Annexe – Cours Visual Basic

Manipulation des fichiers en Visual Basic

VILLETTE Charles FAVRE Olivier

I) Introduction

Visual Basic.net a intégré des classes liées à la manipulation des fichiers et des dossiers.
L'idée était de s'écarter de Visual Basic, puisque la manipulation de fichiers fait partie (encore sous .net) des points faibles de Visual Basic.

Nous allons ici nous intéresser à la manipulation de fichier/dossier sous Visual Basic.net, puis la manipulation de fichiers sous Visual Basic.

II)Visual Basic.net

Visual Basic.net se base sur cinq classe pour manipuler des fichiers/dossiers :

- `Directory`
- `DirectoryInfo`
- `Path`
- `File`
- `FileInfo`

Pour les utiliser il faut importer l'espace de nom System.IO (`Imports System.IO`)

1) Propriétés communes à `Directory` et `File`

Nous listons ici les ressemblances entre ces deux classes, parmi les fonctions “utiles”.

Membres communs aux classes `Directory` et `File`	
Méthode	Description
`GetCreationTime`	Date et heure de création du dossier/fichier
`GetLastAccessTime`	Date et heure du dernier accès au dossier/fichier
`GetLastWriteTime`	Date et heure du dernier accès en écriture
`Move`	Déplace un dossier/fichier et son contenu vers le dossier spécifié
`SetCreationTime`	Définit la date et l'heure de la création du dossier/fichier
`SetLastAccessTime`	Définit la date et l'heure du dernier accès au dossier/fichier
`SetLastWriteTime`	Définit la date et l'heure du dernier accès en écriture du dossier/fichier

2) Classe `Directory`

Cette classe comme son nom l'indique s'occupe exclusivement de la manipulation de dossier.
Voici les méthodes importantes qui la compose :

Membres caractéristiques de la classe `Directory`	
Méthode	Description
`CreateDirectory`	Créer le dossier spécifié par un chemin (`Path`)
`Delete`	Supprime un dossier **et** son contenu
`Exists`	Détermine si le chemin d'accès spécifié existe
`GetCurrentDirectory`	Correspond au dossier de travail de l'application, il peut être modifié par `SetCurrentDirectory`
`GetDirectories`	Tableaux des noms et des sous-dossiers du dossier spécifié
`GetDirectoryRoot`	Retourne le dossier racine du chemin d'accès spécifié. Typiquement, cela renvoi la lettre initiale (`"C:\"`, `"D:\"`...)
`GetFiles`	Tableaux des noms de fichiers du dossier spécifié
`GetLogicialDrives`	Tableaux des noms de lecteurs disponibles (ex : C:\, D:\...)
`GetParent`	Récupère le dossier parent du dossier spécifié
`SetCurrentDirectory`	Définit le dossier de travail de l'application

Exemples :
1 – Récupérer tous les fichiers d'un dossier :

```
Dim fichiers As String = Directory.getFiles("C:\")
Dim unFichier As String

For Each unFichier In fichiers
          'Manipulation du fichier unFichier ici
Next
```

2 – Récupérer tous les sous-dossiers d'un dossier :

```
Dim dossiers As String = Directory.getDirectories("C:\")
Dim unDossier As String

For Each unDossier In dossiers
          'Manipulation du dossier unDossier ici
Next
```

3) Propriétés communes à `DirectoryInfo` et `FileInfo`

Nous listons ici les ressemblances entre ces deux classes, parmis les fonctions “utiles”.

Membres communs aux classes DirectoryInfo et FileInfo		
Type	Membre	Description
Propriétés	`Exists`	`Boolean` indiquant si le dossier/fichier représenté par l'instance de la classe existe ou non
	`Name`	Nom du dossier/fichier
Méthodes	`Create`	Créer le dossier/fichier s'il n'existe pas
	`Delete`	Supprimer le dossier/fichier, ainsi que le contenu d'un dossier dans le cas de `DirectoryInfo`
	`MoveTo`	Déplace le dossier/fichier, ainsi que le contenu d'un dossier dans le cas de `DirectoryInfo`

4) Classe `DirectoryInfo`

La classe `Directory` est une classe statique (on y accède sans l'instancier), tandis que la classe `DirectoryInfo` doit être instanciée.
La classe `DirectoryInfo` *représente* un dossier, elle est donc à utiliser lorsqu'il s'agit de manipuler un dossier.
On utilise plutôt `Directory` lorsqu'il s'agit de faire un parcours dans les dossiers, tandis que l'on utilise `DirectoryInfo` lorsque l'on cherche à obtenir des informations détaillées sur un dossier.
Il est bien entendu tout à fait possible de combiner les deux.

Membres caractéristiques de la classe `DirectoryInfo`		
Type	Membre	Description
Propriétés	`Parent`	Dossier parent du dossier représenté par l'instance de la classe
	`Root`	Partie racine du dossier
	`FullName`	Nom complet (path complet du fichier)
Méthodes	`CreateSubdirectory`	Créer le sous-dossier spécifié. Si les dossiers parent spécifiés dans le path n'existent pas, ils sont également créés.
	`GetDirectories`	Retourne les sous-dossiers du dossier
	`GetFiles`	Retourne la liste des fichiers du dossier
	`GetFileSystemInfos`	Retourne un tableau d'objets `FileSystemInfo`

Exemple :
```
Dim di As DirectoryInfo = new DirectoryInfo("C:\Windows\system32")
di.Parent                'renvoie C:\Windows
di.FullName              'renvoie C:\Windows\system32
```

5) Classe File

La classe `File` et la classe `FileInfo` suivent le même schéma que la classe `Directory` et la classe `DirectoryInfo`. La première est statique et s'utilise plutôt sur un groupe de fichier, tandis que la deuxième s'intéresse à un fichier en particulier, et est instanciable.

Ces deux classes peuvent être vues comme ambiguës, pour un soucis de vitesse et de légèreté du code, elles embarques des fonctions normalement associées à un `StreamReader` (lecture du contenu), et un `StreamWriter` (écriture de contenu).
Dans Visual Basic.net, il est donc possible d'écrire dans un fichier sans avoir recours à l'une de ces classes, c'est un gain de temps considérable **mais** dans la pratique on se dirigera toujours vers un `StreamReader/Writer`, plus complet et plus approprié.

Membres caractéristiques de la classe `File`	
Méthode	Description
`AppendText`	Créer un objet `StreamWriter` pour ajouter du texte dans un fichier existant
`Copy`	Copie un fichier existant
`Create`	Créer le fichier spécifié
`CreateText`	Ouvre un fichier existant ou créer un nouveau fichier pour y écrire du texte
`Delete`	Supprime le fichier spécifié, ne lève pas d'exception si le fichier n'existe pas.
`Exists`	Détermine si le fichier existe ou non
`GetAttributes`	Obtient les attributs du fichier spécifié. Les attributs sont précisés par l'énumération de constantes `FileAttributes` (voir ci-dessous)
`Open`	Ouvre un objet `FileStream` à partir du fichier spécifié
`OpenRead`	Ouvre un fichier existant en lecture
`OpenText`	Ouvre le fichier texte existant en lecture
`OpenWrite`	Ouvre un fichier existant en écriture
`SetAttributes`	Définit les attributs du fichier spécifié

6) Classe `FileInfo`

Membres caractéristiques de la classe `FileInfo`		
Type	Membre	Description
Propriétés	`Attributes`	Attributs du fichier (voir `FileAttributes`)
	`Directory`	Retourne le dossier parent
	`DirectoryName`	Représente le chemin du fichier sans son nom
	`Length`	Taille du fichier en octets
Méthodes	`AppendText`	Créer un objet `StreamWriter` pour ajouter du texte dans un

		fichier
	`CopyTo`	Copie un fichier existant
	`CreateText`	Créer un objet `StreamWriter` pour écrire dans un fichier texte
	`Open`	Ouvre un fichier avec divers privilèges de lecture/écriture/partage
	`OpenRead`	Créer un objet `FileStream` en lecture seule
	`OpenText`	Créer un objet `StreamReader` pour lire le contenu d'un fichier texte existant
	`OpenWrite`	Créer un objet `FileStream` en écriture

7) Classe `FileAttributes`

Cette classe est à mettre en relation avec les différentes parties `Attributes` que l'on retrouve dans ces objets, elle est à utiliser avec des `And` pour l'énumération :

Valeur de l'énumération `FileAttributes`	
Attribut	Description
`Archive`	Les applications utilisent cet attribut afin de marquer les fichiers pour la sauvegarde
`Compressed`	Fichier compressé
`Directory`	Dossier
`Encrypted`	Le fichier ou dossier est crypté. Pour un dossier cela signifie que tous les nouveaux fichiers et dossiers créés seront cryptés par défaut
`Hidden`	Fichier masqué, non compris dans les collections de fichiers ordinaires
`Normal`	Aucun attribut défini. Cet attribut est valable uniquement s'il utilisé seul
`ReadOnly`	Fichier en lecture seule
`System`	Le fichier est un fichier système. Il fait partie du système d'exploitation ou est utilisé en mode exclusif par celui-ci

8) Classe `Path`

On ne s'attardera pas sur cette classe car ses méthodes sont explicites. Il est ici question de manipulation de `String` de type chemin d'accès. Il est par exemple possible d'obtenir l'extension d'un fichier, de la changer...

9) Lire et écrire un fichier texte

Les classes `StreamReader` et `StreamWriter` sont respectivement dédiées à l'écriture ou à la lecture dans des fichiers texte.
Ces classes font l'usage d'un buffer à la manière de Java/C, mais que celui-ci est transparent, il faut donc penser à utiliser la méthode `Flush` commune aux buffers qui permet de déclencher l'opération d'écriture/lecture à un instant précis.
Si cet appel n'est pas réalisé, la fermeture du fichier fait automatiquement appel à cette fonction.

Classe `StreamWriter` :

Les membres caractéristiques de la classe `StreamWriter`		
Type	Classe	Description
Propriétés	`AutoFlush`	`Boolean` indiquant si l'objet est écrit puis vide le contenu de sa mémoire tampon (buffer) dans le fichier, après chaque appel aux méthodes `Write` et `WriteLine`
	`Encoding`	Représente l'encodage des caractères (ANSI, UTF-8, Unicode...) du fichier
Méthodes	`Close`	Ferme l'objet `StreamWriter`. Cette méthode appelle `Flush`.
	`Flush`	Écrit puis efface le contenu de la mémoire tampon (buffer) dans le fichier
	`Write`	Écrit une donnée dans le fichier. Cette méthode est surchargée de manière à accepter tous les types de données prédéfinis
	`WriteLine`	Idem que `Write`, ajoute un saut de ligne à la fin de l'écriture

Classe `StreamReader` :

Les membres caractéristiques de la classe `StreamReader`		
Type	Membre	Description
Méthodes	`Close`	Ferme et libère les ressources occupées par le fichier
	`Peek`	Retourne le prochain caractère disponible mais n'avance pas le pointeur
	`Read`	Lit le nombre de caractères spécifiés
	`ReadLine`	Lit une ligne de caractères
	`ReadToEnd`	Lit le contenu du fichier puis la position en cours jusqu'à la fin

Exemples
1-Lecture d'un fichier :

```
'A partir d'un streamReader
Dim monFichier As New StreamReader("C:\fichier.txt")

'équivalent : a partir de la classe File
Dim monFichier As StreamReader = File.OpenText("C:\fichier.txt")

'lecture jusqu'à la fin du fichier
```

```
Do Until monFichier.peek = -1
          'Récupérer la ligne courante comme suit :
          monFichier.ReadLine()
Loop
' S'il s'agit de lire tout le fichier, jusqu'à la fin, il est possible
' d'utiliser : monFichier.ReadToEnd
```

2-Ecriture d'un fichier :

```
'Création d'un fichier (s'il n'existe pas), et écriture
Dim monFichier As New StreamWriter("C:\fichier.txt", True)
'Idem via File
'S'il n'existe pas
Dim monFichier As StreamWriter = File.CreateText("C:\fichier.txt")
'En ajout
Dim monFichier As StreamWriter = File.AppendText("C:\fichier.txt")

'Ecriture d'une ligne "Ligne1"
monFichier.writeLine("Ligne1")

'Fermeture du fichier
monFichier.Close()
```

Quelques exemples de classe de lecture de fichiers spécifiques :

Classe	Description
`BinaryWriter`	Fichiers binaires en écriture
`BinaryReader`	Fichiers binaires en lecture
`XmlTextWriter`	Fichier XML en écriture
`XmlTextReader`	Fichier XML en lecture

On ne détaillera pas ici l'utilisation de ces différentes classes, l'idée reste toujours la même : ouverture puis parcours/écriture du document.

III) Visual Basic

Visual Basic n'intègre pas tout ce dont dispose Visual Basic.net, c'est d'ailleurs son plus gros point faible. Il est cependant possible de faire à peu près les mêmes choses :

1) Lister les fichiers et sous dossier d'un dossier

Sous Visual Basic, pour manipuler des arborescences de document, on se réfère à la fonction `Dir`, qui permet de récupérer l'ensemble des fichiers/dossiers d'un dossier.

La syntaxe de cette fonction (voir MSDN) est présentée sous cette forme :

```
Public Overloads Function Dir( _
   Optional ByVal PathName As String, _
```

```
   Optional ByVal Attributes As FileAttribute = FileAttribute.Normal _
) As String
```

On remarque donc que le deuxième argument, lorsqu'il n'est pas précisé, est automatiquement mis à Normal, ce qui indique que `Dir` ne renverra que les fichiers *normaux* c'est à dire les fichiers n'étant pas cachés ou système...

Pour pouvoir sélectionner les fichiers que l'on veut obtenir, ce tableau présente les différentes options prises par le second argument de `Dir`. Il suffit de les concaténer avec l'opérateur + si l'on veut avoir plusieurs types de fichiers :

Constante	Valeur
`vbNormal`	0
`vbReadOnly`	1
`vbHidden`	2
`vbSystem`	4
`vbVolume`	8
`vbDirectory`	16

Il est possible de préciser soit la constante soit la valeur, Visual Basic ne fait aucune différence.

Voici un exemple pour scanner l'ensemble des fichiers d'un dossier précisé (*path*) :

```
Dim FileName As String
Dim Path As String
Dim Filter As String

Path = "c:\temp\*.*"

FileName = Dir(Path)
While FileName <> ""
          MsgBox  FileName
          FileName = Dir
Wend
```

Comme vous pouvez le constater, la fonction `Dir` est très puissante puisqu'elle est apte à supporter une recherche par expression régulière !
Par exemple :

`*.bak`	Uniquement les fichiers d'extension `.bak`
`??*.c`	Fichier commencant par deux lettres, suivit de caractères, et d'extension `.c`
`*.*`	Tous les fichiers
`([0-9]+)`	Des dossiers/fichiers n'ayant que des chiffres dans leur nom

2) Lecture d'un fichier

Pour ouvrir un fichier, on se rapproche ici d'un fonctionnement algorithmique standard :

```
Dim Chaine As String
Dim NumFichier As Integer
NumFichier = FreeFile
Open "C:\test.txt" For Input As #NumFichier     'Ouverture
Do While Not EOF(NumFichier)                    'Fin de fichier
  Line Input #NumFichier, Chaine                'Lecture et stockage dans Chaine
  MsgBox Chaine                                 'Affichage de ce que l'on a lu
Loop
Close #NumFichier                               'Fermeture du fichier
```

Il faut ici bien comprendre le principe : on ouvre un fichier à l'aide de `Open`, le `For Input` précise que l'on va écrire dedans et le `As #NumFichier` est un numéro permettant de connaître le fichier ouvert (une sorte d'id que l'on utilise à chaque opération sur le fichier).
Cet id **doit** être unique, pour cela on utilise la fonction `FreeFile` qui stocke tous les id déjà utilisés, et qui renvoi donc un id non utilisé, cela permet de s'assurer que l'id que l'on reçoit de cette fonction n'est pas utilisé à un autre endroit dans le programme.
Le `EOF` indique que le marqueur de fin de fichier à été levé : tant que l'on ne croise pas `EOF`, on continu à lire le fichier. Enfin, `Line Input` s'occupe de lire une ligne du fichier.
La fonction `Close` prend en paramètre l'id du fichier et non son nom.

3) Écriture d'un fichier

Pour écrire un fichier, le fonctionnement est très proche de la lecture vu ci-dessus :

```
Dim Chaine As String
Dim x As Integer
Dim NumFichier As Integer
x = 3
NumFichier = FreeFile
Open "C:\test.txt" For Output As #NumFichier    'Ouverture
  Write #NumFichier, x, 5                       'Ecrit « 3, 5 »
  Write #NumFichier "This is a test !"          'Ecrit « "This is a test !" »
  Print #NumFichier "This is a new test !"      'Ecrit « This is a new test ! »
Close #NumFichier                               'Fermeture du fichier
```

De la même façon, il est possible d'employer le mot clef `Append` en lieu et place de `Ouput`, ainsi le fichier ne serait pas effacé lors de l'écriture mais complété.
Notez la différence entre `Write` et `Print`, l'un enregistre les guillemets tandis que l'autre non, suivant les situations vous pourrez donc être amené à utilisé tantôt l'un et tantôt l'autre.

IV) Ecriture distante

Un sujet que nous n'avons pas ici abordé mais qui occupe une place parfois non négligeable est l'écriture à distance. Comprenez par là l'écriture sur un réseau. Il est en effet possible d'écrire sur un réseau de la même façon que l'on écrit un fichier en local.
Par exemple, dans Visual Basic pour écrire/lire un fichier à travers un réseau il est on ne peut plus simple de le faire :

```
Open "\\unserveur\share\monfichier.txt" For Output As #NumFichier
```

Nous n'irons pas plus loin sur ce détail tant les possibilités sont nombreuse, il est cependant tout à fait possible de trouver de nombreuses pages parlant de l'écriture distante avec Visual Basic/.net