Un livre de Wikilivres.

Programmation Visual Basic .NET

Une version à jour et éditable de ce livre est disponible sur Wikilivres,
une bibliothèque de livres pédagogiques, à l'URL :
http://fr.wikibooks.org/wiki/Programmation_Visual_Basic_.NET

Vous avez la permission de copier, distribuer et/ou modifier ce document selon les termes de la Licence de documentation libre GNU, version 1.2 ou plus récente publiée par la Free Software Foundation ; sans sections inaltérables, sans texte de première page de couverture et sans Texte de dernière page de couverture. Une copie de cette licence est inclue dans l'annexe nommée « Licence de documentation libre GNU ».

Introduction

Présentation

Visual Basic .NET fait partie du framework .NET de Microsoft. Inspiré de Visual Basic, il n'assure cependant aucun compatibilité ascendante avec : les scripts VB6 ne seront pas compilé en VB.NET.

Il propose les fonctionnalités suivantes :

Orienté objet

Comme tous les langages .NET, VB.NET supporte pleinement les concepts orientés objets comme l'héritage. Tout est un objet, y compris les primitives (`Short, Integer, Long, String, Boolean`, etc.) les types, et évènements. Tout hérite donc de la classe `Object` de base.

Programmation événementielle

Toutes les précédentes version de Visual Basic géraient les événements, mais cette fonctionnalité a été améliorée dans le framework .NET. En effet, ils ne sont plus reconnus par convention de nommage (*ObjectName_EventName*), mais déclarés par clause *Handles ObjectName.EventName*. Les gestionnaires d'évènements peuvent aussi être déclarés comme *runtime* utilisant la commande *AddHandler*.

Framework .NET

Comme son nom l'indique, VB.NET utilise le framework .NET Microsoft, ce qui signifie que le langage a pleinement accès à toutes les classes supportées par le framework.

Autres

VB.NET 10.0 supporte classes génériques et anonymes, les fonctions lambda, les types anonymes, et bien d'autres.

Installation

Sous Windows

- Le compilateur gratuit Microsoft Visual Studio Express est téléchargeable depuis le site officiel : http://msdn.microsoft.com/fr-fr/express/aa975050.aspx.
- Sinon il existe plusieurs versions payantes de Microsoft Visual Studio : https://msdn.microsoft.com /subscriptions/buy/buy.aspx.
- Il est également possible de lancer des programmes VB.NET depuis Mono, l'alternative open-source à .NET multiplateforme (Windows, Linux et Mac OSX), présenté dans le paragraphe suivant.
- Enfin, SharpDevelop Télécharger (http://www.icsharpcode.net/OpenSource/SD/Download/) [archive] constitue une autre alternative open-source.

Sous Linux et Mac OSX

MonoDevelop Télécharger (http://www.monodevelop.com) [archive] : bien que ne supportant pas toutes les fonctionnalités du langage, il autorise le développement multiplateforme.

Visual Studio

Si vous n'utilisez pas Microsoft Windows vous pouvez passer au chapitre suivant.

Visual Studio est environnement de développement intégré (IDE), c'est-à-dire un logiciel qui aide les programmeurs à développer. Par exemple il indique les erreurs de compilation en temps réel sous le code.

Contenu de l'IDE

L'environnement Visual Studio est composé de plusieurs sections, ou outils, visibles pendant la programmation. Dans un nouveau projet trois sections sont généralement visibles :

- La boîte à outils sur la gauche
- L'explorateur de solutions sur la droite
- La vue du code ou design au milieu.

Boîte à outils

La boîte à outils est une palette d'objets développeur, ou de contrôles, qui sont placées sur les formulaires ou pages web, puis le code est ajouté pour permettre à l'utilisateur d'interagir avec eux. Un exemple sont les contrôles *TextBox*, *Button* et *ListBox*, avec lesquels le développeur peut autoriser des acquisitions de l'utilisateur dans application, par exemple entrer un texte puis l'ajouter à une liste sur pression d'un bouton.

Explorateur de solutions

Cette section permet de voir et modifier les contenus du projet. Un projet Visual Studio *Application Windows Forms* contient généralement un formulaire avec une page de code associée, avec potentiellement des composants et modules de code employés par l'application.

Fenêtre propriétés

La fenêtre propriétés montre les propriétés des contrôles (comme les *TextBox*) changeable lors du design. La plupart peuvent aussi être déclenchées par du code, mais basiquement elles changent l'affichage du contrôle dans l'application.

Explorateur d'objets

En pressant F2 ou en le choisissant dans le menu *View* menu, il est possible d'explorer tous les objets des bibliothèques (types, fonctions...).

Vue du code ou design

Cette vue permet d'afficher sur plusieurs onglets les formulaires tels que les verront les utilisateurs, et le code. C'est ici que l'on dispose les objets sélectionnés dans la boîte à outils.

Raccourcis clavier

Tout comme en Visual Basic :

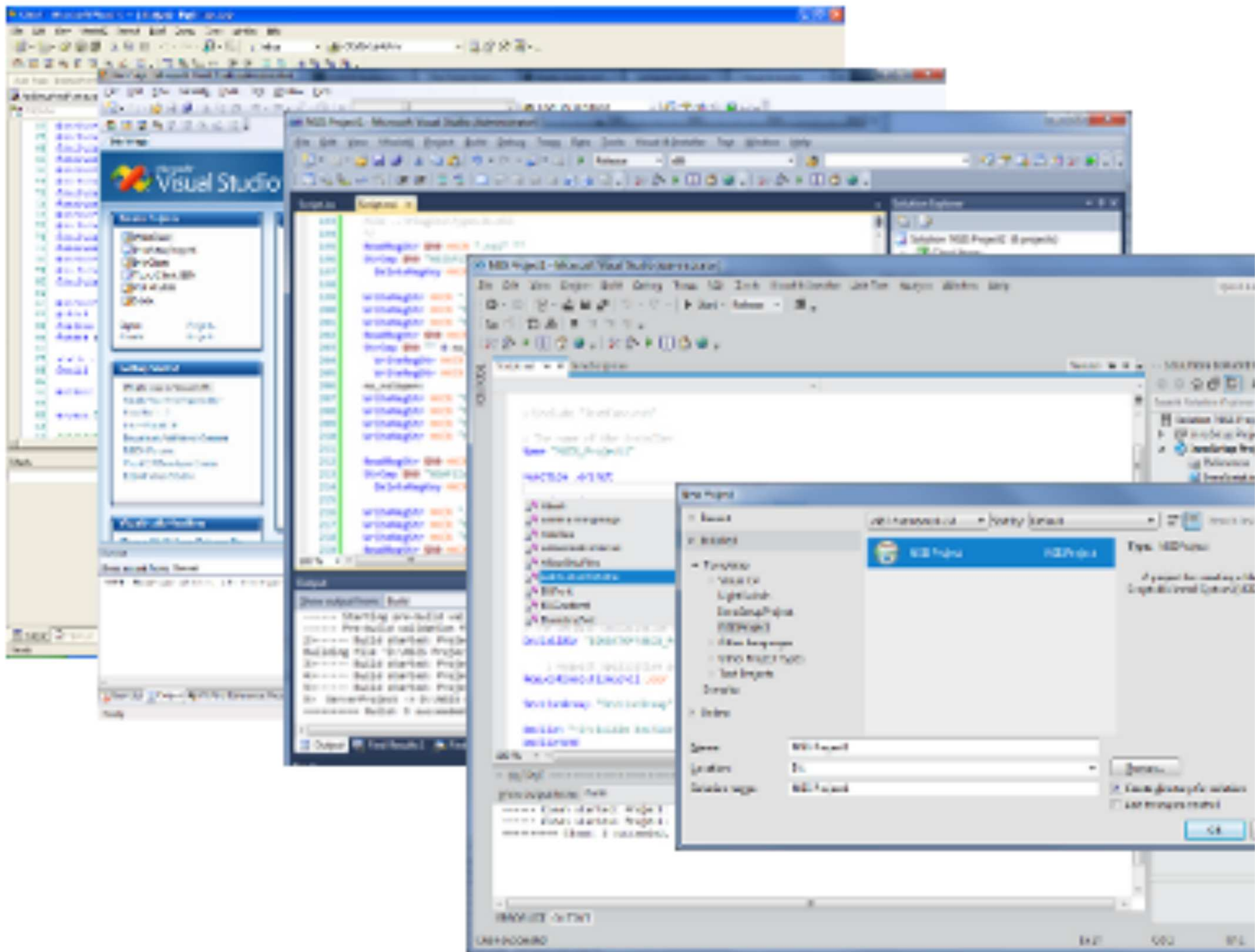
- F5 exécute le code.
- F8 le fait en pas à pas.

Remarque : en cliquant dans la marge il est possible de définir des "points d'arrêt", qui stopperont l'exécution du programme jusqu'à ce qu'on lui demande de la continuer.

Premier script

Application Windows Forms

Pour commencer à programmer en Visual Basic .NET, lancer l'interface choisie dans le premier chapitre et créer un nouveau projet de type *Application Windows Forms*.



Double-cliquer sur le formulaire vierge pour faire apparaitre son code dans un autre onglet, qui doit être une classe vierge :

```
Public Class Form1
    Inherits System.Windows.Forms.Form

[Windows Form Designer generated code]

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Ha

    End Sub
End Class
```

Y ajouter la ligne suivante dans la fonction *Form_Load* (entre les lignes *Private Sub* et *End Sub*) :

```
MessageBox.Show("Hello World!")
```

En pressant F5 ou en cliquant dans le menu *Debug* sur *Démarrer le programme*, cela devrait afficher une boite contenant le message "Hello World!", qui si on clique dessus laisse place au formulaire vierge. On peut le fermer ensuite avec le bouton de croix en haut à droite.

Application console

Ajouter un projet via le menu *File*, puis *New*, et *Project...* puis sélectionner *Application console*.

Ajouter dans la fonction *Main()* :

```
Console.WriteLine("Hello World!")
Console.ReadLine()
```

Pour que l'exécution du programme ne déclenche plus le formulaire du paragraphe précédent, mais la console à la place, effectuer un clic droit sur la nouvelle *ConsoleApplication1* dans l'*Explorateur de solutions*, puis *Définir comme projet de démarrage*.

Ensuite en pressant F5 la console se lance.

Symboles de base

Tout comme en Visual Basic, VB.net utilise :

1. ' avant les commentaires
2. _ pour scinder une commande sur la ligne suivante
3. : pour démarrer une nouvelle commande sur la même ligne.

Variables

Déclaration de variables

En Visual Basic .NET, les variables sont déclarées avec la commande `Dim` (de l'anglais *dimension*). Voici la syntaxe :

```
Dim NomVariable As TypeVariable
```

NomVariable est le nom de la variable (le plus explicite possible).

TypeVariable est son type de données (ex : *string, integer, double, boolean*, etc.).

Par exemple, pour déclarer un nombre entier :

```
Dim NomNombre As Integer
```

Par défaut, la casse des variables ne permet pas de les distinguer, ainsi *nomnariable* sera automatiquement convertie en *NomVariable* par l'IDE si elle est déclarée comme telle.

Par contre, pour que le programme ignore la casse dans les valeurs chaines de caractères, il faut définir `Option Compare Text`.

```
Option Compare Text    ' En commentant cette ligne le résultat devient False
Module Module1
```

```
    Sub Main()
        Dim chaine1 As String = "a"
        Dim chaine2 As String = "A"
        MsgBox(chaine1 = chaine2)
    End Sub
End Module
```

Types de données

Les types incorporés par VB.NET sont :

Entiers

Alias VB	Type .NET	Taille	Valeurs
SByte	System.SByte	8 bits (1 octet)	-128 à 127
Byte	System.Byte	8 bits (1 octet)	0 à 255
Short	System.Int16	16 bits (2 octets)	-32 768 à 32 767
UShort	System.UInt16	16 bits (2 octets)	0 à 65 535
Integer	System.Int32	32 bits (4 octets)	-2 147 483 648 à 2 147 483 647
UInteger	System.UInt32	32 bits (4 octets)	0 à 4 294 967 295
Long	System.Int64	64 bits (8 octets)	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807
ULong	System.UInt64	64 bits (8 octets)	0 à 18 446 744 073 709 551 615

Nombres à virgule flottante

Alias VB	.NET Type	Taille	Précision	Valeurs
Single	System.Single	32 bits (4 octets)	7 chiffres	$1,5 \times 10^{-45}$ à $3,4 \times 10^{38}$
Double	System.Double	64 bits (8 octets)	15-16 chiffres	$5,0 \times 10^{-324}$ à $1,7 \times 10^{308}$
Decimal	System.Decimal	128 bits (16 octets)	28-29 places décimales	$1,0 \times 10^{-28}$ à $7,9 \times 10^{28}$

Autres types prédéfinis

Alias VB	.NET Type	Taille	Valeurs
Char	System.Char	16 bits (2 octets)	Un symbole Unicode entre 0 et 65 535
Boolean	System.Boolean	32 bits (4 octets)	True ou False
Object	System.Object	32/64 bits (4/8 octets)	Dépendant de la plateforme
Date	System.DateTime	64 bits (8 octets)	Du 1 janvier 0001 12:00:00 AM au 31 décembre 9999 11:59:59 PM
String	System.String	80 + [16 * longueur] bits (10 + [2 * longueur] octets)	Une chaine Unicode avec une longueur maximum de 2 147 483 647 caractères.

Pour connaitre le type d'une variable, utiliser sa méthode `.GetType()`.

Utiliser les variables

Assignation

'*Les valeurs sont assignées aux variables par le signe égal.*

Suffixe pour les nombres littéraux

Les nombres intégralement littéraux comme 42 et 1000, sont de type Integer par défaut. Les chaines et caractères littéraux, comme "Hello World!" et "A", sont de type String.

Pour spécifier le type d'un littéral, on utilise donc les suffixes. Ces derniers sont accolés immédiatement après les littéraux (sans espace), comme ceci : <littéral><suffixe>.

Exemples

Les chaines de caractères doivent être entre guillemets pour ne pas être considérées comme des noms de variables :

```
strNomVariable = "Une chaine"
chrNomVariable = "À"
```

Pour les variables dates, utiliser des croisillons autour de la valeur, au format #<jour>/<mois>/<année> <heure>:<minute>:<seconde> <AM|PM>#:

```
dtNomVariable = #14/07/1789 12:01:50 PM#
```

Pour tous les autres, ne pas entourer les valeurs :

```
bytNomVariable = 1
sbytNomVariable = -2
shrtNomVariable = 10S
ushrtNomVariable = 10US
intNomVariable = 100
uintNomVariable = 100UI
lngNomVariable = 1000L
ulngNomVariable = 1000UL
sngNomVariable = 1.234F
dblNomVariable = 1.567R
decNomVariable = 1234567.89D
boolNomVariable = True
objNomObjet = New Object
```

Valeur initiale

Il est possible d'assigner une valeur à une variable pendant sa déclaration :

```
Dim NomVariable As String = "Valeur de la chaine"
```

Visual Basic assigne toujours la valeur de droite dans la variable à gauche du signe égal, sans modifier la première.

Constantes

Les constantes sont considérées comme des variables qui ne changent pas. Elles sont déclarées avec le mot clé `Const`, et leurs valeurs sont définies durant leur déclaration :

```
Const cnstNomConstante As String = "Longue chaine de caractères"
```

Exemple :

```
Const cnstPi As Single = 3.14159265F
```

Pour convertir des radians en degrés on peut créer la constante 180/Pi :

```
Const cnstRadDeg As Single = 57,29579143
...
Degres = Radians / cnstRadDeg
```

Cette constante serait utile aux fonctions Sin, Cos, Tan, Arctan, etc.

Tableaux

Tableaux à une dimension : les listes

Le type du tableau est imposé à tous ses éléments (un tableau Integer ne pourra stocker que des entiers). Voici un tableau de taille six (d'indice 0 à 5) :

```
Dim NomTableau(5) As Integer
```

Cette déclaration engendre la création de cet objet :

Index	Données
00	Nothing
01	Nothing
02	Nothing
03	Nothing
04	Nothing
05	Nothing

Que l'on peut parcourir ainsi : NomTableau(0), NomTableau(1),..., NomTableau(10).

Pour assigner des valeurs pendant la déclaration on crée une instance de la classe du type (ici Integer) :

```
Module Module1
```

```
  Sub Main()
    Dim NomTableau() As Integer = New Integer(4) { 1, 2, 3, 4, 5 }

    ' Affichage du tableau déclaré
    For ligne = 0 To UBound(NomTableau)
      Console.WriteLine(NomTableau(ligne))
    Next
    Console.ReadLine()
  End Sub
End Module
```

Voici un autre exemple avec une instance de String dont les valeurs sont attribuées ensuite :

```
    Dim NomTableau As System.Array
    NomTableau = System.Array.CreateInstance(GetType(String), 4)
    NomTableau(0) = "a"
    NomTableau(1) = "b"
    NomTableau(2) = "c"
    NomTableau(3) = "d"

    ' Affichage du tableau déclaré
    For ligne = 0 To NomTableau.Length
      Console.WriteLine(NomTableau(ligne))
    Next
    Console.ReadLine()
```

Pour afficher tous les éléments du tableau d'un coup on utilise un énumérateur :

```
      Dim En As System.Collections.IEnumerator
      Dim NomTableau As System.Array
      NomTableau = System.Array.CreateInstance(GetType(String), 4)
      NomTableau(0) = "a"
      NomTableau(1) = "d"
      NomTableau(2) = "b"
      NomTableau(3) = "c"

      ' Affichage du tableau déclaré
      En = NomTableau.GetEnumerator
      Do While En.MoveNext
          Console.WriteLine(En.Current())
      Loop

      Console.WriteLine("Presser entrée pour continuer...")
      Console.ReadLine()
```

Classer le tableau :

```
      Dim NomTableau As System.Array
      Dim En As System.Collections.IEnumerator
      NomTableau = System.Array.CreateInstance(GetType(String), 4)

      NomTableau(0) = "a"
      NomTableau(1) = "d"
      NomTableau(2) = "b"
      NomTableau(3) = "c"
      En = NomTableau.GetEnumerator
      Console.WriteLine("Avant classement")
      Do While En.MoveNext
          Console.WriteLine(En.Current())
```

```
        Loop
        Array.Sort(NomTableau)
        En = NomTableau.GetEnumerator
        Console.WriteLine("Après classement")
        Do While En.MoveNext
            Console.WriteLine(En.Current())
        Loop

        Console.WriteLine("Presser entrée pour continuer...")
        Console.ReadLine()
```

Classer le tableau par ordre décroissant :

```
Module Module1
    Sub Main()
        Dim NomTableau As System.Array
        Dim En As System.Collections.IEnumerator
        Dim DescSortCompare = New DescSortCompareClass
        NomTableau = System.Array.CreateInstance(GetType(String), 4)

        NomTableau(0) = "a"
        NomTableau(1) = "d"
        NomTableau(2) = "b"
        NomTableau(3) = "c"
        En = NomTableau.GetEnumerator
        Console.WriteLine("Avant classement décroissant")
        Do While En.MoveNext
            Console.WriteLine(En.Current())
        Loop
        Array.Sort(NomTableau, DescSortCompare)
        En = NomTableau.GetEnumerator
        Console.WriteLine("Après classement décroissant")
        Do While En.MoveNext
            Console.WriteLine(En.Current())
        Loop

        Console.WriteLine("Presser entrée pour continuer...")
        Console.ReadLine()
    End Sub


    Public Class DescSortCompareClass
        Implements IComparer

        Function Compare(ByVal x As Object, ByVal y As Object) As Integer Implements IComp
            Return x > y
        End Function
    End Class
End Module
```

Renverser le tableau :

```
        Dim NomTableau As System.Array
        Dim En As System.Collections.IEnumerator
        NomTableau = System.Array.CreateInstance(GetType(String), 4)

        NomTableau(0) = "a"
        NomTableau(1) = "d"
        NomTableau(2) = "b"
        NomTableau(3) = "c"
        En = NomTableau.GetEnumerator
```

```
        Console.WriteLine("Avant renversement du tableau")
        Do While En.MoveNext
            Console.WriteLine(En.Current())
        Loop
        Array.Reverse(NomTableau)
        En = NomTableau.GetEnumerator
        Console.WriteLine("Après renversement du tableau")
        Do While En.MoveNext
            Console.WriteLine(En.Current())
        Loop

        Console.WriteLine("Presser entrée pour continuer...")
        Console.ReadLine()
```

Une autre façon de classer de façon décroissante est de renverser le tableau croissant :

```
        Dim NomTableau As System.Array
        Dim En As System.Collections.IEnumerator
        NomTableau = System.Array.CreateInstance(GetType(String), 4)

        NomTableau(0) = "a"
        NomTableau(1) = "d"
        NomTableau(2) = "b"
        NomTableau(3) = "c"

        En = NomTableau.GetEnumerator
        Console.WriteLine("Avant classement décroissant")
        Do While En.MoveNext
            Console.WriteLine(En.Current())
        Loop
        Array.Sort(NomTableau)
        Array.Reverse(NomTableau)
        En = NomTableau.GetEnumerator
        Console.WriteLine("Après classement décroissant")
        Do While En.MoveNext
            Console.WriteLine(En.Current())
        Loop

        Console.WriteLine("Presser entrée pour continuer...")
        Console.ReadLine()
```

La ligne `Option Strict On` impose d'utiliser `SetValue` pour affecter des valeurs aux tableaux :

```
Option Strict On
Module Module1
  Sub Main()
        Dim NomTableau As System.Array
        Dim En As System.Collections.IEnumerator
        NomTableau = System.Array.CreateInstance(GetType(String), 4)

        NomTableau.SetValue("a", 0)
        NomTableau.SetValue("d", 1)
        NomTableau.SetValue("b", 2)
        NomTableau.SetValue("c", 3)

        For ligne = 0 To NomTableau.Length - 1
           Console.WriteLine(NomTableau(ligne))
        Next
        Console.ReadLine()
  End Sub
End Module
```

Tableaux à plusieurs dimensions

On peut déclarer des tableaux 2D, 3D, 4D etc.

```
' Tableau 2D aux coordonnées (X, Y)
Dim NomTableau2D(2, 2) As Integer
```

Cela génère cet objet :

Index	Data
0, 0	Nothing
0, 1	Nothing
0, 2	Nothing
1, 0	Nothing
1, 1	Nothing
1, 2	Nothing
2, 0	Nothing
2, 1	Nothing
2, 2	Nothing

Pour le parcourir on peut utiliser des boucles imbriquées, ou bien un énumérateur :

```
        Dim En As System.Collections.IEnumerator
        Dim NomTableau2D(2, 1) As String
        NomTableau2D(2, 0) = "3"
        NomTableau2D(2, 1) = "c"
        NomTableau2D(1, 0) = "2"
        NomTableau2D(1, 1) = "b"
        NomTableau2D(0, 0) = "1"
        NomTableau2D(0, 1) = "a"

        ' Affichage du tableau déclaré
        En = NomTableau2D.GetEnumerator
        Do While En.MoveNext
            Console.WriteLine(En.Current())
        Loop

        Console.WriteLine("Presser entrée pour continuer...")
        Console.ReadLine()
```

Le résultat sera :

```
1
a
2
b
3
c
Presser entrée pour continuer...
```

Méthodes de manipulation de tableaux

- `Split()` : pour transformer une chaine de caractères en tableau, selon le séparateur placé en paramètre.
- `Join()` : convertit un tableau en chaine.
- `Filter()` : filtre les entrées d'un tableau.

Exemple :

```
Module Module1
  Sub Main()
    Dim NomTableau() As String = Split("lorem ipsum dolor sit", " ")
    Console.WriteLine(NomTableau(1))  ' affiche "ipsum"

    NomTableau(4) = "amet"
    Dim ContenuTableau As String = String.Join(" ", NomTableau)
    Console.WriteLine(ContenuTableau)

    Dim SousTableau = Filter(NomTableau, "o", True, CompareMethod.Text)
    Console.WriteLine(String.Join(" ", SousTableau))  ' affiche les mots contenant des "

    Console.ReadLine()
  End Sub
End Module
```

Voir aussi

- Toutes les opérations sur les tableaux (http://msdn.microsoft.com/fr-fr/library/system.array_methods%28v=vs.80%29.aspx) [archive]

Conditions

Conditions

Liste des conditions :

1. `If/Else/Endif`
2. `If/ElseIf/Else/Endif`
3. `Select Case/End Select`

`If...Else...EndIf`

Soit les variables et les assertions suivantes :

```
   Dim Entier1 as Integer = 4
   Dim Entier2 as Integer = 7

   MsgBox(Entier1 = 4) ' affiche True
   MsgBox(Entier1 = 5) ' False
   MsgBox(Entier1 > 7) ' True
   MsgBox(Entier1 < Entier2) ' True

   If Entier1 = 4 Then MsgBox("Entier1 n'a pas changé")
```

S'il y a plusieurs instructions derrière le `Then`, il est obligatoire de leur allouer une ligne chacune, puis de terminer la condition par un `End if` :

```
 If Entier1 = 4 Then
    Entier1 = Entier1 + 3     ' Entier1 = 7
    Entier2 = Entier2 + 1     ' Entier2 = 8
 End if
```

Si des instructions doivent s'appliquer quand la condition n'est pas vérifiée, utiliser `Else` :

```
IF Entier1 = 4 Then
    (exécute le code True)
Else
    (exécute le code False)
End if

If Not Entier1 = 4 Then             ' avec une clause "not"
    (exécute le code True)          ' si Entiers1 est différent de 4
Else
    (exécute le code False)         ' si Entiers1 = 4
End if
```

Attention à bien prioriser avec des parenthèses :

```
If Not Entier1 = 3 Or Entier2 = 7 Then MsgBox("Entier1 est différent de 3 et Entier2 =
If Not (Entier1 = 3 Or Entier2 = 7) Then MsgBox("Entier1 est différent de 3 et Entier2
```

`If...ElseIf...Else...EndIf`

`If/ElseIf` permet de tester plusieurs conditions dans l'ordre séquentiel programmé. Exemple :

```
Dim x As Integer
Dim y As Integer
'...
If x = y Then
  MsgBox("x = y")
ElseIf x < y Then
  MsgBox("x < y")
Else
  MsgBox("x > y")
End If
```

`Select Case`

Ce procédé permet de raccourcir la syntaxe de longues chaines `If/ElseIf/.../ElseIf/Else`.

Il décrit en effet les différentes valeurs d'une variable (nombres ou chaines de caractères) :

```
Dim CPU as Integer
Select Case CPU
    Case 0
        MsgBox "Aucun processeur"
    Case 1
```

```
        MsgBox "Un processeur"
    Case 2
        MsgBox "Deux processeurs"
    Case 4
        MsgBox "Quatre processeurs"
    Case 3, 5 To 8
        MsgBox "3, 5, 6, 7, ou 8 processeurs"
    Case Else
        MsgBox "> 8 processeurs"
End Select
```

Opérateurs booléens

Les opérateurs booléens en Visual Basic .NET peuvent être testés tels quels, sans fonction `isTrue()` que l'on peut trouver dans d'autres langages. Par exemple :

```
functionA() and functionB()
```

Dans cette instruction le circuit court fait en sorte que la seconde fonction ne s'exécute que si la première renvoie *True*.

L'ordre des conditions est donc important pour optimiser la vitesse du programme.

Outre les opérateurs compatibles avec les précédentes version de Visual Basic, il existe ceux-ci :

```
Not
And
Or
Xor
```

Plus deux utilisant les circuits courts d'évaluation :

```
AndAlso
OrElse
```

Boucles

Il existe plusieurs structures de boucles en VB.NET.

Boucle `Do...Loop Until`

Un `Do...Loop Until` est une boucle qui itère tant que sa condition de sortie est fausse, cette dernière est testée à chaque fin de boucle, donc le programme y passe forcément au moins une fois.

```
Dim Entier1 As Integer = 4
Do
  Entier1 = Entier1 + 1
Loop Until Entier1 > 3
```

```
MsgBox(Entier1)  ' 5
```

Boucle `Do...Loop While`

Une boucle `Do...Loop While` tourne tant que sa condition d'itération est vraie, cette dernière est testée à chaque fin de boucle, donc le programme y passe forcément au moins une fois.

```
Dim Entier1 As Integer = 4
Do
  Entier1 = Entier1 + 1
Loop While Entier1 < 3
MsgBox(Entier1)  ' 5
```

Boucle `Do Until...Loop`

Un `Do Until...Loop` est une boucle qui itère tant que sa condition de sortie est fausse (comme `Do Until...Loop`), cette dernière est testée à chaque début de boucle, donc le programme n'y passe pas forcément.

```
Dim Entier1 As Integer = 4
Do Until Entier1 > 3
    Entier1 = Entier1 + 1
Loop
MsgBox(Entier1) ' 4
```

Boucle `Do While...Loop`

De la même manière, `Do While...Loop` tourne tant que sa condition d'itération est vraie (comme `Do Until...Loop`) et la teste en amont de chaque boucle.

```
Dim Entier1 As Integer = 4
Do While Entier1 < 3
  Entier1 = Entier1 + 1
Loop
MsgBox(Entier1)  ' 4
```

Boucle `For`

Une boucle `For` s'exécute un certain nombre de fois, relativement prédéfinie car la valeur du compteur peut changer en cours de rouge.

```
Dim Entier1 As Integer = 4
For a = 1 To 10
  Entier1 = Entier1 + 1
Next
MsgBox(Entier1)  ' 14
```

Cet exemple s'exécute 10 fois, a = 1, puis 2... puis 10.

Pour changer le sens du compteur :

```
For a = 10 To 1 Step -1
  Entier1 = Entier1 + 1
Next
MsgBox(Entier1)  ' 14
```

S'exécute aussi 10 fois, mais a = 10, puis 9... puis 1.

Remarque : à tout moment il est possible de sortir de la boucle sans attendre sa fin avec `Exit for`.

Boucle `For Each`

Une boucle `For Each` itère pour chaque entrée d'un tableau ou autre objet itérable, qui doit implémenter les interfaces `IEnumerable` ou `IEnumerator`. L'implémentation de `IList` par `Array` autorise un tableau à être utilisé comme un objet, car `IList` hérite de `IEnumerable`.

```
Dim Liste As Integer() = { 1, 2, 4 }
Dim Ligne As Integer
For Each Ligne In Liste
     MsgBox(Ligne)  ' affiche 1, puis 2, puis 4
Next
```

Assignation et comparaison

Assignation

L'opérateur "=" sert pour l'affectation et la comparaison.

Pour définir une variable :

```
x = 7     ' x est à présent égal à sept
x = -1294
x = "exemple"
```

Comparaison

Pour comparer deux valeurs (constantes ou variables) :

```
If 4 = 9 Then
  End  ' sortie du programme
End If
If 1234 = 1234 Then
  MsgBox("1234 est identique à 1234 !")
End If
```

```
MsgBox("Sept égal deux est " & (7 = 2) & ".")  ' False
```

```
MsgBox("Sept égal sept est " & (7 = 7) & ".")  ' True
```

Cas particulier : plusieurs signes égaux consécutifs :

```
Dim x As Boolean
x = 2 = 7
```

Le second opérateur sera exécuté en premier, renvoyant False, puis l'assignation de x à False.

Autres opérateurs de comparaison

```
(x <> y) ' Différent
(x < y) ' Inférieur
(x > y) ' Supérieur
(x <= y) ' Inférieur ou égal
(x >= y) ' Supérieur ou égal
```

Arithmétique

Opérateurs arithmétiques

Visual Basic .NET fournit différent opérateurs arithmétiques :

```
&    ' Concaténation
+    ' Addition ou concaténation
-    ' Soustraction
*    ' Multiplication
/    ' Division
\    ' Division entière
Mod  ' Modulo (reste de division)
^    ' Exponentiation

"7" & "2" ' "72"
"7" + "2" ' "72"
7 + 2     ' 9
7 - 2     ' 5
7 * 2     ' 14
7 / 2     ' 3.5
7 \ 2     ' 3
7 Mod 2   ' 1
7 ^ 2     ' 49
```

Exemple (le séparateur décimal est le point contrairement en français où on utilise la virgule) :

```
Dim Commission As Single
Dim Ventes As Single
Ventes = 3142.51
Commission = 0.3 * Ventes  ' Calcule 30 % de commission.
MsgBox(Commission)  ' 942,753
```

Remarque : ces symboles sont les mêmes que dans la calculatrice scientifique Windows.

Addition

```
x = 7 + 2      ' 9
x = 25 + -4    ' 21
Dim Chaine As String
Chaine = "Chaine1" + "Chaine2"   ' Chaine1Chaine2
```

L'opérateur "+=" incrémente la variable d'une valeur donnée :

```
Dim x As Integer = 54
x += 89        ' 143
x += 7         ' 150
```

Il fonctionne aussi en concaténation :

```
Dim x As String = "Un renard"
x += " saute"          ' Un renard saute
x += " la barrière"   ' Un renard saute la barrière
```

Soustraction

"-" soustrait deux nombres :

```
Dim x As Integer
x = 7 - 2     ' 5
x = 25 - -4  ' 29
```

Multiplication

Le symbole est l'étoile :

```
Dim x As Integer
x = 7 * 2    ' 14
x = 25 * -4 ' -100
```

Division

Il existe plusieurs types de divisions.

Division standard

Opérateur "/" :

```
Dim x As Single  ' Type acceptant les nombres décimaux
```

```
x = 7 / 2  ' 3,5
x = 25 / 4 ' 6,25
```

Division entière

Si le type ne le permet pas, l'opérateur "/" arrondit à l'inférieur : il fournir le résultat de la division sans le reste :

```
Dim x As Integer
x = 7 \ 2     ' 3
x = 25 \ 4    ' 6
```

Modulo

Cet opérateur fourni le reste de la division :

```
Dim x As Integer
x = 7 Mod 2   ' 1 (7-2*3)
x = 25 Mod 4 ' 1 (25-4*6)
```

Exponentiation

L'opérateur "^" élève un nombre à une puissance :

```
Dim x As Integer
x = 7 ^ 2    ' 7² = 49
```

Si la puissance est de 0,5 il peut aussi servir à calculer la racine carré d'un nombre :

```
Dim x As Single
x = 7 ^ 0.5 ' 2,645
```

Attention aux types des variables :

```
Dim x As Integer
x = 7 ^ 0.5 ' 3
```

La racine Nième se calcule donc généralement ainsi :

```
Dim x As Single
Dim n As Single
n = 7
x = 2 ^ (1 / n)
```

Car $x^{\frac{1}{n}} = \sqrt[n]{x}$.

Arrondis

- `Round()` : arrondit à l'entier le plus proche.
- `Floor()` : arrondit à l'inférieur.
- `Ceiling()` : arrondit au supérieur.
- `Truncate()` : tronque les chiffres décimaux.
- `isNaN()` : *not a number*, si ce n'est pas un nombre.

```
Module Module1
    Sub Main()
        Dim Nombre1 As Single = 1.5
        Console.WriteLine(Math.Round(Nombre1))  ' 2
        Console.WriteLine(Math.Floor(Nombre1))  ' 1
        Console.WriteLine(Math.Ceiling(Nombre1))  ' 2
        Console.WriteLine(Math.Truncate(Nombre1))  ' 1
        Console.ReadLine()
    End Sub
End Module
```

Remarque : avec *Imports Math*, plus besoin d'appeler ces fonctions avec ce préfixe.

Comparaisons

- `Max()`
- `Min()`

```
Module Module1
    Sub Main()
        Console.WriteLine(Math.Max(3, 4))
        Console.ReadLine()  ' 4
    End Sub
End Module
```

Autres

- `Abs()` : valeur absolue.
- `Pow()` : puissance.
- `Sqrt()` : racine carrée (square root).

Chaines de caractères

Joindre des chaines de caractères

Concaténation

L'opérateur "&" joint deux chaines ensemble :

```
Dim Chaine1 As String = "123"
Dim Chaine2 As String = "456"
Dim Chaine3 As String
```

```
Chaine3 = Chaine1 & Chaine2  ' 123456.
```

L'opérateur "+" peut être utilisé à la place de "&", mais non recommandé pour éviter la confusion avec les additions.

Concat()

La méthode `String.Concat()` est une alternative à l'opérateur :

```
Dim Chaine1 As String = "123"
Dim Chaine2 As String = "456"
Dim Chaine3 As String = "789"
Dim Resultat As String
Resultat = String.Concat(Chaine1,Chaine2,Chaine3)  ' 123456789
```

Cela fonctionne aussi avec les tableaux :

```
Dim TableauChaines As String() = {"1", "2", "3", "4", "5"}
Resultat = String.Concat(TableauChaines)  ' 12345
```

Méthodes sur les chaines de caractères

- Lcase() : convertit en bas de casse.
- Ucase() : convertit en haut de casse.

Pour créer des sous-chaines, il existe toujours les fonctions VB6 :

- Left() : partie de gauche.
- Right() : partie de droite.
- Mid() : partie du milieu.
- InStr() : emplacement d'une chaine sans une autre.
- Replace() : remplace une sous-chaine par une autre.

```
  Sub Main()
      Dim NomChaine As String = "lorem ipsum dolor sit amet"
      NomChaine = NomChaine.Replace("i", "o")
      Console.WriteLine(Mid(NomChaine, InStr(NomChaine, " "), 6))
      Console.ReadLine()  ' Affiche " opsum"
  End Sub
```

Logique

Opérateurs logiques

Not

L'opérateur Not renvoie True quand la condition est False sinon False :

```
If Not (1 = 2) Then
   MessageBox.Show("(1 = 2) est False. Donc Not False = True")
End If
```

Truth Table

Condition	Not *Condition*
True	False
False	True

And

And retourne True quand ses deux opérandes sont True, sinon False. Il les évalue toutes les deux avant de se prononcer.

```
If (1 = 1) And (2 = 2) Then
   MessageBox.Show("(1 = 1) est True. (2 = 2) est True. Donc True And True = True")
End If
```

Truth Table

Condition1	*Condition2*	*Condition1* And *Condition2*
True	True	True
True	False	False
False	True	False
False	False	False

AndAlso

AndAlso est comme And sauf qu'il économise du temps : il dit False quand la condition qui le précède est False, puis sinon True si les deux opérandes sont True, sinon False. Cet ordre s'appelle un circuit court logique.

Truth Table

Condition1	*Condition2*	*Condition1* AndAlso *Condition2*
True	True	True
True	False	False
False	-	False

Or

Or est True quand au moins l'une de ses opérandes est True, sinon False. Il évalue les deux (comme And).

Truth Table

Condition1	*Condition2*	*Condition1* Or *Condition2*

True	True	True
True	False	True
False	True	True
False	False	False

OrElse

OrElse est le circuit logique court de Or : il donne True quand sa première opérande est True, sinon il teste la seconde est renvoie True si elle est True, et False si les deux sont finalement False.

Truth Table

Condition1	*Condition2*	*Condition1* OrElse *Condition2*
True	-	True
False	True	True
False	False	False

Xor

Xor ("ou exclusif", de l'anglais "exclusive or") est True uniquement si l'une des deux opérandes est True, mais pas quand elles le sont toutes les deux.

Truth Table

Condition1	*Condition2*	*Condition1* Xor *Condition2*
True	True	False
True	False	True
False	True	True
False	False	False

Gestion d'exception

Levées d'exception

Pour éviter que des erreurs bloquent l'exécution du programme il est possible de définir un comportement quand elles surviennent.

`On Error GoTo`

La méthode utilisée en VBA est toujours disponible. Celle-ci n'était pas structuré car fait appel à un Goto vers une étiquette :

```
    Public Sub Main()
        On Error GoTo Etiquette1
        Dim Resultat As Integer = 2
        Dim a = 1
        Dim b = 0
        Resultat = a / b  ' Division par zéro
Etiquette1:
        MessageBox.Show(Resultat) ' renvoie 2
    End Sub
```

Pour éviter d'avoir à définir des étiquettes on peut tout simplement ignorer les erreurs un moment (puis les réactiver avec `On Error GoTo 0` :

```
        On Error Resume Next
        Dim Resultat As Integer = 2
        Dim a = 1
        Dim b = 0
        Resultat = a / b
        On Error GoTo 0
        MessageBox.Show(Resultat)
```

Try...Catch...Finally

Le gestionnaire d'erreur structuré s'appelle `Try...Catch...Finally`[1]. De nombreux types d'erreur y sont disponible par héritage de la classe `Exception`.

```
    Public Sub Main()
        Dim Resultat As Integer = 2
        Dim a = 1
        Dim b = 0
        Try
            Resultat = a / b
        Catch ex As Exception
            MsgBox(ex.ToString)
        Finally
            MessageBox.Show(Resultat)
        End Try
    End Sub
```

Remarque : pour relancer une exception dans un bloc `Catch`, utiliser le mot clé `Throw` sans argument. En effet il reset la propriété `.StackTrace` de l'objet exception en cas d'argument[2].

Références

1. anglais http://msdn.microsoft.com/fr-fr/library/fk6t46tz%28v=vs.71%29.aspx
2. anglais Blog d'Eric Lippert (http://blogs.msdn.com/ericlippert/archive/2010/03/04/too-much-reuse.aspx) [archive]

Classes

Introduction

Tout comme en VB6, les classes sont utilisables pour la programmation orientée objet.

Pour en créer une, dans l'Explorateur de solutions, après un clic droit sur l'application, *Add*, puis *Class*. Ou bien dans le menu *Project*, choisir *Ajouter une classe...*.

Champs

Les champs sont des variables déclarées directement dans la classe (pas dans ses fonctions ou propriétés), et que l'on ne peut pas appeler en dehors d'elle :

```
Public Class customer
   Private Nom As String
   Private Adresse As String
   Private Age As Integer
   ...
```

Les classes peuvent appeler ces variables privées à l'aide de Me :

```
...
Public Function AfficherNom()
   Return Me.Nom
End Function
...
```

Constructeurs

Une fois la classe déclarée, il convient d'initialiser ses champs avec un constructeur :

```
Public Class Client
   Public Nom As String
   Private Adresse As String
   Private Identifiant As String
   ' Constructeur avec paramètre
   Public Sub New(ByVal Nom As String)
      Me.Nom = Nom
   End Sub
   ' Constructeur sans paramètre
   Public Sub New()
   End Sub
   ...
```

Propriétés

Les propriétés sont divisées en deux catégories : *getter* et *setter*. Un getter renvoie la valeur dans une classe, et un setter définit une valeur dans une classe.

```
  Public Property nom() As String
    Get
        Return Me.Nom
```

```
    End Get
    Set(By Val valeur As String)
        Me.Nom = valeur
    End Set
End Property
```

La classe étant publique on peut y accéder en dehors de la classe, contrairement aux champs.

Méthodes

Les méthodes sont les sous-routines de la classe, et peuvent être appelées un nombre illimité de fois.

Instanciation

On utilise le mot New.

```
Dim Client1 = New Client("Untel")
MsgBox(Client1.Nom)
```

Namespaces

Namespaces

Les espaces de noms permettent de séparer les différents programmes importés, notamment quand ils sont volumineux comme les bibliothèques.

Dans un module, entrer simplement (sans nom de module) :

```
Namespace Paquet1
  Class ClasseExterne
    Public Nom As String = "Défaut"
  End Class
End Namespace
```

Pour les importer ensuite, utiliser `Imports`.

Par exemple depuis un autre module du projet *ConsoleApplication1* :

```
Imports ConsoleApplication1.Paquet1
Module Module1
    Sub Main()
        Dim NomLocal = New ClasseExterne
        Console.WriteLine(NomLocal.Nom)
        Console.ReadLine()  ' Affiche "Défaut"
    End Sub
End Module
```

En commentant la ligne de l'importation, l'erreur suivante apparait : *Type ClasseExterne non défini*.

Références

- http://msdn.microsoft.com/fr-fr/library/zt9tafza.aspx

Héritage

Héritage d'objets

L'héritage se traduit par le mot clé `Inherits`. On peut ensuite étendre et modifier une classe existante avec des propriétés et méthodes additionnelles.

Par exemple en partant de la classe ci-dessous :

```
Public Class Personne

     Public Prenom As String
     Public Nom As String
     Public DateDeNaissance As Date
     Public Genre As String

     Public ReadOnly Property NomEntier() As String
         Get
             Return Prenom & " " & Nom
         End Get
     End Property

 End Class
```

Sachant qu'un client est une personne avec un type et un numéro de client, il est plus rapide qu'il hérite de la classe *Personne* que de déclarer toutes ses propriétés et méthodes en repartant de zéro :

```
 Public Class Client
     Inherits Personne
     Public IDClient As String
     Public TypeClient As String
 End Class
```

De plus, si la classe *Personne* est modifiée il ne sera pas nécessaire de mettre à jour la classe *Client*.

Interfaces

Utiliser les interfaces

Une Interface est l'ensemble des méthodes publiques d'une classe. Elles se déclarent avec le mot clé `Implements`.

Pour créer une interface :

```
Public Interface Interface1
   Function Fonction1() As String
End Interface
```

Pour l'utiliser dans une autre classe :

```
Public Class ClasseDepuisInterface
    implements Interface1()
    ' en appuyant sur entrée l'IDE crée automatiquement les éléments de l'interface implé
    Public Function Fonction1() As String Implements Interface1.Fonction1
        ' ...
    End Function
End Class
```

Références

- (anglais) http://msdn.microsoft.com/en-us/library/28e2e18x%28v=vs.71%29.aspx
- (anglais) http://www.youtube.com/watch?v=jV4_VhOCMuU

IDisposable

L'interface IDisposable

IDisposable est implémentée quand un objet a besoin d'être réinitialisé. Généralement s'il contient une méthode *Dispose*, cette réinitialisation est nécessaire.

La plus simple façon de le faire est d'utiliser le mot clé `Using`.

```
    Using f As New Form
        f.Show
    End Using
```

Quand un objet IDisposable est un formulaire, il doit être disposé dans l'évènement `Form_Closed`.

```
    Public Class Formulaire1
        Private FormulaireEnfant As Form
        Private Sub Formulaire1_Load(ByVal sender As System.Object, ByVal e As System.Eve
            FormulaireEnfant = New Form
            FormulaireEnfant.Text = "Enfant"
            FormulaireEnfant.Show()
        End Sub
        Private Sub frmMain_FormClosed(ByVal sender As Object, ByVal e As System.Windows.
            FormulaireEnfant.Dispose()
        End Sub
    End Class
```

Contrôles

Comme son prédécesseur, Visual Basic .NET excelle à la création d'interfaces graphiques. En travaillant sur des formulaires, il suffit de faire glisser les contrôles désirés depuis la ToolBox, de changer leur taille et de les configurer via la fenêtre des propriétés. Le gestionnaire d'évènement de chaque contrôle renvoie à la fenêtre de code.

Contrôles visibles

TextBox

La TextBox affiche du texte, sa fenêtre propriété permet de changer de police de caractère, sa taille, couleur, etc.

Mots de passe

Pour contrôler la validité d'un mot de passe, une TextBox doit contenir un caractère de masquage dans `PasswordChar`. On trouve généralement * ou ● mais toute lettre peut faire l'affaire.

Presse-papier

La méthode `TextBox.Copy()` permet de remplir le presse-papier de l'utilisateur avec le contenu de la boite, et `TextBox.Paste()` d'en récupérer dedans (coller). `TextBox.Cut` coupe le texte de la TextBox dans le presse-papier.

Restreindre du texte

Pour restreindre certains mots dans la TextBox il faut créer un gestionnaire d'évènement de KeyPress.

L'exemple suivant n'autorise que des chiffres (de 0 à 9), la virgule et le signe moins :

```
Private Sub SubName (ByVal sender As Object, ByVal e As System.Windows.Forms.KeyPressEven
    Select Case Asc(e.KeyChar)
        Case 48 To 57, 8
        Case 45
            If InStr(sender.text, "-") <> 0 Or sender.selectionStart <> 0 Then
                e.Handled = True
            End If
        Case 46
            If InStr(sender.Text, ",") <> 0 Then
                e.Handled = True
            End If
        Case Else
            e.Handled = True
    End Select
End Sub
```

Labels

Les Labels affichent du texte non modifiable. Généralement, ils servent à décrire les autres contrôles du formulaire.

Buttons

Les Buttons déclenchent une action prédéfinie quand on clique dessus.

```
Public Class Form1
    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles But
        MsgBox("Bouton 1 pressé")
    End Sub
End Class
```

Checkbox

Une CheckBox indique un choix deux états (True/False) que l’utilisateur peut cocher ou décocher. Bien sûr le code peut modifier sur le titre de la boite selon sa valeur.

```
Public Class Form1
    Private Sub CheckBox1_CheckedChanged(sender As System.Object, e As System.EventArgs) I
        If CheckBox1.Checked Then
            MsgBox("Case 1 cochée")
        End If
    End Sub
End Class
```

RadioButton

Le RadioButton permet d'effectuer un choix parmi une liste d'options.

Pour déclencher une action sur une case cochée :

```
Public Class Form1
    Private Sub RadioButton1_CheckedChanged(sender As System.Object, e As System.EventArg
        If RadioButton1.Checked Then
            MsgBox("Bouton 1 coché")
        End If
    End Sub

    Private Sub RadioButton2_CheckedChanged(sender As System.Object, e As System.EventArg
        If RadioButton2.Checked Then
            MsgBox("Bouton 2 coché")
        End If
    End Sub
End Class
```

RichTextBox

Ce type de boite permet d'afficher avec `openfile(Richtextbox1.openfile(CheminFichier))`, ou de remplir via `method(richtextbox1.savefile(CheminFichier))`, le contenu d'un fichier texte.

Autres contrôles

Certains contrôles sont stockés dans des sous-menus de la ToolBox.

Boîtes de dialogue

Elles se placent sous le formulaire en mode design, et ne se voient pas en mode lecture.

OpenFileDialog

Une `OpenFileDialog` régit le parcourt du disque pour ouvrir un fichier, sans ses propriétés.

SaveFileDialog

Permet par exemple de prédéfinir une extension de de la sauvegarde des fichiers.

FontDialog

Liste des polices de caractères disponibles dans les boites texte de l'utilisateur.

Énumérations

Introduction

Une énumération est un ensemble de constantes du même type. Par défaut il s'agit d'entiers dont le premier est à 0 et le deuxième à 1, etc.

Créer des énumérations

```
Enum Termes
    30jours ' = 0
    60jours ' = 1
    90jours ' = 2
End Enum
```

Avec affectations :

```
Enum ValeurTermes As Short
    30jours = 30
    60jours = 60
    90jours = 90
End Enum
```

Utiliser des énumérations

Les énumérations sont appelables en préfixant le nom de leur collection :

```
Dim t As Termes = Termes.30jours                    't = 0
Dim i As Integer = CInt(Termes.30jours)             'i = 0
Dim valeur As Integer = CInt(ValeurTermes.60jours)  'valeur = 60
Dim nom As String = ValeurTermes.30jours.ToString   'nom = "30"
```

Collections

Une collection est comme un tableau qui pourrait stocker plusieurs types d'entrées. De plus certaines collections ont leurs propres méthodes.

Par exemple pour tout objet `Collection`, il existe les méthodes :

- `Add()`
- `Remove()`
- `Item()`
- `Clear()`

ArrayList

`ArrayList` est un tableau dynamique, sa taille varie automatiquement selon son contenu, et peut stocker des objets de différents types.

Méthode `Add`

La méthode `System.Collections.ArrayList.Add(Object)` peut incrémenter des `ArrayList` :

```
Module Module1
Sub Main()
    Dim Tableau1 As System.Collections.ArrayList = New ArrayList()
    Tableau1.add(New String("a"))
    Tableau1.add(New String("b"))
    Tableau1.add(New String("c"))
    Tableau1.add(New String("d"))
    Console.WriteLine(Tableau1(0))
    Console.WriteLine("Presser entrée pour continuer")
    Console.ReadLine()
End Sub
End Module
```

Queue

Une queue est une collection FIFO, elle a deux uniques méthodes pour empiler et dépiler ses informations. En utilisant `System.Collections.Queue.Enqueue`, un objet peut être ajouté à la fin de la collection, et retiré s'il est au début via `System.Collections.Queue.Dequeue`.

Stack

Une pile (stack en anglais) est une collection en LIFO. Elle prend en charge trois méthodes de stockage et restitution de l'information.

Utiliser `System.Collections.Stack.Push` pour ajouter un objet au sommet de la pile, poussant celui qui s'y retrouvait en-dessous.

`System.Collections.Stack.Pop` renvoi l'objet su sommet de la pile et le retire.

`System.Collections.Stack.Peek` est similaire à `Pop` sauf qu'il ne retire pas l'objet renvoyé.

SortedList

Une `SortedList` est une collection d'objets classée selon un index (comme pour les tableaux) et aussi une clé, pouvant adopter n'importe quel type.

Dictionary

Tableau à au moins deux colonnes dont une clé.

Hashtable

Dictionnaire dont la clé est issue d'une fonction de hachage.

Generics

Les `Generics` permettent de stocker des objets de types plus spécifiques, comme `String` et `Integer`.

List

Une List(d'un type) donne accès à un ensemble d'entrée comme un tableau, ou comme une collection.

```
Dim Jours As New List(Of String)
Jours.Add("Lundi")
Jours.Add("Mardi")
Jours.Add("Mercredi")
Jours.Add("Jeudi")
Jours.Add("Vendredi")

Console.WriteLine("Accès aux membres par index...")
For i As Integer = Jours.Count - 1 To 0 Step -1
    Console.WriteLine(Jours(i))
Next i
Console.WriteLine("Accès aux membres par collection...")
For Each j As String In Jours
    Console.WriteLine(s)
Next j
```

Références

- http://plasserre.developpez.com/cours/vb-net/?page=langage-vb3#LV-I-6

GDI+

Objet graphique

Un objet `System.Drawing.Graphics` représente une surface de dessin.

Pour obtenir un tel objet dans un contrôle *PictureBox*, utiliser la propriété `.Graphics` du paramètre

PaintEventArgs dans l'évènement Paint() de la PictureBox.

```
Private Sub PictureBox1_Paint(ByVal sender As Object, ByVal e As System.Windows.Forms
    Using g as Graphics = e.Graphics
        ' g permet ensuite de dessiner
    End Using
End Sub
```

Pour charger une image puis dessiner dessus, utiliser Graphics.FromImage() :

```
Using MonImage As Bitmap = Bitmap.FromFile("C:\Temp\MonImage.BMP")
    Using g as Graphics = Graphics.FromImage(MonImage)
        ' g permet ensuite de dessiner
    End Using
End Using
```

La méthode .Save() peut être utilisée pour sauvegarder le bitmap dans un fichier ou un Stream (flux). Remarque : Bitmap conserve le fichier image ouvert, il faut donc copier le fichier dans un MemoryStream pour ne pas avoir cette liaison.

Méthodes de dessin

des méthodes comme Graphics.DrawLine et Graphics.DrawString pour tracer lignes et textes.

Mesures

Graphics.MeasureString permet de mesurer la hauteur ou largeur d'une partie de texte, affichable avec .DrawString.

Échelle

Échelle simple

Utiliser la propriété Graphics.PageUnit comme une façon simple de changer les coordonnées du système.

Coordonnées des systèmes World, Page et Device

Les coordonnées d'un système VB.NET font partie de celles d'un World. Par défaut ces unités sont en pixels, mais on peut les redéfinir.

La transformation de World convertit ses coordonnées en celles d'une Page. Ces dernières expriment la distance visible sur le moniteur ou imprimée sur du papier.

La transformation de Page convertit ses coordonnées en celles d'une Device. Cela permet à l'image de rester la même quel que soit le périphérique sur laquelle elle est envoyée.

Il est possible de définir une matrice de traduction pour convertir les points spécifié dans le résultat final. Cette transformation peut traduire (offset), échelle, rotation, inclinaison dans le résultat, elle est appelée *transformation affine*.

Convertir depuis Twips

Les précédentes versions de Visual Basic utilisaient Twips comme unité de mesure : 1,440 Twips = 1 pouce. Toutefois, VB.NET utilise plutôt les pixels. Pour convertir les Twips en pixels dans un objet graphique, il faut connaitre :

- `XTwips` qui renvoie la mesure en Twips.
- `XPixels` qui renvoie celle en pixels.

Si `g` est l'objet `Graphics` approprié :

```
XPixels = XTwips * g.DpiX / 1440
```

Utiliser `g.DpiY` pour calcuer les coordonnées de l'axe des Y.

GFDL

Vous avez la permission de copier, distribuer et/ou modifier ce document selon les termes de la **licence de documentation libre GNU**, version 1.2 ou plus récente publiée par la Free Software Foundation ; sans sections inaltérables, sans texte de première page de couverture et sans texte de dernière page de couverture.

Récupérée de « https://fr.wikibooks.org/w/index.php?title=Programmation_Visual_Basic_.NET/Version_imprimable&oldid=440905 »

Dernière modification de cette page le 15 février 2014 à 19:48.
Les textes sont disponibles sous licence Creative Commons attribution partage à l'identique ; d'autres termes peuvent s'appliquer.
Voyez les termes d'utilisation pour plus de détails.
Développeurs