimal count ; count = 0.42M ;
string	Chaîne de caractères	string count ; count = "42" ;
char	Caractère unique	char count ; count = '4' ;
bool	Booléen (*true* ou *false*)	bool count ; count = true ;

Les tableaux

Déclaration et affichage:

```
int[ ] count = {1,2,3,...} ;
Console.Out.WriteLine(count[1]) ;
```

Seulement la deuxième case du tableau sera affiché c'est-à-dire 2 (on peut aussi créer des tableau de type string,...).

```
foreach (int row in count)
{
        Console.Out.WriteLine(row);
}
```

Ici le tableau sera totalement affiché grâce au « foreach » et affiché à l'écran.

```
for (int i = 0 ; i != count.Length ; i++)
{
        Console.Out.WriteLine(count[i]) ;
}
```

Ici le tableau sera totalement affiché grâce au « for » et affiché à l'écran.

Propriété « Length » :

Cette propriété nous donne le nombre de cases d'un tableau.
count.**Length ;**

Remplacer des valeurs dans le tableau :

int[ ] *count* **= {1,2,3,...} ;**
count.**SetValue(**<*Nouvelle valeur*>**,** <*Numéro de case*>**) ;**

Il existe encore beaucoup de propriété pour le tableau, mais je ne montre que les bases.

Conversion

Conversions explicites :

La conversion explicite permet de convertir un plus grand type vers un plus petit type (ex. de long vers byte).

Syntaxe :
(type)expression

Exemple:
byte *a*;
long *b* **= 4564;**
a **= (byte)***b*;
L'exemple ci-dessus prend la variable b, qui est de type long et la convertit en byte.

Méthode « Parse » :

La méthode « Parse » permet de convertir une string en une valeur dont le type est différent de string.

Si nous voulons convertir une string représentant une valeur de type double (par exemple 3.14), nous ferons ainsi :

string *strPi* **= "3.14" ;**
double *pi* **= Double.Parse(***strPi***) ;**

Méthode « ToString » :

La méthode « ToString » permet de convertir une valeur numérique en string.

Double *pi* **= 3.14 ;**
string *strPi* **=** *pi*.**ToString() ;**

Séquence d’échappement

« \" »

Permet d’afficher un guillemet.

```
Console.Out.WriteLine("Voila un guillemet \" ") ;
```
Cela donne :
Voila un guillemet "

« \n »

Renvoie le texte à la ligne.

```
Console.Out.WriteLine("A la ligne\nNouvelle ligne ") ;
```
Cela donne :
A la ligne
Nouvelle ligne

Lecture des données

```
string entre = Console.In.ReadLine() ;
int nombre = Int32.Parse(entre) ;
```
Attention l’entrée doit être un nombre entier, sinon le programme va générer une erreur.

Une solution pour la gestion des erreur sur l’entrée est d’ajouter un bloc « try-catch ».
```
try
{
        string entre = Console.In.ReadLine() ;
        int nombre = Int32.Parse(entre) ;
}
catch
{
        Console.Out.WriteLine("Erreur sur le nombre") ;
}
```
Si l’utilisateur entre autre chose qu’un entier le message « Erreur sur le nombre » apparaît.

Le bloc « try-catch » peut être utilisé n’importe où.

Voilà vous êtes capable d’entrer quelque chose, et de contrôler des erreur.

Opérateurs

Opérateur	Description	Exemple d'utilisation
*	multiplication	multiplication = 3*4 ;
/	division	division = 4/2 ;
+	addition	addition = 3+2 ;
-	soustraction	soustraction = 4-3 ;
%	modulo (reste)	modulo = 4%3 ;
!	unaire (pas)	!unaire ;
<	inférieur à	inferieur1 < inferieur2 ;
<=	inférieur ou égal à	inferieur1 <= inferieur2 ;
>	supérieur à	superieur1 > superieur2 ;
>=	supérieur ou égal à	superieur1 >= superieur2 ;
==	égal à	egal1 == egal2 ;
!=	n'est pas égal à	pasEgal1 != pasEgal2 ;
=	affectation	affectation = 3 ;

Conditions

L'instruction « if »

```
int premier = 1;
int deuxieme = 2 ;

if (premier == deuxieme)
{
        Console.Out.WriteLine("premier est égal à deuxieme") ;
}
```

le message n'apparaîtra jamais car 1 ne sera jamais égal à deux, donc il nous faut une autre instruction (else)

```
int premier = 1;
int deuxieme = 2 ;

if (premier == deuxieme)
{
        Console.Out.WriteLine("premier est égal à deuxieme") ;
}
else
{
        Console.Out.WriteLine("premier n'est égal à deuxieme") ;
}
```

Ici le deuxième message sera afficher car 1 n'est pas égal à 2.

L'instruction « else if »

```
int premier = 1;
int deuxieme = 2 ;

if (premier == deuxieme)
{
        Console.Out.WriteLine("premier est égal à deuxieme") ;
}
else if (premier < deuxieme)
{
        Console.Out.WriteLine("premier est inférieur à deuxieme") ;
}
else
{
        Console.Out.WriteLine("premier n'est égal à deuxieme") ;
}
```

Ici le deuxième message sera afficher car 1 est inférieur à 2 et comme else if est avant else celui-ci est ignoré (dans notre exemple).

L'instruction « switch »

```
int nombre = 2;
switch (nombre)
{
        case 0 :
                Console.Out.WriteLine("nombre est 0") ;
                break;
        case 1 :
                Console.Out.WriteLine("nombre est 1") ;
                break ;
        case 2 :
                Console.Out.WriteLine("nombre est 2") ;
                break;
        default :
                Console.Out.WriteLine("nombre est autre chose que 0,1 et 2") ;
                break ;
}
```

Ici le 3ème message sera affiché car *nombre* est égal à 2.

Les boucles

L'instruction « for »

```
for (int i = 0 ; i <= 10 ; i++)
{
        Console.Out.WriteLine(i) ;
}
```

Ici tant que *i* ne sera pas supérieur à 10 il va être affiché.

L'instruction « while »

```
int i = 0 ;
while (i <= 10)
{
        Console.Out.WriteLine(i) ;
        i++ ;
}
```

Ici tant que *i* ne sera pas supérieur à 10 il va être affiché.

L'instruction « do...while »

```
int i = 0 ;
do
{
        Console.Out.WriteLine(i) ;
        i++ ;
}
while (i <= 10) ;
```

Ici tant que *i* ne sera pas supérieur à 10 il va être affiché. Faites attention *i* sera de toute façon au moins affiché une fois, car ici le test et fait à la fin.

Résumé

Vous êtes capable maintenant de :

- Écrire la syntaxe d'un programme.
- Afficher quelque chose à l'écran.
- Entrer quelque chose à l'écran.
- Créer un tableau.
- Et utiliser les instructions de test ou les boucles.

Maintenant à vous d'avoir des idées et de « grailler » pour vos prochains programmes. Maintenant ce qui suit a été entièrement réalisé avec Visual Studio .NET.

1.1 Faire apparaître une nouvelle Form :

```
Form <Nom de la Form> = new <Nom de la Form>();
<Nom de la Form>.Show();
```

1.2 Cacher une Form :

```
<Nom de la Form>.ActiveForm.Hide();
```

1.3 Fermer une Form :

```
<Nom de la Form>.ActiveForm.Close();
```

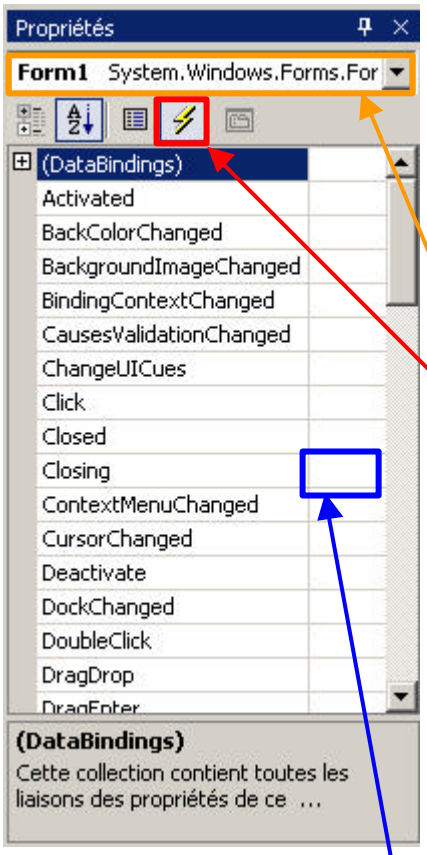
1.4 Arrêter une application :

```
Application.Exit();
```

2. Événements

2.1 Créer un nouvel événement

Cette explication permet de créer un autre événement que celui par défaut (événement par défaut pour une *form* : ***Load*** et pour un *bouton* : ***Click***,...), sans devoir tout copier le code soi-même.



Choisissez votre élément, cliquez sur et pour finir double-cliquez sur l'espace vide à droite de l'événement désiré.

3. Boîte de dialogue « MessageBox »

3.1 MessageBox par défaut :

MessageBox.Show("*MessageBox par défaut***") ;**

3.2 MessageBox avec un titre :

MessageBox.Show("*MessageBox avec un titre***", "***Avec un titre***") ;**

3.3 MessageBox avec une icône :

MessageBox.Show("*MessageBox avec une icône***", "***Avec une icône***",**
MessageBoxButtons.OK, ***MessageBoxIcon.Exclamation***) ;**

3.4 Vérification :

Ce code permet de savoir sur quel bouton l'utilisateur a cliqué.

DialogResult ***key*** **= MessageBox.Show("***Oui ou Non***", "***Résultat***",**
MessageBoxButtons.YesNo);
bool ***Oui*** **= (***key*** == DialogResult.Yes) ;**
Si l'utilisateur clique sur « oui » alors la variable *Oui* sera à *true* sinon elle sera à *false.*

4. Random

4.1 Fonction Random :

La fonction Random permet de sortir un nombre au hasard compris entre un nombre minimal (int min) et un nombre maximal (int max).

Random *<Nom de la variable1>* **= new Random();**
int *<Nom de la variable2>* **=** *<Nom de la variable1>***.Next(int min, int max);**

Attention pour avoir le vrai nombre maximal il faut faire « int max – 1 ».

5. Date

6.1 Quelques fonctions :

6.1.1

int *<Nom de la variable>* **= DateTime.DaysInMonth(int year, int month);**
Retourne le nombre de jour du mois(int month(compris entre 1 et 12)) de l'année(int year).

bool *<Nom de la variable>* **= DateTime.IsLeapYear(int year) ;**
Retourne si l'année(int year) est une année bissextile.

6.1.2

DateTime *<Nom de la variable>* **= DateTime.Now ;**
Retourne la date et l'heure actuel.
<Nom de la variable> **.** *<Choisir une option>* **;**
Permet de choisir des options pour les dates.

6.1.3

DateTime *<Nom de la variable>***;**
<Nom de la variable> **.** *<Choisir une option>* **;**
Permet de travailler avec les dates(ex. addition de date,...).

Faire attention à la conversion ! ! !

7. Ouverture d'un programme

7.1 Fonction Process.Start() :

Tout d'abord insérer la directive(tout en haut du code) *« using System.Diagnostics; »*.

Permet l'ouverture d'un programme quelconque installé sur la machine.

Process.Start("*<Chemin + nom du programme>***") ;**

OU

Process *<Nom de la variable>* **= new Process() ;**
*<Nom de la variable>***.StartInfo = new ProcessStartInfo() ;**
*<Nom de la variable>***.StartInfo.FileName = "***<Chemin + nom du programme>***" ;**
*<Nom de la variable>***.Start() ;**

<Nom de la variable> doit être le même partout ! ! !

8. Information Système

8.1 Trouver les informations système

Ce code permet de trouver certaines informations système.

SystemInformation.*<Choisir une option>* **;**

9. Gestion des fichiers

9.1 Ouvrir :

Permet d'ouvrir n'importe quelle sorte de fichiers.

```
FileStream File  = new FileStream("<Chemin + nom du fichier>", FileMode.Open,
FileAccess.ReadWrite) ;

int nombreCaractere ;
int tailleFichier  = (int)File.Length ;
byte[ ] contenu = new byte[tailleFichier] ;
nombreCaractere = File.Read(contenu, 0, tailleFichier) ;

for (int i = 0 ; i  <  tailleFichier ; i++)
{
        text1.Text += (char)contenu[ i ] ;
}

File.Close() ;
```

Les mots en italique sont des variables, donc ils ne doivent pas forcément être les mêmes que dans l'exemple.

9.2 Enregistrer :

Permet d'enregistrer n'importe quel sorte de fichiers.

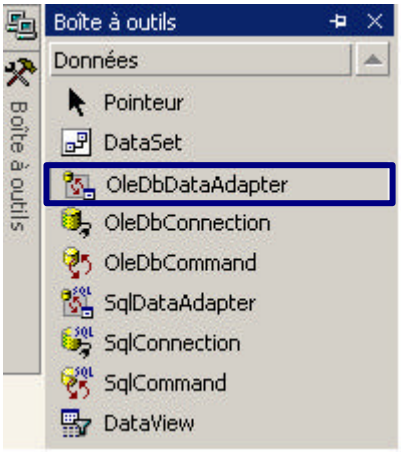
```
FileStream File  = new FileStream("<Chemin + nom du fichier>", FileMode.Create,
FileAccess.Write) ;
StreamWriter flux = new StreamWriter(File) ;

string contenu = text1.Text ;
flux.Write(contenu) ;

flux.Close() ;
File.Close() ;
```

Les mots en italique sont des variables, donc ils ne doivent pas forcément être les mêmes que dans l'exemple.

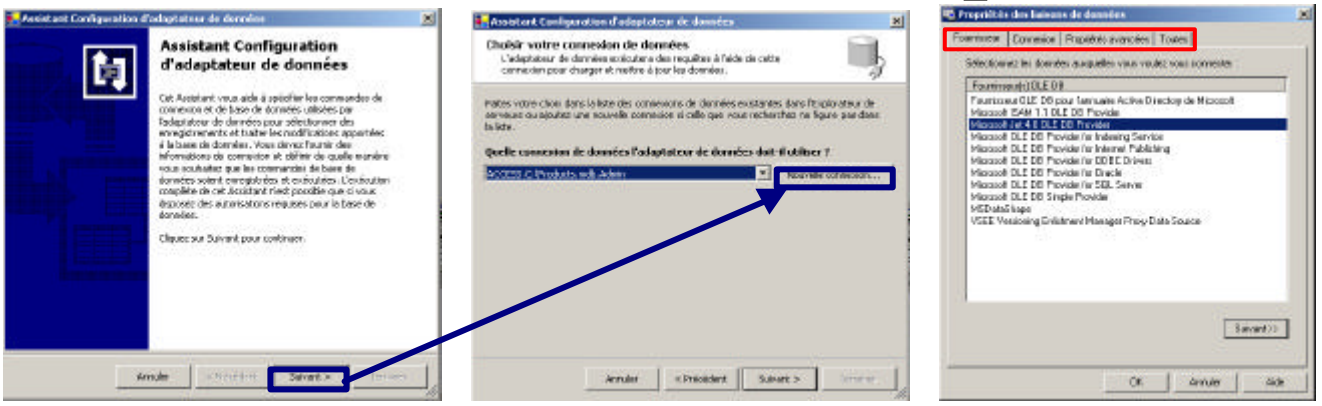
10. Base de données

10.1 Ajout des objets :

Ajoutez un objet « *OleDbDataAdapter* » comme le montre l'image ci-dessous.



Après cela un assistant s'ouvre, suivez les instructions pour créer totalement le contrôle souhaité. Cet assistant propose plusieurs options comme par exemple le choix de la base de données (Access, SQL,...), comme le montre l'image ci-dessous.



Fournisseur – Connexion – Propriétés avancées(laisser par défaut) – Toutes(laisser par défaut)

Ensuite générez un groupe de données comme le montre l'image.

10.2 Afficher des données :

Ce code permet d'afficher le contenu d'une base de données.

Adapter.**Fill(***dataSet***) ;**
Adapter et dataSet sont les noms des différents objets du chapitre 8.1.

text1.Text **=** *dataSet*.**Tables["***<Nom de la cache du dataSet>***"].Rows[***<N° de ligne>***]["***<Nom du champ>***].ToString() ;**
Affiche le contenu de la base de données dans *text1.Text*.

10.3 Modifier des données :

Ce code permet de modifier le contenu d'une base de données.

dataSet.Tables["*<Nom de la cache du dataSet>*"].Rows[*<N° de ligne>*]["*<Nom du champ>*] = *text1.Text*;
Le contenu de la base de données prend la valeur de text1.Text

Adapter.Update(*dataSet*) ;
Adapter et dataSet sont les noms des différents objets du chapitre 8.1.

10.4 Supprimer un enregistrement :

Ce code permet de supprimer un enregistrement d'une base de données.

dataSet.Tables["*<Nom de la cache du dataSet>*"].Rows[*<N° de ligne>*].Delete() ;
Adapter.Update(*dataSet*) ;

10.5 Ajouter un enregistrement :

Ce code permet d'ajouter un enregistrement dans une base de données.

DataRow *Enregistrement* = *dataSet*.Tables["*<Nom de la cache du dataSet>*"].NewRow() ;
Enregistrement["*<Nom du champ>* "] = *text1*.Text ;
dataSet.Tables["*<Nom de la cache du dataSet>*"].Rows.Add(*Enregistrement*) ;
Adapter.Update(*dataSet*) ;

10.6 Création d'un base de données avec le code :

La procédure du haut est un peu pour les débutants. Il y a une autre façon de gérer une base de données un peu plus professionnellement (en se connectant et en gérant la base de données seulement avec du code).

10.6.1

Ajoutez la directive *using System.Data.OleDb;*.

Ensuite ajoutez les 3 variables suivantes dans la classe principal juste en dessus du constructeur :

```
private int row;
private DataSet dataSet ;
private OleDbDataAdapter adapter ;
```

10.6.2

1. Créez une méthode pour la connexion (dans l'exemple *Connexion*).

```
private void Connexion()
{
        string connection = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=<Nom + chemin de la base de données>" ;
        OleDbConnection myConnection = new OleDbConnection(connection) ;
        myConnection.Open() ;
        string requete = "SELECT * FROM <Nom de la table>" ;
        dataSet = new DataSet() ;
        adapter = new OleDbDataAdapter(requete, myConnection) ;
        OleDbCommandBuilder command = new OleDbCommandBuilder(adapter) ;
        adapter.Fill(dataSet, "<Nom de la cache du dataSet>") ;
        myConnection.Close() ;
}
```

2. Créez une méthode pour la lecture (dans l'exemple *Lecture*).

```
private void Lecture()
{
        text1.Text = dataSet.Tables["<Nom de la cache du dataSet>"].Rows[row][ "<Nom du champ1>"].ToString() ;

        text2.Text = dataSet.Tables["<Nom de la cache du dataSet>"].Rows[row][ "<Nom du champ2>"].ToString() ;
}
```

3. Insérez la méthode *Connexion* et *Lecture* dans l'événement *Form_Load* et affectez la valeur 0 à *row*. Comme ci-dessous :

```
row = 0 ;
Connexion() ;
Lecture() ;
```

10.6.3

Pour updater les champs de la base de données, insérez le code ci-dessous dans un bouton par exemple.

```
dataSet.Tables["<Nom de la cache du dataSet>"].Rows[row]["<Nom du champ1>"] = text1.Text ;

dataSet.Tables["<Nom de la cache du dataSet>"].Rows[row]["<Nom du champ2>"] = text2.Text ;

adapter.Update(dataSet, "<Nom de la cache du dataSet>") ;
```

10.6.4

Pour supprimer les champs de la base de données, insérez le code ci-dessous dans un bouton par exemple.

```
dataSet.Tables["<Nom de la cache du dataset >"].Rows[row].Delete() ;

adapter.Update(dataSet, "<Nom de la cache du dataSet>") ;
```

10.6.5

Pour ajouter des champs de la base de données, insérez le code ci-dessous dans un bouton par exemple.

```
DataRow Enregistrement = dataSet.Tables["<Nom de la cache du dataSet>"].NewRow() ;

Enregistrement["<Nom du champ1>"] = text1.Text ;
Enregistrement["<Nom du champ2>"] = text2.Text ;

dataSet.Tables["<Nom de la cache du dataSet>"].Rows.Add(Enregistrement) ;

adapter.Update(dataSet, "<Nom de la cache du dataSet>") ;
```

Je conseille de donner comme nom *cache* au *<Nom de la cache du dataSet>*. Toutes les variables écrites en italique bleu peuvent avoir un autre nom.

10.6.6

Je conseille de mettre un bloc « try-catch » dans chaque instruction pour gérer les différentes erreur.

11. Réseau

11.1 Serveur :

Le serveur attend un client pour ensuite accepter la connexion.

Il faut ajouter les directives suivantes pour que ça fonctionne :

```
using System.Net ;
using System.Net.Sockets ;
using System.Threading ;
```

```
Thread MyThread = new Thread(new ThreadStart(Ecoute)) ;
MyThread.Start() ;
```

Ce code permet de ne pas faire planter l'application. A insérer par exemple dans un bouton.

Ensuite créez une nouvelle méthode (dans l'exemple « Ecoute »).

```
public void Ecoute()
{
    try
    {
        int port = 5555 ;
        TcpListener SocketEcoute = new TcpListener(port) ;
        SocketEcoute.Start() ;
        Socket MySocket = SocketEcoute.AcceptSocket() ;
        MessageBox.Show("Connexion établie") ;
    }

    catch
    {
        MessageBox.Show("Erreur") ;
    }
}
```

Le bloc « try » se produit lorsqu'il n'y pas d'erreur, mais dès que la connexion subit une erreur quelconque le bloc « catch » se produit.

11.2 Client :

Permet de se connecter à un serveur grâce à l'adresse IP.

Il faut ajouter les directives suivantes pour que ça fonctionne :

```
using System.Net ;
using System.Net.Sockets ;
```

```
try
{
    int port = 5555 ;
    TcpClient ClientSocket = new TcpClient("<IP>", port) ;
    NetworkStream ns = ClientSocket.GetStream() ;
    MessageBox.Show("Connexion établie") ;
}

catch
{
    MessageBox.Show("Connexion non établie") ;
}
```

Code à insérer par exemple dans un bouton.

11.3 Trouver l'adresse IP

Ce code permet de trouver l'adresse IP en fonction du HostName en la stockant dans un label.

```
IPHostEntry ipEntry = Dns.GetHostByName ("<HostName>");
IPAddress[] IpAddr = ipEntry.AddressList;

for (int i = 0; i < IpAddr.Length; i++)
{
        label1.Text += IpAddr[i].ToString() ;
}
```

12. Texte

12.1 Sélectionner un caractère :

Ce code permet de sélectionner du texte (*textBox*.Text) caractère par caractère, et de le copier dans un label (*label1*.Text).

```
string texte = textBox.Text ;
char caractere ;

for (int i = 0 ; i < texte.Length ; i++)
{
    caractere = texte[i] ;
    label1.Text += caractere ;
}
```

13. Classe

13.1 Création d'une classe :

Ce code permet de créer une classe avec une méthode de type « double ».

```
class Mathematique
{
    public static double addition(double d)
    {
            d = d + 2 ;
            return d ;
    }
}
```

13.2 Accéder à une méthode :

Ce code permet d'accéder à la méthode de la classe du chapitre 15.1 et de stocker la valeur dans un label.

```
label1.Text = Mathematique.addition(10).ToString ;
```
La sortie sera : (10+2) 12

13.3 Méthode de retour :

Ce code permet de créer une méthode de retour.

```
public void addition(double d)
{
    d = d + 2 ;
}
```

13.4 Accéder à une méthode de retour :

Ce code permet d'accéder à la méthode de retour du chapitre 15.3.

```
addition(10) ;
```

13.5 Accesseurs « get – set » :

Ce code permet de créer la méthode « get – set ».

```
class GetSet
{
    int Variable1 ;
    public int Exemple
    {
            get
            {
                    return Variable1 ;
            }
            set
            {
                    Variable1 = value ;
            }
    }
}
```

« get » permet la lecture et « set » l'écriture.

13.6 Appel des accesseurs « get – set » :

Ce code permet d'appeler la méthode « get – set », de modifier *Variable1* du chapitre 15.3 et de l'afficher.

```
GetSet Variable2 = new GetSet() ;

Variable2.Exemple = 2 ;
```

Modifie *Variable1* grâce au « set »

```
label1.Text = Variable2.Exemple.ToString() ;
```

Affiche *Variable1* dans *label1* grâce au « get »

13.7 Bloc « try – catch » :

Le bloc « try – catch » permet de contrôler les erreurs d'une application.

« try » se produit lorsqu'il n'y pas d'erreur, et dès que l'application subit une erreur quelconque l'instruction « catch » se produit.

```
try
{
    // code
}

catch
{
    // code
}
```

Ce bloc est vraiment très pratique, donc n'hésitez pas de à l'insérer un peu partout.

14. Menu contextuel

14.1 Création d'un menu contextuel :

Pour créer un menu contextuel, ajoutez l'élément « *ContextMenu* », puis en haut du formulaire un menu apparaît avec comme titre « *Menu contextuel* », cliquez sur ce titre. Ensuite en dessous du titre une zone « *Tapez ici* » apparaît, ajoutez vos « *menus* » à cet emplacement avec les noms voulus et double-cliquez sur ceux-ci pour ajouter un événement (« *Click* » par défaut).

14.2 Affichage d'un menu contextuel :

Pour pouvoir afficher vos « *menus* », allez dans la propriété « *ContextMenu* » d'un « *textBox* » par exemple et sélectionnez le nom de votre contrôle (*contextMenu1* par défaut).
Lorsque vous êtes dans l'application, cliquez avec le bouton droit de la souris sur ce « *textBox* » pour afficher les menus.

15. Barre des tâches

Cette procédure permet d'afficher une icône de votre application dans la barre des tâches.

15.1 Insertion du contrôle :

Ajoutez simplement l'élément « *NotifyIcon* » de la boîte à outils.

15.2 Affichage de l'icône :

Il suffit juste de mettre sa propriété « *Visible* » à « *true* » et le tour est joué.

16. Navigateur Web

Cette procédure permet de créer un navigateur web.

16.1 Insertion du contrôle :

Personnalisez la boîte à outils en ajoutant l'élément « *Navigateur Web Microsoft* » et insérez-le dans votre « *Form* ».

16.2 Accéder à un site

Ce code permet de se connecter à un site web (à insérer par exemple dans un bouton).

```
System.Object nullObject = 0 ;
System.Object nullObjStr = "" ;

NavigateurWeb.Navigate("<URL>", ref nullObject, ref nullObjStr, ref nullObjStr,
ref nullObjStr) ;
```

16.3 Autres propriétés

Il y a plusieurs autres propriétés pour le navigateur web.

```
NavigateurWeb.<Choisir la propriété> ;
```

17. Liens

Tous ces liens sont des sites contenant des codes-sources.

http://www.csharpfr.com
http://www.csharphelp.com/index.html
http://www.codeproject.com/csharp/
http://www.c-sharpcorner.com/