Le langage Visual Basic .NET à grands pas

Ce document présente en abrégé, l'essentiel du langage Visual Basic.NET. Il est en aucun cas exhaustif et ne présente pas tous les détails et subtilités du langage. Il contient les éléments de base nécessaire pour démarrer la programmation d'une application avec ce langage.

Commentaires

Le caractère **'** ou le mot clé ***Rem*** est utilisé pour définir un commentaire. Tout le texte qui suit ce caractère n'est pas compilé.

Caractères de continuation de ligne d'instruction

Le caractère _ est utilisé en fin de ligne lorsqu'une instruction s'étend sur plus d'une ligne. Voici un exemple:

```
Dim sjour As String = {"Lundi", "Mardi", "Mercredi", "Jeudi", _
                       "Vendredi", "Samedi", "Dimanche"}
```

Variables

Les différentes types de variables simples

Type	Valeurs	Octets
Byte	0 à 255	1
Short	-32 768 à 32 767	2
Integer	-2 147 483 648 à 2 147 483 647	4
Long	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807	8
Single	$-3{,}402823 \times 10^{38}$ à $-1{,}401298 \times 10^{-45}$ et $1{,}401298 \times 10^{-45}$ à $3{,}402823 \times 10^{38}$	4
Double	$-1{,}79769313486231 \times 10^{308}$ à $-4{,}94065645841247 \times 10^{-324}$ et $4{,}94065645841247 \times 10^{-324}$ à $1{,}79769313486231 \times 10^{308}$	8
Char	1 caractère (ASCII ou Unicode)	2
String	n caractères ($0<n<2 \times 10^{9}$)	10+2n
Boolean	True (0) ou False (-1)	2
Date	1 janvier 100 à 31 décembre 9999	8

Par défaut, la déclaration des variables n'est pas nécessaire dans VB.NET. Ce comportement est défini par:

```
Option Explicit {Off | On}
```

Cette option est par défaut à Off et n'est pas recommandée. Lorsque **Option Explicit On** est ajoutée en début de programme toutes les variables doivent être explicitement déclarées, ce qui évite des erreurs très délicates à retracer.

Déclaration d'une variable

La déclaration d'une variable en VB.NET est réalisée simplement de la manière suivante:

```
Dim identifiant As Type [=  valeur]
```

Exemples:

```
Dim icompteur As Integer
Dim snomFichier As String
Dim bnouveau As Boolean
Dim dtdateDebut As Date = #3 mai 1961#
```

Tableaux

Pour déclarer un tableau, il faut en plus du type des éléments qui le composent, définir le nombre d'éléments qui le composent. Voici les trois manières de déclarer un tableau:

```
Dim identifiant(nombre_éléments) As Type 'Déclare tableau taille donnée
Dim identifiant() As Type             'Déclare tableau taille dynamique
ReDim identifiant(nombre_éléments)    'Modifie taille d'un tableau
```

Les éléments du tableau sont toujours indexés de 0 à nombre_éléments-1. Un tableau peut être initialisé lors de la déclaration:

```
Dim identifiant() As Type = {valeur0, valeur1, …,valeurN}
```

Exemples:

```
Dim sjour As String = {"Lundi", "Mardi", "Mercredi", "Jeudi", _
                       "Vendredi", "Samedi", "Dimanche"}
```

Affectation d'une variable

L'instruction **Set** n'existe plus. Une affectation se fait de la manière suivante:

```
identifiant = valeur
identifiantObjet = New unObjet()
```

Constantes

En VB.NET, une constante est une variable en lecture uniquement. Une fois affectée, sa valeur ne peut plus être modifiée. Étant une variable, la constante a un type.

```
Const LANG As String = "Français"
```

Opérateurs

Les opérateurs suivants sont utilisables dans VB.NET:

=	affectation
+	Addition
-	Soustraction
*****	Multiplication
/	Division
****	Division entière
Mod	Modulo
&	Concaténation de chaîne de caractères
^	Puissance

Les raccourcis suivants sont aussi possibles: **+=** , **-=** , ***=** , **/=** , **&=** , par exemple:

```
iVal += 2 est équivalent à iVal = iVal +2
```

Fonctions intégrées

Fonctions de conversion de type

VB.NET propose de nombreuses fonctions intégrées pour réaliser la conversion explicite de type entre variables. Parmi celles-ci se trouvent les fonctions de conversion de types suivantes:

Fonction	Type résultant	Commentaires
CBool	Boolean	True, sauf pour 0 qui donne False
CByte	Byte	Valeur > 255 ou fractionnaire est perdue
CChar	Char	Valeur > 65 536 est perdue. 1^{ier} char d'une String
CDate	Date	Reconnaît tous les formats de saisie d'une date
CDbl	Double	
CInt	Integer	Fraction arrondie à la valeur la plus proche
CLng	Long	Fraction arrondie à la valeur la plus proche
CSht	Short	Fraction arrondie à la valeur la plus proche
CStr	String	Si c'est une date alors le résultat format Short Date
CType	TOUS	VarCible = CType (varSource, NouveauType)

Attention: Utiliser des fonctions de conversion de type qui font du sens. CType permet toutes les formes de conversion même sur les énumérations, les structures, les objets.

La conversion de type peut être réalisée automatiquement par VB.NET si la ligne suivante est présente en début de fichier (dangereux!):

```
Option Strict Off
```

Cette option par défaut n'est pas recommandée. Lorsque **option strict on** est placée en début de programme, le compilateur ne fait plus de conversion de type par défaut, ce qui évite des erreurs difficiles à tracer. Cette option implique aussi que toutes les variables soient explicitement déclarées. Elle inclue donc aussi **option explicit on**.

Fonction de manipulation de chaînes de caractères

Dans VB.NET, la plupart des fonctions de manipulation de chaînes sont des méthodes de la classe String:

Fonction	Description	Exemple
Len	Retourne la longueur de la chaîne	iLng = Len ("Bonjour") iLng contient 7
Chr	Retourne le caractère correspondant au code ASCII ou UNICODE entrée	cCar = Chr (56) cCar contient la lettre A
Asc	Retourne le code ASCII ou UNICODE du caractère entrée	iVal = Asc ("A") iVal contient 56
Left	strDest = Left (strSrc, nbr) retourne les nbr caractères de gauche de strSrc	strDest = Left ("Bonjour",3) strDest contient "Bon"

Right	strDest = Right (strSource, nbr) retourne les nbr caractères de droite de strSrc	strDest = Right ("Bonjour",4) strDest contient "jour"
Mid	strDest = Mid (strSrc, prem, [nbr]) retourne les nbr caractères à partir du prem ^{ième} de strSrc ou jusqu'à la fin de la chaîne	strDest = Mid ("Bonjour",3,2) strDest contient "nj" strDest = Mid ("Bonjour",4) strDest contient "jour"
Instr	iPos = Instr (i, strSrc, strRec, [Tc]) Recherche la chaîne strRec dans strSrc à partir du i ^{ième} caractère. Si Tc = 0, la comparaison est en mode binaire (strict) sinon la casse n'est pas prise en compte (voir doc VB.NET pour détail)	iPos = Instr(2,"Concorde", "co") iPos contient 4 iPos = Instr(1,"Concorde','co",0) iPos contient 4 iPos = Instr(1,"Concorde','co") iPos contient 1
InstrRev	Idem à Instr mais en partant de la fin de la chaîne iPos = InstrRev (strSrc, strRec, i,[Tc])	j = InstrRev("Concorde", "co") j contient 4 j = InstrRev("Concorde','co",4,0) j contient 0 j = InstrRev("Concorde','co",4) j contient 1
LCase	strDest = LCase (strSrc) retourne la chaîne en minuscule	strDest = LCase ("Bonjour") strDest contient "bonjour"
UCase	strDest = UCase (strSrc) retourne la chaîne en majuscule	strDest = UCase ("Bonjour") strDest contient "BONJOUR"
LTrim	strDest = LTrim (strSrc) retourne la chaîne sans les espace du début	strDest = LTrim (" Bonjour") strDest contient "Bonjour"
RTrim	strDest = RTrim (strSrc) retourne la chaîne sans les espace de la fin	strDest = RTrim ("Bonjour ") strDest contient "Bonjour"
Trim	strDest = Trim (strSrc) retourne la chaîne sans les espace du début et de la fin	strDest = Trim (" Bonjour ") strDest contient "Bonjour"

Autres fonctions

Le tableau suivant récapitule quelques autres fonctions utiles.

Fonction	**Description**
IsArray	Retourne True si la variable est un tableau
IsDate	Retourne True si la variable est une date
IsNumeric	Retourne True si la variable est un nombre
IsObject	Retourne True si la variable est un objet
TypeName	Retourne sous forme de chaîne de caractère le type du paramètre
Now	Retourne la date et l'heure du moment
Today	Retourne la date du moment et l'heure est 0:00:00

Fonctions et sous programmes

En VB.NET, il existe deux formes de sous-programmes, l'une retourne une valeur (**function**), l'autre ne retourne rien (**subroutine** ou **sub**). La syntaxe est la suivante:

```
Sub NomSubroutine (param1 As type, param2 As type, …)
      'traitement
End Sub

Function NomFunction (param1 As type, param2 As type, …) As type
      'traitement
      Return valeurRetournee
End Function
```

La syntaxe pour les appels des sous-programmes est la suivante:

```
NomSubroutine (param1, param2, …)
NomVar = NomFunction (param1, param2, …)
```

Les paramètres peuvent être passés par valeur (byVal) ou …….

Exemple:

```
Sub AfficheMessage (ByVal mess As String)
      Console.WriteLine (mess)
End Sub

AfficheMessage ("Bonjour")
```

Portée des identifiants

Ce terme définit la visibilité (donc l'accessibilité) d'un identifiant dans un programme qui est contrôlée par les mots clés **Private** et **Public**. En règle générale, il faut déclarer les variables à l'emplacement où elles sont utiles et restreindre leur portée pour éviter toute modification accidentelle. **Private** restreint l'accès au contexte dans lequel l'élément a été déclaré. **Public** aucune restriction ne s'applique à la visibilité de cet élément. Pour plus de détail, consulter la documentation de VB.NET.

Contrôle de flux – Structuration du code

Instruction if

Cette instruction possède plusieurs variations au niveau de sa syntaxe et de sa forme.

```
If (condition) then
      'traitement exécuté si la condition est vrai
End if

If (condition) then
      'traitement exécuté si la condition est vrai
Else
      'traitement exécuté si la condition est fausse
End if

If (condition) then
      'traitement exécuté si la condition est vrai
Else If (condition2) then
'traitement exécuté si la condition2 est vrai
Else
```

```
'traitement exécuté si la condition1 et condition2 sont fausses
End i
```

Opérateurs de comparaison

Les opérateurs de comparaison suivants sont utilisables pour définir les expressions booléennes:

> supérieur à
< inférieur à
= égal à
<> non égal à
>= supérieur ou égal à
<= inférieur ou égal à

Ces opérateurs peuvent être utilisés avec des variables numériques mais également avec des caractères et des chaînes de caractères.

Like permet aussi la comparaison entre deux chaînes de caractères avec l'utilisation des caractères spéciaux suivants:

***** remplace un nombre quelconque de caractères quelconque
? remplace un caractère quelconque
remplace un chiffre
[-] une plage de caractères, exemple [f-k], tous les caractères de f à k

Opérateurs logiques

VB.NET propose les quatre opérateurs suivants (NOT, AND, OR, XOR):

A	B	NOT A	A AND B	A OR B	A XOR B
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

Instruction Select Case

Si une variable peut prendre plusieurs valeurs différentes exigeant chacune un traitement spécifique, l'instruction **Select Case** est à utiliser.

```
Select Case NomVariable
      Case valeur1
            'traitement pour valeur1
      Case valeur2    'valeur2 to valeur3 est une autre forme possible
      …
      Case else
End Select
```

La clause **Case Else** est exécutée si aucune autre ne l'a été.

Boucle For … Next

Ce type de boucle convient pour exécuter un traitement un nombre prédéterminé de fois.

```
For variable = valeurDepart to valeurFin [Step = valPas]
      'traitement à répéter
```

```
Next variable
```

Boucle While … End While

Cette structure est mieux adaptée lorsqu'un traitement doit être exécuté tant qu'une condition est satisfaite.

```
While (condition)
      'Traitement à réaliser
End While
```

Boucle Do

C'est la boucle de base. Elle offre plusieurs possibilités:

```
Do
      'Traitement à réaliser
      'Condition de sortie de la boucle
Loop

Do While (condition)
      'Traitement à réaliser
Loop

Do Until (condition)
      'Traitement à réaliser
Loop

Do
      'Traitement à réaliser
Loop Until (condition)

Do
      'Traitement à réaliser
Loop While (condition)
```

Conditions de sortie des boucles

Il est possible de forcer la sortie de chaque type de boucle avec l'instruction appropriée **Exit For**, **Exit Do**, **Exit While**. Le traitement continue à la ligne suivant la fin de la boucle. Le recours à cette instruction doit être limitée.

Dans la condition d'une boucle, pour les expressions longues à évaluer, il est recommandé de mettre une partie simple en début de condition, puis la suite de l'expression après les mots clés **ANDALSO** ou **ORELSE**.

Divers

Récursivité

VB.NET supporte l'appel récursif d'une fonction. La récursivité est parfois une alternative pour simplifier du code.

Gestion des exceptions

VB.NET offre un mécanisme intégré de gestion structurée des exceptions qui surviennent durant l'exécution du code. La partie de code à risque est encadré par les instructions **Try** et **End Try**. La syntaxe complète est la suivante:

Try

    'code à risque

Catch Ex As exception [When expression] 'condition optionnelle supplémentaire

'traitement exécuté en cas d'erreur

[Exit Try] 'saute après le End Try

[Finally]

'traitement optionnel toujours exécuté avant de quitter un bloc Try …. End Try

End Try

Exemple:

```
Public Sub TryExemple()
   Dim x As Integer = 5
   Dim y As Integer = 0
   Try
      x /= y   ' Cause "Divide by Zero" erreur
   Catch ex As Exception When y = 0
      MsgBox(ex.toString)   'Affiche le message d'erreur en clair
   Finally
      Beep()   ' Beep après le traitement avec ou sans erreur
   End Try
End Sub
```

Convention d'écriture

L'attribution des noms des variables peut respecter les conventions suivantes pour permettre une reconnaissance plus aisée de son type:

Type de la variable	**Préfixe**	**Exemple**
Byte	byt	bytSemaine
Short	sht	shtCompteur
Integer	i, int	iMois
Long	l, lng	lDistance
Single	sng	sngDimension
Double	d, dbl	dCoordonneeX
Char	C	cAbrev
String	s, str	sNomFichierEntree
Boolean	B	bNouveau
Date	Dt	dtEntree
UDT (user defined type)	2 ou 3 car.	pntDepart pour type Point
Énumération	2 ou 3 car.	colFace pour type Couleur
Classe	Cls	clsVecteur pour classe Vecteur

Les noms des constantes sont écrits en majuscule. On utilise une majuscule pour chaque nouveau mot intervenant dans le nom de la variable

VB.NET et les objets

Programmer avec des objets

Cette approche de développement de programme consiste à décomposer une application en objets réels ou virtuels capables de comportement en plus de mémoriser un état.

La programmation orientée objet (POO) produit du code plus facile à comprendre, maintenir et réutiliser.

VB.NET est une version totalement nouvelle du langage Visual Basic supportant réellement les concepts de la POO.

Il est essentiel de comprendre les rudiments de la POO car VB.NET fait appel à de nombreuses classes d'objets prédéfinis dans le Framework. Il en est de même pour les API des systèmes de CAO.

La POO s'appuie sur les concepts fondamentaux suivants:

- La classe et ses instances
- L'encapsulation
- L'héritage
- Le polymorphisme

Classe et instance

La classe est un des éléments de base de la POO. C'est avant tout un type au même titre que Integer ou Boolean. Une classe décrit un ensemble d'objets ayant les mêmes attributs (propriétés) et comportements (méthodes). Lors de la conception d'une application, le programmeur définit donc à partir des spécifications, des classes d'objets. Une classe est en fait un modèle de l'objet. Chaque classe sert ensuite à créer des objets individuellement identifiables appelés des instances de la classe. Chaque instance d'une classe partage la même description mais chacune a des valeurs propres pour ses attributs (valeur de ses propriétés).

Classe dans VB.NET

Une classe dans VB.NET contient des propriétés et des méthodes. Une propriété est déclarée par le mot clé **Property** et **Public**, pour être visible de l'extérieur de la classe. Une propriété est composée de deux parties **Set** qui permet de définir sa valeur et **Get** qui permet d'extraire la valeur. Une propriété peut aussi être déclarée en lecture seule avec **ReadOnly** et dans ce cas Set n'est pas défini ou en écriture seule avec **WriteOnly** et Get n'est pas défini, ce qui est plus rare. Il ne faut pas confondre les propriétés et les variables internes d'une classe qui ne doivent pas être déclarées **Public** pour ne pas contrevenir au principe de l'encapsulation (la classe est vue comme une boite noire de l'extérieure, ce qui est à l'intérieure doit être invisible).

La définition d'une méthode revient à définir une fonction ou un sous-programme à l'intérieur de la définition de la classe. Une fonction définie comme **Public** est accessible à l'extérieur de la classe, avec **Private**, elle ne peut être utilisée que dans la classe.

Exemple:

```
Public Class clsPoint
      Dim m_Coord(2) as Double      'une variable interne de la classe
      Public Property CoordX as Double    'une propriété de la classe
            Get
                  Return m_Coord(0)
            End get
            Set (ByVal valX as Double)
                  m_Coord(0) = valX
            End Set
      End Property
      Public Sub translateXY (ByVal valX As Double, ByVal valY As Double)
            m_Coord(0) = valX
            m_Coord(1) = valY
```

```
      End Sub
End Class
```

Références et instance

Une variable objet (**Dim** unObjetA **As** ObjetA) ne contient pas l'objet lui-même mais en fait une référence à cet objet. C'est donc totalement différent d'une variable de type Integer par exemple qui contient directement sa valeur. Il est donc possible que plusieurs variables objets fassent référence au même objet, ou dit différemment, pointent le même objet.

De ce fait la création d'une instance d'une classe se fait en deux étapes. Il faut définir une variable de référence et l'instance elle-même:

```
Dim NomRefInstance As NomClasse 'définit une variable de référence à un objet
NomRefInstance = New NomClasse 'définit un nouvelle objet
```

Exemple:

```
Dim unPoint As clsPoint 'référence sur clsPoint (pointe sur rien!)
unPoint = New clsPoint  'instance de clsPoint, affecte adresse à unPoint
Dim unAutrePoint As clsPoint
unAutrePoint = unPoint  'les deux variables pointent la même instance
```

Il est extrêmement aisé d'accéder à une propriété d'une classe en lecture ou en écriture:

```
NomRefInstance.NomPropriété = valeur
valeur = NomRefInstance.NomPropriété
```

Exemple:

```
Dim dblX As Double = 5.
unPoint.CoordX = dblX
unPoint.translate (2,4)
dblX = unAutrePoint.CoordX          'dbl contient 7
```

Surcharge des méthodes

La surcharge permet de donner le même nom à plusieurs fonctions tant que la liste des arguments permet de les différentier. Il faut ajouter le mot clé **Overloads** en début de ligne pour toutes les fonctions.

Héritage

Ce concept nouveau dans VB est fondamental pour créer un système réellement basé sur les objets. L'héritage permet de spécialiser une classe d'objet en lui conférant des méthodes et des propriétés plus spécifiques tout en conservant les caractéristiques héritées de la classe de base ou parent. Une classe héritée est créée avec le mot clé **Inherit**.

Exemple:

```
Public Class clsCercle
      Inherits clsPoint
      Dim m_rayon As Double
      Public Property Rayon As Double
            Get
                  …
            End Get
            Set
                  …
            End Set
      End Property
End Class
```

Une méthode de la classe dérivée déclarée avec **Overrides** se substitut à une méthode de la classe de base si elle a été déclarée remplaçable par le mot **Overridable**.

L'héritage d'une classe et la réécriture d'une méthode peuvent être rendus obligatoire par les mots clés **MustInherit** et **MustOverride**. Si une méthode doit être redéfinie, alors la classe doit être dérivée.

Dans VB.NET, tous les objets héritent de la classe de base **Systeme.Object**.

Constructeur

Lors de la création d'une instance d'un objet, **New** réserve la place nécessaire en mémoire pour contenir l'objet. L'initialisation de l'objet est réalisée par l'un des sous-programmes appelés constructeurs de la classe. Les constructeurs peuvent aussi être surchargés.

Exemple:

```
Public Class clsPoint
      …
      Public Sub New (ByVal valX As Double, ByVal valY As Double)
            m_coord(0) = valX
            m_coord(1) = valY
      End Sub
End Class

Dim unPoint As clsPoint
unPoint = New clsPoint (5,7)
```

Espace des noms

Les espaces de noms permettent une structuration hiérarchique des objets et des modules d'un programme. Il contribue à la lisibilité et à la modularité. La raison principale pour l'existence des espaces de noms est d'assurer l'unicité des noms des identifiants. En fait, il est impossible de garantir qu'un même nom n'a pas été utilisé ailleurs, par exemple dans le Framework.NET ou dans une API d'un système de CAO. Les espaces de noms contrôlent la portée des identifiants.

Exemple

```
Namespace MGA802.Exemple
      Public Class clsPoint
            …
      End Class
End Namespace
```

Cette déclaration est identique à la suivante:

```
Namespace MGA802
      Namespace Exemple
            Public Class clsPoint
                  …
            End Class
      End Namespace
End Namespace


Dim unPoint As MGA802.Exemple.clsPoint
```

Membres et objets partagés

Pour travailler avec un objet, il faut normalement créer une instance d'abord. Cependant, il est possible d'utiliser directement les éléments qu'une classe partage avec l'ensemble de ses instances sans même créer d'instance. Ces éléments sont identifiés par le mot clé **Shared**.

Exemple:

```
Public Class clsPoint
      ReadOnly Shared Property nbr as Integer = 0     'nombre de points créés
End Class
```

Dans VB.NET, les modules sont en fait des classes dont tous les membres sont partagés.

Bibliographie

Visual Basic.NET en 21 jours, Ducan Mackenzie, Kent Sharkey, Campus Press, 2002,
ISBN: 2-7440-1369-2