Lire et modifier un document Word OpenXML en C#

Par Florian Casabianca

Date de publication : 18 juin 2007

Dernière mise à jour : 18 juin 2007

Avec l'arrivée de la nouvelle version de Microsoft Office 2007, Microsoft introduit le nouveau format de document Office Open XML pour Word, Excel et PowerPoint et qui succèdent aux formats de fichier binaires d'Office (.doc, .xls et .ppt) apparus avec la sortie d'Office 97. Cet article présente les bases pour la lecture et la modification d'un fichier Word OpenXML en .Net.

Introduction

Avec l'arrivée de la nouvelle version de Microsoft Office 2007, Microsoft introduit le nouveau format de document *Office Open XML* pour Word, Excel et PowerPoint et qui succèdent aux formats de fichier binaires d'Office (.doc, .xls et .ppt) apparus avec la sortie d'Office 97.

Grâce à ce nouveau format **standardisé** à l'ECMA, les fichiers Word deviennent de simples packages zip contenant des fichiers XML. Ainsi, il n'est plus nécessaire de posséder Microsoft Word pour créer ou visualiser des fichiers, un simple éditeur de texte ou une application « maison » suffit.

Le but de cet article est de vous montrer comment lire un document Word au format Open XML. Nous verrons plus spécifiquement, au travers de plusieurs exemples, le code nécessaire afin de charger un document OpenXML, d'en récupérer les propriétés (auteur, date de création, etc.), d'y rechercher l'existence d'un mot ou encore d'extraire les images qu'il contient.

I - Rappel sur la structure d'un document WordProcessingML

WordprocessingML est un ensemble de conventions pour représenter un document Word au format Open XML. Pour les documents Excel il existe SpreadsheetML et pour les documents PowerPoint il s'agit de PresentationML.

Cet article n'a pas pour but de vous présenter l'architecture et la structure d'un document au format Open XML. Vous devez en avoir pris connaissance avant de lire cet article. Si tel n'est pas le cas, je vous conseille d'aller visiter ces quelques liens : les **livres blancs** et **Structure d'un document Open XML**. Cependant, nous allons tout de même faire un petit rappel sur la structure d'un document (ou package) Open XML de type WordprocessingML que vous pouvez visualiser notamment en ajoutant l'extension .zip à un fichier *.docx* et en l'ouvrant avec votre lecteur de fichiers zip.

Les trois principaux composants du nouveau format sont :

- Les ***parts*** : chaque fichier contenu dans l'arborescence est une Part. La plupart sont des fichiers XML mais il peut aussi y avoir des fichiers binaires (images, vidéos, objets OLE etc.) ou même d'autres fichiers Open XML multimédia si le document Word en contient.
- Les éléments ***type de contenu*** : ce sont des métadonnées contenues dans le fichier ***[Content_Types].xml*** et permettant de décrire le type de contenu stocké dans une *Part* (fichier jpeg, fichier de styles, fichier de relations, etc.). On peut ainsi savoir quelle méthode de lecture employer pour lire une *Part*.
- Les éléments ***relation*** : ils permettent de définir les associations entre une *Part* source et une *partie* cible. Les relations spécifient comment les *parts* se mêlent pour former un document. Les relations sont définies dans les fichiers **.rels**.

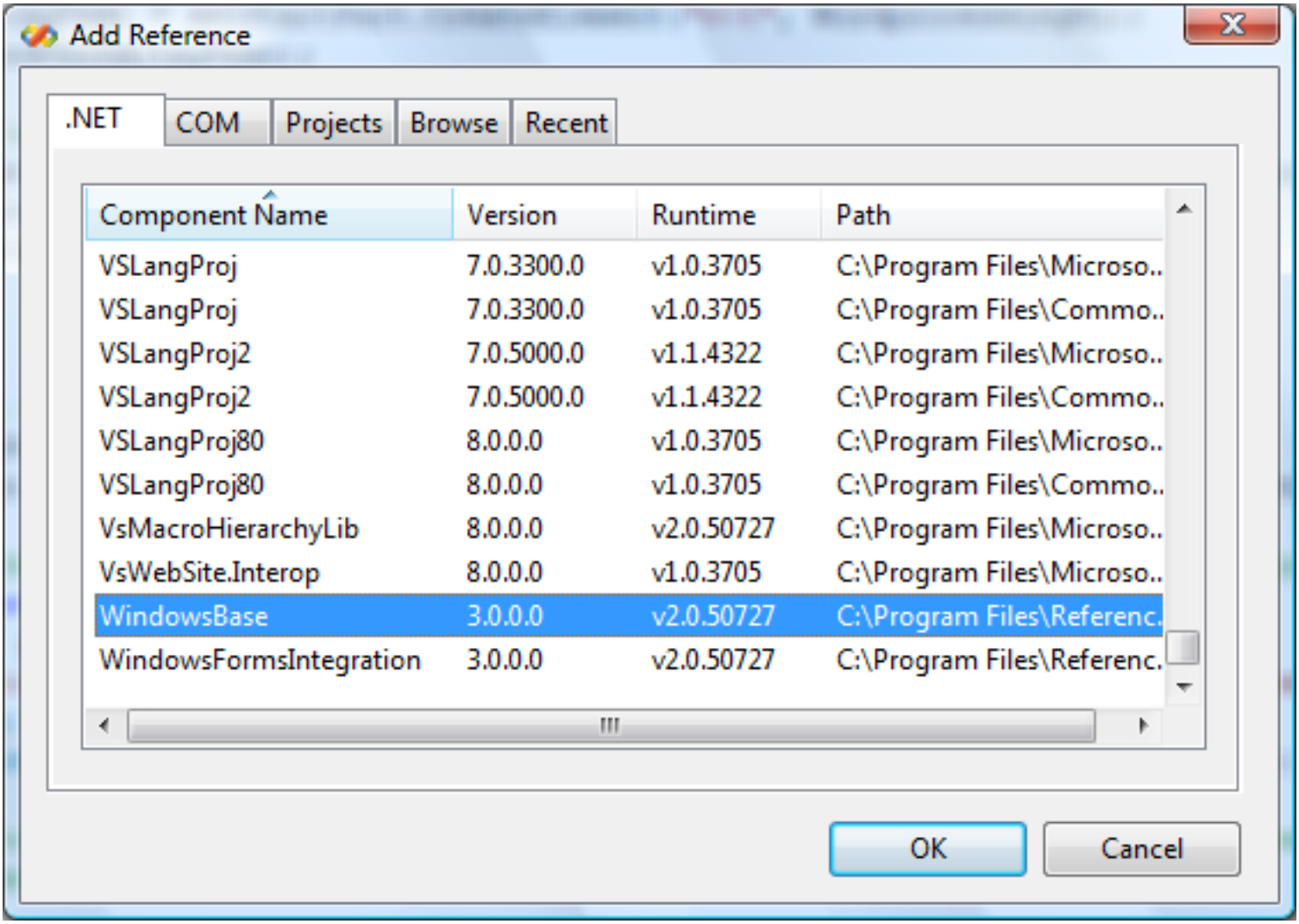
Le dossier ***docProps*** contient les fichiers de propriétés du document.
Le fichier ***document.xml*** est la *Part* principale d'un document WordprocessingML et contient le texte du corps du document.

II - Pré requis nécessaires

Pour nous faciliter la tâche, le Framework 3.0 de Microsoft .NET inclut la nouvelle API de packaging fournie dans l'assembly WindowsBase.dll. Les classes qui constituent l'API packaging sont contenues dans l'espace de noms System.IO.Package.

Vous aurez donc besoin du **Framework 3.0** et de son **SDK** (non obligatoire).

Pour ajouter une référence à votre projet sous Visual Studio : dans le menu *projet*, cliquez sur *ajouter une référence*. Si la dll *WindowsBase* ne se trouve pas dans l'onglet *.Net*, choisissez l'onglet *Parcourir* et aller la chercher dans *\Program Files\Reference Assemblies\Microsoft\Framework\v3.0*.



Vous devrez aussi référencer les espaces de noms *System.IO* et *System.Xml* dans votre projet :

```
using System.IO;
using System.Xml;
using System.IO.Packaging;
```

III - Ouverture d'un document Word OpenXML

La première chose à faire pour pouvoir lire un document OpenXML, est de le charger dans un objet de type ***Package***. Cela se fait en une ligne grâce à la fonction ***Open*** de la classe ***Package***. N'oubliez pas à la fin de vos traitements, d'appeler la méthode ***Close*** pour fermer le package. Voici le code permettant d'ouvrir en lecture/écriture un document OpenXML:

```
docWord = @"C:\monFichier.docx";

//overture du package en lecture/écriture
Package officePackage = Package.Open(docWord, FileMode.Open, FileAccess.ReadWrite);

//mettre ici le code des traitements

//fermeture du package
officePackage.Close();
```

Maintenant que nous disposons de notre objet ***Package***, nous allons voir comment récupérer les différentes parties composant le document.

IV - Lecture des propriétés du document

IV-A - Présentation

Les fichiers contenant les propriétés du document sont stockés dans le répertoire *docProps* situé à la racine du package.

Le fichier *core.xml* contient un ensemble de propriétés communes à tous les fichiers Open XML. Ces propriétés incluent le nom du créateur, la date de création, le titre, et la description. Ainsi, que vous traitiez un document docx, xlsx ou encore pptx, ces propriétés seront toujours placées à cet endroit.

Le fichier *app.xml* contient des propriétés spécifiques pour chaque type de package Open XML. Par exemple, pour un package *WordprocessingML*(docx), ces propriétés incluent le nombre de caractères, de mots, de lignes, de paragraphes, et de pages dans le document. Pour un package de type *Spreadsheet* (xlsx), ces propriétés incluent les titres des feuilles. Pour un package de type *Presentation* (pptx), ces propriétés incluent le format de présentation, le nombre de diapositives, le nombre de notes.

Nous allons uniquement nous intéresser dans cette partie au fichier *core.xml*.

Voici un exemple de ce que pourrait être le contenu de ce fichier :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cp:coreProperties
  xmlns:cp="http://schemas.openxmlformats.org/package/2006/metadata/core-properties"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:dcmitype="http://purl.org/dc/dcmitype/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dc:title>OpenXML et dotnet</dc:title>
  <dc:subject>Lire un fichier Word 2007</dc:subject>
  <dc:creator>Florian</dc:creator>
  <dc:description>commentaires</dc:description>
  <dcterms:created xsi:type="dcterms:W3CDTF">2007-05-20T10:16:00Z</dcterms:created>
  <dcterms:modified xsi:type="dcterms:W3CDTF">2007-05-20T10:16:00Z</dcterms:modified>
  <cp:category>Article dotnet</cp:category>
  <cp:contentStatus>En cours</cp:contentStatus>
  <cp:keywords>dotnet OpenXML</cp:keywords>
  <cp:revision>2</cp:revision>
</cp:coreProperties>
```

Notez l'utilisation de différents namespaces.

IV-B - Récupérer la partie core.xml

Nous allons tout d'abord récupérer la partie correspondant au fichier *core.xml* sous forme d'un objet ***PackagePart***. L'objet ***Package*** (que nous avons construit précédemment lors du chargement du document OpenXML) possède une fonction ***GetRelationshipsByType*** qui prend en paramètre un type de contenu (nous donnerons le type correspondant à la partie *core.xml*) et renvoie une liste d'objets de type ***PackageRelationship***. Ces objets représentent une association entre une source et une cible (dans notre cas, une partie de type *core-properties*). Étant donné qu'il n'y a qu'une seule partie de type *core-properties* dans un package OpenXML, la fonction ***GetRelationshipsByType*** renverra une liste ne contenant au plus qu'un seul élément. Soyez conscient que cette liste peut être vide car les fichiers de propriétés ne sont pas obligatoires dans un package OpenXML. L'objet ***PackageRelationship*** contient une propriété ***TargetUri*** qui permet de récupérer l'*Uri* relative de la cible (ici l'*Uri* de la partie *core.xml*). A partir de cette *Uri* il est ensuite facile de récupérer la partie correspondante grâce à la méthode ***GetPart*** de l'objet ***Package***.

Voici l'illustration de ce principe par le code:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cp:coreProperties
  xmlns:cp="http://schemas.openxmlformats.org/package/2006/metadata/core-properties"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:dcmitype="http://purl.org/dc/dcmitype/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dc:title>OpenXML et dotnet</dc:title>
  <dc:subject>Lire un fichier Word 2007</dc:subject>
  <dc:creator>Florian</dc:creator>
  <dc:description>commentaires</dc:description>
  <dcterms:created xsi:type="dcterms:W3CDTF">2007-05-20T10:16:00Z</dcterms:created>
  <dcterms:modified xsi:type="dcterms:W3CDTF">2007-05-20T10:16:00Z</dcterms:modified>
  <cp:category>Article dotnet</cp:category>
  <cp:contentStatus>En cours</cp:contentStatus>
  <cp:keywords>dotnet OpenXML</cp:keywords>
  <cp:revision>2</cp:revision>
</cp:coreProperties>
```

Vous aurez compris que ce code va pouvoir être réutilisé pour récupérer n'importe quelle partie d'un package OpenXML. Il suffira de remplacer le type de contenu afin d'obtenir la partie (objet ***PackagePart***) correspondante.

Note importante pour la suite:

Nous avons récupérer les objets ***PackageRelationship*** en partant de l'objet ***Package*** (*officePackage*). Ainsi, la propriété ***TargetUri*** des objets ***PackageRelationship*** renverra une *Uri* relative au package (à la racine du package). Pour la partie *core.xml* cette *Uri* sera donc "*docProps/core.xml* ". Pour pouvoir récupérer la partie *core.xml* en utilisant la méthode ***GetPart*** il nous faut son *Uri*absolue. C'est pourquoi nous utilisons la méthode ***ResolvePartUri*** qui retourne l'*Uri* absolue d'une partie à partir de l'*Uri* d'une source (ici "/", c'est-à-dire le package) et de l'*Uri* relative de cette partie.

IV-C - Lecture des propriétés

Maintenant que nous avons l'objet ***PackagePart*** correspondant au fichier *core.xml*, nous allons pouvoir charger son contenu dans un objet ***XmlDocument*** afin d'en lire le contenu. A partir de là il ne s'agit plus que de la simple lecture d'un fichier XML.

Voici le code permettant de charger la partie ***core.xml*** dans un ***XmlDocument*** et d'en récupérer quelques informations (l'auteur, le titre, une liste de mots clefs et la date de création du document):

```
if (corePart != null)
{
    //construction d'un XmlNamespaceManager contenant les namespaces utilisés
    NameTable nt = new NameTable();
    XmlNamespaceManager nsmgr = new XmlNamespaceManager(nt);
    nsmgr.AddNamespace("dc", dcPropertiesSchema);
    nsmgr.AddNamespace("cp", cpPropertiesSchema);
    nsmgr.AddNamespace("dcterms", dctermsPropertiesSchema);

    //chargement de la partie dans un XmlDocument
    XmlDocument doc = new XmlDocument(nt);
    doc.Load(corePart.GetStream());

    XmlNode nodeAuteur = doc.DocumentElement.SelectSingleNode("//dc:creator", nsmgr);
    if (nodeAuteur != null)
        labelAuteur.Text = nodeAuteur.InnerText;

    XmlNode nodeTitre = doc.DocumentElement.SelectSingleNode("//dc:title", nsmgr);
    if (nodeTitre != null)
        labelTitre.Text = nodeTitre.InnerText;

    XmlNode nodeMotsClefs = doc.DocumentElement.SelectSingleNode("//cp:keywords", nsmgr);
    if (nodeMotsClefs != null)
        labelMotsClefs.Text = nodeMotsClefs.InnerText;

    XmlNode nodeDate = doc.DocumentElement.SelectSingleNode("//dcterms:created", nsmgr);
```

```
    if (nodeDate != null)
        labelDate.Text = DateTime.Parse(nodeDate.InnerText).ToShortDateString();
}
```

On construit tout d'abord un ***XmlNamespaceManager*** contenant les différents namespaces utilisés dans le fichier ***core.xml***. On instancie ensuite un objet ***XmlDocument*** en lui passant en paramètre le ***XmlNamespaceManager***. Enfin, on appelle la méthode ***GetStream*** pour charger le contenu de la partie en mémoire et on rempli l'objet ***XmlDocument*** grâce à ce flux.

On récupère ensuite les éléments voulus grâce à la méthode ***SelectSingleNode***.

IV-D - Modification des propriétés

Après avoir lu les propriétés du document nous allons voir comment modifier leurs valeurs. Il n'y a rien de compliqué à cela. Il suffit simplement de se servir des ***XmlNode*** que nous avons créés précédemment et d'en modifier leur valeur.

Ainsi, pour modifier le nom de l'auteur il suffit d'écrire:

```
//  Met à jour la valeur:
nodeAuteur.InnerText = "Toto";
```

Ces modifications s'effectuent sur le ***XmlDocument*** chargé en mémoire. Il nous faut donc ensuite le sauvegarder dans le package OpenXML. Pour cela on utilise la méthode ***Save*** de l'objet ***XmlDocument***:

```
//  Enregistre le XML des propriétés dans sa partie:
doc.Save(corePart.GetStream(FileMode.Create, FileAccess.Write));
```

Voilà, c'est tout !

V - Lecture de la partie principale d'un document OpenXML

La partie principale d'un document Word OpenXML est générale représentée par le fichier *document.xml* contenu dans le dossier word (voir l'arborescence présentée au début de ce tutorial). Tout le contenu textuel d'un document Word s'y trouve. Nous allons illustrer ici la lecture de cette partie par l'écriture d'une méthode qui aura pour but de rechercher un mot (ou une expression) quelconque dans un fichier Word.

La première chose à faire est donc de récupérer la partie principale du document OpenXML au travers d'un objet ***PackagePart***. Pour cela il nous suffit de réutiliser la méthode vue précédemment en changeant simplement le type de contenu à rechercher:

```
//type de contenu pour la partie principale
const String officeDocRelType = @"http://schemas.openxmlformats.org/officeDocument/2006/relationships/
officeDocument";

PackagePart mainPart = null;
Uri documentUri = null;
//on récupère la partie contenant les propriétés
foreach (PackageRelationship relationship in officePackage.GetRelationshipsByType(officeDocRelType))
{
    // Il n'y a qu'une seule partie de type partType dans le package
    documentUri = PackUriHelper.ResolvePartUri(new Uri("/", UriKind.Relative), relationship.TargetUri);
    mainPart = officePackage.GetPart(documentUri);
    break;
}
```

"http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument" correspond au type de contenu de la partie principale d'un document OpenXML.

Nous pouvons ensuite charger le contenu de cette partie dans un objet ***XmlDocument*** et y rechercher un mot grâce à la méthode ***Contains***:

```
if (mainPart != null)
{
    //chargement de la partie dans un XmlDocument
    XmlDocument doc = new XmlDocument();
    doc.Load(mainPart.GetStream());

    //sensible à la casse
    if (doc.DocumentElement.InnerText.Contains(textBoxRechercher.Text))
    {
        MessageBox.Show("Texte trouvé dans le document");
    }
    else
    {
        MessageBox.Show("Impossible de trouver le texte dans le document");
    }
}
```

Et ce n'est pas plus compliqué que ça !

VI - Extraire les images d'un document Word OpenXML

Le traitement des images va nécessiter un peu plus de code (mais vraiment pas beaucoup plus). Le principe reste en fait toujours le même: les images sont des parties comme les autres qu'il va falloir récupérer à l'aide de la technique vu précédemment. Bien évidemment, leur lecture différera de celle d'un ficher XML.

La première chose à faire est de récupérer la partie principale du document OpenXML. Nous l'avons fait juste au-dessus, il est donc inutile de remettre le code. Peut-être vous demandez-vous pourquoi nous avons besoin de cette partie. En effet, jusqu'à présent nous récupérions directement la partie qui nous intéressait grâce à notre bout de code passe-partout et en changeant uniquement le type de contenu. Pourquoi donc ne pas simplement refaire la même chose ?

Pour une raison très simple: le package ignore où se trouve ces images. Pour mieux comprendre, jetez de nouveau un coup d'oeil à l'arborescence d'un fichier OpenXML qui se trouve en début d'article. A la racine se trouve un dossier *_rels* avec un fichier *.rels* à l'intérieur. Voici un extrait de ce qu'il contient:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="rId3"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/extended-properties"
   Target="docProps/app.xml"/>
<Relationship Id="rId2"
   Type="http://schemas.openxmlformats.org/package/2006/relationships/metadata/core-properties"
   Target="docProps/core.xml"/>
<Relationship Id="rId1"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument"
   Target="word/document.xml"/>
</Relationships>
```

On y retrouve quelques vieux amis... Ce fichier indique au package où se trouvent certain types de contenu (comme la partie principale ou les parties propriétés). Comme vous pouvez le voir, aucune trace d'une quelconque image. Déplaçons nous maintenant dans le dossier *_rels* se trouvant dans le dossier *word* et ouvrons le fichier *document.xml.rels* (qui rassemble les liaisons existantes entre *document.xml* et d'autres parties).

Voilà enfin nos fameuses images ! Notez au passage la référence à la partie des styles; cela indique que sa récupération se fera de la même manière que pour les images.

Que faut-il retenir de tout ceci ? Simplement que certaines parties (comme *document.xml* ou *core.xml*) sont liées directement au package alors que d'autres (comme les images) sont liées à une autre partie (ici la partie *document.xml*). Il faut voir les choses de cette manière: un package OpenXML est liée à une partie principale, de parties de propriétés, etc. La partie principale peut elle-même être liée à d'autres parties (styles, images, videos, etc.). Ainsi les images sont connues de la partie principale qui est elle-même connue du package. Bref, pour avoir les images il faut avoir la partie principale.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="rId1"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styles"
   Target="styles.xml"/>
<Relationship Id="rId5"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image"
   Target="media/image2.png"/>
<Relationship Id="rId4"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image"
   Target="media/image1.png"/>
</Relationships>
```

Tel qu'il est écrit, le code permettant de récupérer une partie n'autorise que la récupération d'une partie liée au package. Nous allons donc le modifier un peu pour qu'il puisse récupérer une partie liée à la partie principale du package. Dans notre cas nous voulons récupérer **les** images liées. En voici la nouvelle version:

```
//type de contenu pour une image
const String imageRelType = @"http://schemas.openxmlformats.org/officeDocument/2006/relationships/
image";


List<PackagePart> listePackageParts = new List<PackagePart>();
Uri imageUri = null;
//on récupère les parties correspondantes aux images. Les images sont relatives à la mainPart
foreach (PackageRelationship relationship in mainPart.GetRelationshipsByType(imageRelType))
{
    //relationship.TargetUri contient media/image1.jpg (par exemple)
    imageUri = PackUriHelper.ResolvePartUri(new Uri(mainPart.Uri.ToString(), UriKind.Relative),
      relationship.TargetUri);
    listePackageParts.Add(officePackage.GetPart(imageUri));

    //il n'y a pas forcément qu'une seule image donc on ne fait pas de break !!!!
}
```

Quelle sont les nouveautés ? Nous avons bien entendu modifié le type de contenu pour y mettre le type correspondant aux images. Nous avons ensuite déclaré une liste d'objets ***PackagePar***. En effet, cette fois-ci il n'y aura pas qu'une seule partie à récupérer mais plusieurs (tout dépendra évidemment du nombre d'images dans le package). La nouveauté suivante concerne la méthode ***GetRelationshipsByType***. Nous ne l'appelons plus depuis l'objet ***Package*** mais depuis la partie principale (objet ***PackagePart***). Comme dit précédemment nous voulons récupérer les relations entre la partie principale et les parties de type image.

La propriété TargetUri des objets ***PackageRelationship*** va renvoyer une *Uri* relative d'une partie de type image par rapport à la partie principale (par exemple *media/image1.jpg*). Pour récupérer la partie il va nous falloir l'*Uri* absolue que l'on construit grâce à la méthode ***ResolvePartUri*** à laquelle on donne en paramètre l'*Uri* source (la partie principale) et l'*Uri* relative de l'image.

Bien, nous voilà à présent propriétaire d'une liste de ***PackagePart*** correspondant aux différentes images du package. Comme pour les précédents exemples, nous pouvons en récupérer le contenu grâce à la méthode ***GetStream*** (sauf qu'évidemment cette fois fois-ci on ne le charge pas dans un fichier XML.).

Plusieurs choix s'offrent maintenant à vous. Vous pouvez par exemple construire un objet de type Image:

```
foreach (PackagePart imagePart in listePackageParts)
{
   Image image = Image.FromStream(imagePart.GetStream());
   // traitement quelconque avec image
}
```

Ou bien enregistrer les images sur le disque dur:

```
// pour chaque partie image, on l'enregistre
foreach (PackagePart imagePart in listePackageParts)
{
    // pour récupérer le nom des images
    String[] tab = imagePart.Uri.ToString().Split(new Char[] { '/' });
    String nomImage = tab[tab.Length-1];

    using (Stream sourceStream = imagePart.GetStream())
    {
        // répertoire de destination
        String path = Path.Combine(@"C:\", "images");
        if (!Directory.Exists(path))
            Directory.CreateDirectory(Path.Combine(@"C:\", "images"));
        using (FileStream targetStream = new FileStream(Path.Combine(path, nomImage),
   FileMode.Create, FileAccess.Write))
        {
            byte[] buffer = new byte[1024];
            int nrBytesWritten = sourceStream.Read(buffer, 0, 1024);
            while (nrBytesWritten > 0)
            {
```

```
                targetStream.Write(buffer, 0, nrBytesWritten);
                nrBytesWritten = sourceStream.Read(buffer, 0, 1024);
            }
        }
    }
}
```

VII - Pour aller plus loin

Microsoft a sorti un package de snippets utilisable avec Visual Studio 2005 destiné au format de documents Open XML (Word, Excel et PowerPoint). Et ça se trouve ici : **snipets**. Ces modèles de code sont une véritable mine d'informations pour mieux comprendre comment manipuler le format OpenXML.

VIII - Conclusion

Nous avons donc vu au travers de cet article les bases pour la lecture de documents Word au format Open XML en utilisant les briques du Framework 3.0 autour de l'assembly System.IO.Package.

La lecture de documents Office (Word dans cet exemple) peut désormais être exécutée sans qu'Office ne soit installé et sans devoir utilisé les *Primary Interop Assemblies* d'Office comme c'était le cas avec les anciens formats.

Vous pouvez dorénavant lire simplement le contenu d'un fichier Word OpenXML afin d'en récupérer ou d'en changer le contenu.

IX - Liens

Le site Open XML chez Microsoft

Introducing the Office (2007) Open XML File Formats

How to: Manipulate Office Open XML Formats Documents

Structure d'un document Open XML

Open XML sur le blog de Julien Chable

openxmldeveloper.org

Spécifications du format à l'ECMA

Les webcast sur Open XML

Structure des fichiers OpenXML (developpez.com)

Lecture des fichiers OpenXML avec PHP 5 (developpez.com)

Remerciements

J'adresse ici tous mes remerciements à l'équipe de rédaction de "developpez.com" pour le temps qu'ils ont bien voulu passer à la correction et à l'amélioration de cet article.

Contact

Si vous constatez une erreur dans le tutorial, dans les sources, dans la programmation ou pour toutes informations, n'hésitez pas à me contacter par le **forum**.