

Les chaînes de caractères

(J-C Armici, 2006)

D'autres documents et exemples concernant la programmation, sur les sites:
www.unvrai.com, www.unfaux.com, ica.developpez.com et www.developpez.com

Ce document basé sur le Framework .NET 2.0 et Visual Studio 2005.
--

Les chaînes de caractères	1
Introduction.....	2
Création de strings	2
Méthode ToString()	2
Manipulation de chaînes de caractères.....	2
Description des principaux membres	2
Quelques exemples.....	4

Introduction

C# offre une gestion complète des chaînes de caractères (string) et les traite en tant qu'objets qui encapsulent toutes les manipulations telles que tris, recherches, etc.

Création de strings

La manière la plus commune de créer un string est de lui assigner une chaîne littérale (entre guillemets):

```
string prenom = "Alfred";
```

Une chaîne littérale peut contenir des caractères d'"échappement" tels que \n (nouvelle ligne) ou \t (tabulation).

Pour introduire le caractère \ dans un string, il faut le doubler: \\.

Une particularité intéressante et fort pratique de C# est pouvoir faire précéder une chaîne littérale par le caractère @. Dans ce cas le contenu de la chaîne qui suit est pris de manière littérale, y compris les sauts de lignes:

```
string chemin1 = "c:\\temp\\test.txt"
```

est équivalent à:

```
string chemin1 = @"c:\temp\test.txt"
```

et

```
string ligne1 = "Ligne un \nLigne deux";
```

est équivalent à:

```
string ligne2 = @"Ligne un  
Ligne deux";
```

Méthode ToString()

Une autre facilité offerte par C# est la méthode ToString() dont disposent tous les types prédéfinis. Cette méthode permet, par exemple, de convertir une valeur numérique en chaîne de caractères:

```
double nombre=3.45;  
string chaine=nombre.ToString();
```

Il est également possible d'écrire:

```
128.ToString();
```

Manipulation de chaînes de caractères

Description des principaux membres

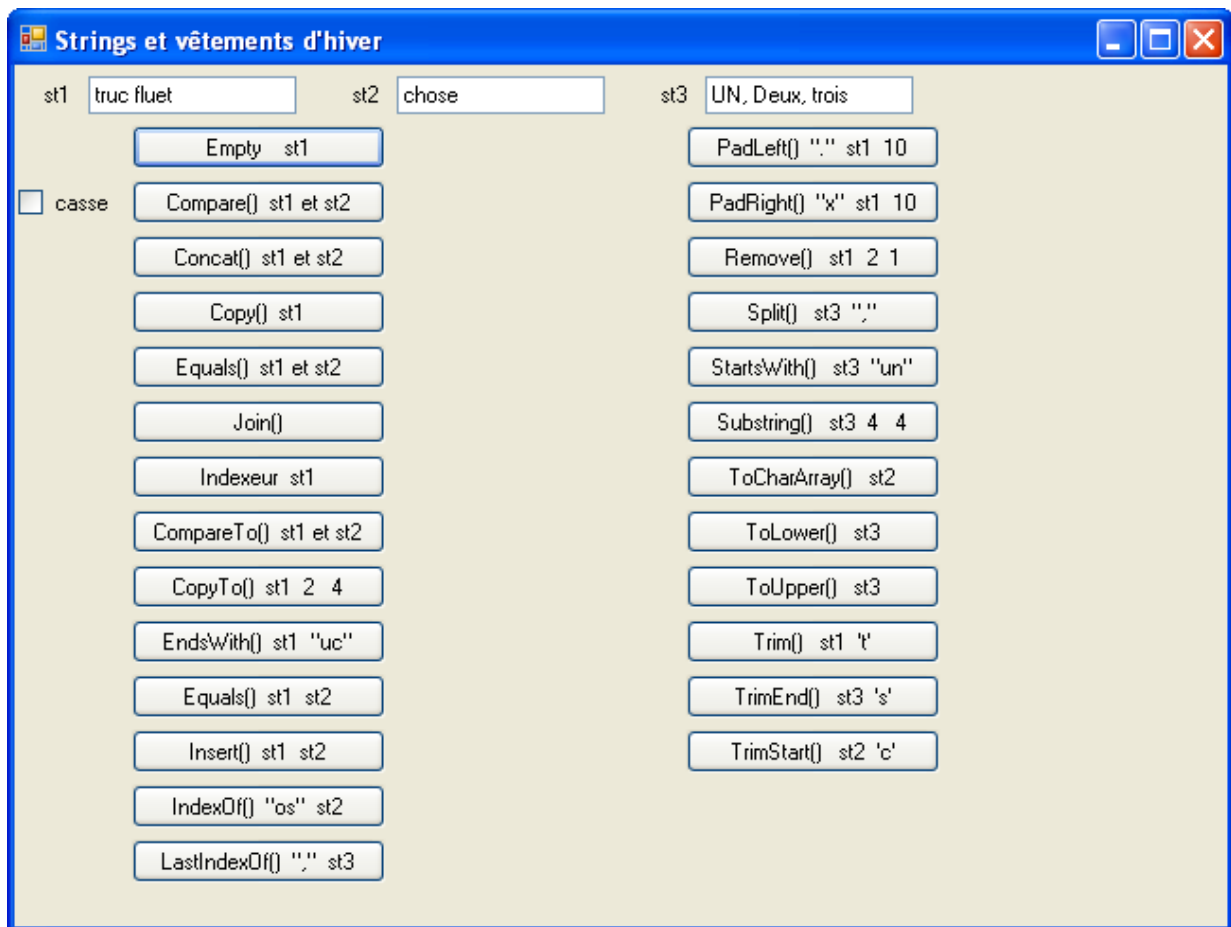
La classe string dispose d'un certain nombre de méthodes, dont les principales sont décrites dans le tableau suivant:

Membres	Description
Empty	Représente une chaîne vide.

Compare()	Compare deux chaînes de caractères.
Concat()	Crée une nouvelle chaîne obtenue par la concaténation de deux ou plusieurs chaînes.
Copy()	Crée une nouvelle chaîne par copie d'une autre chaîne.
Equals()	Détermine si deux chaînes ont la même valeur.
Format()	Permet de formater une chaîne en utilisant des paramètres de formatage.
Join()	Concatène la chaîne spécifiée à l'aide d'un tableau de strings.
Length	Retourne le nombre de caractères d'une chaîne.
CompareTo()	Compare le string courant à un autre.
CopyTo()	Copie le nombre de caractères spécifiés dans un tableau de caractères unicode.
EndsWith()	Détermine si un string spécifié se trouve à la fin du string courant.
Equals()	Détermine si deux chaînes ont la même valeur.
Insert()	Retourne un nouveau string dans lequel a été inséré le string spécifié.
IndexOf()	Retourne la position du caractère spécifié dans la chaîne.
LastIndexOf()	Retourne la dernière position du caractère spécifié dans la chaîne.
PadLeft()	Aligne la chaîne courant à droite dans une zone de taille spécifiée en la complétant à gauche par des espaces ou autre caractère spécifié.
PadRight()	Aligne la chaîne courant à gauche dans une zone de taille spécifiée en la complétant à droite par des espaces ou autre caractère spécifié.
Remove()	Supprime le nombre de caractères spécifié.
Split()	Retourne dans un tableau de strings les sous-chaînes délimitées par un caractère spécifié.
StartsWith()	Détermine si un string spécifié se trouve au début du string courant.
Substring()	Retourne une sous-chaîne depuis une position et d'une longueur spécifiées.
ToCharArray()	Crée un tableau de caractères avec les caractères du string courant.
ToLower()	Retourne le string courant en minuscules.
ToUpper()	Retourne le string courant en majuscules.
Trim()	Supprime tous les espaces (ou autre caractère spécifié) se trouvant au début et à la fin d'une chaîne.
TrimEnd()	Supprime tous les espaces (ou autre caractère spécifié) se trouvant à la fin d'une chaîne.
TrimStart()	Supprime tous les espaces (ou autre caractère spécifié) se trouvant au début d'une chaîne.

Quelques exemples

Nous allons reprendre les éléments du tableau précédent en les illustrant à l'aide d'exemples dans le programme suivant:



Dans ce programme, chaque bouton exécute des instructions mettant en évidence les diverses possibilités de manipulation de chaînes de caractères. Selon les besoins, les trois chaînes st1, st2 et st3 seront utilisées. S'agissant de textBoxes, l'utilisateur a tout loisir d'en modifier le contenu. Le programme ne comportant pas de contrôle d'erreur, des paramètres inappropriés peuvent conduire à des résultats erronés.

Afin de travailler de manière concise, le programme travaille directement sur des chaînes de caractères non pas sous forme de variable de type string, telles que:

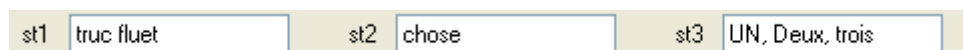
```
string st1, st2;  
st2 = st1.ToUpper();
```

mais plutôt sur les composants, par exemple:

```
lToUpper.Text = txtSt3.Text.ToUpper();
```

ce qui revient absolument au même.

Enfin, les descriptions ci-dessous supposent que les valeurs des trois chaînes d'exemple sont celles par défaut:

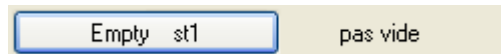


Empty

Code:

```
if (txtSt1.Text == string.Empty)
    lEmpty.Text = "vide";
else
    lEmpty.Text = "pas vide";
```

Résultat:

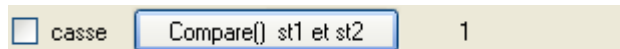
A screenshot of a UI element with a light beige background. It contains a button with a blue border and the text "Empty st1". To the right of the button is the text "pas vide".

Compare()

Code:

```
int resultat;
resultat = string.Compare(txtSt1.Text,txtSt2.Text, cbCasse.Checked);
lCompare.Text = resultat.ToString();
```

Résultat:

A screenshot of a UI element with a light beige background. It contains a checkbox with the label "casse" to its left. To the right of the checkbox is a button with a blue border and the text "Compare() st1 et st2". To the right of the button is the text "1".

Note:

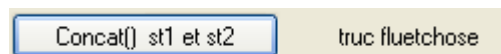
Contrairement à **CompareTo()** cette méthode possède deux paramètres.

Concat()

Code:

```
lConcat.Text = string.Concat(txtSt1.Text, txtSt2.Text);
```

Résultat:

A screenshot of a UI element with a light beige background. It contains a button with a blue border and the text "Concat() st1 et st2". To the right of the button is the text "truc fluetchose".

Note:

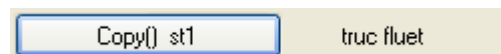
Concat() est équivalente à l'utilisation de l'opérateur +.

Copy()

Code:

```
lCopy.Text = string.Copy(txtSt1.Text);
```

Résultat:

A screenshot of a UI element with a light beige background. It contains a button with a blue border and the text "Copy() st1". To the right of the button is the text "truc fluet".

Equals()

Code:

```
lEquals.Text = txtSt1.Text.Equals(txtSt2.Text).ToString();
```

Résultat:

Equals() st1 et st2 False

Join()

Code:

```
string[] tableau = new string[4] {"Une", "fois", "par", "jour."};  
lJoin.Text = string.Join(" ", tableau);
```

Résultat:

Join() Une fois par jour.

Indexage des caractères

Code:

```
lIndex.Text="";  
string resultat="";  
for (int i = 0; i < txtSt1.Text.Length; i++)    // accès par caractère  
    resultat += txtSt1.Text[i] + "-";  
  
lIndex.Text = resultat.Remove(resultat.Length - 1);    // suppression du  
                                                         // dernier '-'
```

Résultat:

Indexeur st1 t-r-u-c- -f-l-u-e-t

CompareTo()

Code:

```
int resultat;  
resultat = txtSt1.Text.CompareTo(txtSt2.Text);  
lCompareTo.Text = resultat.ToString();
```

Résultat:

CompareTo() st1 et st2 1

CopyTo()

Code:

```
char[] destination = { 'L', 'e', ' ', 't', 'a', 'b', 'l', 'e', 'a',  
                        'u',  
                        ' ', 'd', 'e', ' ', 'c', 'h', 'a', 'r' };  
  
txtSt1.Text.CopyTo(2, destination, 4, 1);  
for (int i=0;i<destination.Length;i++)  
    lCopyTo.Text+=destination[i];
```

Résultat:

CopyTo() st1 2 4 Le tableau de char

EndsWith()

Code:

```
lEndsWith.Text = txtSt1.Text.EndsWith("uc").ToString();
```

Résultat:

EndsWith() st1 "uc"	False
---------------------	-------

Equals()

Code:

```
lEquals2.Text = string.Equals(txtSt1.Text, txtSt2.Text).ToString();
```

Résultat:

Equals() st1 st2	False
------------------	-------

Insert()

Code:

```
lInsert.Text = txtSt1.Text.Insert(3, txtSt2.Text);
```

Résultat:

Insert() st1 st2	truchosec fluet
------------------	-----------------

IndexOf()

Code:

```
lIndexOf.Text = txtSt2.Text.IndexOf("os").ToString();
```

Résultat:

IndexOf() "os" st2	2
--------------------	---

LastIndexOf()

Code:

```
lLastIndexOf.Text = txtSt3.Text.LastIndexOf(",").ToString();
```

Résultat:

LastIndexOf() "," st3	8
-----------------------	---

PadLeft()

Code:

```
lPadLeft.Text = txtSt1.Text.PadLeft(15, '.');
```

Résultat:

PadLeft() "" st1 15truc fluet
---------------------	-----------------

PadRight()

Code:

```
lPadRight.Text = txtSt1.Text.PadRight(15, 'x');
```

Résultat:

PadRight() 'x' st1 15	truc fluetxxxxx
-----------------------	-----------------

Remove()

Code:

```
lRemove.Text = txtSt1.Text.Remove(2, 1);
```

Résultat:

Remove() st1 2 1	trc fluet
------------------	-----------

Note:

Dans cet exemple on supprime 1 caractère à partir de la position 2.

Split()

Code:

```
char[] sep = new char[] { ',' };  
lSplit.Text = "-";  
foreach (string st in txtSt3.Text.Split(sep))  
    lSplit.Text += st + "-";
```

Résultat:

Split() st3 ","	-UN- Deux- trois-
-----------------	-------------------

StartsWith()

Code:

```
lStartsWith.Text = txtSt3.Text.StartsWith("un").ToString();
```

Résultat:

StartsWith() st3 "un"	False
-----------------------	-------

Substring()

Code:


```
lSubstring.Text = txtSt3.Text.Substring(4, 4);
```

Résultat:

Substring() st3 4 4	Deux
---------------------	------

Note:

Dans cet exemple, 4 caractères sont copiés à partir de la position 4.

ToCharArray()

Code:

```
lToCharArray.Text="-";  
foreach (char c in txtSt2.Text.ToCharArray())  
    lToCharArray.Text += c + "-";
```

Résultat:

ToCharArray() st2	-c-h-o-s-e-
-------------------	-------------

Note:

Dans cet exemple on parcourt le tableau de caractère à l'aide d'une boucle **foreach**.

ToLower()

Code:

```
lToLower.Text = txtSt3.Text.ToLower();
```

Résultat:

ToLower() st3	un, deux, trois
---------------	-----------------

ToUpper()

Code:

```
lToUpper.Text = txtSt3.Text.ToUpper();
```

Résultat:

ToUpper() st3	UN, DEUX, TROIS
---------------	-----------------

Trim()

Code:

```
lTrim.Text = txtSt1.Text.Trim('t');
```

Résultat:

Trim() st1 't'	ruc flue
----------------	----------

Note:

Habituellement on utilise les méthodes **Trim()** pour supprimer des espaces, mais il est aussi possible de spécifier le caractère à supprimer, comme dans cet exemple.

TrimEnd()

Code:

```
lTrimEnd.Text = txtSt3.Text.TrimEnd('s');
```

Résultat:

TrimEnd() st3 's'	UN, Deux, troi
-------------------	----------------

TrimStart()

Code:

```
lTrimStart.Text = txtSt2.Text.TrimStart('c');
```

Résultat:

TrimStart() st2 'c'	hose
---------------------	------