

# Pilotage de l'application Office Excel (Excel Automation)

par Skalp ([Accueil](#))

Date de publication : 22 août 2007

Dernière mise à jour :

Lire et/ou écrire dans un fichier Office Excel est chose courante dans nos développements. Cependant, ce n'est pas évident pour le commun des développeurs. Cet article fournit une base très simple permettant de lire et d'écrire dans un fichier Excel.

- I - Introduction
- II - Référencer Excel dans son projet
- III - Lire un fichier Excel (ExcelReader)
  - III-A - Champs privés
  - III-B - Champ public
  - III-C - Constructeur
  - III-D - Méthodes publiques
  - III-E - Méthodes privées
- IV - Ecrire dans un fichier Excel (ExcelWriter)
  - IV-A - Champs privés
  - IV-B - Champ public
  - IV-C - Constructeur
  - IV-D - Méthodes publiques
  - IV-E - Méthodes privées
- V - Exemple d'utilisation
- VI - Vers l'infini... et au-delà !
  - VI-A - La MSDN
  - VI-B - Adapter des macros Excel
  - VI-C - Utiliser la documentation du modèle objet Excel
- VI - Conclusion
- VIII - Téléchargement
- IX - Références
- X - Liens utiles

## I - Introduction

Manipuler une application Office n'est pas chose évidente pour le commun des développeurs. Or, dans nos développements, il n'est pas rare que nous ayons besoin de **lire** ou **écrire** dans un **fichier Excel** qui est souvent utilisé comme fournisseur de données. Nous allons voir comment référencer Excel dans son projet de développement, puis nous allons piloter Excel à l'aide de deux classes effectuant les opérations de base de lecture (reader) et d'écriture (writer). Ces classes sont adaptées des classes StreamReader et StreamWriter fournies avec le framework.NET. Enfin, nous allons voir comment découvrir et piloter l'ensemble des fonctions Excel grâce à quelques astuces.

Il existe deux façons de contrôler une application Office. Ce sont la **liaison tardive** et la **liaison anticipée**. La différence est expliquée dans un article de l'Aide et Support Microsoft intitulé "Comment faire pour automatiser Microsoft Excel à l'aide de Visual Basic" ([Référence : Comment faire pour automatiser Microsoft Excel à l'aide de Visual Basic - cliquez ici](#)) : "Avec la liaison tardive, les méthodes ne sont pas liées jusqu'au temps d'exécution et le serveur Automation [(ici, l'application Office)] est déclaré comme Objet. Avec la liaison anticipée, votre application sait au moment de la conception le type exact d'objet avec lequel elle va communiquer, et peut déclarer ses objets comme un type spécifique. [...] La liaison anticipée [est] considérée comme meilleure dans la plupart des cas car elle permet de meilleures performances et une meilleure sécurité de type.". Il sera question dans cet article de liaison anticipée uniquement.

Le code a été rédigé avec Microsoft © Visual C# 2005 Express Edition.

## II - Référencer Excel dans son projet

Afin de piloter automatiquement les différentes fonctions d'Excel, il est nécessaire de **référencer les bibliothèques** Office Excel dans son projet. Pour cela, je vous renvoie à l'article de Jean-Marc "bidou" Rabilloud intitulé "Piloter Microsoft Office 2000 en .NET" ([Référence : Piloter Microsoft Office 2000 - cliquez ici](#)) :

"Dans VS.NET, je crée un nouveau projet. Dans le menu 'projet', je choisis 'Ajouter une référence' et dans la boîte de dialogue, je sélectionne l'onglet COM. Là, je choisis 'Microsoft Excel Object Library X.0' (X dépendant de la version ; 8 pour Excel 97 et 9 pour Excel 2000) et je valide. Dans les références de mon projet il existe alors deux références interop.excel et interop.office. Il y a eu création dans le répertoire de mon projet d'un excel.dll et d'un office.dll [...]."

Je rajoute que :

- Pour les versions plus récentes d'Excel, vous aurez des valeurs de X plus élevées (10 pour Excel 2002 et 11 pour Excel 2003).
- Selon la version, les références Office et les fichiers .dll peuvent avoir des noms légèrement différents de ceux suscités.

N'oubliez pas de **référencer** les bibliothèques dans le fichier (avec un using en C#, ou un imports en VB).

Pour les classes lecteur et écrivain, j'utilise la bibliothèque Excel 2000 (version 9) extraite à partir d'un fichier d'installation (c'est-à-dire que je n'ai pas eu besoin d'installer Office Excel 2000) selon l'astuce proposée à plusieurs reprises par MadMarc52 sur le forum ([Référence : Using the Type Library - cliquez ici](#)).

## III - Lire un fichier Excel (ExcelReader)

La classe ExcelReader permet de lire un fichier Excel.

### III-A - Champs privés

```
/** EXCEL */
// Application :
private _Application m_Application;
// Identifiant du processus :
private int m_ProcessId;
// Classeur :
private _Workbook m_Workbook;
// Feuille :
private _Worksheet m_Worksheet;

/** FEUILLE */
// Nombre total de colonnes :
private int m_ColumnCount;
// Index courant de colonne :
private int m_ColumnIndex;
// Fin du flux :
private bool m_EndOfStream;
// Nombre total de lignes :
private int m_RowCount;
// Index courant de ligne :
private int m_RowIndex;
// Valeur de cellule ou de ligne :
private string m_Value;
```

Les premiers champs sont relatifs à l'application Excel, les autres font référence à la feuille active.

### III-B - Champ public

```
public bool EndOfStream
{
    get { return m_EndOfStream; }
}
```

Cette propriété indique si le lecteur est arrivé à la fin de la feuille, c'est-à-dire après la cellule de la dernière ligne (RowCount) et de la dernière colonne (ColumnCount).

### III-C - Constructeur

```
public ExcelReader()
{
    this.m_ProcessId = -1;
    this.m_EndOfStream = false;
    this.m_RowIndex = -1;
    this.m_RowCount = -1;
    this.m_ColumnIndex = -1;
    this.m_ColumnCount = -1;
}
```

Le constructeur se contente d'initialiser les champs privés à une valeur par défaut. Il ne lance pas l'application car elle prend des ressources non négligeables (entre 15 et 20 Mo de mémoire), sans parler de l'ouverture (lorsqu'une variable est déclarée, elle n'est pas forcément utilisée immédiatement).



```
{
    // Sélectionner la feuille active :
    this.m_Worksheet = (_Worksheet)this.m_Workbook.ActiveSheet;
}
else
{
    // Sélectionner la feuille donnée :
    this.m_Worksheet = this.GetWorksheet(WorksheetName);
}

// Mettre à jour le nombre de lignes et colonnes :
this.SetWorksheetAttributes();
}
```

Lorsque la feuille est donnée, le lecteur est placé à son niveau. Le nombre de colonnes et de lignes renseignées est dénombré et le lecteur est placé à la première ligne et la première cellule de la feuille. Les alertes déclenchées par Excel sont désactivées (`DisplayAlerts = false`).

## Peek

```
public int Peek()
{
    if (!this.m_EndOfStream)
    {
        if (this.m_ColumnIndex + 1 <= this.m_ColumnCount)
        {
            return ((this.m_ColumnCount * (this.m_RowIndex - 1)) + this.m_ColumnIndex);
        }
        else
        {
            if (this.m_RowIndex + 1 <= this.m_RowCount)
            {
                return ((this.m_ColumnCount * (this.m_RowIndex - 1)) + this.m_ColumnIndex);
            }
            else
            {
                this.m_EndOfStream = true;
                return -1;
            }
        }
    }
    else
    {
        return -1;
    }
}
```

Retourne la position de la prochaine cellule, sans avancer le lecteur. Retourne -1 si le lecteur est à la fin de la feuille.

## Read

```
public string Read()
{
    if (!this.m_EndOfStream)
    {
        this.m_Value = string.Empty;

        // Récupérer le contenu de la cellule :
```

```
        this.m_Value = ((Range)this.m_Worksheet.Cells[this.m_RowIndex, this.m_ColumnIndex]).Value2
        != null ? ((Range)this.m_Worksheet.Cells[this.m_RowIndex, this.m_ColumnIndex]).Value2.ToString() :
        string.Empty);

        // Avancer le lecteur :
        if (this.m_ColumnIndex == this.m_ColumnCount)
        {
            if (this.m_RowIndex < this.m_RowCount)
            {
                this.m_RowIndex++;
                this.m_ColumnIndex = 1;
            }
            else
            {
                this.m_EndOfStream = true;
            }
        }
        else
        {
            this.m_ColumnIndex++;
        }

        return this.m_Value;
    }
    else
    {
        throw new EndOfStreamException();
    }
}
```

Retourne le contenu de la cellule et avance le lecteur. Lève une exception si le lecteur est déjà à la fin de la feuille.

## ReadLine

```
public string ReadLine()
{
    if (!this.m_EndOfStream)
    {
        this.m_Value = string.Empty;

        // Récupérer toute la ligne :
        for (int i = 1; i <= this.m_ColumnCount; i++)
        {
            this.m_Value += ((Range)this.m_Worksheet.Cells[this.m_RowIndex, i]).Value2 != null ?
            ((Range)this.m_Worksheet.Cells[this.m_RowIndex, i]).Value2.ToString() + ";" : ";";
        }

        // Avancer le lecteur :
        if (this.m_RowIndex < this.m_RowCount)
        {
            this.m_RowIndex++;
            this.m_ColumnIndex = 1;
        }
        else
        {
            this.m_EndOfStream = true;
        }

        return this.m_Value.Substring(0, this.m_Value.Length - 1);
    }
    else
    {
        throw new EndOfStreamException();
    }
}
```

}

Retourne le contenu d'une ligne entière sous la forme d'une chaîne de caractères dont les valeurs des cellules sont séparés par des points-virgules. Lève une exception si le lecteur est déjà à la fin de la feuille.

## III-E - Méthodes privées

### GetColumncount

```
private int GetColumnCount(int FirstColumnIndex, int LastColumnIndex, int MiddleColumnIndex)
{
    if ((Range)this.m_Worksheet.Cells[1, MiddleColumnIndex] != null &&
        ((Range)this.m_Worksheet.Cells[1, MiddleColumnIndex]).Value2 != null)
    {
        if ((Range)this.m_Worksheet.Cells[1, MiddleColumnIndex + 1] != null &&
            ((Range)this.m_Worksheet.Cells[1, MiddleColumnIndex + 1]).Value2 == null)
        {
            // La ligne suivante n'est pas renseignée, le résultat est :
            return MiddleColumnIndex;
        }
        else
        {
            // La ligne suivante est renseignée, rechercher dans l'intervalle supérieur :
            return GetColumnCount(MiddleColumnIndex, LastColumnIndex,
                (int)Math.Ceiling((double)(MiddleColumnIndex + ((LastColumnIndex - MiddleColumnIndex) / 2))));
        }
    }
    else
    {
        // La ligne n'est pas renseignée, rechercher dans l'intervalle inférieur :
        return GetColumnCount(FirstColumnIndex, MiddleColumnIndex,
            (int)Math.Ceiling((double)(FirstColumnIndex + ((MiddleColumnIndex - FirstColumnIndex) / 2))));
    }
}
```

Retourne le nombre de colonnes renseignées sur la première ligne (recherche récursive et dichotomique, il n'y a pas plus rapide :).

### GetProcessId

```
private int GetProcessId(Process[] ExcelProcessesBefore, Process[] ExcelProcessesAfter)
{
    bool IsMyProcess = false;
    int Result = -1;

    // Si mon processus est le seul à être instancié
    // inutile de parcourir le tableau, il n'y a qu'une seule instance :
    if (ExcelProcessesBefore.Length == 0 && ExcelProcessesAfter.Length == 1)
    {
        Result = ExcelProcessesAfter[0].Id;
    }
    else
    {
        // Parcours des processus après instanciation de l'objet :
        foreach (Process ProcessAfter in ExcelProcessesAfter)
        {
            // Parcours des processus avant instanciation de l'objet :
            IsMyProcess = true;
            foreach (Process ProcessBefore in ExcelProcessesBefore)
            {
```

```
        // Si je le retrouve, ce n'est pas celui que je cherche :
        if (ProcessAfter.Id == ProcessBefore.Id)
        {
            IsMyProcess = false;
        }
    }

    // J'ai retrouvé mon processus :
    if (IsMyProcess)
    {
        Result = ProcessAfter.Id;
    }
}

return Result;
}
```

Retourne l'identifiant du processus présent dans le deuxième paramètre et absent du premier.

## GetRowCount

```
private int GetRowCount(int FirstRowIndex, int LastRowIndex, int MiddleRowIndex)
{
    if ((Range)this.m_Worksheet.Cells[MiddleRowIndex, 1] != null &&
        ((Range)this.m_Worksheet.Cells[MiddleRowIndex, 1]).Value2 != null)
    {
        if ((Range)this.m_Worksheet.Cells[MiddleRowIndex + 1, 1] != null &&
            ((Range)this.m_Worksheet.Cells[MiddleRowIndex + 1, 1]).Value2 == null)
        {
            // La ligne suivante n'est pas renseignée, le résultat est :
            return MiddleRowIndex;
        }
        else
        {
            // La ligne suivante est renseignée, rechercher dans l'intervalle supérieur :
            return GetRowCount(MiddleRowIndex, LastRowIndex,
                (int)Math.Ceiling((double)(MiddleRowIndex + ((LastRowIndex - MiddleRowIndex) / 2))));
        }
    }
    else
    {
        // La ligne n'est pas renseignée, rechercher dans l'intervalle inférieur :
        return GetRowCount(FirstRowIndex, MiddleRowIndex, (int)Math.Ceiling((double)(FirstRowIndex
            + ((MiddleRowIndex - FirstRowIndex) / 2))));
    }
}
```

De la même façon que pour les colonnes, retourne le nombre de lignes renseignées sur la première colonne (recherche récursive et dichotomique, il n'y a pas plus rapide :)).

## GetWorksheet

```
private _Worksheet GetWorksheet(string WorksheetName)
{
    foreach (_Worksheet Worksheet in this.m_Workbook.Worksheets)
    {
        if (Worksheet.Name == WorksheetName)
        {
            this.SetWorksheetAttributes();
            return Worksheet;
        }
    }
}
```

```
    }  
  }  
  return (_Worksheet)this.m_Workbook.ActiveSheet;  
}
```

Retourne l'instance de la feuille donnée en paramètre, sinon la feuille active par défaut.

### SetWorksheetAttributes

```
private void SetWorksheetAttributes()  
{  
    // Les attributs spécifiques à la feuille sont :  
    // - Le nombre de lignes :  
    this.m_RowCount = this.GetRowCount(1, this.m_Worksheet.Rows.Count,  
(int)Math.Ceiling((double)((this.m_Worksheet.Rows.Count - 1) / 2)));  
    this.m_RowIndex = 1;  
    // - Le nombre de colonnes :  
    this.m_ColumnCount = this.GetColumnCount(1, this.m_Worksheet.Columns.Count,  
(int)Math.Ceiling((double)((this.m_Worksheet.Columns.Count - 1) / 2)));  
    this.m_ColumnIndex = 1;  
}
```

Met à jour les nombres de lignes et de colonnes et la position du lecteur.

## IV - Ecrire dans un fichier Excel (ExcelWriter)

La classe ExcelWriter permet d'écrire dans un fichier Excel.

### IV-A - Champs privés

```
/** EXCEL */
// Application :
private _Application m_Application;
// Identifiant du processus :
private int m_ProcessId;
// Classeur :
private _Workbook m_Workbook;
// Feuille :
private _Worksheet m_Worksheet;

/** FEUILLE */
// Nombre total de colonnes :
private int m_ColumnCount;
// Index courant de colonne :
private int m_ColumnIndex;
// Fin du flux :
private bool m_EndOfStream;
// Nombre total de lignes :
private int m_RowCount;
// Index courant de ligne :
private int m_RowIndex;
```

Les premiers champs sont relatifs à l'application Excel, les autres font référence à la feuille active.

### IV-B - Champ public

```
public bool EndOfStream
{
    get { return m_EndOfStream; }
}
```

Cette propriété indique si le lecteur est arrivé à la fin de la feuille, c'est-à-dire après la cellule de la dernière ligne (RowCount) et de la dernière colonne (ColumnCount).

### IV-C - Constructeur

```
public ExcelWriter()
{
    this.m_ProcessId = -1;
    this.m_EndOfStream = false;
    this.m_RowIndex = -1;
    this.m_RowCount = -1;
    this.m_ColumnIndex = -1;
    this.m_ColumnCount = -1;
}
```

Le constructeur se contente d'initialiser les champs privés à une valeur par défaut. Il ne lance pas l'application car elle prend des ressources non négligeables (entre 15 et 20 Mo de mémoire), sans parler de l'ouverture (lorsqu'une variable est déclarée, elle n'est pas forcément utilisée immédiatement).

## IV-D - Méthodes publiques

### Close

```
public void Close()
{
    this.m_Worksheet = null;
    this.m_Workbook.Save();
    this.m_Workbook.Close(false, Type.Missing, Type.Missing);
    this.m_Workbook = null;
    this.m_Application.DisplayAlerts = true;
    this.m_Application.Quit();
    this.m_Application = null;
    Process.GetProcessById(this.m_ProcessId).Kill();
}
```

Ferme l'application de la même façon que pour ExcelReader à ceci près que les modifications sont sauvegardées avant la fermeture.

### NewLine

```
public void NewLine()
{
    if (!this.m_EndOfStream)
    {
        // Avancer le lecteur :
        if (this.m_RowIndex < this.m_RowCount)
        {
            this.m_RowIndex++;
            this.m_ColumnIndex = 1;
        }
        else
        {
            this.m_EndOfStream = true;
        }
    }
    else
    {
        throw new EndOfStreamException();
    }
}
```

L'écrivain change de ligne.

### Open

Surchargé, deux signatures. Les paramètres sont le chemin du fichier à ouvrir et le nom de la feuille à sélectionner.

Ouvre le processus Excel et le fichier donné.

```
public void Open(string FilePath)
{
    this.Open(FilePath, null);
}

public void Open(string FilePath, string WorksheetName)
```

```
{
    // Ouvrir Excel :
    Process[] ExcelProcessesBefore = Process.GetProcessesByName("EXCEL");
    this.m_Application = new ApplicationClass();
    Process[] ExcelProcessesAfter = Process.GetProcessesByName("EXCEL");
    this.m_ProcessId = this.GetProcessId(ExcelProcessesBefore, ExcelProcessesAfter);
    this.m_Application.DisplayAlerts = false;

    // Ouvrir le classeur :
    this.m_Workbook = this.m_Application.Workbooks.Add(Type.Missing);
    this.m_Workbook.SaveAs(FilePath, Type.Missing, Type.Missing, Type.Missing, Type.Missing,
        Type.Missing, XlSaveAsAccessMode.xlNoChange, Type.Missing,
        Type.Missing, Type.Missing, Type.Missing);

    // Sélectionner la feuille :
    if (string.IsNullOrEmpty(WorksheetName))
    {
        // Sélectionner la feuille active :
        this.m_Worksheet = (_Worksheet)this.m_Workbook.ActiveSheet;
    }
    else
    {
        // Sélectionner la feuille donnée :
        this.m_Worksheet = this.GetWorksheet(WorksheetName);
    }

    // Mettre à jour le nombre de lignes et colonnes :
    this.SetWorksheetAttributes();
}
```

Lorsque la feuille est donnée, l'écrivain est placé à son niveau. Le nombre de colonnes et de lignes renseignées est dénombré et l'écrivain est placé à la première ligne et la première cellule de la feuille.

## Write

```
public void Write(string Value)
{
    if (!this.m_EndOfStream)
    {
        // Renseigner le contenu de la cellule :
        ((Range)this.m_Worksheet.Cells[this.m_RowIndex, this.m_ColumnIndex]).Value2 = Value;

        // Avancer le lecteur :
        if (this.m_ColumnIndex == this.m_ColumnCount)
        {
            if (this.m_RowIndex < this.m_RowCount)
            {
                this.m_RowIndex++;
                this.m_ColumnIndex = 1;
            }
            else
            {
                this.m_EndOfStream = true;
            }
        }
        else
        {
            this.m_ColumnIndex++;
        }
    }
    else
    {
        throw new EndOfStreamException();
    }
}
```

```
}
```

Ecrit la valeur donnée dans la cellule courante et avance l'écrivain. L'écrivain change de ligne s'il arrive à la fin d'une ligne.

## WriteLine

```
public void WriteLine(string[] Values)
{
    if (!this.m_EndOfStream)
    {
        // Renseigner les cellules :
        foreach (string str in Values)
        {
            if (this.m_ColumnIndex <= this.m_ColumnCount)
            {
                ((Range)this.m_Worksheet.Cells[this.m_RowIndex, this.m_ColumnIndex]).Value2 = str;
                if (this.m_ColumnIndex < this.m_ColumnCount)
                {
                    this.m_ColumnIndex++;
                }
            }
        }

        // Avancer le lecteur :
        if (this.m_RowIndex < this.m_RowCount)
        {
            this.m_RowIndex++;
            this.m_ColumnIndex = 1;
        }
        else
        {
            this.m_EndOfStream = true;
        }
    }
    else
    {
        throw new EndOfStreamException();
    }
}
```

Ecrit un ensemble de valeurs donné sous forme de tableau dans des cellules, puis change de ligne à la fin.

## IV-E - Méthodes privées

Les méthodes privées sont les mêmes que pour ExcelReader, sauf pour GetColumnCount et GetRowCount qui n'y sont pas et pour SetWorksheetAttributes.

### GetProcessId

Voir la méthode privée d'ExcelReader ([Référence : GetProcessId - cliquez ici](#))

### GetWorksheet

Voir la méthode privée d'ExcelReader ([Référence : GetWorksheet - cliquez ici](#))

## SetWorksheetAttributes

```
private void SetWorksheetAttributes()
{
    // Les attributs spécifiques à la feuille sont :
    // - Le nombre de lignes :
    this.m_RowCount = this.m_Worksheet.Rows.Count;
    this.m_RowIndex = 1;
    // - Le nombre de colonnes :
    this.m_ColumnCount = this.m_Worksheet.Columns.Count;
    this.m_ColumnIndex = 1;
}
```

## V - Exemple d'utilisation

```
using System;
using System.Collections.Generic;
using System.Text;
using AutomationExcel;

namespace AutomationExcelTest
{
    class Program
    {
        static void Main(string[] args)
        {
            /** EXCELREADER ***/
            // Déclarer et ouvrir le lecteur :
            ExcelReader ER = new ExcelReader();
            ER.Open(@"C:\Classeur1.xls");

            // Afficher les valeurs :
            while (ER.Peek() >= 0)
            {
                Console.WriteLine(ER.ReadLine());
            }

            // Ne pas oublier de fermer le lecteur :
            ER.Close();

            /** EXCELWRITER ***/
            // Déclarer et ouvrir l'écrivain :
            ExcelWriter EW = new ExcelWriter();
            EW.Open(@"C:\Classeur2.xls");

            string[] mesValeurs = new string[6]{ "Piloteage", "de", "l'application", "Office",
            "Excel", "(Automation Excel)"};

            // Je peux parcourir ce tableau pour renseigner mon fichier Excel :
            foreach (string str in mesValeurs)
            {
                EW.Write(str);
                EW.NewLine();
            }

            // Je peux envoyer le tableau entier pour écrire une ligne :
            EW.WriteLine(mesValeurs);

            // Ne pas oublier de fermer l'écrivain, c'est là qu'il enregistre :
            EW.Close();


            Console.ReadLine();
        }
    }
}
```

Voilà un exemple d'utilisation simple du lecteur et de l'écrivain.

## VI - Vers l'infini... et au-delà !


Pour découvrir plus amplement ce qu'il est possible de faire avec la bibliothèque de pilotage Excel, voici quelques astuces auxquelles on ne pense pas forcément mais qui sont fort utiles :

### VI-A - La MSDN

 La msdn fournit une documentation assez complète sur l'automation d'Office en général. Ce sont les "Visual Studio Tools pour Office" ([Référence : Visual Studio Tools pour Office - cliquez ici](#)). A partir de là, vous avez accès à :

- une vue d'ensemble du modèle objet Excel
- la description de tâches couramment exécutées
- la description de procédures pas à pas utilisant Excel (pour des scénarios plus ou moins courants)

### VI-B - Adapter des macros Excel

 Il est très pratique d'enregistrer une macro Excel et ensuite accéder au code VBA pour voir les fonctions, champs ou valeurs auxquels fait appel Excel pour exécuter l'opération. Pour enregistrer une macro, allez dans Outils > Macros > Nouvelle macro. Effectuez les opérations via l'interface Excel. Lorsque vous avez terminé, allez dans Outils > Macros > Arrêter l'enregistrement. Pour visionner le code VBA associé, allez dans Outils > Macros > Macros (Alt + F8), sélectionnez la macro que vous venez d'enregistrer et cliquez sur Modifier. Exemple avec la modification des bordures d'une cellule (encadrée en rouge) :

```
Sub Macro1()  
'  
' Macro1 Macro  
' Macro enregistrée le 30/07/2007 par Skalp  
'  
    Selection.Borders(xlDiagonalDown).LineStyle = xlNone  
    Selection.Borders(xlDiagonalUp).LineStyle = xlNone  
    With Selection.Borders(xlEdgeLeft)  
        .LineStyle = xlContinuous  
        .Weight = xlThin  
        .ColorIndex = 3  
    End With  
    With Selection.Borders(xlEdgeTop)  
        .LineStyle = xlContinuous  
        .Weight = xlThin  
        .ColorIndex = 3  
    End With  
    With Selection.Borders(xlEdgeBottom)  
        .LineStyle = xlContinuous  
        .Weight = xlThin  
        .ColorIndex = 3  
    End With  
    With Selection.Borders(xlEdgeRight)  
        .LineStyle = xlContinuous  
        .Weight = xlThin  
        .ColorIndex = 3  
    End With  
End Sub
```

Même si l'on ne connaît pas le langage VBA, on comprend assez facilement ce qui est effectué. Ainsi, les bordures sont représentées par l'objet Borders, la couleur rouge correspond à la valeur 3, ...

Les variables préfixées d'un "xl" sont des constantes Excel. Vous pouvez accéder à une liste exhaustive en consultant la documentation du modèle objet Excel ([Référence : Utiliser la documentation du modèle objet Excel - cliquez ici](#)).

D'une façon plus générale, le langage VBA vous aidera également à découvrir le modèle objet Excel. Ainsi, la FAQ VBA pourra vous être utile ([Référence : FAQ VBA - cliquez ici](#)).

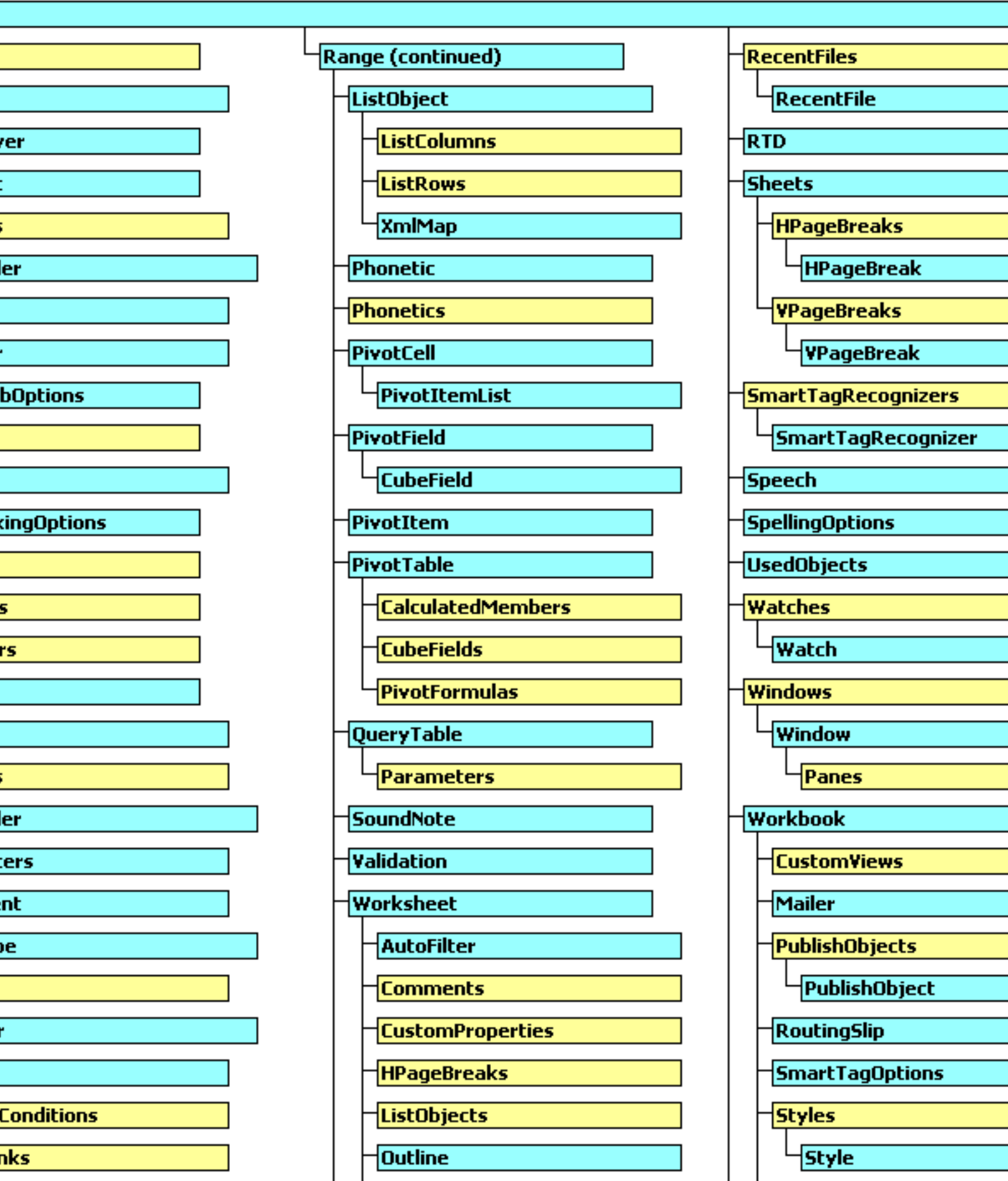
## VI-C - Utiliser la documentation du modèle objet Excel



*L'Aide et Support de Microsoft nous fournit un article nous permettant de trouver et d'utiliser la documentation du modèle objet Office depuis la version 97 jusqu'à la version 2003 ([Référence : How to find and use Office object model documentation - cliquez ici](#)). Cette documentation se présente sous la forme de fichiers d'aide windows (.CHM ou .HLP) disponibles dans le dossier d'installation Office.*

*Voici un exemple de modèle avec celui d'Excel 2003 :*

## et Microsoft Excel



Chaque cadre de couleur envoie vers une description commentée de l'objet. Exemple avec l'objet Borders :

## Collection

[Propriétés](#)    [Méthodes](#)    [Événements](#)



Les objets **Border** qui représentent les quatre bordures d'un objet **Range** ou **Style**.

### La collection Borders

La propriété **Borders** pour renvoyer la collection **Borders** contenant les quatre bordures. L'exemple suivant montre comment ajouter une bordure double à la cellule A1 de la feuille de calcul 1.

```
WorksheetFunction.Range("A1").Borders.LineStyle = xlDouble
```

Sur un seul objet **Border**, spécifiez **Borders(index)**, *index* identifiant la bordure. L'exemple suivant montre comment affecter la couleur rouge à la bordure inférieure des cellules A1:G1.

```
WorksheetFunction.Range("Sheet1").Range("A1:G1").Borders(xlEdgeBottom).Color = RGB(255, 0, 0)
```

Utilisez les constantes **XlBordersIndex** suivantes : **xlDiagonalDown**, **xlDiagonalUp**, **xlEdgeBottom**, **xlEdgeLeft**, **xlEdgeRight**, **xlEdgeTop**, **xlInsideHorizontal** ou **xlInsideVertical**.

Sur un objet **Range** et **Style**, vous pouvez définir les propriétés pour une seule bordure. Dans les autres objets à bordure (par exemple, les formes graphiques), la bordure est traitée comme une seule entité, quel que soit le nombre de côtés. Pour ces objets, les propriétés de bordure sont définies pour l'ensemble de la bordure. Pour plus d'informations, consultez la rubrique relative à l'objet **Border**.

## VI - Conclusion

Cet article n'est qu'une **base de développement** que j'ai voulue **simple** et **disponible** afin que vous puissiez l'**inclure simplement** dans vos projets et la **compléter** selon vos besoins et grâce aux astuces fournies.

Have fun !

## VIII - Téléchargement

**Télécharger les bibliothèques Automation Excel** à utiliser directement.

**Télécharger les sources des bibliothèques Automation Excel en C#**

**Télécharger la solution complète en C#** (les sources AutomationExcel et AutomationExcelTest)

## IX - Références

### Comment faire pour automatiser Microsoft Excel à l'aide de Visual Basic

Comment créer et manipuler Excel à l'aide de l'Automation à partir de Visual Basic.

 <http://support.microsoft.com/kb/219151/fr-fr>

### Piloter Microsoft Office 2000 en .NET

Comment communiquer avec les composants d'applications Microsoft Office comme Word ou Excel.

 <http://dotnet.developpez.com/cours/office/>

### Using the Type Library

Comment extraire les Interop Assemblies Excel depuis la bibliothèque de type Excel (Excel Type Library).

 <http://www.devcity.net/Articles/163/1/article.aspx>

### Visual Studio Tools pour Office

Documentation sur la programmabilité Office (Automation Office).

 [http://msdn2.microsoft.com/fr-fr/library/d2tx7z6d\(VS.80\).aspx](http://msdn2.microsoft.com/fr-fr/library/d2tx7z6d(VS.80).aspx)

Vue d'ensemble du modèle objet Excel.

 [http://msdn2.microsoft.com/fr-fr/library/wss56bz7\(VS.80\).aspx](http://msdn2.microsoft.com/fr-fr/library/wss56bz7(VS.80).aspx)

Tâches couramment exécutées.

 [http://msdn2.microsoft.com/fr-fr/library/syyd7czh\(VS.80\).aspx](http://msdn2.microsoft.com/fr-fr/library/syyd7czh(VS.80).aspx)

Procédures pas à pas utilisant Excel.

 [http://msdn2.microsoft.com/fr-fr/library/d7f63219\(VS.80\).aspx](http://msdn2.microsoft.com/fr-fr/library/d7f63219(VS.80).aspx)

## How to find and use Office object model documentation

Comment rechercher et utiliser la documentation de modèle d'objet Office.



<http://support.microsoft.com/kb/222101/en-us>



<http://support.microsoft.com/kb/222101/fr-fr> (traduction automatique)

## FAQ VBA

[FAQ VBA](#)

## X - Liens utiles

### Liens utiles developpez.com

Comment piloter Excel à partir de Visual Basic.

 <http://drq.developpez.com/vb/tutoriels/Excel/>

Visual Studio 2005 Tools for Microsoft Office: L'automation Office en .NET.

 <http://morpheus.developpez.com/vsto2/>

### Autres liens

Pour plus d'informations sur les liaisons tardive et anticipée :

Early and Late Binding.

 [http://msdn2.microsoft.com/en-us/library/0tcf61s1\(vs.80\).aspx](http://msdn2.microsoft.com/en-us/library/0tcf61s1(vs.80).aspx)

Using early binding and late binding in Automation.

 <http://support.microsoft.com/kb/245115>

API toutes prêtes de pilotage Excel :

Koogra project (gratuit). Attention : pas de documentation !

 <http://sourceforge.net/projects/koogra/>

Excel Reader (gratuit après inscription).

 <http://www.codeproject.com/office/ExcelReader.asp>

GemBox.Spreadsheet (version gratuite limitée, version payante)

 <http://www.gemboxsoftware.com/GBSpreadsheet.htm>

