



# ASP.Net et Excel - Nouvelle version

(Avec code source à télécharger)

*Mis à jour le 06/05/2005*  
Par Elise Dupont

## **Droit de diffusion:**

L'ensemble ou partie de ce document ainsi que le code mis à disposition, ne peut être diffusé sur d'autres sites Web sans l'autorisation au préalable de son créateur.

## **Avant Propos :**

Cet article est une refonte totale d'un ancien article, qui traitait de la génération de fichiers Excel depuis une application ASP.Net. Cet article n'était vraiment pas complet dans le sujet, c'est pourquoi j'ai décidé de traiter le sujet différemment. Vous pouvez cependant toujours consulter l'ancien article.

Le but de cet article est de vous présenter de façon un peu plus complète toutes les actions possibles autour d'ASP.net et Excel, que ce soit la génération de fichiers Excel ou la lecture de fichiers excel. Vous verrez que la solution que je présentais dans mon ancien article comporte beaucoup d'inconvénients, et quand dans certains cas il existe des solutions plus fiables.

## **Sommaire:**

- [1. Lire un fichier Excel avec ADO](#)
- [2. Générer et lire un fichier Excel en COM \(automation office\)](#)
- [3. Générer un fichier Excel en passant par XML](#)
- [4. Générer un fichier Excel en passant par CSV](#)
- [5. Tableau comparatif des solutions](#)
- [6. Graphiques](#)

## **1. Lire un fichier Excel avec ADO**

La première solution, est utilisée quand vous devez lire le contenu d'un fichier excel. Elle est très simple, et consiste à passer par OLEDB. Il faut construire une connexion qui utilise Microsoft.Jet.OLEDB.4.0 et ensuite effectuer une requête. Un exemple détaillé est disponible dans le code source à télécharger. Dans le répertoire ADO de l'application web, vous trouverez 2 exemples : pour lire

un fichier Excel, et pour lire un fichier CSV

Dans les deux cas, le code vous montre comment lire le contenu d'un fichier Excel ou CSV, et comment filtrer le contenu d'un fichier Excel ou CSV d'après une colonne.

Pour lire le contenu d'un fichier Excel, le code est simple :

```
public static DataSet GetExcelWorkSheet(string pathName, string fileName, int
workSheetNumber)
{
    OleDbConnection ExcelConnection = new
OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source="+pathName+"\ "+fileName+";Extended Properties=Excel 8.0;");
    OleDbCommand ExcelCommand = new OleDbCommand();
    ExcelCommand.Connection = ExcelConnection;
    OleDbDataAdapter ExcelAdapter = new OleDbDataAdapter(ExcelCommand);

    ExcelConnection.Open();
    DataTable ExcelSheets =
ExcelConnection.GetOleDbSchemaTable(System.Data.OleDb.OleDbSchemaGuid.Tables, new
object[] {null, null, null, "TABLE"});
    string SpreadSheetName =
"["+ExcelSheets.Rows[workSheetNumber]["TABLE_NAME"].ToString()+"]";

    DataSet ExcelDataSet = new DataSet();
    ExcelCommand.CommandText = @"SELECT * FROM "+SpreadSheetName;

    ExcelAdapter.Fill(ExcelDataSet);

    ExcelConnection.Close();
    return ExcelDataSet;
}
```

Pour lire un fichier CSV, c'est aussi simple. Cette solution n'est pas dépendante de la version du fichier Excel. Cela fonctionne normalement pour toutes les versions (sauf oublié de ma part).

## 2. Générer et lire un fichier Excel en COM (automation office)

Cette partie là n'a rien de vraiment nouveau. Pour plus de détail sur la solution technique, voir l'article que j'avais écrit il y a quelques temps ([ASP.Net : Générez des Rapport Excel depuis votre application Web](#)).

Petit rappel cependant sur la solution technique : elle permettait grâce à l'intérop de lire un fichier Excel, de le modifier, d'exécuter des macros (qui par exemple génèrent des graphiques), et de renvoyer au client l'Excel.

Mais cette fois-ci, je vais plutôt vous parler des inconvénients de ce choix technique, et des éventuelles solutions pour les contourner.

Le premier inconvénient, c'est la stabilité/sécurité : faire de l'intérop Excel sur un serveur web, c'est rendre potentiellement votre serveur web très instable. Pour plus d'informations, allez directement sur le site de Microsoft : <http://support.microsoft.com/kb/257757/fr>

D'ailleurs cette technique n'est pas supportée par Microsoft.

Le deuxième inconvénient, est un problème de processus. Si vous aviez testé la solution que j'ai proposé dans mon ancien article,

vous avez peut être remarqué un bel effet de bord : au bout de quelques temps si on regarde les liste des processus sur le serveur, on peut voir une quantité de processus Excel.

En fait, quand on exécute le code, le processus Excel n'est jamais vraiment terminé correctement.

Il existe une méthode (pas très propre) pour contourner le problème, je vais pour l'exposer ici :

Quand vous fermez votre document Excel, exécutez ce petit code :

```
oBook.Close(False, xlsPath, oRien)
xl.xlApp.Quit()
oExcelApp = Nothing
System.Diagnostics.Process.GetProcessById(xl.ProcId).Kill()
xl = Nothing
GC.Collect()
```

Cela tuera le processus Excel associé, afin de ne pas laisser le serveur web plein de processus Excel fantômes. Mais ce code nécessite des droits un peu plus élevés que le compte ASP.Net qui est utilisé en général sur un serveur. Cela peut donc vulnérabiliser votre serveur. Comme je l'explique dans un article précédant concernant la sécurité ([Sécurisez vos applications .Net : Partie 2](#)), donner plus de privilèges à un compte asp.net signifie donner plus de privilèges en cas d'attaque.

Pour les versions d'Excel, reportez vous à l'article en question, vous verrez que tout dépend de la version d'Excel que vous utilisez.

Le dernier inconvénient c'est que la configuration du serveur et réussir à faire fonctionner le code semble un peu long. J'ai eu pas mal de retour par email suite à mon 1er article sur le sujet. Il semblerait que réussir à faire fonctionner le code que je vous ai fourni demande parfois un peu de peine. Personnellement, je n'ai pas eu de soucis sans solution.

### 3. Générer un fichier Excel en passant par XML

Voilà une solution qui peut être utile quand vous avez besoin de générer un fichier Excel peu évolué, et de l'envoyer au client, sans lire coté serveur un fichier Excel. C'est une solution assez simple (pour peu que l'on soit familier avec le XML), et à laquelle on ne pense pas forcément.

Il s'agit d'utiliser le OfficeXML afin de générer un fichier XML qui sera compréhensible par Excel.

Vous pouvez faire l'expérience vous-même en ouvrant un fichier excel et en le sauvegardant au format XML. Voici un exemple du contenu d'un fichier au format OfficeXML :

```

<?xml version="1.0"?>
<?mso-application progid="Excel.Sheet"?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:x="urn:schemas-microsoft-com:office:excel"
  xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:html="http://www.w3.org/TR/REC-html40">
  <DocumentProperties xmlns="urn:schemas-microsoft-com:office:office">
    <LastAuthor>DOTNET-TECH</LastAuthor>
    <Created>2005-05-06T16:09:34Z</Created>
    <Version>11.6360</Version>
  </DocumentProperties>
  <ExcelWorkbook xmlns="urn:schemas-microsoft-com:office:excel">
    <ProtectStructure>False</ProtectStructure>
    <ProtectWindows>False</ProtectWindows>
  </ExcelWorkbook>
  <Styles>
    <Style ss:ID="Default" ss:Name="Normal">|
    </Style>
  </Styles>
  <Worksheet ss:Name="export">
    <Table ss:ExpandedColumnCount="7" ss:ExpandedRowCount="15" x:FullColumns="1"
      x:FullRows="1">
      <Row>
        <Cell><Data ss:Type="String">Nom</Data></Cell>
        <Cell><Data ss:Type="String">Ville</Data></Cell>
        <Cell><Data ss:Type="String">Code Produit</Data></Cell>
        <Cell><Data ss:Type="String">Produit</Data></Cell>
        <Cell><Data ss:Type="String">Quantité</Data></Cell>
        <Cell><Data ss:Type="String">Prix</Data></Cell>
        <Cell><Data ss:Type="String">Total</Data></Cell>
      </Row>
      <Row>
        <Cell><Data ss:Type="String">Durand</Data></Cell>
        <Cell><Data ss:Type="String">Bordeaux</Data></Cell>
        <Cell><Data ss:Type="String">PA-250</Data></Cell>
        <Cell><Data ss:Type="String">POWERBALL Ambre 250 HZ Pro</Data></Cell>
        <Cell><Data ss:Type="Number">1</Data></Cell>
        <Cell><Data ss:Type="Number">36.96</Data></Cell>
        <Cell><Data ss:Type="Number">36.96</Data></Cell>
      </Row>
      <Row>

```

Etant donné qu'Excel peut sauvegarder ou lire un fichier au format OfficeXML (qui est un XML spécifique à office), rien ne vous empêche de générer un fichier XML qui suit cette syntaxe, et ensuite de générer un flux de type Excel vers le client afin qu'Excel prenne le relais au moment de l'ouverture du fichier sur le client.

Mieux encore, au moment d'envoyer le fichier XML au client, vous pouvez le renommer en .XLS afin que l'ouverture dans Excel soit automatique. Il faudra tout de même vous familiariser avec la syntaxe du OfficeXML. Si c'est une contrainte pour votre projet, vous pouvez toujours vous diriger vers la dernière solution proposée dans cet article : générer un fichier CSV. Vous pouvez toujours déporter la syntaxe OfficeXML dans un fichier XSL qui s'occupera de la transformation, et n'avoir à générer qu'un fichier XML de votre propre format.

Sachez cependant qu'en « SpreadsheetML » (le langage XML pour Excel), vous avez la possibilité de définir des formules par exemple, que vous ne pourrez pas faire en CSV.

Pour plus d'informations sur le OfficeXML, je vous conseille ce livre chez O'Reilly : <http://www.oreilly.com/catalog/officexml/> Le but n'est pas de vous donner un cours sur le SpreadsheetML, mais de vous fournir des éléments de décision quand à la technologie

que vous retiendrez.

## 4. Générer un fichier Excel en passant par CSV

Le principe est très proche de la solution précédente : générer du texte dans un format compréhensible par excel. Ici il s'agit du CSV, un format classique et passe partout, bien interprété par toute version d'Excel.

Dans le code source à télécharger, je vous expose un petit exemple. Il suffit de séparer le texte par des virgules.

Vous pourrez trouver un article plus complet sur le sujet, qui fournit du code source afin d'exporter vos données au format CSV ici : <http://www.codeproject.com/aspnet/ExportClassLibrary.asp> (en anglais)

Cette solution est très simple, et permet de générer un fichier pour Excel très légèrement. Il n'y a aucune contrainte spécifique. Cependant un fichier CSV est plus limité qu'un fichier Excel (pas de possibilité de faire un graphique par exemple).

## 5. Tableau comparatif des solutions

Technique	Fiabilité	Lecture	Ecriture / Création	Définition de formules	Appels de macros	Simplicité
<b>Automation</b>	-	OUI	OUI	OUI	OUI	*** (surtout lors de la configuration)
<b>ADO</b>	Ok	OUI	NON	NON	NON	*
<b>SpreadsheetML</b>	Ok	-	OUI	OUI	NON	**
<b>CSV</b>	Ok	-	OUI	NON	NON	*

## 6. Graphiques

Vous verrez dans les solutions proposées que la manière la plus simple d'obtenir des graphiques directement dans Excel est d'utiliser la solution de l'automation. Cependant, si l'automation ne vous convient pas pour des raisons de sécurité et/ou fiabilité, il existe d'autres moyens d'obtenir un graphique sympathique. Ce n'est pas le sujet de cet article, mais je vous incite à jeter un coup d'œil du côté de SVG.

■ Article en anglais : <http://www.123aspx.com/redirect.aspx?res=31844>

■ Article en français mais qui expose uniquement des exemples pour l'ASP classique ou le PHP : [http://www.asp-php.net/scripts/asp-php/generer\\_svg.php](http://www.asp-php.net/scripts/asp-php/generer_svg.php)

■ Le portail francophone du SVG : <http://www.svgfr.org/>



L'ensemble ou partie de ce document ainsi que le code mis à disposition, ne peut être diffusé ailleurs sans autorisation préalable

[elise.dupont@europe.com](mailto:elise.dupont@europe.com) - [www.dotnet-tech.com](http://www.dotnet-tech.com) - 2003-2005