

OpenXML en VB6 Lire et Ecrire des fichiers .xlsx

par bbil ([La page à bbil](#))

Date de publication : 02 février 2009

Dernière mise à jour :

Le but de cet article est la lecture/écriture de documents OpenXML, depuis un programme écrit en VB6.
Pour l'exemple nous traiterons la lecture/écriture de valeurs dans un document Excel (.xlsx), cependant la méthode est transposable aux autres types de documents OpenXml (Word, Powerpoint...).

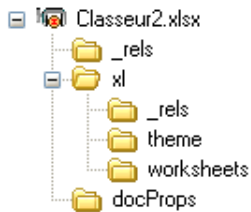
I - Avant-propos.....	3
I-A - Le format Open XML.....	3
I-B - Principe retenu.....	4
Gestion des packages.....	4
Gestion des fichiers XML.....	4
II - Pré-requis.....	5
II-A - Interaction VB6 et VB.NET.....	5
II-B - Installation du Wrapper.....	8
II-B-1 - Installation des outils en ligne de commande.....	8
II-B-2 - Enregistrement du Wrapper.....	9
II-C - Télécharger vb6-OpenXML.dll.....	10
III - La Classe oxWorkbook.....	11
III-A - Fonction ouverture classeur : Open.....	11
III-B - Lecture de Workbook.xml.....	11
III-C - Lecture de l'Xml d'une feuille.....	12
III-D - Enregistrer les changements de la feuille.....	12
III-E - Fermer le classeur.....	12
III-F - Liste de chaînes.....	13
III-G - Télécharger les sources du projet vb6_OpenXML.....	13
IV - Gestion des parties XML.....	14
IV-A - Ouverture du classeur.....	14
IV-A-1 - Fonctionnement de l'Ouverture du classeur.....	14
IV-A-2 - Utilisation du fichier sharedstring.xml.....	15
IV-A-3 - La fonction Ouverture classeur.....	16
IV-B - Lecture/Ecriture d'une Cellule.....	17
IV-B-1 - Contenu des parties feuilles : sheetX.xml.....	17
IV-B-2 - Contenu d'une Cellule.....	18
IV-B-3 - Ouverture de la feuille de calcul.....	18
IV-B-4 - Lecture du contenu d'une cellule.....	19
IV-B-5 - Ecriture dans une cellule.....	21
Formatage des données à écrire.....	21
Extraction Numéro ligne et colonne.....	22
Cas où la cellule n'existe pas.....	23
Si la ligne existe :.....	23
Si la ligne n'existe pas :.....	24
Mise à jour des données cellules.....	24
IV-C - Fermer la feuille.....	24
IV-D - Fermer classeur.....	25
V - Exemple d'application : prjOpenXML.....	26
V-A - fenêtre principale.....	26
V-B - Fonctionnement.....	26
V-C - Télécharger l'exemple d'application prjOpenXML.....	28
VI - Liens utiles.....	29
OpenXML Schéma.....	29
Open SDK 2.0.....	29
Interaction .Net Framework et VB6.....	29
MSXML.....	29
VII - Conclusions.....	29
VII-A - Remerciements.....	29
VII-B - Vos commentaires.....	29

I - Avant-propos

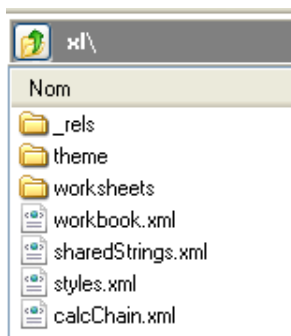
I-A - Le format Open XML

L' OpenXML est le nouveau format de fichier utilisé par la suite Office (version 2007), Dans le présent article nous nous limiterons aux fichiers Excel (.xlsx) .

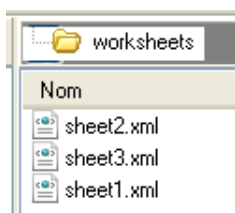
Le fichier .xlsx est en fait un ensemble de fichiers compressés au format .zip, nommé "package" son arborescence ce présente ainsi :



Les "fichiers" qui nous permettrons de lire et écrire dans notre classeur sont situés dans le dossier xl.



On trouve là le fichier WorkBook.xml, qui contient entre autres la liste des feuilles contenues dans notre classeur, et le fichier sharedStrings.xml, qui regroupe les valeurs de type "chaînes de caractères" contenues dans le classeur. Dans le dossier worksheets, se trouve un fichier xml par feuille de calcul :



I-B - Principe retenu

Pour traiter la lecture/écriture dans les fichiers, open XML , j'ai choisi d'effectuer cela en deux grande parties :

Gestion des packages

Le "package" , est l'ensemble de fichiers qui une fois compressé, compose le document OpenXML. Microsoft fourni gratuitement un "SDK OpenXML" qui permet une gestion plus simple des packages, implémentant en autres les fonctions d'extractions/incorporations de "parties" de documents.

Ce SDK nécessite Ce SDK nécessitant l'utilisation du Framework .Net 3.5, cet article nous donnera en plus, un exemple d'utilisation de librairie .Net en VB6.

Gestion des fichiers XML

Une fois l'XML extrait du package, l'on utilisera la librairie Microsoft XML , MSXML2 en version 3.0, pour lire ou modifier les données contenues dans ces XML.

II - Pré-requis

Pour commencer, ayant choisi d'utiliser la dernière version du SDK OpenXML (2.0) celle-ci nécessite d'installer le Framework 3.5 ,

téléchargeable ici : **NET Framework 3.5 SP1**

et ensuite, le SDK : **OpenXML SDK 2.0**

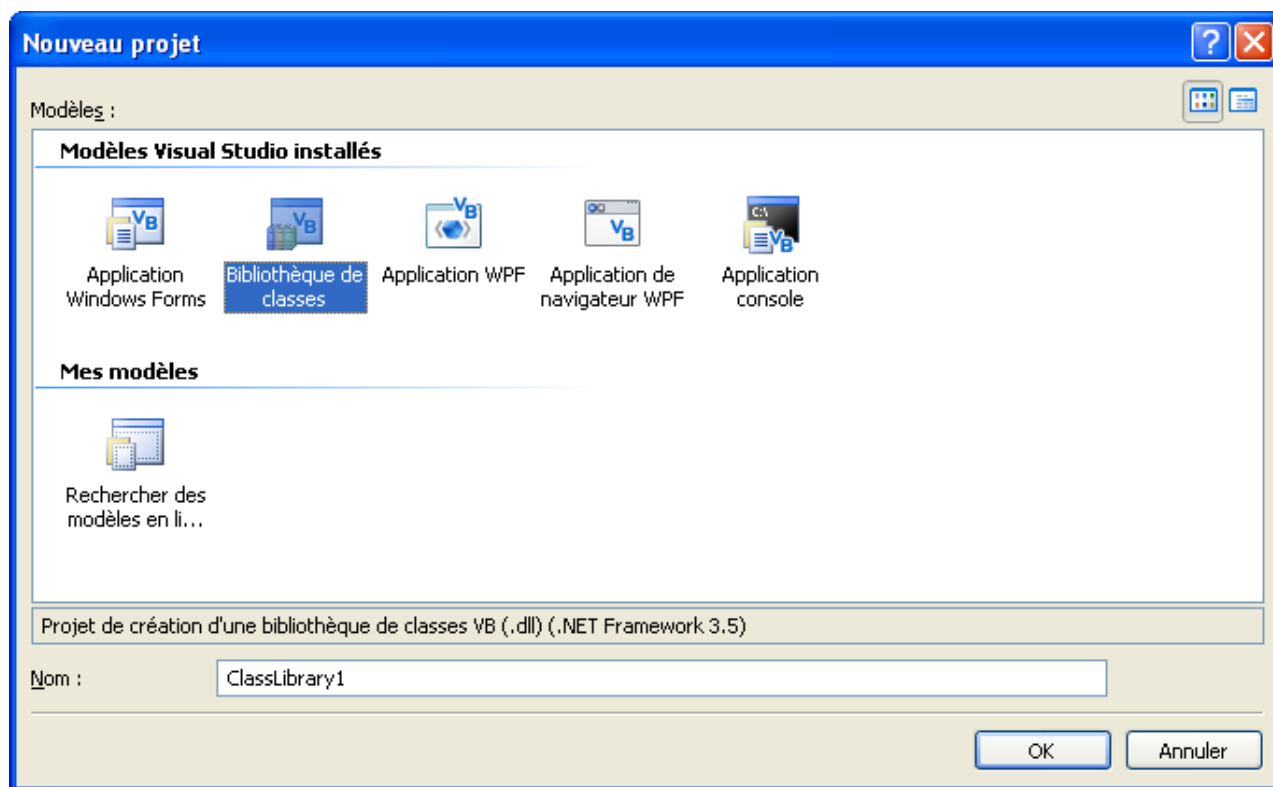
II-A - Interaction VB6 et VB.NET

L'appel des fonctions SDK d'OpenXML depuis VB6, repose sur un objet COM, qui sert de 'Wrapper'.

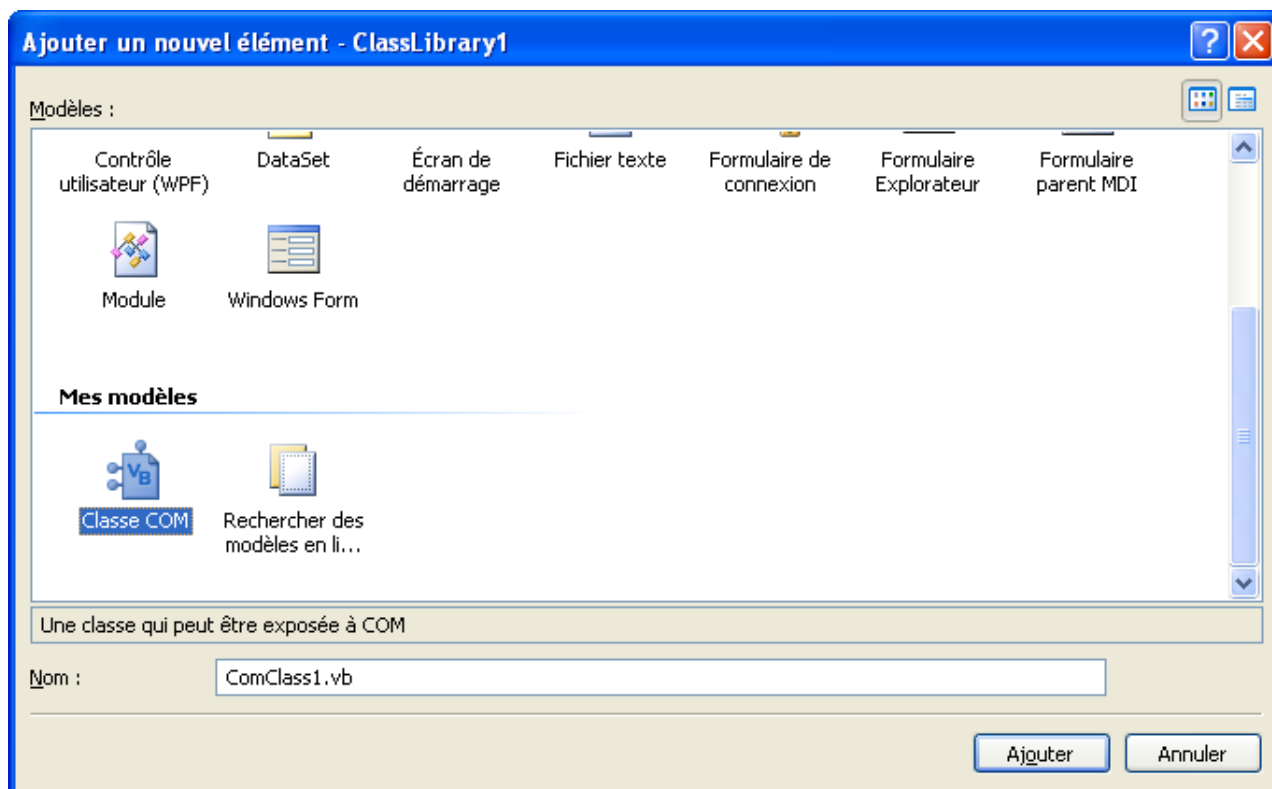
Le "Wrapper" est écrit en .Net dans notre exemple en VB.Net. Pour accéder à son code source vous pouvez utiliser la version gratuite , "Microsoft Visual Basic 2008 Express édition",

disponible sur le site de Microsoft :  **Télécharger Visual Basic 2008 Express**

Ensuite, pour pouvoir créer votre wrapper, vous devez utiliser une bibliothèque de classe



Supprimez la Class1.vb, créée par défaut et grâce au menu "projet ajouter un nouvel élément" et au modèle "Classe Com", ajoutez une nouvelle classe à votre projet :

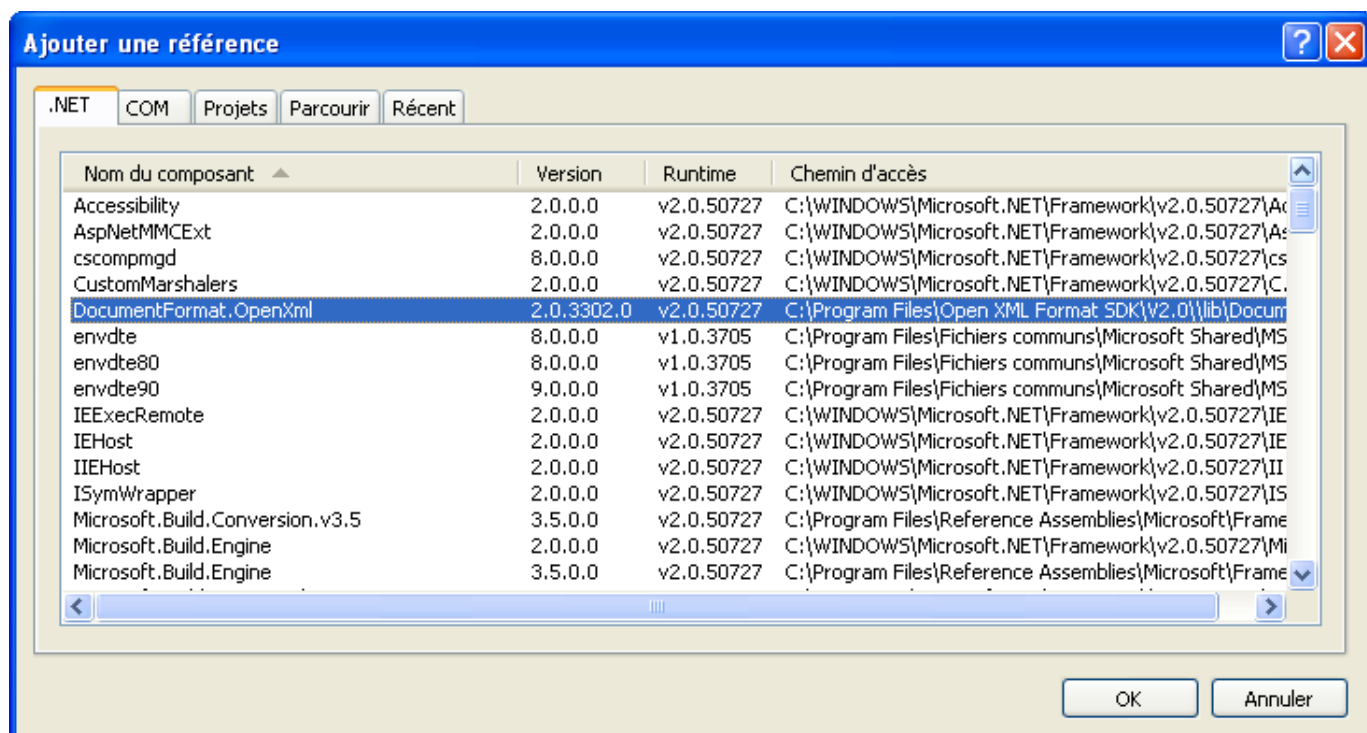


Le modèle [classe com](#) n'est pas installé par défaut avec Visual basic 2008 Express, mais il est téléchargeable avec le source disponible dans l'article : [Appel du .NET Framework à partir d'applications Visual Basic 6.0 existantes](#)

Une fois vbfusion05.exe décompressé, copiez le fichier comclass.zip dans le sous répertoire :

```
\Mes documents\Visual Studio 2008\Templates\ItemTemplates\Visual Basic\
```

Pour avoir accès à l'OpenXML SDK, il faut ensuite ajouter la référence à DocumentFormat.OpenXML



Voilà, les bases du projet VB sont posées. Ce projet fait partie de l'archive téléchargeable en fin d'article.

II-B - Installation du Wrapper

Il y a deux manière d'installer le Wrapper : soit en utilisant Visual basic 2008 Express, et en compilant la source vb6-openXML fournie avec cet article,

soit en utilisant les outils en ligne de commande installés avec le SDK Windows

II-B-1 - Installation des outils en ligne de commande

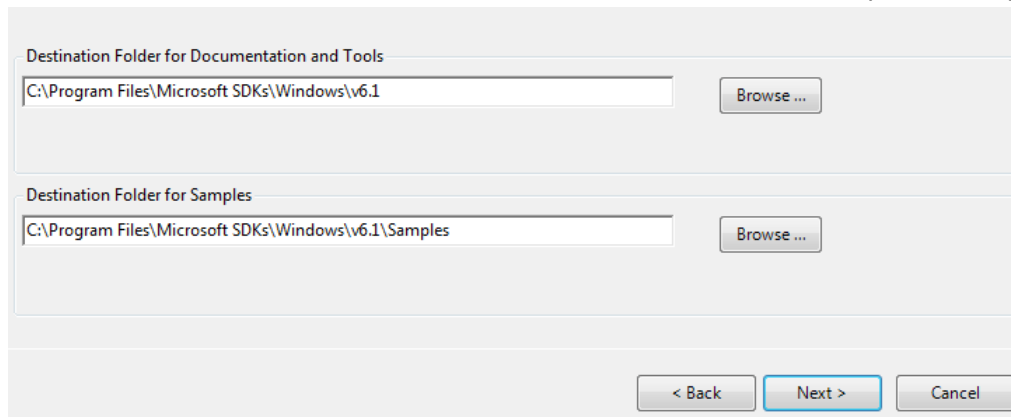
Les outils en ligne de commande sont installés simultanément avec les applications Visual Express. Si vous ne désirez pas installer une de ces applications

utilisez le lien suivant :  **Kit de développement logiciel (SDK) Windows pour Windows Server 2008 et .NET Framework 3.5**

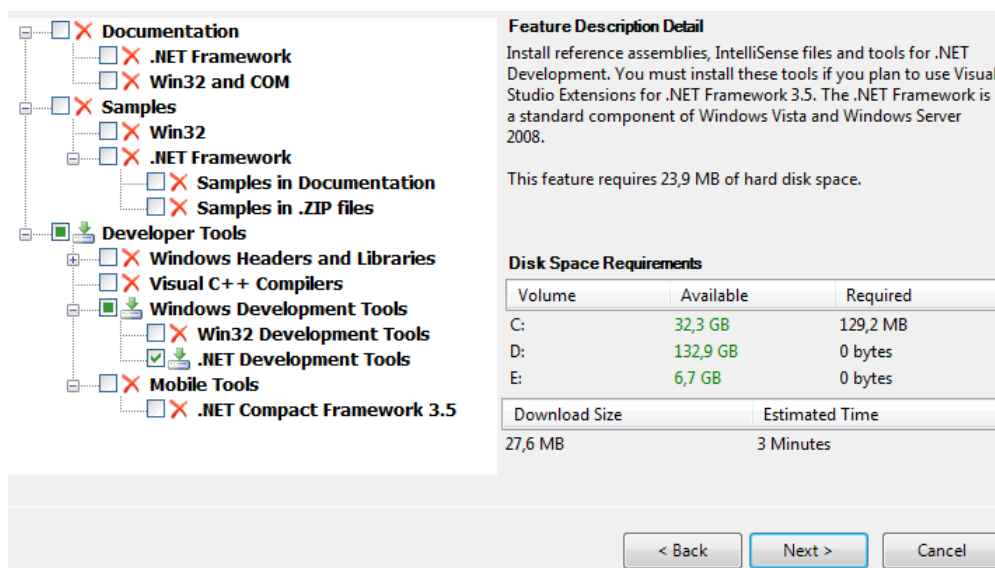
Lors de la procédure d'installation, notez le chemin de destination du kit qui devrait être :

C:\Program Files\Microsoft SDKs\Windows\v6.1\

si le chemin est différent il faudra modifier le fichier install.bat, fourni avec vb6-openXML.dll pour son enregistrement.

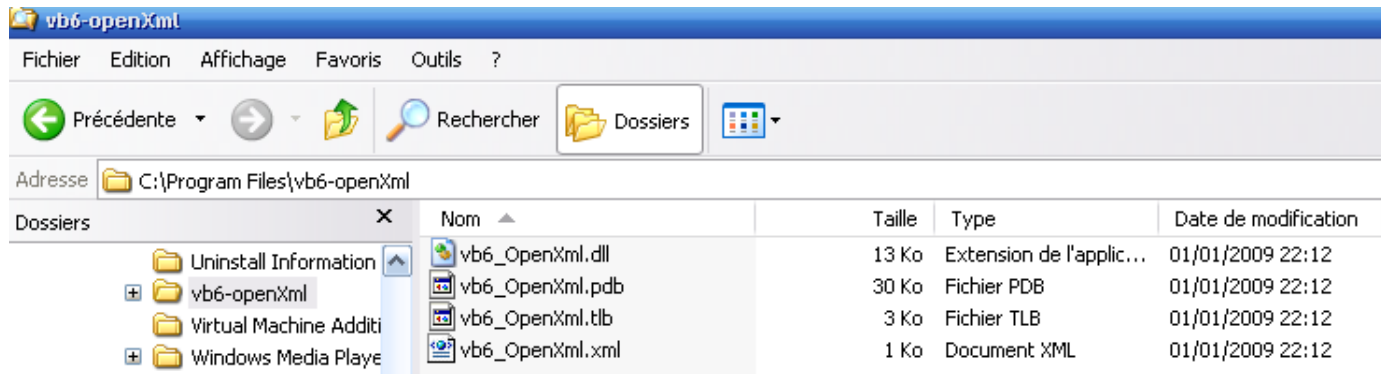


Sélectionner ensuite les "Outils de développement .Net" :



II-B-2 - Enregistrement du Wrapper

Pour installer le Wrapper il suffit de copier tous les fichiers du sous répertoire release de l'application VB.NET vb6-openXML, présent dans le **zip distribution** à la fin de cette section. Copiez les fichiers, par exemple dans un sous répertoire de program Files, vb6-openXML.



Deux utilitaires RegAsm et GacUtil vous permettront d'effectuer cet enregistrement. N'ayant pas trouvé le fichier sdkvars.bat, censé définir les variables d'environnement permettant l'utilisation de ces outils, je vous ai mis, à la fin de cette section, un fichier install.bat permettant de lancer l'enregistrement de vb6_OpenXml, il convient de vérifier les répertoires en fonction de votre installation :

```

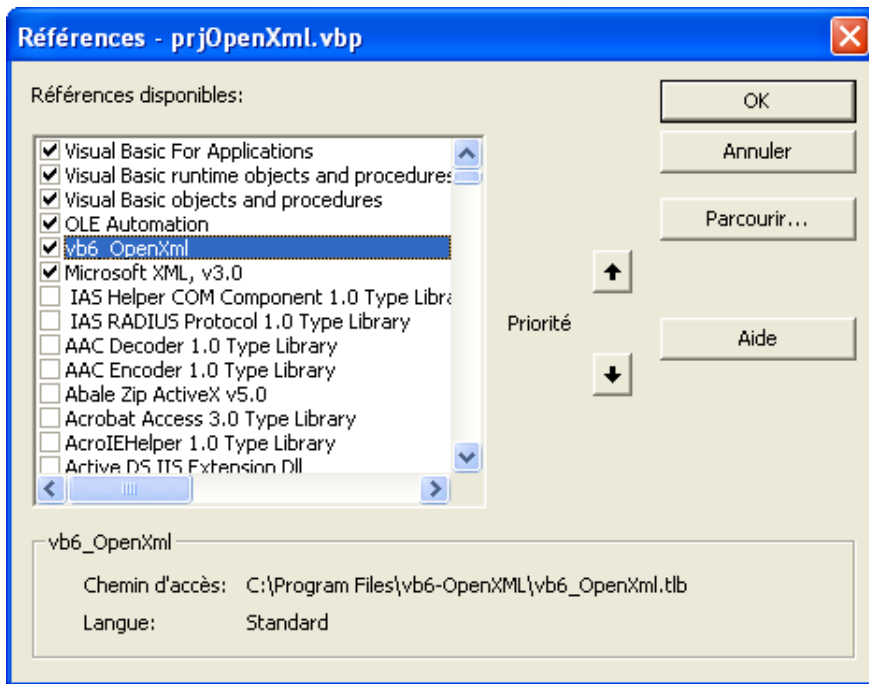
aperçu d'install.bat

@Echo supprime les enregistrement précédent
"C:\Program Files\Microsoft SDKs\Windows\v6.1\bin\gacutil" -u VB6_OpenXML
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\regasm /u "C:\Program Files\vb6-OpenXML\vb6_OpenXml.dll"

@Echo Enregistrement
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\regasm /tlb "C:\Program Files\vb6-OpenXML
\vb6_OpenXml.dll"
"C:\Program Files\Microsoft SDKs\Windows\v6.1\bin\gacutil" -i "C:\Program Files\vb6-OpenXML
\vb6_OpenXml.dll"

pause
    
```

Suite à cela, l'objet .COM vb6-openXML est maintenant accessible en VB6 et il suffit de l'ajouter aux références du projet.



II-C - Télécharger vb6-OpenXML.dll

Le fichier suivant contient, outre la dll vb6-OpenXML.dll, la version 2.0.3930.0 de l'assembly OpenSDK (DocumentFormat.OpenXml.dll) afin d'éviter d'éventuels problèmes lors de changement de versions d'OpenXML.

 **vb6-OpenXML.dll et install.bat**

III - La Classe oxWorkbook

Le projet vb6_OpenXML ne comporte qu'une seule classe : **oxWorkbook**.

III-A - Fonction ouverture classeur : Open

La première fonction mise à disposition par la classe com permet d'initialiser l'ouverture du classeur Excel :

```

'-----'
' Fonction d'ouverture du classeur, renvoie 0 si Ok ou numero d'erreur
' Initialise la variable "globale" xlDoc qui pointe alors sur le Classeur.
'
Function Open(ByVal stNom As String, Optional ByVal bLectureSeule As Boolean = False) As Integer
    Try
        xlDoc = SpreadsheetDocument.Open(stNom, Not (bLectureSeule))
    Catch
        Open = Err.Number 'Renvoi n°Erreur
        Exit Function
    End Try
    Open = 0 'OK
End Function

```

Cette fonction permet l'initialisation d'une variable globale à la classe : **xlDoc**, d'un type SpreadsheetDocument, fourni par le SDK OpenXML. Elle retourne 0 lors d'un succès ou le numéro de l'erreur généré par l'ouverture du classeur.

III-B - Lecture de Workbook.xml

Une fois le classeur ouvert, la variable en lecture seule **wkXml**, met à disposition de l'application hôte le contenu Xml du fichier workbook.xml extrait du package :

```

'
' Propriété en lecture seule retournant XML workbook.xml du classeur Ouvert
'
Public ReadOnly Property wkXml() As String
    Get
        If Not xlDoc Is Nothing Then
            Dim oSr As StreamReader
            oSr = New StreamReader(xlDoc.WorkbookPart.GetStream)
            Return oSr.ReadToEnd
            oSr = Nothing
        Else
            Return ""
        End If
    End Get
End Property

```

Nous verrons plus loin le contenu de workbook.xml (liste des feuilles du classeur).

III-C - Lecture de l'Xml d'une feuille

La fonction **sh_Open** permet l'extraction de l'Xml d'une feuille du classeur ouvert. Elle prend en paramètre l'identifiant de la feuille extrait au préalable par l'application hôte du fichier workbook.xml

```
'  
' Lecture Xml feuille dans classeur ouvert.  
'  
Function Sh_Open(ByVal stSheetID As String) As String  
    Sh_Open = ""  
    Dim oSr As StreamReader  
    oSr = New StreamReader(xlDoc.WorkbookPart.GetPartById(stSheetID).GetStream)  
    Sh_Open = oSr.ReadToEnd  
    oSr = Nothing  
End Function
```

III-D - Enregistrer les changements de la feuille

La fonction **sh_close** permet de fermer la feuille, en remplaçant l'Xml du package associé à cette feuille, par l'Xml passé en paramètre à la fonction.

```
'  
' Ferme et sauve la feuille ouverte par sh_open.  
'  
Function sh_close(ByVal stXml As String, ByVal stSheetID As String) As Boolean  
    Dim oSw As StreamWriter  
    oSw = New StreamWriter(xlDoc.WorkbookPart.GetPartById(stSheetID).GetStream(FileMode.Create))  
    oSw.Write(stXml)  
    oSw.Flush()  
    oSw = Nothing  
    sh_close = True  
End Function
```

III-E - Fermer le classeur

La procédure **Close** permet la libération du classeur et la fermeture du package

```
Sub Close()  
    xlDoc.Close()  
    xlDoc = Nothing  
End Sub
```

III-F - Liste de chaînes

Pour optimiser la taille des fichiers archivés, une partie du document, **sharedStrings.xml**, est utilisée pour stocker les diverses chaînes de caractères contenues dans le classeur.

La propriété **SharedStringXml** permet à l'application hôte d'obtenir l'XML ou d'en modifier le contenu.

```
'
' sharedStrings.xml
'
Public Property SharedStringXml() As String

    Get
        If Not xlDoc Is Nothing Then
            Dim oSr As StreamReader
            oSr = New StreamReader(xlDoc.WorkbookPart.SharedStringTablePart.GetStream)
            Return oSr.ReadToEnd
            oSr = Nothing
        Else
            Return ""
        End If
    End Get
    Set(ByVal value As String)
        If Not xlDoc Is Nothing Then
            Dim oSw As StreamWriter
            oSw = New
StreamWriter(xlDoc.WorkbookPart.SharedStringTablePart.GetStream(FileMode.Create))
            oSw.Write(value)
            oSw.Flush()
            oSw = Nothing
        End If
    End Set
End Property
```

III-G - Télécharger les sources du projet vb6_OpenXML



Sources Visual Basic 2008 de vb6_OpenXML

IV - Gestion des parties XML

Pour la gestion des parties XML, on utilisera le parseur XML de Microsoft MSXML en version 3.0. Il est très probable que celui-ci soit déjà disponible sur votre machine Windows car MSXML est installé avec la version 6.0 d'Internet Explorer par exemple.

IV-A - Ouverture du classeur

IV-A-1 - Fonctionnement de l'Ouverture du classeur

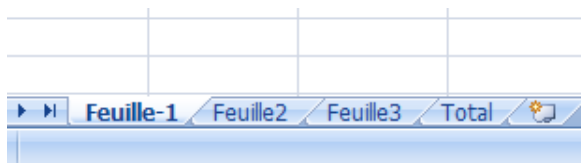
La fonction **OuvreClasseur** consiste à exploiter les données contenues dans les fichiers workbook.xml et sharedstring.xml.

Tout d'abord, une variable globale **wk**, du type **oxWorkbook**, est initialisée de façon à permettre l'accès aux propriétés et méthodes de cette classe.

```
Set wk = New vb6_OpenXml.oxWorkbook
```

Le fichier workbook.xml contient les données relatives aux différentes feuilles du classeur.

Dans l'application on l'utilise d'une part pour récupérer la liste des feuilles du classeur, d'autre part lors de l'ouverture d'une feuille pour récupérer dans ce fichier l'identifiant de la feuille, qui nous servira ensuite de paramètre à la fonction d'ouverture feuille.



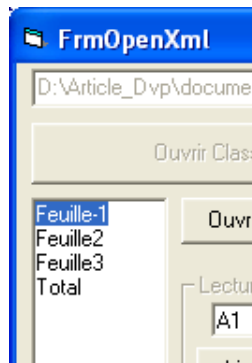
workbook.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<workbook xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main"
  xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships">
  <fileVersion appName="xl" lastEdited="4" lowestEdited="4" rupBuild="4505" />
  <workbookPr defaultThemeVersion="124226" />
  <bookViews>
    <workbookView xWindow="120" yWindow="120" windowWidth="15135" windowHeight="8040" />
  </bookViews>
  <sheets>
    <sheet name="Feuille-1" sheetId="1" r:id="rId1" />
    <sheet name="Feuille2" sheetId="2" r:id="rId2" />
    <sheet name="Feuille3" sheetId="3" r:id="rId3" />
    <sheet name="Total" sheetId="4" r:id="rId4" />
  </sheets>
  <calcPr calcId="124519" />
</workbook>
```

Une boucle For ... Each est utilisée pour parcourir la liste des feuilles et dans un premier temps récupérer le nom de ces feuilles, pour mettre à jour la list box de sélection feuille :

```

(...)
Set wkXml = New MSXML2.DOMDocument
If wkXml.loadXML(wk.wkXml) Then 'Lecture Workbook.xml
Set xmlListeFeuilles = wkXml.selectSingleNode("/workbook/sheets").childNodes
For Each xmlFeuille In xmlListeFeuilles
    Lst.AddItem xmlFeuille.getAttribute("name")
Next
(...)
    
```



liste feuilles

IV-A-2 - Utilisation du fichier sharedstring.xml

Le fichier sharedstring.xml reprend la liste des "valeurs chaînes de caractères" utilisées dans le classeur :

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<sst xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" count="35" uniqueCount="34">
  <si>
    <t>Chaine 1</t>
  </si>
  <si>
    <t>Chaine 2</t>
  </si>
</sst>
    
```

A l'ouverture du classeur, une variable DomDocument est initialisée avec l'XML sharedstring.

```

Set SharedStringXml = New MSXML2.DOMDocument
SharedStringXml.loadXML wk.SharedStringXml
    
```

On utilisera ensuite la variable SharedStringXml pour la lecture des valeurs chaînes de caractères contenues dans les cellules du classeur.

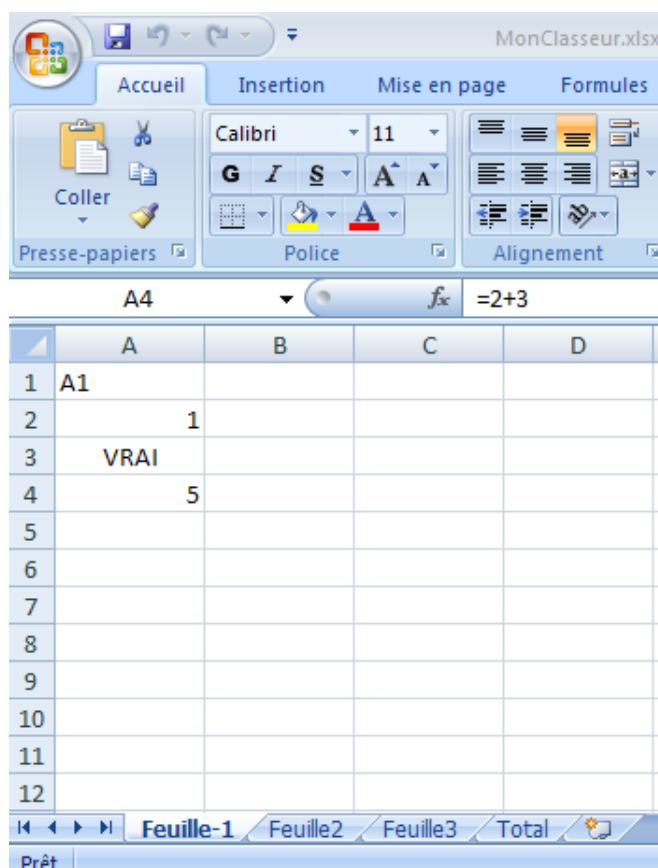
IV-A-3 - La fonction Ouverture classeur

```
'  
' Ouverture classeur  
'  
Function OuvreClasseur(stNom As String, Lst As ListBox) As Long  
Set wk = New vb6_OpenXml.oxWorkBook  
Dim xmlFeuille As IXMLDOMElement  
Dim lRet As Long  
Lst.Clear  
bSSModif = False  
lRet = wk.Open(stNom)  
If lRet = 0 Then 'Ouverture classeur  
Set wkXml = New MSXML2.DOMDocument  
If wkXml.loadXML(wk.wkXml) Then 'Lecture Workbook.xml  
Set xmlListeFeuilles = wkXml.selectSingleNode("/workbook/sheets").childNodes  
For Each xmlFeuille In xmlListeFeuilles  
Lst.AddItem xmlFeuille.getAttribute("name")  
Next  
Set SharedStringXml = New MSXML2.DOMDocument  
SharedStringXml.loadXML wk.SharedStringXml  
Else  
lRet = -1  
wk.Close  
End If  
End If  
OuvreClasseur = lRet  
End Function
```

IV-B - Lecture/Ecriture d'une Cellule

IV-B-1 - Contenu des parties feuilles : sheetX.xml

Le package .xlsx contient un fichier par feuille calcul, placé dans le dossier xl/worksheets, par exemple la partie sheet1.xml du classeur fourni avec cet article :



partie sheet1.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<worksheet xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships">
  <dimension ref="A1:A4" />
  <sheetViews>
    <sheetView tabSelected="1" workbookViewId="0">
      <selection activeCell="A5" sqref="A5" />
    </sheetView>
  </sheetViews>
  <sheetFormatPr baseColWidth="10" defaultRowHeight="15" />
  <sheetData>
    <row r="1" spans="1:1">
      <c r="A1" t="s">
        <v>0</v>
      </c>
    </row>
    <row r="2" spans="1:1">
      <c r="A2">
        <v>1</v>
      </c>
    </row>
    <row r="3" spans="1:1">
      <c r="A3" t="b">
        <v>1</v>
      </c>
    </row>
  </sheetData>
</worksheet>
```

partie sheet1.xml

```

</row>
<row r="4" spans="1:1">
  <c r="A4">
    <f>2+3</f>
    <v>5</v>
  </c>
</row>
</sheetData>
<pageMargins left="0.7" right="0.7" top="0.75" bottom="0.75" header="0.3" footer="0.3" />
</worksheet>
    
```

Les données sont regroupées par ligne

```
<row r="2" spans="1:1"> ...</row>
```

avec l'attribut **r** utilisé pour stocker le numéro de ligne.

Dans chaque ligne l'on retrouve les cellules non-vides de la forme :

```

<c r="A2">
  <v>1</v>
</c>
    
```

IV-B-2 - Contenu d'une Cellule

* **L'élément c a obligatoirement un attribut "r", qui prend pour valeur l'adresse de la cellule.**

* **Et de façon facultative un attribut "t" pour le type de donnée :**

- pas d'attribut "t" ; type par défaut (nombre)
- "s" : type chaîne de caractères (lié au xml sharedstring)
- "b" : type booléen (VRAI ou FAUX)

* **Chaque élément cellule à obligatoirement un 'sous-élément' : 'v' , qui reprend :**

- pour un type de donnée numérique, la valeur contenue dans la cellule
- pour un type de donnée chaîne (s), l'index de la chaîne dans sharedstring.xml
- pour un type de donnée booléen (b), 1 pour VRAI où 0 pour FAUX.

* **si la cellule contient une formule, un élément "f" contient cette formule :**

```

<c r="A4">
  <f>2+3</f>
  <v>5</v>
</c>
    
```

IV-B-3 - Ouverture de la feuille de calcul

La fonction **LireFeuille** permet de récupérer un DomDocument : shXML, initialisé par l'XML de la feuille sélectionnée. Il faut récupérer dans workbook.xml l'identifiant **r:id** de la feuille sélectionnée, sachant qu'à l'ouverture du classeur la variable de type IXMLDOMNodeList: xmlListeFeuilles a été initialisée avec les "noeuds enfants" de l'élément <sheets> du classeur, on récupère l'attribut r:id correspondant à l'index de la feuille sélectionnée dans cette variable :

```

Dim oNode As IXMLDOMElement
Set oNode = xmlListeFeuilles(iNum)
shRelID = oNode.getAttribute("r:id")
    
```

La variable chaîne **shRelID** est ensuite utilisée comme paramètre lors de l'appel à la fonction **sh_Open** de la classe **oxWorkbook**

```
LireFeuille = shXml.loadXML(wk.Sh_Open(shRelID)) 'Lecture XML
```

La variable objet **shXML** est ensuite disponible pour les fonctions de lecture/écriture dans les cellules.

La fonction complète utilisée à l'ouverture de la feuille:

```
'
' Ouverture d'une feuille en lecture
'
Function LireFeuille(iNum As Integer) As Boolean
    Dim oNode As IXMLDOMEElement
    Set oNode = xmlListeFeuilles(iNum)
    Set shXml = New DOMDocument
    shModif = False
    shRelID = oNode.getAttribute("r:id")
    LireFeuille = shXml.loadXML(wk.Sh_Open(shRelID)) 'Lecture XML
    Set oNode = Nothing
End Function
```

IV-B-4 - Lecture du contenu d'une cellule

La lecture consiste à "parser" le DomDocument **shXml** préalablement ouvert par la fonction **LireFeuille**. Grâce à une requête XPath l'on extrait le noeud "c" concernant la cellule à lire, c'est à dire dont l'attribut "r" a pour valeur l'adresse de la cellule en lecture :

```
'
' Test Existence Cellule, la renvoie si existe
'
Function ExisteCell(stAdd As String) As IXMLDOMEElement
    On Error Resume Next ' Rennoi Nothing si Inexistant.
    Set ExisteCell = shXml.selectSingleNode("/worksheet/sheetData/row/c[@r='" & stAdd & "']")
End Function
```

La fonction **ExisteCell** renvoie un **IXMLDOMEElement**, pointant sur ce noeud ou Nothing si l'adresse n'existe pas. La suite des opérations consiste à récupérer les attributs "v", "t" et "f" du noeud renvoyé par la fonction **ExisteCell**.

```
'
' Lecture d'une cellule
'
Function LectCell(stAdd As String, MaCell As tyCell) As Boolean
    Dim oCell As MSXML2.IXMLDOMEElement
    Set oCell = ExisteCell(stAdd)
    If Not oCell Is Nothing Then
        On Error Resume Next
        MaCell.v = oCell.selectSingleNode("v").Text 'Val
        MaCell.t = oCell.getAttribute("t") 'Type
        MaCell.f = oCell.selectSingleNode("f").Text 'Formule
        LectCell = True
    End If
End Function
```

Pour les cellules contenant un texte, avec un attribut t="s", la valeur chaîne de caractère est alors déterminée dans le DomDocument pointant sur sharedstring.xml en fonction de l'attribut "v" du noeud :

Appel de la fonction d'extraction chaîne partagée

```
If MaCell.t = "s" Then
    lbVal.Caption = GetShared(CInt(MaCell.v))
```

Fonction d'extraction chaine partagée

```
'  
' Récupère liste chaines partagées..  
'  
Function GetShared(iNum As Integer) As String  
  Dim oSelect As IXMLDOMElement  
  On Error Resume Next  
  Set oSelect = SharedStringXml.selectSingleNode("/sst/si[" & iNum & "]")  
  If Not oSelect Is Nothing Then  
    GetShared = oSelect.Text  
  Else  
    GetShared = "#ERR#"  
  End If  
  Set oSelect = Nothing  
End Function
```

IV-B-5 - Ecriture dans une cellule

L'écriture dans une cellule est effectuée en agissant sur le DomDocument ShXml (qui a été initialisé avec la partie xml sheetX.xml correspondant à la feuille ouverte)

Formatage des données à écrire

Avant le passage en paramètre à la fonction d'écriture cellule, les données relatives à la cellule sont formatées. Dans le code fourni en exemple, une combobox : **cbType** permet à l'utilisateur de définir le type de donnée à écrire (cbType=0 : numérique, 1 : Chaîne, 2 : Booléen), la fonction **FormateDonnees** permet de préparer la valeur à écrire en fonction de ce choix :

Pour une valeur numérique (le . est utilisé comme séparateur décimal dans l'XML feuille , les virgules sont donc remplacées par des .)

```
(...)
  If FormateDonnees(cbType.ListIndex, txtVal, CellEcr, stMess) Then
    (...)
  '
  'Formatage des données à écrire
  '
  Function FormateDonnees(iType As Integer, stVal As String, ByRef CellEcr As tyCell, ByRef
    stMess As String) As Boolean
    Dim bOk As Boolean
    Select Case iType
      stVal = Replace(stVal, ",", ".")
      If IsNumeric(Replace(stVal, ".", ",")) Then
        bOk = True
        CellEcr.v = stVal
      Else
        bOk = False
        stMess = "Saisir une valeur 'Numérique'"
      End If
    End If
  End Function
```

Pour une valeur de type chaîne :

```
Case 1 'Chaîne
  If Trim(stVal) <> "" Then
    CellEcr.t = "s"
    CellEcr.v = ChercheChaîne(stVal)
    bOk = True
  Else
    bOk = False
    stMess = "Saisir une chaîne"
  End If
```

une fonction cherchechaîne permet de retrouver dans le "DomDocument" xmlSharedString, la chaîne dont l'index est passé en paramètre :

```
'
' Cherche chaîne partagée et la rajoute si n'existe pas .
'
Function ChercheChaîne(stVal As String) As Integer
Dim oSelect As IXMLDOMElement
Dim osst As IXMLDOMElement
Dim osi As IXMLDOMElement
Dim iTrouve As Long
Dim iIndex As Long
Dim lCount As Long
Dim lUniqueCount As Long
iIndex = 0
iTrouve = -1
```

```

Set osst = SharedStringXml.selectSingleNode("/sst")
If Not osst Is Nothing Then
    On Error Resume Next
    lCount = osst.getAttribute("count")
    lUniqueCount = osst.getAttribute("uniqueCount")
    On Error GoTo 0
End If

For Each oSelect In SharedStringXml.selectNodes("/sst/si/t")
    If oSelect.Text = stVal Then
        iTrouve = iIndex 'Memorise index trouvé
        Exit For
    End If
    iIndex = iIndex + 1
Next

If iTrouve < 0 Then
    'incrémente unique count et count
    iTrouve = iIndex
    'Ajoute nouvelle chaîne

    Set osi = shXml.createElement(MSXML2.NODE_ELEMENT, "si", worksheetSchema)
    osi.appendChild shXml.createElement(MSXML2.NODE_ELEMENT, "t", worksheetSchema)
    osi.selectSingleNode("t").Text = stVal
    osst.appendChild osi
    osst.setAttribute "uniqueCount", lUniqueCount + 1

End If

ChercheChaîne = iTrouve
osst.setAttribute "count", lCount + 1
'sauve
bSSModif = True
End Function
    
```

Si la chaîne n'existe pas, elle est ajoutée au DomDocument SharedStringXML, et la valeur de l'attribut count du noeud "sst" est incrémenté d'un. Dans tous les cas l'attribut "uniqueCount" du noeud "sst" est aussi incrémenté.

Si le type sélectionné est 2 (booléen) la valeur à écrire devient 0 pour FAUX et 1 pour VRAI

```

Case 2 'Boolean
    If txtVal = "VRAI" Then txtVal = 1
    If txtVal = "FAUX" Then txtVal = 0
    If txtVal = 0 Or txtVal = 1 Then
        CellEcr.v = Cint(txtVal)
        CellEcr.t = "b"
        Ok = True
    Else
        Ok = False
        stMess = "Saisir une valeur Booléenne 0 ou 1 (FAUX ou VRAI)"
    End If
    
```

Extraction Numéro ligne et colonne

Comme on le verra plus loin, l'écriture d'une cellule réclame de connaître le numéro de ligne et de colonne de la dite cellule. Une procédure permet le calcul de ces deux valeurs en fonction de l'adresse de la cellule.

```

Type tyADD
    iR As Long ' Numéro de ligne
    iC As Long ' Numéro de colonne
End Type
'
' Transformation adresse de type A1 en ligne colonne ..
'
Sub CalcAdd(stAdd As String, Add As tyADD)
    Dim stC As String
    Dim stR As String
    
```

```

Dim i As Integer
Dim j As Long 'Multiplicateur
i = 1
While Not IsNumeric(Mid(stAdd, i, 1))
    stC = stC & Mid(stAdd, i, 1)
    i = i + 1
Wend
stR = Mid(stAdd, i)
Add.iR = CLng(stR)
Add.iC = 0
j = 1
For i = Len(stC) To 1 Step -1
    Add.iC = Add.iC + (CInt(Asc(Mid(stC, i, 1)) + 1) - Asc("A")) * j
    j = j * 26
Next
End Sub
    
```

Par exemple CalcADD "Z10", MonAdd
renvoi : MonAdd.IC= 26
et MonAdd.IR = 10

Cas où la cellule n'existe pas

Création du noeud cellule :

```
Set oCell = shXml.createElement(MSXML2.NODE_ELEMENT, "c", worksheetSchema)
```

Ensuite test de l'existence du noeud <row> encadrant cette cellule :

```
Set oRow = shXml.selectSingleNode("/worksheet/sheetData/row[@r='" & MonAdd.iR & "']")
```

Si la ligne existe :

Les cellules doivent être placées dans l'ordre croissant des numéros de colonnes pour cela le code suivant :

```

If Not (oRow Is Nothing) Then 'Cas ou la ligne existe
    'L'ordre des cellules doit-être respecté, on cherche donc cellule suivante pour se placer
    'avant
    For Each oSuiv In oRow.childNodes
        CalcAdd oSuiv.getAttribute("r"), AddSuiv
        If AddSuiv.iC > MonAdd.iC Then
            bSuivant = True
            Exit For
        End If
    Next
    (...)
    
```

Parcourt les noeuds des cellules contenues dans la ligne (row) courante et stoppe à la première cellule dont le numéro de colonne est supérieur au numéro de colonne à écrire, si cette cellule est trouvée (**bsuivant** à 1), ajout du nouveau noeud cellule avant la cellule trouvée,
et dans le cas où aucune cellule n'a été trouvée la nouvelle cellule est ajoutée "en fin" de ligne:

```

(...)
If bSuivant Then
    oRow.insertBefore oCell, oSuiv
Else
    oRow.appendChild oCell
End If
(...)
    
```

Si la ligne n'existe pas :

Le noeud ligne doit alors être créé et placé dans l'ordre croissant des lignes :

```
(...)  
Else ' Cas o ligne n'existe pas  
Set oRow = shXml.createElement(MSXML2.NODE_ELEMENT, "row", worksheetSchema)  
oRow.setAttribute "r", MonAdd.iR  
bSuivant = False  
For Each oSuiv In shXml.selectNodes("/worksheet/sheetData/row")  
    If CLng(oSuiv.getAttribute("r")) > MonAdd.iR Then  
        bSuivant = True  
        Exit For  
    End If  
Next  
If bSuivant Then  
    shXml.selectSingleNode("/worksheet/sheetData").insertBefore oRow, oSuiv  
Else  
    shXml.selectSingleNode("/worksheet/sheetData").appendChild oRow  
End If  
(...)
```

avant de placer le noeud cellule sur ce nouveau noeud "row"

```
(...)  
oRow.appendChild oCell  
End If '-- Fin test existence oRoww..  
End I  
(...)
```

Mise à jour des données cellules

Dans tous les cas la fonction **EcritCellule** se termine par une mise à jour des données associées à cette cellule

```
(...)  
'-- Mise à jour des donnée cellule  
oCell.setAttribute "r", stAdd  
If MaCell.t <> "" Then oCell.setAttribute "t", MaCell.t  
Set oVal = shXml.createElement(MSXML2.NODE_ELEMENT, "v", worksheetSchema)  
oVal.Text = MaCell.v  
oCell.appendChild oVal  
EcritCellule = True  
shModif = True  
(...)
```

IV-C - Fermer la feuille

Pour libérer la feuille on effectue un appel à **fermerFeuille** :

```
'  
' Fermer feuille  
'  
Sub FermerFeuille(Optional bSauve = False)  
If bSauve And shModif Then  
    If wk.sh_close(shXml.xml, shRelID) And bSSModif Then  
        ' Sauve chaines partagées.  
        wk.SharedStringXml = SharedStringXml.xml  
        bSSModif = False  
    End If  
End If  
Set shXml = Nothing  
shModif = False  
End Sub
```

Si le paramètre **bSauve** est à 1, la feuille est sauvée par un appel à la fonction **sh_close** de l'objet com **oxWorkbook** :

```
(...)  
wk.sh_close(shXml.xml, shRelID)  
(...)
```

IV-D - Fermer classeur

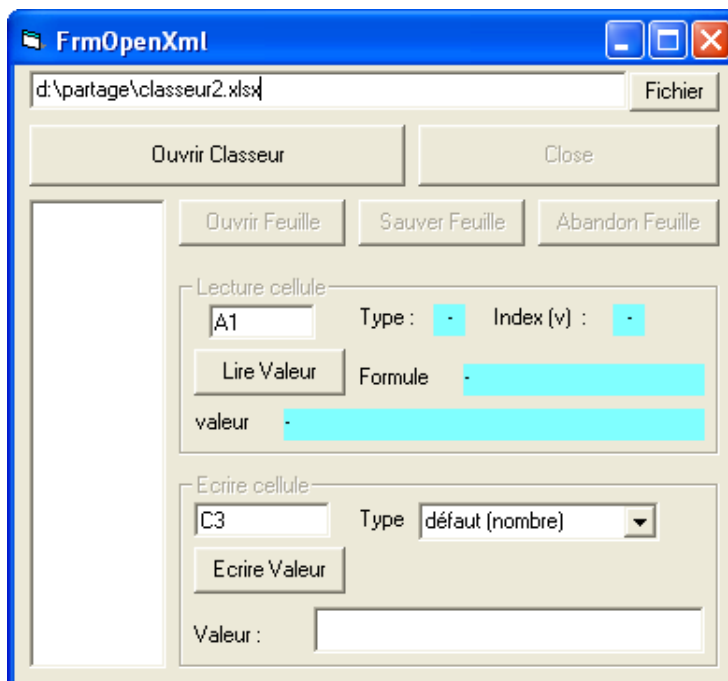
Pour finir, l'appel à la fonction **FermerClasseur** permet d'effectuer la mise à jour du package et la libération du classeur:

```
'  
' Ferme le classeur  
'  
Sub FermerClasseur()  
Set shXml = Nothing  
If Not wk Is Nothing Then  
    wk.Close  
    Set wk = Nothing  
End If  
Set SharedStringXml = Nothing  
End Sub
```

V - Exemple d'application : prjOpenXML

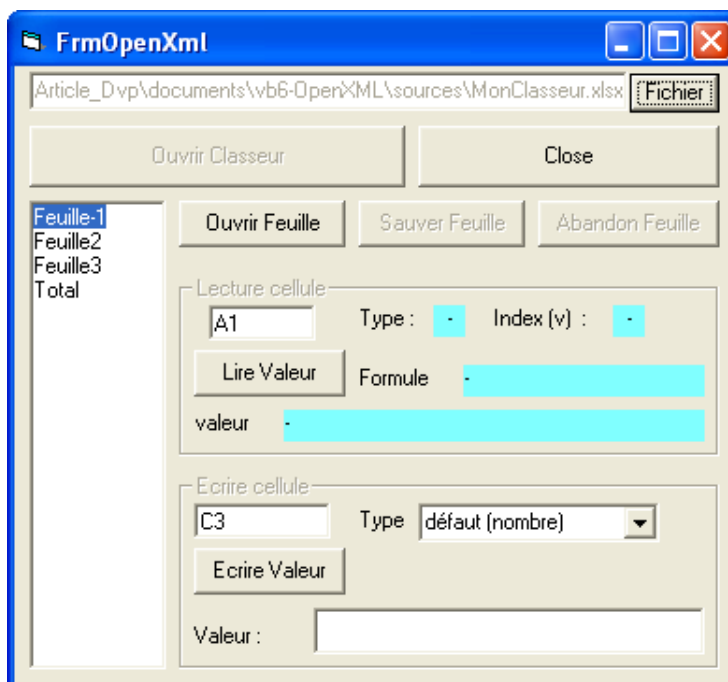
V-A - fenêtre principale

Pour illustrer cet article vous trouverez le projet VB6 prjOpenXML téléchargeable à la fin de ce chapitre. La fenêtre principale :

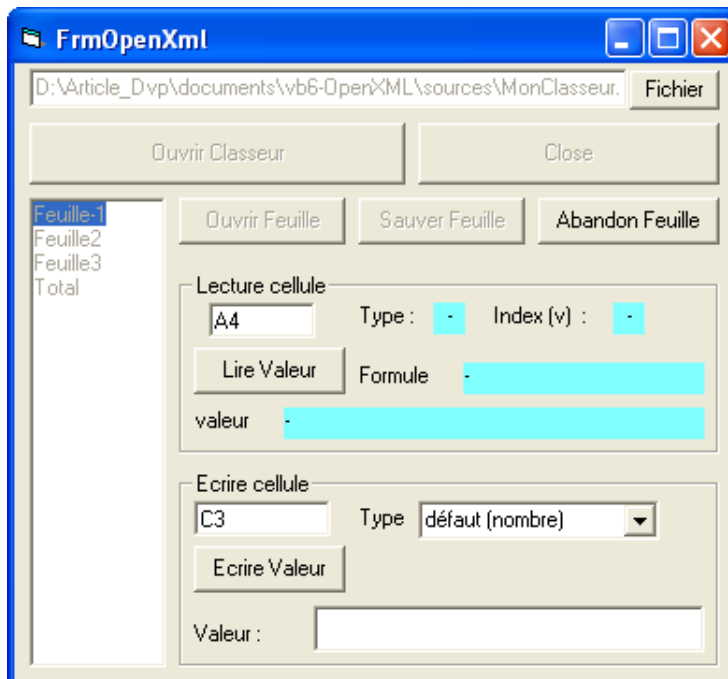


V-B - Fonctionnement

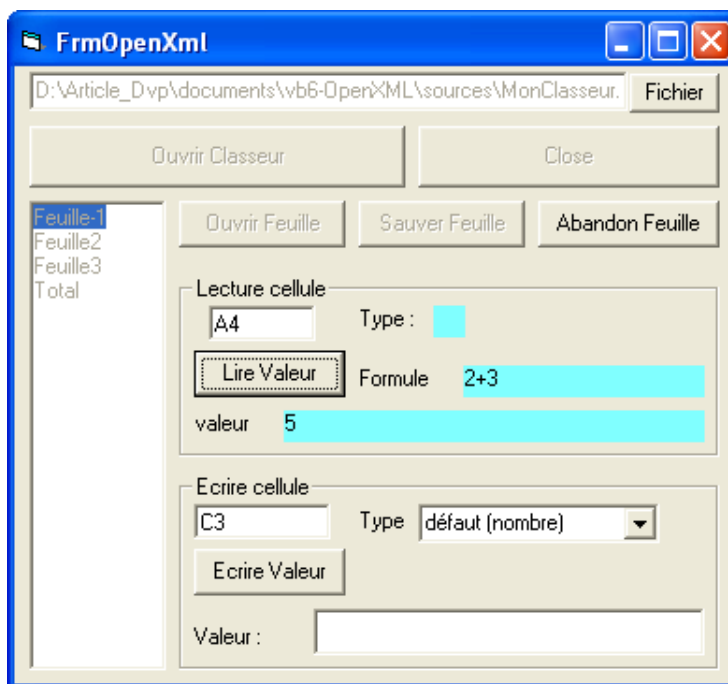
Utilisez le bouton Fichier pour sélectionner le classeur .xlsx, puis le bouton "Ouvrir le classeur"



Après l'ouverture du classeur il est possible de sélectionner une feuille dans la liste de gauche, et enfin d'actionner le Bouton Ouvrir, puis de saisir l'adresse de la cellule à lire par exemple A4 :

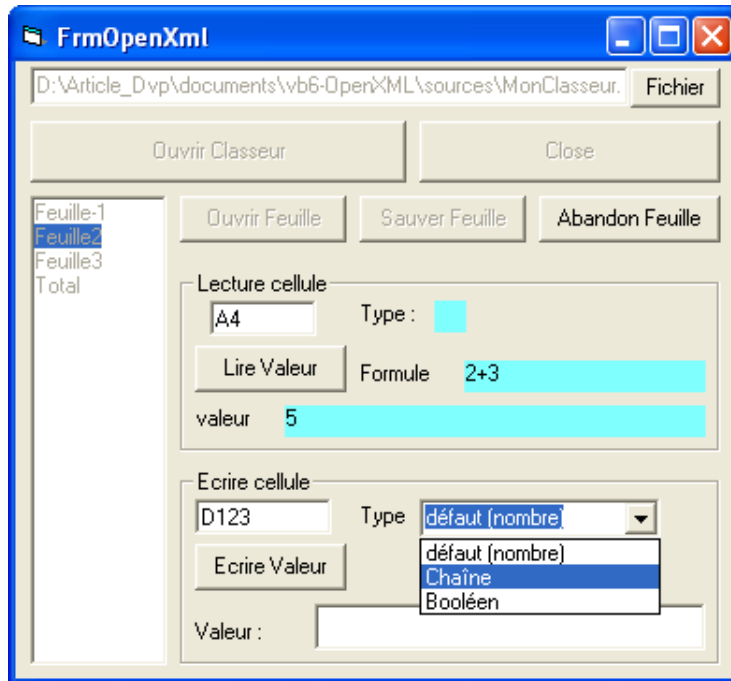


Actionnez ensuite le bouton "Lire Valeur" pour afficher les données de la cellule.

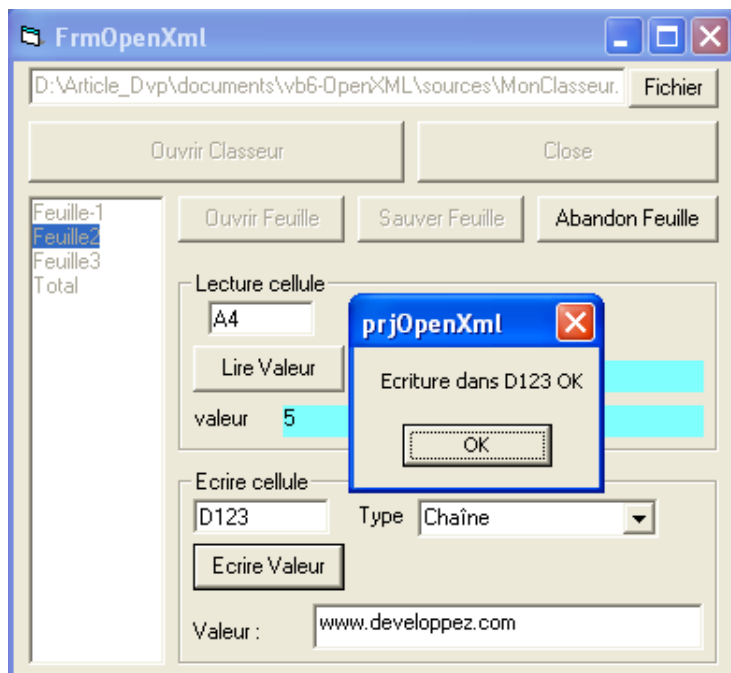


Les boutons "Abandon feuille" puis "Close" permettent ensuite de libérer le classeur

Pour l'écriture, après l'ouverture du classeur et de la feuille, saisir l'adresse dans "Ecrire cellule", dans l'exemple ci-dessous D123, sélectionner le type de donnée, puis saisir la valeur à écrire.



Validez ensuite l'écriture grâce au bouton "Ecrire Valeur"



Sauvez vos modifications grâce au bouton "Sauver Feuille" puis libérer le classeur grâce au bouton "Close"

V-C - Télécharger l'exemple d'application prjOpenXML



Les sources VB6 du projet prjOpenXML

Un classeur .xlsx créé avec Excel 2007 au format OpenXML pour vos tests :



MonClasseur.xlsx

VI - Liens utiles

OpenXML Schéma

-  [Le XML dans Microsoft Office \(OpenXML\)](#)
-  [Structure des fichiers OpenXML](#)
-  [La section OpenXML de Microsoft.com](#)
-  [Standard ECMA-376 Office Open XML File Formats](#)

Open SDK 2.0

-  [Introduction au SDK Open XML](#)
-  [MSDN : Open XML Format SDK 2.0](#)

Interaction .Net Framework et VB6

-  [Appel du .NET Framework à partir d'applications Visual Basic 6.0 existantes](#)

MSXML

-  [Visual Basic 6.0 et le format XML](#)
-  [Manipuler des fichiers XML en VBScript](#)

VII - Conclusions

VII-A - Remerciements

Tous mes remerciements à **Jeannot45** et **Pierre Fauconnier** pour leur relecture.

VII-B - Vos commentaires

J'ai ouvert une discussion, sur le forum :
j'espère y trouver vos commentaires, et un débat sur l'avenir du code de ce tutoriel....