Manipuler un document XML

Le framework .NET expose deux API ayant pour objectif la manipulation de documents XML.

Le **modèle DOM** existe depuis les débuts du framework .NET et est basé sur les recommandations du W3C. Chaque balise, attribut ou contenu textuel formant le document XML est ainsi considéré comme un nœud d'une structure hiérarchique. La recherche d'un nœud avec cette API peut être manuelle ou utiliser la navigation **XPath**. Les fonctionnalités du modèle DOM sont implémentées dans les types de l'espace de noms `System.Xml`.

L'apparition de la version 3.5 du framework .NET a introduit **LINQ to XML**, qui utilise pour sa part une approche plus moderne. Les balises, attributs ou espaces de noms sont tous considérés comme des éléments qui peuvent exister indépendamment, ce qui permet de traiter aisément des fragments de code XML. Ces éléments peuvent être interrogés de la même manière qu'une base de données à l'aide de requêtes LINQ. Les types associés à cette API sont implémentés dans l'espace de noms `System.Xml.Linq`.

Les exemples exposés dans cette section sont réalisés à partir du fichier `commandes.xml` dont le contenu est le suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<Clients web="http://www.masociete.com/clients/web">
  <web:Client Id="ALFKI">
    <Societe>Alfreds Futterkiste</Societe>
    <NomContact>Alfreds Maria Anders</NomContact>
    <Adresse>Obere Str. 57</Adresse>
    <CodePostal>12209</CodePostal>
    <Ville>Berlin</Ville>
    <DerniereCommande Date="09/04/1998">
      <Produits>
        <Produit>
          <Nom>Escargots de Bourgogne</Nom>
          <PrixUnitaire>13,25</PrixUnitaire>
          <Quantite>40</Quantite>
        </Produit>
        <Produit>
          <Nom>Floremysost</Nom>
          <PrixUnitaire>21,50</PrixUnitaire>
          <Quantite>20</Quantite>
        </Produit>
      </Produits>
    </DerniereCommande>
  </web:Client>
</Clients>
```

Ce fichier est disponible en téléchargement depuis la page Informations générales.

1. Utilisation de DOM

La première étape dans l'utilisation de DOM consiste à charger un document XML en mémoire. Il est représenté par un objet de type `XmlDocument` contenant l'ensemble de l'arborescence du document.

La lecture et le chargement d'un fichier XML sont effectués à travers l'utilisation des méthodes `Load` ou `LoadXml` du type `XmlDocument`.

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

```
//La chaîne de caractères doit contenir une structure de document XML
string xml = "...";
doc.LoadXml(xml);
```

Il n'est toutefois pas obligatoire de créer cet objet en mémoire à partir d'un fichier. L'API DOM définit plusieurs classes et méthodes permettant la génération d'un document XML à l'aide de code C#. Le code suivant crée un objet `XmlDocument` contenant la même arborescence que le fichier `commandes.xml`.

```
XmlDocument doc = new XmlDocument();

//Création du noeud de déclaration XML
XmlDeclaration declarationXml = doc.CreateXmlDeclaration("1.0",
"utf-8", null);
doc.AppendChild(declarationXml);

//Création du noeud Clients
XmlElement racine = doc.CreateElement("Clients");
//Création de l'attribut qui définit le namespace web
racine.SetAttribute("xmlns:web",
"http://www.masociete.com/clients/web");
doc.AppendChild(racine);

//Création du noeud Client et indication de l'espace de noms associé
XmlElement client1 = doc.CreateElement("Client",
"http://www.masociete.com/clients/web");
client1.Prefix = "web";
racine.AppendChild(client1);
//Création de l'attribut Id du noeud Client
client1.SetAttribute("Id", "ALFKI");

//Création du noeud Societe
XmlElement societe = doc.CreateElement("Societe");
//Création du noeud texte contenant la valeur de Societe
XmlText valeurSociete = doc.CreateTextNode("Alfreds
Futterkiste");
societe.AppendChild(valeurSociete);
client1.AppendChild(societe);

//Création du noeud NomContact
XmlElement nomContact = doc.CreateElement("NomContact");
//Création du noeud texte contenant la valeur de NomContact
XmlText valeurNomContact = doc.CreateTextNode("Alfreds Maria
Anders");
nomContact.AppendChild(valeurNomContact);
client1.AppendChild(nomContact);
```

```
XmlElement adresse = doc.CreateElement("Adresse");
XmlText valeurAdresse = doc.CreateTextNode("Obere Str. 57");
adresse.AppendChild(valeurAdresse);
client1.AppendChild(adresse);
XmlElement codePostal = doc.CreateElement("CodePostal");
XmlText valeurCodePostal = doc.CreateTextNode("12209");
codePostal.AppendChild(valeurCodePostal);
client1.AppendChild(codePostal);
XmlElement ville = doc.CreateElement("Ville");
XmlText valeurVille = doc.CreateTextNode("Berlin");
ville.AppendChild(valeurVille);
client1.AppendChild(ville);

//Création du noeud DerniereCommande
XmlElement derniereCommande =
doc.CreateElement("DerniereCommande");
client1.AppendChild(derniereCommande);
//Création de l'attribut Date du noeud DerniereCommande
derniereCommande.SetAttribute("Date", "09/04/1998");

//Création du noeud Produits
XmlElement produits = doc.CreateElement("Produits");
derniereCommande.AppendChild(produits);
//Création du premier noeud Produit
XmlElement produit1 = doc.CreateElement("Produit");
produits.AppendChild(produit1);
//Création du noeud Nom
XmlElement nom1 = doc.CreateElement("Nom");
XmlText valeurNom1 = doc.CreateTextNode("Escargots de Bourgogne");
nom1.AppendChild(valeurNom1);
produit1.AppendChild(nom1);
//Création du noeud PrixUnitaire
XmlElement prixUnitaire1 = doc.CreateElement("PrixUnitaire");
XmlText valeurPrixUnitaire1 = doc.CreateTextNode("13,25");
prixUnitaire1.AppendChild(valeurPrixUnitaire1);
produit1.AppendChild(prixUnitaire1);
//Création du noeud Quantite
XmlElement quantite1 = doc.CreateElement("Quantite");
XmlText valeurQuantite1 = doc.CreateTextNode("40");
quantite1.AppendChild(valeurQuantite1);
produit1.AppendChild(quantite1);

//Création du second noeud Produit
XmlElement produit2 = doc.CreateElement("Produit");
produits.AppendChild(produit2);
//Création du noeud Nom
XmlElement nom2 = doc.CreateElement("Nom");
XmlText valeurNom2 = doc.CreateTextNode("Floremysost");
nom2.AppendChild(valeurNom2);
produit2.AppendChild(nom2);
//Création du noeud PrixUnitaire
XmlElement prixUnitaire2 = doc.CreateElement("PrixUnitaire");
XmlText valeurPrixUnitaire2 = doc.CreateTextNode("21,50");
prixUnitaire2.AppendChild(valeurPrixUnitaire2);
```

```
produit2.AppendChild(prixUnitaire2);
//Création du noeud Quantite
XmlElement quantite2 = doc.CreateElement("Quantite");
XmlText valeurQuantite2 = doc.CreateTextNode("20");
quantite2.AppendChild(valeurQuantite2);
produit2.AppendChild(quantite2);
```

Analysons les différents éléments utilisés dans cet exemple.

`XmlElement` représente une balise XML pour laquelle il est nécessaire de fournir un nom. Le moyen le plus simple pour créer un objet de ce type est d'appeler la méthode `CreateElement` de l'objet `XmlDocument` contenant l'arborescence XML. La propriété `Prefix` (utilisée ici sur `client1`) du type `XmlElement` permet de définir le préfixe d'espace de noms à utiliser pour l'élément.

La création d'attributs pour un objet `XmlElement` est effectuée en ajoutant des objets de type `XmlAttribute` à sa collection `Attributes`. Il est toutefois généralement plus approprié d'utiliser sa méthode `SetAttribute`, plus simple et plus lisible. Celle-ci accepte deux paramètres représentant respectivement le nom de l'attribut ainsi que sa valeur. C'est d'ailleurs en utilisant cette dernière méthode sur l'élément racine que l'espace de noms `web` est ajouté au document.

L'imbrication des différents constituants du document résulte de l'utilisation de la méthode `AppendChild`. Celle-ci est utilisée sur le nœud destiné à être le parent du nœud passé en paramètre. Cette méthode est définie sur la classe `XmlNode`, de manière à ce que tous ses types dérivés puissent l'utiliser. On trouve parmi eux les types `XmlDocument`, `XmlElement` et `XmlAttribute`.

Le type `XmlText` est quant à lui légèrement différent puisqu'il ne peut pas contenir d'autre élément. Il est le véritable conteneur de données, quand les autres ne permettent de définir que la structure du document. Les objets de ce type sont créés par un appel à la méthode `CreateTextNode` de l'objet `XmlDocument`. Le paramètre passé à cette fonction est la donnée qui doit être écrite.

Une fois ce document chargé ou généré, il est possible de rechercher certains éléments par leur nom ou leur identifiant.

La méthode `GetElementsByTagName` de la classe `XmlNode` est utilisée pour retrouver tous les nœuds XML dont le nom correspond à la valeur passée en paramètre. Elle effectue la recherche dans toute l'arborescence enfant d'un élément. La liste de nœud retournée peut donc contenir des éléments de tout niveau : nœuds enfants, petits-enfants ou plus bas encore.

L'exemple de code suivant affiche à l'écran le prix unitaire de chacun des produits commandés.

```
XmlNodeList prixUnitaires =
doc.GetElementsByTagName("PrixUnitaire");
foreach (XmlNode noeud in prixUnitaires)
{
   Console.WriteLine(noeud.InnerText);
}
```

Lorsque le document contient une DTD, il est possible de lister les éléments correspondant à un identifiant à l'aide de la méthode `GetElementById`. La DTD doit pour cela définir un ou plusieurs attributs de type `ID`.

L'insertion de la DTD suivante entre la déclaration XML et le nœud racine du document définit l'attribut `Id` des éléments de type `web:Client` comme étant l'identifiant pour la balise.

```
<!DOCTYPE Clients [
  <!ELEMENT Clients ANY>
  <!ELEMENT web:Client ANY>
  <!ATTLIST web:Client Id ID #REQUIRED>
]>
```

Une fois cette définition ajoutée, il est possible de rechercher et d'afficher le nœud dont l'identifiant est ALFKI.

```
XmlNode clientALFKI = doc.GetElementById("ALFKI");
Console.WriteLine(clientALFKI.OuterXml);
```

Une fois un nœud sélectionné, il est aisé de modifier un de ses attributs ou de lui ajouter un ou plusieurs nœuds enfants.

```
//Modification de la date et ajout d'un noeud Commercial
//pour chacun des noeuds DerniereCommande
XmlNodeList commandes =
doc.GetElementsByTagName("DerniereCommande");
foreach (XmlElement noeud in commandes)
{
    noeud.Attributes["Date"].Value = "24/10/2001";

    XmlElement commercial = doc.CreateElement("Commercial");
    XmlText valeurCommercial = doc.CreateTextNode("Steven
Buchanan");
    commercial.AppendChild(valeurCommercial);
    noeud.AppendChild(commercial);
}
```

L'enregistrement du document avec les modifications qui lui ont été apportées doit être effectué en appelant la méthode `Save` de l'objet `XmlDocument`. Sans cette opération, toute modification peut être perdue, puisque le document est traité uniquement en mémoire.

```
doc.Save("commandesModifiees.xml");
```

2. Utilisation de XPath

XPath a pour objectif la localisation d'un ou plusieurs éléments de la même manière que l'explorateur Windows permet d'adresser un fichier particulier. L'implémentation de la technologie XPath dans le framework .NET est située dans l'espace de noms `System.Xml.XPath`.

La première étape dans l'utilisation de XPath correspond à l'instanciation d'un objet de type `XPathNavigator`. C'est lui qui fournit la capacité d'adresser les éléments d'un document XML.

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

```
XPathNavigator navigateur = doc.CreateNavigator();
```

Une des méthodes de cet objet, `Select`, offre la possibilité de lister les éléments correspondant au chemin XPath qui lui est fourni en paramètre.

```
XPathNavigator navigateur = doc.CreateNavigator();

XPathNodeIterator produits =
navigateur.Select("/Clients/Client/DerniereCommande/Produits/
Produit");

while (produits.MoveNext())
{
   Console.WriteLine(produits.Current. OuterXml);
}
```

Dans notre cas, ce code n'affiche strictement rien à l'écran, et c'est bien normal : les nœuds `Client` et `web:Client` sont différents. Il est donc nécessaire d'ajouter quelques éléments pour que l'objet `XPathNavigator` puisse retourner les éléments souhaités.

Le chemin passé en paramètre à la méthode Select doit contenir le nom pleinement qualifié de chacun des types de nœuds. Dans le cas présent, cela signifie qu'il faut utiliser `web:Client` au lieu de `Client`.

Pour que le navigateur XPath soit en mesure de comprendre et trouver ce nom qualifié, il est nécessaire de lui fournir un objet de type `XmlNamespaceManager` auquel aura été ajoutée la définition de l'espace de noms `web`.

Le code fonctionnel est le suivant :

```
XPathNavigator navigateur = doc.CreateNavigator();
XmlNamespaceManager manager = new
XmlNamespaceManager(navigateur.NameTable);
manager.AddNamespace("web",
"http://www.masociete.com/clients/web");

XPathNodeIterator produits =
navigateur.Select("/Clients/web:Client/DerniereCommande/
Produits/Produit", manager);

while (produits.MoveNext())
{
    Console.WriteLine(produits.Current.OuterXml);
}
```

La modification de valeurs est à peine plus compliquée puisqu'il suffit d'appliquer la méthode `SetValue` à un élément retourné par la fonction `Select`. Une augmentation arbitraire de 10 % sur le tarif unitaire de chaque produit commandé est ainsi réalisée de la manière suivante.

```
XPathNodeIterator prixUnitaires =
```

```
navigateur.Select("/Clients/web:Client/DerniereCommande/Produits/
Produit/PrixUnitaire", manager);

while (prixUnitaires.MoveNext())
{
    double valeur = double.Parse(prixUnitaires.Current.Value);
    valeur = valeur * 1.10;
    prixUnitaires.Current.SetValue(valeur.ToString());
}
```

Après exécution de cette portion de code, chacun des nœuds `PrixUnitaire` du document XML a bien vu sa valeur changer.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Clients[
  <!ELEMENT Clients ANY>
  <!ELEMENT web:Client ANY>
  <!ATTLIST web:Client Id ID #REQUIRED>
]>
<Clients xmlns:web="http://www.masociete.com/clients/web">
  <web:Client Id="ALFKI">
    <Societe>Alfreds Futterkiste</Societe>
    <NomContact>Alfreds Maria Anders</NomContact>
    <Adresse>Obere Str. 57</Adresse>
    <CodePostal>12209</CodePostal>
    <Ville>Berlin</Ville>
    <DerniereCommande Date="09/04/1998">
      <Produits>
        <Produit>
          <Nom>Escargots de Bourgogne</Nom>
          <PrixUnitaire>14,575</PrixUnitaire>
          <Quantite>40</Quantite>
        </Produit>
        <Produit>
          <Nom>Floremysost</Nom>
          <PrixUnitaire>23,65</PrixUnitaire>
          <Quantite>20</Quantite>
        </Produit>
      </Produits>
    </DerniereCommande>
  </web:Client>
</Clients>
```

3. Utilisation de LINQ to XML

L'arrivée de LINQ dans le framework .NET a apporté de nombreuses innovations dans la manipulation des données issues de collections locales ou de bases de données. Avec le fournisseur LINQ pour SQL Server, Microsoft a également livré un fournisseur pour les documents XML, permettant ainsi de manipuler plus simplement les données enregistrées dans ce format.

Le fournisseur LINQ to XML contient de nouveaux types simplifiant la création de documents XML et intégrant

également les fonctionnalités nécessaires au requêtage des données qu'ils contiennent. Ils sont placés dans l'espace de noms `System.Xml.Linq`.

La première étape dans l'utilisation de LINQ to XML consiste à charger un document XML en mémoire. Cette opération est effectuée en utilisant la méthode statique `Load` de la classe `XDocument`.

```
XDocument doc = XDocument.Load("commandes.xml");
```

Comme avec l'API DOM, il est possible de créer une arborescence XML directement en mémoire. Le code C# suivant instancie et valorise un objet `Xdocument`.

```
XNamespace webNamespace = "http://www.masociete.com/clients/web";
XDocument doc =
    new XDocument(
        new XDeclaration("1.0", "utf-8", null),
        new XElement("Clients",
            new XAttribute(XNamespace.Xmlns + "web", webNamespace),
            new XElement(webNamespace + "Client",
                new XAttribute("Id", "ALFKI"),
                new XElement("Societe", "Alfreds Futterkiste"),
                new XElement("NomContact", "Alfreds Maria Anders"),
                    new XElement("Adresse", "Obere Str. 57"),
                    new XElement("CodePostal", "12209"),
                    new XElement("Ville", "Berlin"),
                    new XElement("DerniereCommande",
                        new XAttribute("Date", "09/04/1998"),
                        new XElement("Produits",
                            new XElement("Produit",
                                XElement("Nom", "Escargots de Bourgogne"),
                                new XElement("PrixUnitaire", "13,25"),
                                new XElement("Quantite", "40")),
                            new XElement("Produit",
                                new XElement("Nom", "Floremysost"),
                                new XElement("PrixUnitaire", "21,50"),
                                new XElement("Quantite", "20")))))));
```

Le code LINQ est **quatre fois plus court** que le code DOM générant la même arborescence, tout en gardant une **expressivité** que l'API DOM est incapable d'égaler. Un autre point positif concerne le **nombre de variables** nécessaires à la création de ce document : deux avec LINQ, contre une trentaine précédemment !

Un document XML est représenté avec LINQ par un objet dont le type est `XDocument`. Il définit un constructeur pouvant accepter un nombre variable de paramètres, ce qui permet de lui ajouter une arborescence enfant entière dès sa création. Le principe est le même pour le type `XElement`, qui représente quant à lui un nœud de l'arborescence. Ce mode de fonctionnement permet de chaîner les créations d'objets, ce qui a pour effet de rendre le code plus concis et plus expressif.

La lecture d'informations dans cette arborescence est elle aussi simplifiée. L'écriture de requêtes LINQ pour la recherche tire parti de certaines méthodes présentes dans la classe `XElement` :

- `Descendants` renvoie la liste des nœuds de l'arborescence enfant du nœud courant.
- `Elements` renvoie uniquement les enfants directs du nœud courant.

- `AncestorsAndSelf` renvoie la liste des nœuds trouvés en remontant du nœud courant jusqu'au nœud racine. Le résultat inclut le nœud courant.
- `Attributes` liste les attributs du nœud courant.
- `DescendantsAndSelf` renvoie la liste des nœuds de l'arborescence enfant du nœud courant et inclut ce dernier dans le résultat.

Le code ci-dessous effectue la recherche des nœuds dont le nom est `PrixUnitaire`.

```
var prixUnitaires =
    from noeud in doc.Descendants("PrixUnitaire")
    select noeud;

foreach (XElement noeud in prixUnitaires)
{
    Console.WriteLine(noeud.Value);
}
```

Il est également possible de rechercher les éléments dont un attribut a une valeur particulière. L'exemple suivant montre comment rechercher le premier nœud dont l'attribut `Id` a pour valeur `"ALFKI"`.

```
var clientALFKI =
    (from noeudClient in doc.Descendants(webNamespace + "Client")
     where noeudClient.Attribute("Id").Value == "ALFKI"
     select noeudClient).FirstOrDefault();

Console.WriteLine(clientALFKI);
```

La modification d'une valeur d'attribut ou d'élément peut être effectuée en affectant une nouvelle valeur à la propriété `Value` de l'élément. Le code suivant modifie les dates de commandes et les prix unitaires enregistrés dans le document.

```
var noeudsCommandeOuPrix =
    from noeud in doc.Descendants()
    where noeud.Name == "DerniereCommande"
       || noeud.Name == "PrixUnitaire"
    select noeud;

foreach (XElement noeud in noeudsCommandeOuPrix)
{
    if (noeud.Name == "DerniereCommande")
    {
        noeud.Attribute("Date").Value = "28/05/2014";
    }
    if (noeud.Name == "PrixUnitaire")
    {
        var nouvelleValeur = double.Parse(noeud.Value) * 1.10;
        noeud.Value = nouvelleValeur.ToString();
    }
}
```

La sauvegarde des modifications est effectuée comme avec l'API DOM, avec l'appel de la méthode `Save` sur l'objet `XDocument`.

```
doc.Save("commandes modifiees avec Linq.xml");
```

Le contenu de ce fichier est le suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<Clients xmlns:web="http://www.masociete.com/clients/web">
  <web:Client Id="ALFKI">
    <Societe>Alfreds Futterkiste</Societe>
    <NomContact>Alfreds Maria Anders</NomContact>
    <Adresse>Obere Str. 57</Adresse>
    <CodePostal>12209</CodePostal>
    <Ville>Berlin</Ville>
    <DerniereCommande Date="28/05/2014">
      <Produits>
        <Produit>
          <Nom>Escargots de Bourgogne</Nom>
          <PrixUnitaire>14,575</PrixUnitaire>
          <Quantite>40</Quantite>
        </Produit>
        <Produit>
          <Nom>Floremysost</Nom>
          <PrixUnitaire>23,65</PrixUnitaire>
          <Quantite>20</Quantite>
        </Produit>
      </Produits>
    </DerniereCommande>
  </web:Client>
</Clients>
```