

XML

Sommaire

XML	1
1 Présentation	2
2 Structure d'un document XML	3
2.1 Constitution	3
2.1.1 Instruction de traitement	3
2.1.2 Commentaire	3
2.1.3 Les noeuds	3
2.2 Validité d'un document XML	4
3 Manipulation basique d'un document XML	5
3.1 Création d'un XmlDocument	5
3.2 Sauvegarder un XmlDocument	6
3.3 Déplacement	7
3.4 Lecture d'information / affichage	8
3.4.1 XmlReader :	10
3.5 Création / Ajout	11
3.5.1 XmlTextWriter	12
3.6 Modification	14
3.7 Suppression	17
4 Avec un DataSet	17
5 Conclusion	19

1 Présentation

Un document XML (eXtended Markup Language) est un document contenant du texte qui représente des données. Ces données sont placés sous forme hiérarchique : elles sont sous forme d'arborescence. Pour une représentation, ce sont des balises qui peuvent en contenir d'autres. Nous verrons les termes employés pour un document XML (comme nœud, nœud parent ...) dans la partie suivante (2 Structures d'un document XML). Nous verrons aussi dans la prochaine partie qu'il y a des critères pour qu'un document XML soit valide. Comme dit plus haut ce genre de document possède des balises. Ces balises ne sont pas prédéfinies. C'est-à-dire que l'on crée les siennes. Il pourrait y en avoir une qui s'appelle « Contacts » comme une qui s'appelle « Lieu ». C'est plus tard que nous gérerons la manipulation de ce document.

Ces documents sont très utilisés pour le stockage de données (pas en quantité importante). Contrairement à un objet qui possède une durée de vie égale à la durée de l'application dans le meilleur des cas, un document XML n'a pas de réelle durée de vie. C'est-à-dire que nous décidons le moment où nous voulons le supprimer.

Par ailleurs puisqu'il y a une hiérarchie et que nous devons définir la manière de se déplacer dans le document, il faut l'avoir fait avec une certaine logique. Voici un petit exemple de ce que cela peut donner avec une liste de fruit.

```
XML

<ListeFruit>
  <Fruit id="1" nom="Poire">
    <couleur>Jaune</couleur>
    <gout>Sucrée</gout>
  </Fruit>
  <Fruit id="2" nom="Raisin">
    <couleur>Rosé</couleur>
    <gout>Sucrée</gout>
  </Fruit>
  <Fruit id="3" nom="Mûre">
    <couleur>Rouge</couleur>
    <gout>Amer</gout>
  </Fruit>
</ListeFruit>
```

La balise ListeFruit contient toute la liste de fruit. Ensuite chaque balise Fruit représente un fruit et possèdent comme attribut l'id du fruit et son nom. Chacune de ces balises Fruit contiennent d'autres balises qui décrivent le fruit.

Le XML est un langage validé par le W3C (World Wide Web Consortium), vous trouverez d'autres informations en visitant leur site <http://www.w3.org/xml> et <http://www.w3.org/TR/xml>.

Passons maintenant à l'explication de la structure de cet exemple.

2 Structure d'un document XML

Un document XML étant un langage validé par le W3C, il possède une syntaxe à respecter ainsi qu'une structure. Nous allons comment il est constitué, structuré, puis quels sont les critères pour avoir un document XML valide.

2.1 Constitution

2.1.1 Instruction de traitement

Tout d'abord, il doit commencer par une instruction de traitement qui définit quel type de fichier c'est et les informations qu'il faut pour le faire fonctionner. Plus exactement pour permettre au processeur XML ou aux applications qui vont vouloir interagir avec ce document de connaître la version, l'encodage ... Une instruction de traitement ressemble à ceci : `<?NomApplication Instruction ?>`

En XML et au moment où ce chapitre est écrit, elle doit être comme suit : `<?xml version="1.0" encoding="utf-8" ?>`. On spécifie dans cette instruction l'encodage du document XML.

2.1.2 Commentaire

Une chose importante lorsque l'on fait du développement c'est de mettre des commentaires. Pour cela, il existe une balise de commentaire

```
<!-- Exemple de commentaire -->
```

Pour ces commentaires on peut mettre n'importe quels caractères à l'intérieur. En revanche, la séquence `--` est interdite.

2.1.3 Les noeuds

Un noeud XML est un conteneur de données ou d'autre noeud. Concrètement ce sont les balises du document XML, par exemple `<ListeFruit></ListeFruit>`. En anglais nœud se dit node. Les noeuds XML répondent à des règles concernant leur forme.

- Il ne doit posséder aucun espace dans son nom, que ce soit en début, milieu ou fin
- Son nom ne peut pas commencer par un chiffre ou un caractère de ponctuation
- Evidemment la balise de fin doit correspondre à la balise de début
- Son nom ne peut pas commencer par XML quel que soit la casse (`<XmlTest>` n'est donc pas valide)
- Il doit posséder au moins un nœud racine (que nous verrons juste après) avec une balise de début et une balise de fin

Vous rencontrerez ou vous entendrez sûrement parler de la classe `XmlElement`. Il est à savoir qu'un élément en XML correspond à un nœud.

Parlons maintenant un peu plus en détail de ces nœuds. Nous allons voir les termes que l'on donne à ces nœuds suivant leurs positions, ce qu'ils contiennent

Le nœud racine :

C'est lui qui contient tout le reste du document. On peut s'en passer mais par soucis de validité il le faut. En anglais il se dit Root Node.

Element racine

```
<?xml version="1.0" encoding="utf-8" ?>
<ListeFruit> <!-- Element racine -->

  <!-- Contenu du document -->

</ListeFruit>
```

Nœud parent/fils :

Un nœud fils (ChildNode) est un nœud contenu dans un autre nœud. Le nœud qui le contient s'appelle le nœud parent.

Nœud parent / fils

```
<Fruit> <!-- Nœud parent du nœud couleur et gout -->
  <couleur>Jaune</couleur> <!-- Nœud fils de Fruit -->
  <gout>Sucrée</gout> <!-- Nœud fils de Fruit -->
</Fruit>
```

Il y a aussi un autre type de nœud qui sont les nœuds adjacents. Ces nœuds sont ceux qui sont situés au même niveau que le nœud courant.

Attribut :

Un nœud peut posséder un ou plusieurs attributs. C'est vous qui créez le nom de l'attribut et sa valeur au même titre que les noms des nœuds.

Attribut

```
<Fruit id="1" nom="Poire">
```

Dans cet exemple id et nom sont des attributs.

2.2 Validité d'un document XML

Que veut dire être valide ? Dans ce cas cela veut dire qu'il respecte la norme du W3C. Il faut donc qu'il soit bien formé. Il lui faut donc une ligne d'instruction de traitement, au moins le nœud racine qui doit être constitué de deux balises. De plus il ne peut y avoir dans le texte des caractères spéciaux comme &<> Pour les mettre il faut les remplacer par ce qui correspond dans le tableau ci-dessous :

&	& ;
<	< ;
>	> ;
'	' ;
"	" ;

A titre d'exemple, `<couleur>Jaune & Vert</couleur>` n'est pas valide, il faut écrire `<couleur>Jaune & ; Vert</couleur>`.

XML permet aussi de vérifier si un document XML est conforme à une syntaxe donnée. Un document DTD (Document Type Definition) est un document type contenant la syntaxe, la grammaire pour vérifier la conformité du document XML. La norme XML n'impose pas l'utilisation d'un DTD. Ainsi un document XML n'ayant pas de DTD mais qui répond aux règles de base est un document bien formé. Un document valide est un document XML comportant un DTD et le respectant. Pour ajouter un DTD il faut ajouter un DOCTYPE tout comme en HTML.

3 Manipulation basique d'un document XML

Pour utiliser les classes XML il faut importer l'espace de noms *System.Xml*.

3.1 Création d'un XmlDocument

Pour pouvoir travailler sur un document XML ou sur une chaîne de caractère XML, il faut créer un objet de type XmlDocument. Il existe deux méthodes du XmlDocument qui nous seront utiles pour remplir notre objet XmlDocument : Load et LoadXml.

Load	<p>Permet de charger le contenu d'un document ou d'un objet contenant du XML dans le XmlDocument.</p> <p>Argument :</p> <ul style="list-style-type: none"> - Un objet Stream : charge le XmlDocument depuis le flux spécifié - String : URL vers le fichier contenant le XML - TextReader : charge depuis un TextReader - XmlReader : charge depuis un XmlReader
LoadXml	Attend en argument une chaîne de caractères contenant du XML.

Voici un exemple d'utilisation de ces deux méthodes :

C#

```
XmlDocument document = new XmlDocument();
document.Load(@"C:\MyProject\ConsoleApplication1\ConsoleApplication1\Fruit.xml");
```

```
XmlDocument document = new XmlDocument();
document.LoadXml("<Fruit id='1' nom='Poire'><couleur>Jaune &
Vert</couleur><gout>Sucrée</gout></Fruit>");
```

Rappel : En C# l'arobase (@) devant une chaîne de caractère permet d'éviter d'avoir à échapper le caractère d'échappement et donc d'avoir \\.

VB.NET

```
Dim document As XmlDocument = New XmlDocument()
document.Load("C:\MyProject\ConsoleApplication1\ConsoleApplication1\Fruit.xml")
```

```
Dim document As XmlDocument = New XmlDocument()
document.LoadXml("<Fruit id='1' nom='Poire'><couleur>Jaune &
Vert</couleur><gout>Sucrée</gout></Fruit>")
```

Pour la validité du document il faut créer la ligne d'instruction de traitement. Pour cela on va utiliser un objet de type `XmlDeclaration` qui nous permet de déclarer l'encodage, la version ...

C#

```
XmlDocument document = new XmlDocument();

XmlDeclaration declar;
declar = document.CreateXmlDeclaration("1.0", "utf-8", null);
```

VB.NET

```
Dim document As XmlDocument = New XmlDocument()

Dim declar As XmlDeclaration
declar = document.CreateXmlDeclaration("1.0", "utf-8", Nothing)
```

Nous verrons dans la partie traitant de l'ajout de données comment ajouter cette ligne à notre document XML.

Passons à la sauvegarde de ce document dans un fichier.

3.2 Sauvegarder un XmlDocument

Maintenant que nous avons appris à créer un `XmlDocument` et à le remplir, nous allons voir comment le sauvegarder. Pour ce faire, on utilise la méthode `Save` du `XmlDocument`. Voir dans le tableau ci-dessous les arguments que peut prendre cette méthode.

Save	<ul style="list-style-type: none"> - Un objet Stream : enregistre dans le flux spécifié - Un String : correspond à une URL, enregistre dans le fichier spécifié - Un objet TextWriter ou un objet XmlWriter : enregistre dans l'objet TextWriter ou XmlWriter spécifié
------	---

Exemple d'utilisation avec un String :

<p><i>C#</i></p> <pre>document.Save(@"C:\MyProject\ConsoleApplication1\ConsoleApplication1\Fruit.xml");</pre>
<p><i>VB.NET</i></p> <pre>document.Save("C:\MyProject\ConsoleApplication1\ConsoleApplication1\Fruit.xml")</pre>

Pour pouvoir sauvegarder le document, il faut que celui-ci soit valide. En particulier qu'il possède un nœud racine. Nous verrons comment créer un nœud racine plus tard. Pour revenir à la méthode Save, si vous ne mettez en String qu'un nom de fichier (Fruit.xml) par exemple, votre fichier se trouvera dans le document *bin\Debug* de votre projet (par défaut).

3.3 Déplacement

On entend par déplacement le fait d'accéder à un endroit précis du document XML pour accéder aux données, aux nœuds enfants etc.

Pour cela on va donner plusieurs méthodes et propriétés qui permettent ceci et on expliquera un exemple qui les regroupera toutes (comme XmlDocument hérite de XmlNode, on retrouve des méthodes dans les deux).

Méthodes	Description
GetElementsByTagName	Méthode de XmlDocument qui permet de récupérer un XmlNodeList contenant tous les éléments qui correspondent au nom spécifié en argument.

Toutes les propriétés qui suivent se retrouvent dans XmlDocument et XmlNode sauf une : DocumentElement.

Propriétés	Description
ChildNodes	Retourne un XmlNodeList. Récupère tous les nœuds enfants du nœud.
DocumentElement	Retourne un XmlNode qui contient le nœud racine du document.
FirstChild	Retourne le premier nœud enfant du nœud courant.
HasChildNodes	Retourne un booléen indiquant si le nœud possède des enfants.
LastChild	Retourne le dernier nœud enfant du nœud courant.
NextSibling	Retourne le premier nœud qui suit le nœud courant.
ParentNode	Retourne le nœud parent.
PreviousSibling	Retourne le nœud précédent le nœud courant.

Pour effectuer une modification, un ajout ou toute autre action, il faut se placer dans le document à un endroit où l'on pourra effectuer notre action et donc il faut récupérer un objet que l'on appellera nœud courant. Cette méthode et ces propriétés permettent de récupérer l'objet qui vous intéressera. Nous verrons des exemples concrets dans les parties suivantes.

3.4 Lecture d'information / affichage

Il existe différents moyens de lire les données contenues dans le fichier XML. Nous allons commencer par voir les méthodes qui permettent de lire les informations d'un nœud ou groupe de nœud. Il y aura aussi des méthodes qui permettent de récupérer des informations qu'il suffira ensuite d'afficher.

Ceci s'applique sur les XmlNode :

Propriétés	Description
Attributes	Récupère un XmlAttributeCollection qui contient les attributs du nœud.
InnerText	Récupère les valeurs (value) de tous les nœuds enfants. Retourne un string
InnerXml	Récupère le XML contenu dans le nœud soit tout le XML qui compose les nœuds enfants de ce nœud. Retourne un string
Name	Récupère le nom du nœud. Accessible en lecture seule.
NodeType	Retourne le type du nœud courant. On peut voir les différents types de nœud possible dans l'enum XmlNodeType.
OwnerDocument	Récupère le XmlDocument auquel ce nœud appartient
Value	Récupère ou définit la valeur du nœud

Exemple d'affichage utilisant le document Xml de fruit vu plus haut :

```

C#

XmlDocument doc = new XmlDocument();
doc.Load(@"C:\MyProject\ConsoleApplication1\ConsoleApplication1\Fruit.xml");

foreach (XmlNode e in doc.ChildNodes)
{
    Console.WriteLine(e.Name);
}

foreach (XmlNode e in doc.DocumentElement.ChildNodes)
{
    Console.WriteLine("    " + e.Name + "    id= " + e.Attributes["id"].Value +
Nom= " + e.Attributes[1].Value);
    foreach (XmlNode i in e.ChildNodes)
    {
        Console.WriteLine("        " + i.Name + "    Valeur : " +
i.InnerText);
    }
    Console.WriteLine("\n\n");
}

Console.Read();

```

```

VB.NET

Dim doc As XmlDocument = New XmlDocument()
doc.Load("C:\MyProject\ConsoleApplication1\ConsoleApplication1\Fruit.xml")

For Each e In doc.ChildNodes
    Console.WriteLine(e.Name)
Next

For Each e In doc.DocumentElement.ChildNodes
    Console.WriteLine("    " + e.Name + "    id= " +
e.Attributes("id").Value + "    Nom= " + e.Attributes(1).Value)
    For Each i In e.ChildNodes
        Console.WriteLine("        " + i.Name + "    Valeur : " +
i.InnerText)
    Next
    Console.WriteLine(vbNewLine + vbNewLine)
Next

Console.Read()

```

Pour l'explication nous avons préféré mettre des phrases ici plutôt qu'en commentaire pour ne pas alourdir le code. La première et la seconde ligne de code sont destinées à créer un XmlDocument puis à le remplir avec le document Xml contenant la liste de fruit. Une fois le document chargé nous voulons l'afficher : pour cela nous avons choisi une boucle foreach. Nous récupérerons donc une liste des nœuds se trouvant dans le document avec la propriété ChildNodes. Si

```

xml
ListeFruit
Fruit id= 1 Nom= Poire
couleur Valeur : Jaune
gout Valeur : Sucrée

Fruit id= 2 Nom= Raisin
couleur Valeur : Rosé
gout Valeur : Sucrée

Fruit id= 3 Nom= Mûre
couleur Valeur : Rouge
gout Valeur : Amer

```

vous regardez ce que cela donne sur l'image du résultat, vous remarquerez qu'il contient la déclaration Xml et le nœud racine.

La seconde boucle elle doit afficher le contenu du nœud racine. On va donc récupérer le root du document avec la propriété DocumentElement et utiliser à nouveau la propriété ChildNodes pour en récupérer tous les nœuds. De là on affiche le nom avec la propriété Name et les attributs avec Attributs qui prend en paramètre soit le numéro de l'attribut (le premier correspond à 0, le second à 1 ...) soit une chaîne de caractère correspondant au nom du nœud. En utilisant la propriété Value sur les attributs que l'on a récupérés, on récupère leurs valeurs pour l'affichage. Si on avait utilisé la propriété Name on aurait alors récupéré le nom de l'attribut.

Dans cette seconde boucle on va créer une troisième boucle pour pouvoir récupérer les nœuds qui contiennent les valeurs sur les fruits. Dans cette boucle on affiche encore une fois le nom des nœuds concaténé avec leurs valeurs que l'on récupère avec la propriété Value. Le Console.Read à la fin permet de voir le résultat affiché dans la console.

3.4.1 XmlReader :

Un autre moyen est d'utiliser un objet de type XmlReader. Il correspond à un lecteur qui fournit un accès en lecture seule, qui n'est pas mis en cache et qui ne peut être lue que dans un seul sens : le sens normal de la lecture (du haut du fichier vers le bas).

```
C#  
  
XmlReader reader = XmlReader.Create("Fruit.xml");  
  
while (!reader.EOF)  
{  
    reader.Read();  
    Console.Write(reader.Value);  
}  
reader.Close();  
  
Console.Read();
```

```
VB.NET  
  
Dim reader As XmlReader = XmlReader.Create("Fruit.xml")  
  
While Not reader.EOF  
    reader.Read()  
    Console.Write(reader.Value)  
End While  
  
reader.Close()  
  
Console.Read()
```

On crée un objet reader de type XmlReader sur lequel on utilise la méthode Create du XmlReader qui permet de créer un flux et qui attend, dans notre exemple, l'URL d'un fichier XML. Notre XmlReader instancié et initialisé nous allons maintenant faire la boucle pour lire les données. La propriété EOF

retourne un booléen qui détermine si le reader est à la fin de la lecture ou non. La méthode Read du XmlReader permet de lire le nœud suivant (puisque le reader se déplace de nœud en nœud).

Après la boucle on ferme le reader avec la méthode Close. Le Console.Read() n'étant là que pour nous permettre de lire ce qui est affiché dans la console.

Maintenant voici les tableaux qui recensent les méthodes et propriétés utiles du XmlReader :

Méthodes	Description
Close	Change ReadState en Closed et ferme l'accès au fichier.
Create	Créer une instance de XmlReader.
GetAttribute	Obtient la valeur de l'attribut spécifié.
Value	Récupère la valeur du nœud en cours.

3.5 Création / Ajout

Méthodes du XmlDocument :

Méthodes	Description
CreateAttribute	Permet de créer un attribut
CreateComment	Créer un XmlComment qui contient les données spécifiées en argument (en String).
CreateNode	Créer et retourne un XmlNode
ReadNode	Créer un nœud à partir des informations contenues dans le XmlReader (plus exactement par rapport au nœud sur lequel il est positionné).

Ce qui suit sont les méthodes du XmlNode qui permettent la création ou l'ajout de nœud dans le Xml.

Méthodes	Description
AppendChild	Insère le nœud spécifié à la fin des nœuds enfants du nœud courant. Il s'utilise de la manière suivante : NomObjetNoeud.AppendChild(NoeudAAjouter)
Clone	Permet de créer un clone du nœud spécifié.
InsertAfter	Insère le nœud spécifié juste après le nœud courant.
InsertBefore	Insère le nœud spécifié juste avant le nœud courant.
PrependChild	Insère le nœud spécifié au début des nœuds enfants du nœud courant.

Il peut être intéressant de voir avec un exemple comment on crée et ajoute des attributs :

C#

```

XmlDocument document = new XmlDocument();
document.Load(path); //On charge le fichier XML : path correspond à l'url du fichier

XmlNode node = document.DocumentElement; // On récupère dans un XmlNode le noeud
racine

XmlAttribute nodeattr = document.CreateAttribute("test"); //On crée un attribut avec
la méthode CreateAttribute du XmlDocument
nodeattr.Value = "Valeur"; //On donne une valeur à l'attribut
node.Attributes.Append(nodeattr); // On ajoute l'attribut au noeud
  
```

VB.NET

```

Dim document As XmlDocument = New XmlDocument()
document.Load(path)

Dim node As XmlNode = document.DocumentElement

Dim nodeattr As XmlAttribute = document.CreateAttribute("test")
nodeattr.Value = "Valeur"
node.Attributes.Append(nodeattr)
  
```

Pour créer un attribut, on instancie un objet XmlAttribute qui va contenir le résultat de la méthode CreateAttribute (qui prend un argument un string qui correspond au nom de l'attribut) du XmlDocument. La Méthode Value du XmlDocument permet d'obtenir ou de définir la valeur de l'attribut. Ensuite on l'ajoute au nœud (dans l'exemple le nœud est le nœud racine).

3.5.1 XmlTextWriter

On a vu que XmlReader permet de récupérer un flux en lecture seule. Ici XmlTextWriter va permettre de récupérer un flux dans lequel on peut écrire.

Tableau des méthodes du XmlTextWriter :

Méthodes	Description
Close	Ferme les flux utilisés
Dispose	Libère les ressources utilisés par le XmlTextWriter
WriteAttributes	Attend en argument un XmlReader. Ecrit les attributs trouvés à la position actuelle du XmlReader.
WriteAttributeString	Attend deux arguments qui sont le nom de l'attribut et sa valeur. Permet d'écrire un attribut dans l'élément en cours.
WriteCData	Permet la création d'un bloc <![CDATA[...]]>
WriteComment	Attend en argument le commentaire à ajouter.
WriteDoctype	Permet la création d'un Doctype.
WriteEndDocument	Termine l'écriture du document (ferme toute les écritures en cours).
WriteEndElement	Termine l'écriture d'un élément
WriteStartDocument	Démarre l'écriture d'un document
WriteStartElement	Démarre l'écriture d'un élément. Attend en paramètre le nom de cet élément.
WriteString	Ecrit le texte passé en argument dans l'élément en cours.

L'exemple qui va suivre va être la création d'un document Xml en utilisant un objet XmlTextWriter :

```
C#

XmlTextWriter rw = new
XmlTextWriter(@"C:\MyProject\ConsoleApplication6\ConsoleApplication6\test.xml",
Encoding.UTF8);

rw.WriteStartDocument();
  rw.WriteStartElement("Root");

    rw.WriteStartElement("Noeud");
      rw.WriteAttributeString("id", "value");
      rw.WriteString("Sa fonctionne");
      rw.WriteComment("Commentaire");
      rw.WriteEndElement();

    rw.WriteEndElement();
  rw.WriteEndDocument();

rw.Close();
```

```
VB.NET

Dim rw As XmlTextWriter = New
XmlTextWriter("C:\MyProject\ConsoleApplication7\ConsoleApplication7\test.xml",
Encoding.Default)

rw.WriteStartDocument()
  rw.WriteStartElement("Root")

    rw.WriteStartElement("Noeud")
      rw.WriteAttributeString("id", "value")
      rw.WriteString("Sa fonctionne")
      rw.WriteComment("Commentaire")
      rw.WriteEndElement()

    rw.WriteEndElement()
  rw.WriteEndDocument()

rw.Close()
```

```
<?xml version="1.0" encoding="Windows-1252"?><Root><Noeud id="value">Sa fonctionne<!--Commentaire--></Noeud></Root>
```

Comme vous pouvez le voir dans l'image ci-dessus, tout se trouve sur une ligne. Aussi nous n'avons pas utilisé de moyen de sauvegarde : puisque c'est un flux il va écrire directement et en temps réel dans le fichier.

Tout d'abord nous avons créé un objet XmlTextWriter dans lequel on définit le fichier qu'on utilisera et le type d'encodage. Pour Encoding il faut avoir importé System.Text. Une fois notre objet instancié, on peut commencer à écrire dedans. Pour cela on utilise des méthodes tels que WriteStartDocument qui permet d'écrire la ligne d'instruction Xml, WriteStartElement qui va commencer l'écriture d'un élément ect ... Le contenu du document, d'un élément ou autre sera compris entre le start et le end (WriteStartElement / WriteEndElement). Pour le reste référez vous au tableau plus haut. On finit par un close pour fermer le flux.

3.6 Modification

Pour modifier une donnée déjà présente dans le document XML, il faut récupérer dans un objet les ou les nœuds correspondant. Pour faire simple nous ne donnerons qu'un tableau avec les méthodes qui permettent de modifier des données sans exemple.

Méthode	Description
ReplaceChild	Remplace par le premier nœud en paramètre le second nœud spécifié.

Propriétés	Description
InnerText	Récupère les valeurs (value) de tous les nœuds enfants. Retourne un string
InnerXml	Récupère le XML contenu dans le nœud soit tout le XML qui compose les nœuds enfants de ce nœud. Retourne un string
Prefix	Obtient ou définit le préfixe de l'espace de nom du nom en cours
Value	Obtient ou définit la valeur du nœud

Exemple utilisant le ReplaceChild et le InnerXml :

C# - Code adapté pour une solution web application

```
DataSet ds = new DataSet("Test"); // Créer un dataset
DataTable dt = new DataTable("myTable"); // Créer un databale

dt.Columns.Add("col1"); // Ajoute une colonne dans la table
dt.Columns.Add("col2"); // Une deuxième colonne
// Ajoute une ligne = des éléments dans les colonnes
dt.Rows.Add("Dans col1", "Dans col2");
// Ajoute la table dans le dataset
ds.Tables.Add(dt);

// On crée une seconde DataTable
DataTable dt2 = new DataTable("mySecondTable");
dt2.Columns.Add("colA"); // Première colonne
dt2.Columns.Add("colB"); // Deuxième colonne
dt2.Rows.Add("ABC", "DEF"); // Insertion d'éléments dans les colonnes

ds.Tables.Add(dt2); // Ajout de la seconde table dans le DataSet

ds.WriteXml(Server.MapPath("test.xml")); // On met crée le document test.xml

ds.Dispose(); // On libère les ressources du DataSet
ds.Reset(); // On nettoie le DataSet
dt.Dispose(); // On libère les ressources du DataTable
dt.Reset(); // On nettoie le DataTable

XmlDocument doc = new XmlDocument(); // On crée un objet XML nommé doc

// L'objet doc est une copie de test.xml
doc.Load(Server.MapPath("test.xml"));
XmlNode root = doc.DocumentElement; // Récupère le noeud racine
// On crée un nouvel élément elem
XmlElement elem = doc.CreateElement("NewTable");
// On crée un nouvel élément underElem
XmlElement underElem = doc.CreateElement("AnotherOne");
underElem.InnerText = "mourek"; // On crée une valeur texte pour underElem
// On définit underElem comme étant un élément fils de elem
elem.AppendChild(underElem);
// On remplace le premier enfant du noeud racine par le noeud elem
root.ReplaceChild(elem, root.FirstChild);
doc.Save(Server.MapPath("test.xml")); // On met à jour le fichier test.xml

// On récupère en lecture dans la DataSet le fichier test.xml
ds.ReadXml(Server.MapPath("test.xml"));
// On récupère dans la DataTable le noeud "NewTable"
dt = ds.Tables["NewTable"];
// On récupère dans la DataTable le noeud "mySecondTable"
dt2 = ds.Tables["mySecondTable"];

// On affiche les 2 tables contenues dans des DataTable dans 2 GridView
GridView myGridView = new GridView();
myGridView.DataSource = dt;
myGridView.DataBind();

GridView myGridView2 = new GridView();
myGridView2.DataSource = dt2;
myGridView2.DataBind();

form1.Controls.Add(myGridView);
form1.Controls.Add(myGridView2);
```

VB.NET - Code adapté pour une solution Web Application

```

Dim ds As DataSet = New DataSet("Test") ' Créer un dataset
Dim dt As DataTable = New DataTable("myTable") ' Créer un datable

dt.Columns.Add("col1") ' Ajoute une colonne dans la table
dt.Columns.Add("col2") ' Une deuxième colonne
' Ajoute une ligne = des éléments dans les colonnes
dt.Rows.Add("Dans col1", "Dans col2")
' Ajoute la table dans le dataset
ds.Tables.Add(dt)

' On crée une seconde DataTable
Dim dt2 As DataTable = New DataTable("mySecondTable")
dt2.Columns.Add("colA") ' Première colonne
dt2.Columns.Add("colB") ' Deuxième colonne
dt2.Rows.Add("ABC", "DEF") ' Insertion d'éléments dans les colonnes

ds.Tables.Add(dt2) ' Ajout de la seconde table dans le DataSet

ds.WriteXml(Server.MapPath("test.xml")) ' On met crée le document test.xml

ds.Dispose() ' On libère les ressources du DataSet
ds.Reset() ' On nettoie le DataSet
dt.Dispose() ' On libère les ressources du DataTable
dt.Reset() ' On nettoie le DataTable

Dim doc As XmlDocument = New XmlDocument() ' On crée un objet XML nommé doc

' L'objet doc est une copie de test.xml
doc.Load(Server.MapPath("test.xml"))
Dim root As XmlNode = doc.DocumentElement ' Récupère le noeud racine
' On crée un nouvel élément elem
Dim elem As XmlElement = doc.CreateElement("NewTable")
' On crée un nouvel élément underElem
Dim underElem As XmlElement = doc.CreateElement("AnotherOne")
underElem.InnerText = "mourek" ' On crée une valeur texte pour underElem
' On définit underElem comme étant un élément fils de elem
elem.AppendChild(underElem)
' On remplace le premier enfant du noeud racine par le noeud elem
root.ReplaceChild(elem, root.FirstChild)
doc.Save(Server.MapPath("test.xml")) ' On met à jour le fichier test.xml

' On récupère en lecture dans la DataSet le fichier test.xml
ds.ReadXml(Server.MapPath("test.xml"))
' On récupère dans la DataTable le noeud "NewTable"
dt = ds.Tables("NewTable")
' On récupère dans la DataTable le noeud "mySecondTable"
dt2 = ds.Tables("mySecondTable")

' On affiche les 2 tables contenues dans des DataTable dans 2 GridView
Dim myGridView As GridView = New GridView()
myGridView.DataSource = dt

myGridView.DataBind()

Dim myGridView2 As GridView = New GridView()
myGridView2.DataSource = dt2
myGridView2.DataBind()

form1.Controls.Add(myGridView)
form1.Controls.Add(myGridView2)

```

Aperçu de la page web générée :

AnotherOne	
mourek	
colA	colB
ABC	DEF

3.7 Suppression

Voici les méthodes permettant de supprimer un nœud ainsi qu'un exemple :

Méthodes	Description
RemoveAll	Supprime tous les nœuds enfants du nœud courant.
RemoveChild	Supprimer le nœud enfant spécifié du nœud courant.

C#

```
XmlDocument doc = new XmlDocument();
doc.Load(@"Fruit.xml");

doc.DocumentElement.RemoveAll();
doc.Save("test.xml");
```

VB.NET

```
Dim doc As XmlDocument = New XmlDocument()
doc.Load("C:\MyProject\ConsoleApplication7\ConsoleApplication7\Xml.xml")

doc.DocumentElement.RemoveAll()
doc.Save("test.xml")
```

On charge dans un XmlDocument le document Xml contenant la liste de fruit. Ensuite on se place au root et on utilise la méthode RemoveAll qui permet de supprimer tous les nœuds enfants. L'image de droite montre le resultat se trouvant dans le fichier test.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<ListeFruit>
</ListeFruit>
```

4 Avec un DataSet

Le DataSet est un objet représentant un schéma d'une base de données entière ou d'une partie de celle-ci. A l'image des bases de données relationnelles, il peut contenir des tables et des relations entre ces tables (voir Chapitre Base de données ADO.NET).

Les DataSet peuvent être sérialisé en XML, et inversement des fichiers XML peuvent être récupérés dans un DataSet via les méthodes *WriteXml()* pour la sérialisation et *ReadXml()* pour la récupération.

```
C#  
  
DataSet ds = new DataSet(); // On crée un dataset  
DataTable dt = new DataTable("myTable"); // On crée un datatable  
  
dt.Columns.Add("col1"); // On ajoute une colonne dans la table  
dt.Columns.Add("col2"); // Une deuxième colonne  
// Ajoute une ligne = des éléments dans les colonnes  
dt.Rows.Add("Dans col1", "Dans col2");  
// Ajoute la table dans le dataset  
ds.Tables.Add(dt);  
  
// On sérialise notre DataSet dans un fichier XML  
// La méthode Server.MapPath() permet de récupérer le chemin racine de votre  
application  
ds.WriteXml(Server.MapPath("test.xml"));  
ds.Dispose(); // On libère les ressources  
ds.Reset(); // Vide l'objet et le remet à son état initial  
  
// On récupère dans un DataSet notre XML que l'on affiche dans une GridView il est à  
noter qu'ici on utilise le même DataSet. Il est bien entendu possible d'en créer un  
autre.  
ds.ReadXml(Server.MapPath("test.xml"));  
  
GridView myGridView = new GridView(); // On crée un nouvelle GridView  
// On lui indique la source de données ici notre DataSet  
myGridView.DataSource = ds;  
// On lie notre DataSet à la gridView  
myGridView.DataBind();  
  
// On ajoute le GridView aux éléments de notre page  
form1.Controls.Add(myGridView);
```

VB.NET

```

Dim ds As DataSet = New DataSet() ' On crée un dataset
Dim dt As DataTable = New DataTable("myTable") ' On crée un datable

dt.Columns.Add("col1") ' On ajoute une colonne dans la table
dt.Columns.Add("col2") ' Une deuxième colonne
' Ajoute une ligne = des éléments dans les colonnes
dt.Rows.Add("Dans col1", "Dans col2")
' Ajoute la table dans le dataset
ds.Tables.Add(dt)

' On sérialise notre DataSet dans un fichier XML
' La méthode Server.MapPath() permet de récupérer le chemin racine de votre
application
ds.WriteXml(Server.MapPath("test.xml"))
ds.Dispose() ' On libère les ressources
ds.Reset() ' Vide l'objet et le remet à son état initial

' On récupère dans un DataSet notre XML que l'on affiche dans une GridView il est à
noter qu'ici on utilise le même DataSet. Il est bien entendu possible d'en créer un
autre.
ds.ReadXml(Server.MapPath("test.xml"))

Dim myGridView As GridView = New GridView() ' On crée un nouvelle GridView
' On lui indique la source de données ici notre DataSet
myGridView.DataSource = ds
' On lie notre DataSet à la gridView
myGridView.DataBind()

' On ajoute le GridView aux éléments de notre page
form1.Controls.Add(myGridView)

```

L'exemple ci-dessus permet de sérialiser un DataSet dans un fichier XML. Une fois les ressources libérées et nettoyé notre DataSet *ds* retourne *NULL* (toutes les valeurs contenues dans le DataSet sont effacées). On peut désormais le réutiliser pour récupérer les valeurs contenue dans *test.xml* fraîchement créé.

Dans le cas où on aurait voulu conserver notre DataSet *ds*, il est tout à fait faisable de créer un autre DataSet afin d'y stocker les données de *test.xml*. Tout dépendra du rôle du DataSet dans notre application.

5 Conclusion

Pour conclure cette partie vous devez savoir manipuler un document Xml. C'est-à-dire savoir récupérer les parties qui sont intéressantes, que ce soit pour de l'affichage ou non, grâce aux méthodes comme *DocumentElement* qui récupère le nœud racine, *ChildNodes* qui récupère une liste des nœuds enfants ... Vous devez aussi savoir comment on modifie les données récupérés, comment on les supprime et comment on les remplace (notamment avec *ReplaceChild*).

Dans le but de bien réussir cela vous devez connaître chaque partie de la constitution d'un document Xml. Entre autre ce sont les *XmlElement*, *XmlNode*, *XmlDocument*, *XmlAttribute*, *XmlComment*

Une autre chose à savoir est le passage du Xml au DataSet et inversement grâce à *ReadXml* et *WriteXml*.