Cours .Net
Lambda Expression
Open XML

Lemettre Arnaud

Arnaud.lemettre@gmail.com

ÉCOLE D'INGÉNIEURS EN INFORMATIQUE

Microsoft® .NET

Sommaire

- Introduction
- Lambda Expression
  - Les différentes utilisations
- OpenXML
  - Création d'un document (docx)
  - Lecture d'un document (docx)

Introduction

- Lambda Expression, rajout de la partie fonctionnelle dans le langage C#. Dans sa version 3.0
- Permet d'écrire du code beaucoup plus vite.
- Rapidité d'exécution dans certains cas.

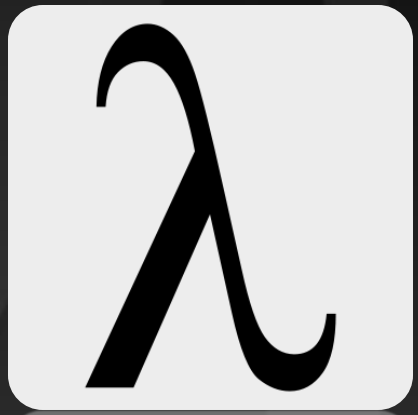
Introduction

- Open xml, est le format de fichier de Microsoft. Nous allons voir la façon de le manipuler et de générer un docx. Bien entendu, le principe de fonctionnement reste le même pour un fichier excel

Lambda Expression

Définition

- Une *expression lambda* est une fonction anonyme qui peut contenir des expressions et des instructions et peut être utilisée pour créer des délégués ou des types d'arborescence d'expression.



Définition

- Le symbole d'une lambda expression est « => ». Cela se lit : conduit à

Utilité …

- On peut les utiliser couplées avec du Linq (Objet, XML, SQL).
- Mais aussi au sein de code.
- Cela permet d'aller plus vite dans certains cas.
- Et de simplifier le code.

Utilisation

Avec Linq To Object

Exemple avec une requête Linq :

```
List<int> list = new List<int>() { 1, 2, 3, 4 };

var resLinq = from x in list
              where x > 2
              select x;
List<int> res = resLinq.ToList();
```

La même chose en Lambda Expression :

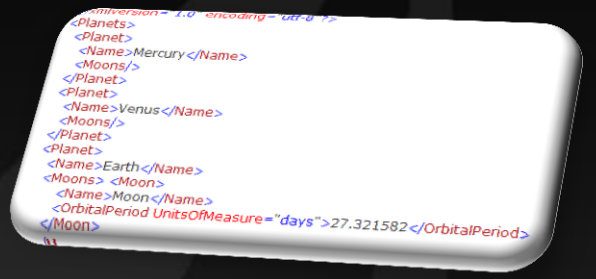
```
List<int> list = new List<int>() { 1, 2, 3, 4 };

List<int> res = list.FindAll(x => x > 2);
```

Utilisation

Avec Linq To XML:

Soit le XDocument suivant :

```
XDocument doc = XDocument.Parse("<root><book>framework
    .net</book><book>asp.net</book><book>IIS</book></root>");
```



Avec une requête en Linq :

```
List<XElement> elts = (from tmpElement in doc.Root.Descendants()
                       where tmpElement.Value.Contains(".net")
                       select tmpElement).ToList();
```

Avec Lambda Expression :

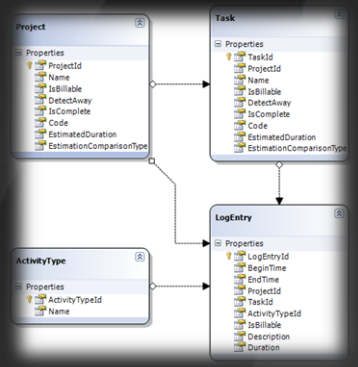
```
List<XElement> elts2 = doc.Root.Descendants().Where(x => x.Value.Contains(".net")).ToList();
```

UTILISATION

Avec Linq To SQL:

En requête Linq :

```
var query = from tmpJoueur in dataContext.T_Joueurs
            where tmpJoueur.id == 1
            select tmpJoueur;

List<T_Joueur> joueurs = query.ToList();
```



En Lambda Expression :

```
DataClassesDataContext dataContext = new DataClassesDataContext();
List<T_Joueur> res = dataContext.T_Joueurs.Where(x => x.id == 1).ToList();
```

Utilisation

- Au sein de code : Cela est très utile pour la déclaration de certains événements ou de thread.

Utilisation

Exemple : Avec un timer

```
private static Timer _timer2;
static void Main(string[] args)
{
     _timer2 = new Timer();
    _timer2.Interval = 10;
    _timer2.Elapsed += new ElapsedEventHandler(timer2_Elapsed);

     Timer timer = new Timer();
    timer.Interval = 10;
    timer.Elapsed += new ElapsedEventHandler((sender, e) =>
    {
        Console.WriteLine("plop");
        timer.Stop();
    });

    timer.Start();
     _timer2.Start();

    Console.ReadKey();
 }

static void timer2_Elapsed(object sender, ElapsedEventArgs e)
{
    Console.WriteLine("Plop");
    _timer2.Stop();
}
```

Possibilité d'avoir accès aux variables au sein de la déclaration de l'expression. Chose infaisable avec une déclaration classique.

Utilisation

Exemple : Avec un thread

```
static void Main(string[] args)
{
    Thread thread = new Thread(new ThreadStart(() =>
        {
            Console.WriteLine("Plop");
        }));

    thread.Start();
    Console.ReadKey();

    Thread thread2 = new Thread(new ThreadStart(Task));
    thread2.Start();
    Console.ReadKey();
}

static void Task()
{
    Console.WriteLine("Work");
}
```

La déclaration des threads de cette manière offre plus de visibilité sur les tâches qu'on accomplit.

UTILISATION

Attention cependant à l'utilisation des lambdas expressions, car si on n'est pas strict sur le nommage des paramètres cela peut vite devenir un code illisible (ex : nom de variables à une lettre dans des expressions imbriquées).
Il ne faut pas non plus les utiliser tout le temps, dans certains cas l'ancienne méthode reste la meilleure solution.

Utilisation

Exemple en vrac :

```
List<int> numbers = new List<int>() { 1, 2, 3, 4 };

numbers.ForEach(n => Console.WriteLine(n));

List<float> floatNumbers = numbers.ConvertAll<float>(n => (float)n);
floatNumbers.ForEach(x => Console.WriteLine(x));

numbers.Sort((x, y) => y - x);

numbers.RemoveAll(n => n % 2 != 0);
numbers.ForEach(n => Console.WriteLine(n));
Console.ReadKey();
```


Open XML

Introduction

- Office Open XML est une norme ISO/IEC (DIS 29500) créée par Microsoft, destinée à répondre à la demande d'interopérabilité dans les environnements de bureautique et à concurrencer la solution d'interopérabilité OpenDocument.

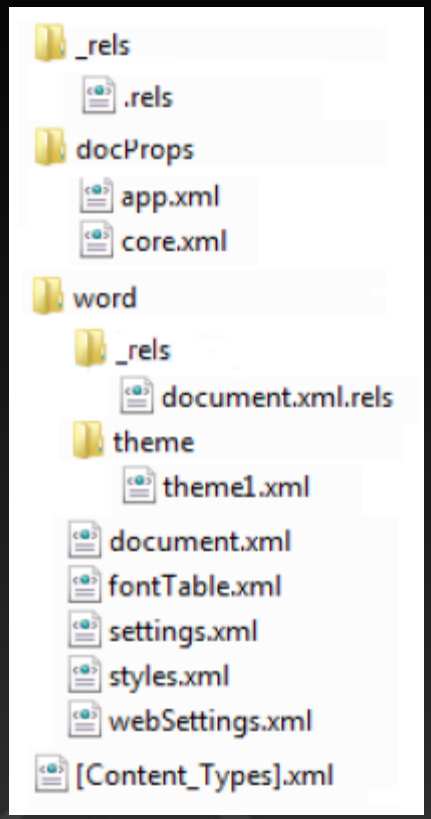
Introduction

- Open XML utilise une structure respectant l'Open Packaging Convention
- Le ZIP permet outre de compresser les documents, de pouvoir stocker les données de façon totalement indépendante dans une architecture segmentée
- Protéger les documents Office Open XML plus efficacement face à la corruption des données

Introduction

- Pourquoi utiliser de l'Open XML en programmation :
  - Permet de générer des rapports, export de bases de données, de faire des graphiques.
  - Avantage par rapport à l'interop : pas besoin d'avoir installé la suite office sur le serveur, donc des économies en licence.

FORMAT



Nom du document	Description
Document.xml	Est l'une des parts les plus importantes du paquet. En effet c'est dans ce fichier que nous retrouverons l'organisation du texte Word et également une partie des données.
Styles.xml	Contient les styles du document.
[Content_Type].xml	Contient les différents types de contenu d'une part.
Custom.xml	Contient les données personnalisables du document.
Images	

Ecriture d'un fichier

- Il existe 2 façons de faire.
  - Manuellement, en insérant les bons nœuds dans le document.xml
  - En utilisant le SDK fourni par Microsoft.

  il existe 2 version, la 1 qui contient quelques fonctionnalités et la 2 passée en release qui offre beaucoup plus de facilité pour le traitement. Cependant les 2 sont encore incomplètes et on doit parfois avoir recours à la 1er méthode.

Ecriture d'un fichier

- Les liens sont les suivants :
  - SDK 2.0
  - Documentation prendre toujours la documentation en anglais (plus complète)
- Dans un premier temps il faut installer le sdk pour qu'il soit disponible dans Visual Studio.
- Si vous utilisez le .NET 4.0 ou supérieur vous pouvez utiliser le SDK 2.5

Ecriture d'un fichier

- Dans le SDK, il y a un outil :
  - OpenXMLtool.exe

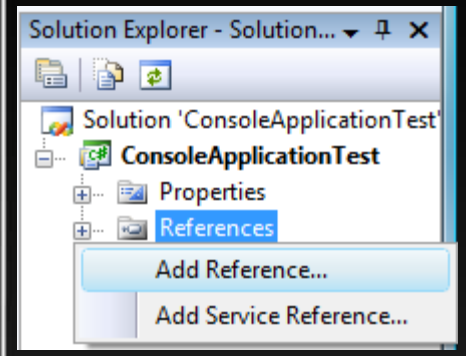
    Une fonction Diff : permet de faire une comparaison entre 2 docx. Utile lorsque l'on génère les docx à la main.

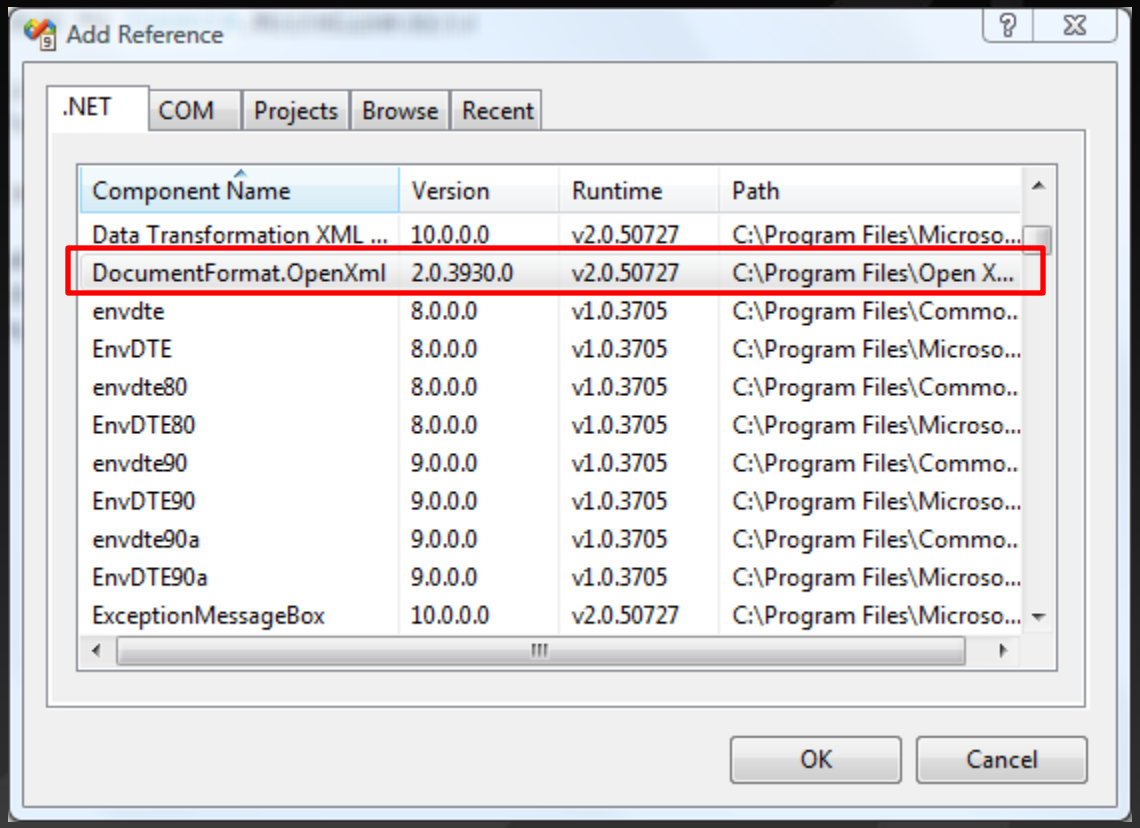
    Une fonction Reflector : permet de générer des documents OpenXML avec le code C# associé.

    Cependant, il faut quand même comprendre ce que génère cet outil

Ecriture d'un fichier

- Utilisation du SDK 2.0





L'ajout de la référence de cette manière suppose l'installation du SDK sur le poste client. On peut directement rajouter la DLL pour qu'elle soit copiée lors du déploiement.

C:\Program Files\Open XML Format SDK\V2.0\lib

Ecriture d'un fichier

- Pour utiliser également le system.IO.packaging il faut ajouter la référence à WindowsBase
- Il faut également rajouter les using suivant :

```
using DocumentFormat.OpenXml.Packaging;
using DocumentFormat.OpenXml;
using DocumentFormat.OpenXml.Wordprocessing;
```

Ecriture d'un fichier

```
string docName = "test.docx";
using (WordprocessingDocument package =
    WordprocessingDocument.Create(docName,
    WordprocessingDocumentType.Document))
{
 //besoin de le faire si c'est une creation
    package.AddMainDocumentPart();

    // Création du document.
    package.MainDocumentPart.Document =
        new Document(
            new Body(
                new Paragraph(
                    new Run(
                        new Text("1er Texte")))));

    // Enregistrer le contenu dans le document
    package.MainDocumentPart.Document.Save();
}
```

Pour rajouter des paragraphes il suffit d'accéder
à la propriété Body du document.

Ecriture d'un fichier

On peut également rajouter des styles aux paragraphes, l'une des méthodes est la suivante :

- Il faut d'abord ajouter le style

```
Styles styles = new Styles(…) ;
```

Puis il faut spécifier la valeur du style dans le paragraphe

```
new Paragraph(new ParagraphProperties(new ParagraphStyleId() {
   Val = "Title" }));
```

Id en anglais du style

Cette partie n'est pas détaillée, mais il y aura une correction. Ca fait partie du bonus ;)

Ecriture d'un fichier

```
string docName = "test.docx";
using (WordprocessingDocument package = WordprocessingDocument.Create(docName,
    WordprocessingDocumentType.Document))
{
    package.AddMainDocumentPart();

    package.MainDocumentPart.Document =
        new Document(
            new Body(
                new Paragraph(
    // il manque le Run
                    new Text("Hello World !"))));
    package.MainDocumentPart.Document.Save();

    //validation
    OpenXmlValidator validator = new OpenXmlValidator();
    var errors = validator.Validate(package);
    //il y aura une erreur
    foreach (ValidationErrorInfo error in errors)
    {
        Console.WriteLine(error.Description);
    }

}
```

Grâce à la dernière version, le SDK offre la possibilité de vérifier l'intégrité des documents. Et de signaler quand il y a des erreurs. Cette signalisation est beaucoup plus précise que celle apportée par Office. Cependant cela ne permet pas encore d'avoir toutes les erreurs.

Ecriture d'un Fichier

- Cette façon de générer le document n'est pas très dynamique, ni évolutif. En effet si on doit changer quelque chose il faut recompiler le projet.
- Pour pallier à cela on peut utiliser les templates de Word.

Ecriture d'un Fichier

- Le principe est de créer le template sous word puis de le remplir en utilisant le code C#. Pour cela il faut ouvrir le document et éditer les 2 fichiers :
  - Document.xml
  - Customs.xml
- Pour cette opération on ne peut pas utiliser le SDK. Il faut passer par du Linq to XML.
- Article sur la création de template

Lecture d'un Fichier

- La lecture de fichier dans une application est beaucoup plus rare, voir quasi inexistante.

- Cependant cela peut s'avérer utile surtout lors de l'utilisation de template et pour compléter certaines données.
- L'autre utilisation serait d'afficher une preview du document.

Lecture d'un Fichier

- La lecture d'un fichier OpenXML se base sur la récupération des parties qui nous intéressent.
- Pour un document Word la partie intéressante est le document.xml qui contient le texte ainsi que certains éléments typographiques.

Lecture d'un Fichier

Permet de récupérer l'ensemble des textes d'un document.

```
string docName = "test.docx";
using (WordprocessingDocument package =
    WordprocessingDocument.Open(docName, false))

{
     List<Run> listText =
    package.MainDocumentPart.Document.Descendants<Run>().ToList();

     foreach (Run item in listText)
     {
       Console.WriteLine( item.InnerText);
     }
}
```


Autres utilisations

- On peut également utiliser l'OpenXML pour faire de la fusion de documents ; pour cela une classe intéressante est :

  ALTCHUNK

QUESTIONS ?