



Dotnet France
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

LINQ to XML

Version 1.1

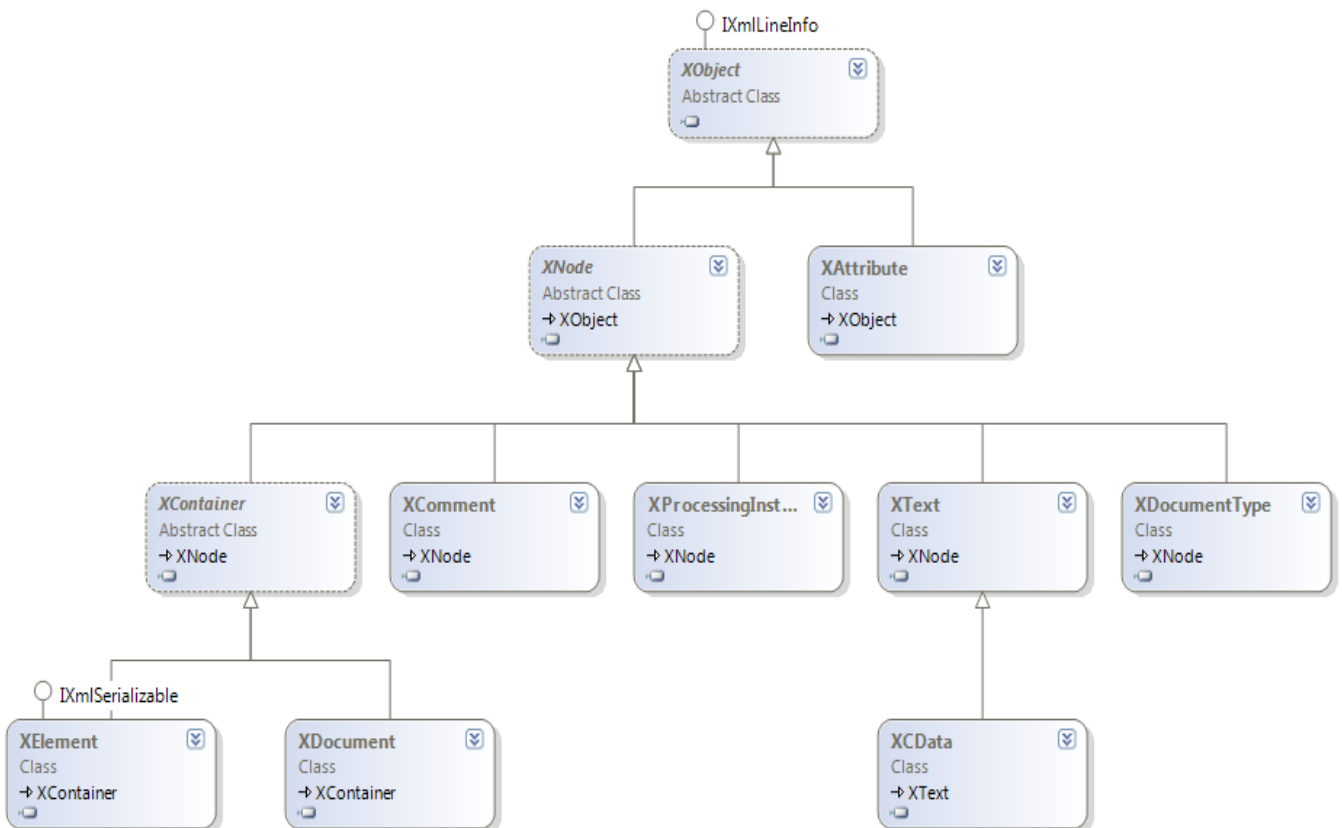
Sommaire

| | | |
|-----|---|----|
| 1 | Présentation | 3 |
| 2 | Les différentes classes de LINQ to XML..... | 4 |
| 2.1 | « XDocument » et « XElement » :..... | 4 |
| 2.2 | « XAttribute » et « XComment » :..... | 5 |
| 3 | Les méthodes et propriétés | 5 |
| 3.1 | Méthodes de « XDocument » (hérite de « XContainer ») :..... | 6 |
| 3.2 | Méthodes de « XElement » (hérité de « XContainer ») : | 7 |
| 4 | Différentes utilisations de LINQ to XML..... | 9 |
| 5 | Exemples..... | 11 |
| 6 | Conclusion | 15 |

1 Présentation

LINQ permet d'établir des requêtes sur du XML de manière générique. C'est-à-dire que LINQ n'est pas un langage qui permet de réaliser ces requêtes exclusivement dans du XML. LINQ permet également la récupération de données depuis une base de données SQL, des objets, etc. Dans ce chapitre, nous allons nous intéresser au LINQ to XML, aussi appelé XLinQ. Dans le but d'utiliser LINQ to XML, il est utile d'ajouter la librairie « *System.Xml.Linq* ». Afin de mieux comprendre ce chapitre, il est conseillé d'avoir déjà lu les chapitres « *Introduction au LINQ* » et « *LINQ to Objects* ». Vous y trouverez notamment des définitions, des mots-clés, la syntaxe, l'utilisation de LINQ dans un projet, etc. De plus, le chapitre Windows Form ou ASP.NET « *XML (ADO.NET)* » permet une meilleure compréhension du langage XML.

Voici un diagramme de classe de *System.Xml.Linq* :



2 Les différentes classes de LINQ to XML

Pour commencer, nous allons apprendre à nous servir de `XLinq` et plus précisément de ses classes. Contrairement au XML où les classes sont préfixées de « *Xml* », dans LINQ to XML celles-ci sont préfixées de « *X* ». Par exemple, « *XmlDocument* » devient « *XDocument* », « *XmlElement* » devient « *XElement* », etc. Afin de mieux découvrir ces classes, nous illustreront nos explications d'exemples. Nous créerons dans un premier temps un document XML, aucune requête LINQ ne sera réalisée pour le moment.

2.1 « *XDocument* » et « *XElement* » :

C#

```
XDocument document = new XDocument(
    new XElement("Racine",
        new XElement("Test", "Contenu"),
        new XElement("Test2",
            new XElement("Fils", "Test réussi")),
        new XElement("Test3", "Contenu"));

Console.WriteLine(document);
```

VB.NET

```
Dim document As XDocument = <?xml version="1.0" encoding="utf-8"?>
    <Racine>
        <Test>Contenu</Test>
        <Test2>
            <Fils>Test réussi</Fils>
        </Test2>
        <Test3>Contenu</Test3>
    </Racine>

Console.WriteLine(document)

Console.Read()
```

« *XDocument* » correspond au document XML. « *XElement* », lui, correspond à un nœud. En VB.NET, la déclaration du document XML doit être placée au début de la création de celui-ci pour qu'il soit correctement considéré comme un « *XDocument* ». Dans le cas contraire, il sera considéré comme un « *XElement* » ce qui provoquera une erreur. En C#, cette ligne de déclaration est rajoutée automatiquement.

```
<Racine>
  <Test>Contenu</Test>
  <Test2>
    <Fils>Test réussi</Fils>
  </Test2>
  <Test3>Contenu</Test3>
</Racine>
```

Les types « *XDocument* » et « *XElement* » héritent tous deux de la classe abstraite « *XContainer* ». Cette classe implémente des méthodes permettant, entre-autre, la recherche d'un nœud suivant ou précédent dans un document XML. Cette classe elle-même hérite de la classe « *XNode* ».

Dans le cas présent, nous utilisons le constructeur de la classe « *XDocument* » qui prend en paramètre tous les éléments de type XML pouvant être inclus dans un document du même type. « *XElement* » est assez similaire à la seule différence qu'il attend en premier argument le nom qui lui sera attribué.

De la même manière qu'en XML, « *XDocument* » crée l'instruction de traitement. Par défaut, l'encodage des caractères est « *UTF-8* » et la version est « *1.0* ».

2.2 « XAttribute » et « XComment » :

C#

```

XDocument document = new XDocument(
    new XElement("Racine",
        new XAttribute("id", "test"),
        new XElement("Test", "Contenu"),
        new XElement("Test2",
            new XAttribute("ID", "0"),
            new XElement("Fils", "Test réussi")),
        new XComment("Voici un commentaire"),
        new XElement("Test3", "Contenu"));

    Console.WriteLine(document);

document.Save(@"C:\MyProject\ConsoleApplication3\ConsoleApplication3\Test.xml");

Console.Read();

```

VB.NET

```

Dim document As XDocument = <?xml version="1.0" encoding="utf-8"?>
    <Racine id="test">
        <Test>Contenu</Test>
        <Test2 ID="0">
            <Fils>Test réussi</Fils>
        </Test2>
        <!--Voici un commentaire-->
        <Test3>Contenu</Test3>
    </Racine>

Console.WriteLine(document)

Console.Read()

```

« XAttribute » crée un attribut et prend comme argument le nom de cet attribut et sa valeur. L'élément « XComment » crée un commentaire et ne prend un « String » en argument représentant le contenu du commentaire.

```

<Racine id="test">
  <Test>Contenu</Test>
  <Test2 ID="0">
    <Fils>Test réussi</Fils>
  </Test2>
  <!--Voici un commentaire-->
  <Test3>Contenu</Test3>
</Racine>

```

L'élément « XDeclaration », quant à lui, permet de spécifier ou d'obtenir la déclaration pour le traitement du document XML. Il est notamment demandé la version du document et l'encodage des caractères.

3 Les méthodes et propriétés

Il est utile de connaître les classes inhérentes à LINQ to XML, mais il sera encore plus intéressant d'en connaître les méthodes et propriétés.

Les tableaux ci-dessous décrivent les méthodes et propriétés disponibles pour les classes vues précédemment.

3.1 Méthodes de « XDocument » (hérite de « XContainer ») :

| Méthodes / Propriétés | Description |
|-----------------------|---|
| CreateReader | Crée un XmlReader pour le nœud courant. |
| CreateWriter | Créer un XmlWriter pour le nœud courant. |
| Declaration | Obtient ou définit la déclaration du document. |
| DescendantNodes | Retourne une collection des nœuds descendant contenus dans le document ou le nœud courant. |
| Descendants | Retourne une collection des éléments descendant contenus dans le document ou le nœud courant. Surchargé. |
| DocumentType | Obtient la définition du type du document(ou DTD) pour ce document. |
| Elements | Retourne une collection qui contient les éléments enfants correspondant au document ou à l'élément passé en argument. Surchargé. |
| Load | Permet de créer un XDocument. Elle prend en argument le lien vers le fichier, un TextReader, ou encore un XmlReader. Surchargé. |
| Nodes | Retourne une collection des nœuds enfants de l'élément courant ou du document. |
| RemoveNodes | Supprime les nœuds enfants de l'élément courant ou du document. |
| ReplaceNodes | Remplace les nœuds enfants de l'élément courant ou du document par les arguments spécifiés. Surchargé. |
| Root | Obtient le nœud racine du document XML. |
| Save | Permet de sauvegarder le XDocument vers le fichier spécifié (s'il n'existe pas il sera créé). Il peut aussi prendre en argument un TextWriter ou un XmlWriter. Surchargé. |

3.2 Méthodes de « XElement » (hérité de « XContainer ») :

| Méthodes / Propriétés | |
|-----------------------|--|
| Add | Ajoute les arguments en tant qu'enfant de cet élément. Surchargé. |
| AddAfterSelf | Ajoute les arguments immédiatement après l'élément. Surchargé. |
| AddBeforeSelf | Ajoute les arguments immédiatement avant l'élément. Surchargé. |
| AddFirst | Ajoute les arguments en tant que premier enfant. Surchargé. |
| Attribute | Retourne l'attribut de cet élément au nom spécifié. |
| Attributes | Retourne une collection de tous les attributs de l'élément courant. Surchargé. |
| CreateReader | Créer un XmlReader pour le nœud courant. |
| CreateWriter | Créer un XmlWriter qui peut être utilisé pour ajouter des nœuds au conteneur courant. |
| Element | Retourne le premier élément enfant correspondant au nom spécifié. |
| ElementsAfterSelf | Retourne une collection contenant les éléments adjacents à cet élément qui se trouvent après celui-ci et qui correspondent au nom spécifié. Surchargé. |
| ElementsBeforeSelf | Retourne une collection contenant les éléments adjacents à cet élément qui se trouvent avant celui-ci et qui correspondent au nom spécifié. Surchargé. |
| FirstNode | Obtient le premier nœud enfant du nœud courant. |
| DescendantNodes | Retourne une collection de tous les nœuds se situant après le nœud courant. |
| Descendants | Retourne une collection de tous les nœuds se situant après le nœud courant qui correspondent à l'élément actuel ou au document. Surchargé. |
| HasAttributes | Retourne un booléen définissant si le nœud en question possède au moins un attribut. |
| HasElements | Retourne un booléen définissant si le nœud contient au moins un élément enfant. |
| IsEmpty | Retourne un booléen définissant si l'élément contient une valeur. |
| LastNode | Obtient le dernier nœud enfant du nœud courant. |
| NextNode | Obtient le nœud adjacent suivant le nœud courant. |

| | |
|-------------------|---|
| NodeType | Obtient le type du nœud courant. |
| Nodes | Retourne une collection des nœuds enfants de l'élément courant ou du document. |
| NodesAfterSelf | Retourne une collection contenant les nœuds adjacents se trouvant après ce nœud. |
| NodesBeforeSelf | Retourne une collection contenant les nœuds adjacents se trouvant avant ce nœud. |
| Parent | Obtient le nœud parent du nœud courant. |
| PreviousNode | Obtient le nœud précédent du nœud courant. |
| Remove | Supprime ce nœud. |
| RemoveAttributes | Supprimer tous les attributs de l'élément courant. |
| RemoveNodes | Supprime les nœuds enfants de l'élément courant ou du document. |
| ReplaceNodes | Remplace les nœuds enfants par les arguments spécifiés. Surchargé. |
| ReplaceWith | Remplace ce nœud par les arguments spécifiés. Surchargé. |
| Save | Sauvegarde l'arborescence XML du nœud dans un fichier, un « <i>XmlTextWriter</i> », un « <i>TextWriter</i> » ou encore un « <i>XmlWriter</i> ». |
| SetAttributeValue | Permet de définir la valeur, de supprimer ou d'ajouter un élément enfant. |
| SetElementValue | Permet de définir la valeur, de supprimer ou d'ajouter un élément enfant. |
| Value | Obtient le contenu de cet élément. |

4 Différentes utilisations de LINQ to XML

Il est conseillé, avant de continuer, d'avoir pris connaissance du chapitre « Introduction au LINQ », du chapitre sur « LINQ to Objects » et du chapitre Windows Form ou ASP.NET « XML (ADO.NET) ». Cela dit, dans un souci de compréhensions, nous effectuerons certains rappels.

LINQ to XML peut être utilisé pour effectuer des requêtes dans un « *XDocument* » ou également dans une base de données dont les informations seront stockées dans un « *XDocument* » ou un « *XElement* ». Des exemples de différentes difficultés sont à venir.

Requête sur un « *XDocument* » :

C#

```
XDocument doc = new XDocument(
    new XElement("Racine",
        new XElement("Element",
            new XAttribute("special", "Admin"), "Utilisateur Admin"),
        new XElement("Element", "Utilisateur Normal"),
        new XElement("PasElement", "Utilisateur"));

IEnumerable<XElement> test = from i in doc.Descendants("Element")
                             select i;

foreach (XElement e in test)
{
    Console.WriteLine(e + "\n");
}

Console.Read();
```

VB.NET

```
Dim doc As XDocument = <?xml version="1.0" encoding="utf-8"?>
    <Racine>
        <Element special="Admin">Utilisateur
Admin</Element>
        <Element>Utilisateur Normal</Element>
        <PasElement>Utilisateur</PasElement>
    </Racine>

Dim test As IEnumerable(Of XElement) = From i In doc.Descendants("Element")
-
                                     Select i

For Each e In test
    Console.WriteLine(e.ToString() + vbNewLine)
Next

Console.Read()
```

Voici le résultat de ces requêtes :

```
<Element special="Admin">Utilisateur Admin</Element>
<Element>Utilisateur Normal</Element>
```

Explication : Tout d'abord, nousinstancions un « *XDocument* » dans lequel nous créons une arborescence XML contenant des données. Cela va nous permettre d'établir des requêtes sur celui-ci. Ensuite, nous créons un objet qui contient la requête. Enfin nous procédons à un affichage des données grâce à un « *foreach* ».

Revenons sur la requête pour un petit rappel. L'élément suivant le mot-clé « *From* » correspond à la variable qui sera utilisée comme alias pour la source de données. Le mot-clé « *In* » définit la source de données contenant les informations souhaitées.

Dans notre cas, la source de données est composée des descendants du document (tous les nœuds du document) qui ont pour nom « *Element* ». Nous pouvons constater que le nœud s'appelant « *PasElement* » n'a pas été pris en compte par la requête et ne s'affiche pas dans la console. Le mot-clé « *Select* » permet de définir ce que l'on veut sélectionner et donc retourner. Ici, nous voulons la valeur « *i* » de type « *IEnumerable* » de « *XElement* ».

Voici une nouvelle requête sur un autre élément qu'un « *XDocument* » :

En définitif, cela revient à effectuer une requête classique que nous allons formater en XML et que nous stockerons dans un « *XDocument* ».

C#

```
XDocument doc = new XDocument(
    new XElement("Utilisateurs",
        from util in bdd.Utilisateur
        select new XElement("Utilisateur",
            new XAttribute("ID", util.ID),
            new XElement("Nom", util.Nom),
            new XElement("Prenom", util.Prenom))));
```

Remarque : Il n'existe aucun équivalent en VB.NET.

Explication : Dans un premier temps, nous créons un document XML à l'aide de l'instruction « *new* ». Ensuite, le premier élément de ce document doit être un nœud racine. « *XElement* » attend comme argument le nom d'un nœud puis son contenu. Dans notre code, nous effectuons une requête LINQ afin de récupérer des valeurs pour nos nœuds suivants et les rattacher au nœud parent (qui est le nœud racine). Ces données peuvent provenir de toute source de données interrogeables avec LINQ. Dans la requête ci-dessus, on récupère une table « *Utilisateur* » dans son intégralité et on crée un alias sur la table qui s'appelle « *util* ». Notre requête se termine par un « *Select new XElement* » permettant de récupérer les données sous ce même type à la place d'un tableau classique. Nous créons ainsi l'arborescence du fichier XML. Les informations de la source de données sont stockées dans des « *XElements* » avec leur « *XAttribute* ». Ils seront donc identifiables par des ID.

Remarque : Il est aussi possible de récupérer cela dans un « *XElement* » plutôt qu'un « *XDocument* ». Cela peut permettre par exemple de l'ajouter dans un document XML déjà existant.

5 Exemples

Dans cette partie, nous présentons quelques exemples d'utilisation de LINQ to XML. Nous réalisons notre premier exemple en utilisant un « *XDocument* ».

C#

```
XDocument doc = new XDocument(  
    new XElement("Racine",  
        new XElement("Utilisateur",  
            new XAttribute("Recherche", "Trouvé")),  
        new XElement("UtilisateurAvecEnfant",  
            new XElement("Utilisateur", "Enfant",  
                new XAttribute("Recherche", "Perdu"))),  
        new XElement("Utilisateur",  
            new XAttribute("Recherche", "Perdu"),  
            "Contenu du noeud"));  
  
Console.WriteLine(doc); //On peut l'afficher pour vérifier si sa fonctionne
```

VB.NET

```
Dim doc As XDocument = <?xml version="1.0" encoding="utf-8"?>  
    <Racine>  
        <Utilisateur Recherche="Trouvé" />  
        <UtilisateurAvecEnfant>  
            <Utilisateur  
Recherche="Perdu">Enfant</Utilisateur>  
        </UtilisateurAvecEnfant>  
        <Utilisateur Recherche="Perdu" />  
    </Racine>  
  
Console.WriteLine(doc)
```

Voici le résultat de l'affichage de ce code :

```
<Racine>  
<Utilisateur Recherche="Trouvé" />  
<UtilisateurAvecEnfant>  
  <Utilisateur Recherche="Perdu">Enfant</Utilisateur>  
</UtilisateurAvecEnfant>  
<Utilisateur Recherche="Perdu">Contenu du noeud</Utilisateur>  
</Racine>
```

Nous allons maintenant créer une requête permettant de récupérer le nœud correspondant à un « *Utilisateur* » qui a l'attribut « *Recherche* » avec la valeur « *Trouvé* ».

C#

```
IEnumerable<XElement> requete = from d in doc.Root.Descendants()  
    where d.Name == "Utilisateur"  
    && d.Attribute("Recherche").Value == "Trouvé"  
    select d;
```

VB.NET

```
Dim requete As IEnumerable(Of XElement) = From d In doc.Root.Descendants()  
-     Where d.Name = "Utilisateur" _  
     And d.Attribute("Recherche").Value = "Trouvé" _  
     Select d
```

L'image ci-contre illustre le résultat retourné par cette commande si nous l'affichons.

```
<Utilisateur Recherche="Trouvé" />
```

Si nous retirons la deuxième condition, il y a deux nœuds « *Utilisateur* » pris en compte par la requête.

L'exemple suivant est un peu plus complet. En effet, nous utiliserons deux documents XML afin de montrer un peu plus les capacités de LINQ to XML.

C#

```

XDocument doc = new XDocument (
    new XElement("Racine",
        new XElement("Utilisateur",
            new XAttribute("id", "0")),
        new XElement("UtilisateurAvecEnfant",
            new XAttribute("id", "1")),
        new XElement("Utilisateur",
            new XAttribute("id", "0"))));

XDocument doc2 = new XDocument (
    new XElement("Racine",
        new XElement("Manager",
            new XAttribute("id", "0")),
        new XElement("Stagiaire",
            new XAttribute("id", "1"))));

IEnumerable<XElement> requete = from d in doc.Root.Descendants()
                                from d2 in doc2.Root.Descendants()
                                where d.Attribute("id").Value ==
                                    d2.Attribute("id").Value
                                select new XElement("Utilisateur",
d2.Name);

foreach (XElement e in requete)
{
    Console.WriteLine(e);
}

```

VB.NET

```

Dim doc As XDocument = <?xml version="1.0" encoding="utf-8"?>
    <Racine>
        <Utilisateur id="0"/>
        <UtilisateurAvecEnfant id="1"/>
        <Utilisateur id="0"/>
    </Racine>

Dim doc2 As XDocument = <?xml version="1.0" encoding="utf-8"?>
    <Racine>
        <Manager id="0"/>
        <Stagiaire id="1"/>
    </Racine>

Dim requete As IEnumerable(Of XElement) = From d In doc.Root.Descendants()
    - From d2 In doc2.Root.Descendants() _
      Where d.Attribute("id").Value = d2.Attribute("id").Value _
      Select New XElement("Utilisateur", d2.Name)

For Each e In requete
    Console.WriteLine(e)
Next

```

Le résultat de la requête est affiché ci-contre. La création de document XML ayant été déjà vue précédemment, nous passons cette partie du code.

```
<Utilisateur>Manager</Utilisateur>
<Utilisateur>Stagiaire</Utilisateur>
<Utilisateur>Manager</Utilisateur>
```

La requête, quant à elle, est plus intéressante à analyser. Pour les deux documents créés, nous récupérons le nœud racine ainsi que ses descendants grâce à l'instruction « `doc.Root.Descendants()` ». L'élément ainsi récupéré est une collection de « XElement » contenant respectivement les utilisateurs et les statuts. Dans le premier document XML, nous pouvons remarquer un « id » qui est, grâce à notre requête, lié à un « id » correspondant au deuxième document, dans le but de récupérer le statut de chaque utilisateur. Pour effectuer cela, nous avons utilisé deux instructions « From », mais nous aurions pu très bien opter pour l'utilisation de l'instruction « Join » qui aurait produit le même résultat. L'instruction « Where » juste après nous permet de créer la liaison entre les « id » des deux documents XML. Enfin, nous utilisons l'instruction « Select » afin de spécifier les informations que l'on souhaite que la requête nous retourne. Pour chaque « Utilisateur » et « Statut » avec un « id » équivalent, nous demandons un nouveau « XElement » ayant pour nom « Utilisateur » et pour valeur le nom du statut de celui-ci.

Pour l'affichage, nous exécutons un « foreach » qui parcourra le « IEnumerable » renvoyé par la requête afin d'afficher chacune des valeurs contenues dans celui-ci.

6 Conclusion

Ce chapitre sur LINQ to XML aborde les notions essentielles sur la manipulation de documents XML dans vos projets Windows Form ou Web Form. Il vous sera aussi utile de prendre connaissance des chapitres « Introduction au LINQ », « LINQ to Objects » et « XML (ADO.NET) » dans les chapitres Windows Form. Les mots-clés de LINQ vous permettront d'établir toute sorte de récupération de données sur des documents XML en utilisant les méthodes et propriétés des classes présentées dans ce chapitre.

Plus d'information sur MSDN : <http://msdn.microsoft.com/fr-fr/library/system.xml.linq.aspx>