

Lecture et écriture dans un fichier XML avec XPath

par Jean-François Determe

Date de publication : 26/08/2007

Dernière mise à jour : 26/08/2007

Cet article traite de la façon de lire et de modifier un document XML avec la technologie XPath.

- 1 - Prérequis
- 2 - De quoi traite ce tutoriel?
- 3 - Présentation du projet
 - 3.1 - Fonctionnalités
 - 3.2 - Techniquement parlant
- 4 - Notions XML de base
 - 4.1 - Noeuds et balises
 - 4.2 - Racine
 - 4.3 - Noeuds enfant et noeuds parent (ou ancêtre)
 - 4.4 - Attributs
- 5 - Syntaxe d'XPath
 - 5.1 - Contexte
 - 5.2 - Sélection noeuds/attributs
 - 5.3 - Conditions
- 6 - Rechercher des données avec XPath
 - 6.1 - Structure du fichier XML
 - 6.2 - Rechercher un noeud à partir de ses attributs
 - 6.3 - Rechercher un noeud à partir d'un noeud enfant
- 7 - Modification de données XML
 - 7.1 - Edition de données XML
 - 7.2 - Insertion de nouveaux noeuds/attributs dans un document XML
 - 7.3 - Suppression d'un noeud et de ses noeuds-enfants.
- 8 - Améliorations possibles du programme-exemple
- 9 - Conclusion
- 10 - Remerciements

1 - Prérequis

- Connaissance du langage C#.
- Les sources du programme-exemple téléchargeables ici : [Source Ici](#).

2 - De quoi traite ce tutoriel?

Ce tutoriel montre comment rechercher, insérer, modifier et supprimer des données XML à l'aide de la technologie XPath. Pour plus d'informations : **Définition XPath**. XPath dispose des avantages suivants : - Les performances: XPath permet de traiter rapidement des documents XML.

- La clarté et la concision du code: XPath permet de coder une même opération en moins de lignes que les autres méthodes. La première partie porte sur l'extraction de données contenues dans un fichier XML.

La seconde partie traite de l'édition d'un fichier XML.

Ce tutoriel se base sur un projet concret.

Les schémas XSD ne sont pas utilisés ici afin de ne pas rendre trop complexe ce tutoriel.

Cependant, la recherche et l'édition des données dans un fichier XML lié à un schéma XSD ne sont pas des techniques difficiles à apprendre une fois que ce qui est présenté dans ce tutoriel est acquis.

3 - Présentation du projet

3.1 - Fonctionnalités

Le projet présenté est un gestionnaire de clients élémentaire.

Chaque enregistrement du fichier XML "Client.xml" contient les champs suivants: ID, Prénom, Nom, Téléphone, DateInscription et Status.

Le programme proposé effectue les actions suivantes:

- Recherche des données concernant un client à partir de son ID, nom, prénom, téléphone ou date d'enregistrement.
- Insertion, édition et suppression des données concernant un client.

3.2 - Techniquement parlant

Ce projet est réparti en trois couches, une qui gère toute l'interface (Interface), une autre qui contient toutes les classes permettant d'accéder aux données (DataLayer) et une dernière qui renferme les classes qui contiennent les données sur un client (StructureLibrary).

L'interface est composée de deux fiches.

4 - Notions XML de base

Cette section décrit les notions XML à connaître pour comprendre la suite de ce tutoriel.

Le fichier XML suivant est utilisé pour expliquer les notions qui suivent:

```
<?xml version="1.0" encoding="utf-8"?>
<Clients>
  <Client id="1">
    <Prenom>Jean-Michel</Prenom>
    <Nom>Laroche</Nom>
    <Telephone>027854632</Telephone>
    <DateInscription>02/06/2007</DateInscription>
    <Status>Activated</Status>
  </Client>
  <Client id="2">
    <Prenom>Smith</Prenom>
    <Nom>Cordwainer</Nom>
    <Telephone>024213296</Telephone>
    <DateInscription>12/06/2007</DateInscription>
    <Status>Activated</Status>
  </Client>
  <Client id="3">
    <Prenom>Frank</Prenom>
    <Nom>Herbert</Nom>
    <Telephone>022354562</Telephone>
    <DateInscription>08/07/2007</DateInscription>
    <Status>Activated</Status>
  </Client>
  <Client id="4">
    <Prenom>Phillipe</Prenom>
    <Nom>Dick</Nom>
    <Telephone>023254789</Telephone>
    <DateInscription>12/07/2007</DateInscription>
    <Status>Deleted</Status>
  </Client>
  <MaxID>4</MaxID>
</Clients>
```

4.1 - Noeuds et balises

Le noeud (node) est l'unité de base d'un fichier XML.

Un noeud est un champ délimité par des balises pouvant contenir une valeur, des attributs et des sous-noeuds (noeuds-enfants).

Dans le fichier XML ci-dessus, **<Clients>...</Clients>**, **<Client id="1">...</Client>**, **<Prenom>...</Prenom>**,... sont des noeuds.

La déclaration d'un noeud ayant comme nom "Telephone" et ayant comme valeur "99999" est celle-ci: **<Telephone>99999</Telephone>** .

Tout élément de la forme **<UnNom>** est appelé **balise** (markup).

Un noeud doit être délimité par deux balises de cette façon: **<NomDuNoeud>Valeur Du Noeud</NomDuNoeud>** .

Il n'est pas obligatoire de donner une valeur à un noeud (noeud vide).


La valeur (textuelle) d'un noeud est le texte contenu entre les deux balises qui délimitent ce noeud et les éventuelles valeurs textuelles de ses noeuds-enfants.

Déclarer plusieurs noeuds ayant le même nom est permis.

4.2 - Racine

La racine d'un document XML ("root" en anglais) est un noeud fictif qui n'est pas déclaré dans le document.

Ce dernier est le noeud avec le niveau hiérarchique le plus élevé.

 *L'élément suivant **<?xml version="1.0" encoding="utf-8"?>** n'est pas considéré comme un noeud.*

C'est en fait une déclaration qui permet de définir la version d'un fichier XML et son encodage (utf-8, ISO,...).

4.3 - Noeuds enfant et noeuds parent (ou ancêtre)

Le noeud "y" est un noeud enfant (child-node) du noeud "x" si le noeud "y" est contenu dans le noeud "x".

Dans notre fichier-exemple, le noeud **<Prenom>Smith</Prenom>** est un noeud enfant du noeud **<Client id="2">...</Client>** .

Le noeud "y" est le noeud parent (ou l'ancêtre, ancestor/parent-node) du noeud "x" si le noeud "y" contient le noeud "x".

Dans l'exemple précédent, Le noeud **<Client id="2">...</Client>** est l'ancêtre du noeud **<Prenom>Smith</Prenom>** .

4.4 - Attributs

L'attribut (attribute) d'un noeud est un noeud déclaré à l'intérieur de la première balise qui délimite ce noeud.

Dans le fichier-exemple, chaque noeud Client possède un attribut "id".

5 - Syntaxe d'XPath

5.1 - Contexte

Le contexte est le référentiel sur lequel la recherche se base.

Pendant la navigation dans l'arborescence d'un fichier XML, l'objet permettant la navigation est toujours positionné sur un noeud ; ce noeud est appelé contexte.

Une expression XPath précédée de "./" se base sur le contexte actuel.

Ex: "./author" sélectionne tous les noeuds enfants de nom "author" en partant du contexte actuel.

Une expression XPath utilisant comme contexte la racine du document XML , est précédée par "/".

Ex: "/book" sélectionne tous les noeuds enfants nommés "book" en utilisant comme contexte la racine du document.

Si la recherche porte sur tous les éléments ayant un nom donné sans tenir compte de leur niveau hiérarchique, l'expression XPath est précédée par "//".

Ex: "//Friend" sélectionne tous les noeuds du documents ayant comme nom "Friend". Si la recherche doit porter sur tous les noeuds d'un nom donné mais doit aussi se baser sur le contexte actuel, il convient de faire précéder l'expression XPath par "./".

Ex: "//Prix" sélectionne TOUS les noeuds qui ont ce nom dans le document alors que "./Prix" ne sélectionne que les noeuds-enfants du contexte actuel ayant pour nom "Prix".

5.2 - Sélection noeuds/attributs

Pour sélectionner tous les noeuds-enfants du contexte actuel, il convient d'utiliser le symbole "*" qui signifie "Tous les noeuds (de type element ou attribut (si précédé de "@"))".

Si la sélection porte sur tous les types de noeuds existants, il faut utiliser node() et non * qui se limite aux noeuds de type éléments et attributs.

Ex: "./*" sélectionne tous les noeuds-enfants de type élément du contexte actuel.

Pour sélectionner un ou plusieurs attributs, il faut utiliser le symbole "@".

Il précède le nom de l'attribut recherché.

Ex: "./book/@price" sélectionne tous les attributs price des noeuds book (en se basant sur le contexte actuel).

5.3 - Conditions

Pour exprimer une condition sur une recherche, il faut utiliser un (des) bloc(s) délimité(s) par des crochets [...].

Exemples:

"//Book[price]" sélectionne tous les noeuds du document ayant un noeud enfant nommé price peu importe la valeur de ce dernier.

"//Book[price = '16']" sélectionne tous les noeuds du document ayant un noeud enfant nommé price dont la valeur est exactement 16.

"//Book[1]" sélectionne uniquement le premier noeud Book parmi les noeuds se nommant Book dans un document XML.

"//Book[price > 15]" sélectionne tous les noeuds book du document dont le noeud enfant price a une valeur plus grande que 15.

"//Book[@author != 'Un mauvais auteur']/price" sélectionne tous les noeuds price pour lesquels la valeur de l'attribut author de leur noeud parent Book est différente de "Un mauvais auteur".

De plus, les opérateurs logiques "or" et "and" permettent d'émettre plusieurs conditions.

Toutes ces notions théoriques seront appliquées à un cas concret dans les sections qui suivent.



Il est conseillé de lire la MSDN sur la syntaxe XPath (lien : [MSDN Library](#)).

Elle est claire et apporte de nombreuses précisions sur la syntaxe XPath ainsi que de nombreux exemples.

6 - Rechercher des données avec XPath

6.1 - Structure du fichier XML

Le fichier XML utilisé est celui-ci:

Clients.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Clients>
  <Client id="1">
    <Prenom>Jean-Michel</Prenom>
    <Nom>Laroche</Nom>
    <Telephone>027854632</Telephone>
    <DateInscription>02/06/2007</DateInscription>
    <Status>Activated</Status>
  </Client>
  <Client id="2">
    <Prenom>Smith</Prenom>
    <Nom>Cordwainer</Nom>
    <Telephone>024213296</Telephone>
    <DateInscription>12/06/2007</DateInscription>
    <Status>Activated</Status>
  </Client>
  <Client id="3">
    <Prenom>Frank</Prenom>
    <Nom>Herbert</Nom>
    <Telephone>022354562</Telephone>
    <DateInscription>08/07/2007</DateInscription>
    <Status>Activated</Status>
  </Client>
  <Client id="4">
    <Prenom>Phillipe</Prenom>
    <Nom>Dick</Nom>
    <Telephone>023254789</Telephone>
    <DateInscription>12/07/2007</DateInscription>
    <Status>Deleted</Status>
  </Client>
  <MaxID>4</MaxID>
</Clients>
```

Chaque noeud Client représente un client et contient cinq noeuds-enfants dont les noms indiquent clairement ce que leurs valeurs représentent; si ce n'est le noeud Status qui prend comme valeur soit "Activated" soit "Deleted".

Si sa valeur est Deleted, ce noeud sera ignoré dans toutes les recherches.

La démarche qui consiste à ne pas supprimer définitivement un noeud permet la restauration des suppressions involontaires et permet aussi de ne pas réaffecter un ID qui a déjà été utilisé auparavant.

En effet, il n'est pas souhaitable qu'un utilisateur dont le compte a été supprimé veuille se reconnecter avec l'ID qui lui a été fourni lors de son enregistrement et ne réalise pas que le compte auquel il essaye de se connecter n'est en fait plus à lui.

Le noeud MaxID permet de faciliter l'insertion d'un nouveau client dans le fichier XML car sans ce noeud, la recherche du premier ID non-utilisé est plus compliquée.

6.2 - Rechercher un noeud à partir de ses attributs

Recherche d'un client à partir de son ID (à partir d'un attribut)

```
public static StructureLibrary.Client SearchCustomerById(string ID)
{
    /* On déclare et on crée une instance des variables nécessaires pour la recherche */
    StructureLibrary.Client Customer = new StructureLibrary.Client();
    XPathDocument XPathDocu = new XPathDocument("Clients.xml");
    XPathNavigator Navigator;
    XPathNodeIterator Nodes;
    /* On affecte false à la variable NoMatches afin de vérifier par la suite
     * si la recherche a été fructueuse*/
    Customer.NoMatches = false;
    /* On crée un navigateur */
    Navigator = XPathDocu.CreateNavigator();
    /* On crée ici l'expression XPath de recherche de client à partir de L'ID */
    ExpXPath = "//Client[@id='" + ID + "' and Status != 'Deleted']";
    /* On lance la recherche */
    Nodes = Navigator.Select(Navigator.Compile(ExpXPath));
    /* On vérifie si la recherche a été fructueuse */
    if (Nodes.Count != 0)
    {
        Nodes.MoveNext(); // NOTE: Nécessaire pour se placer sur le noeud recherché
        /* Encodage des données dans la classe Customer */
        Customer.ID = ID; /* Pas besoin de chercher cette donnée vu que c'est notre
         * critère de recherche, on peut donc directement
         * l'encoder. */
        Nodes.Current.MoveToFirstChild(); /* On se déplace sur le premier noeud
         * enfant "Prenom" */
        Customer.FirstName = Nodes.Current.Value;
        Nodes.Current.MoveNext(); // On se déplace sur le noeud suivant "Nom"
        Customer.Name = Nodes.Current.Value;
        Nodes.Current.MoveNext();
        Customer.PhoneNumber = Nodes.Current.Value;
        Nodes.Current.MoveNext();
        Customer.RegistrationDate = Nodes.Current.Value;
    }
    /* Si aucun client n'a été trouvé */
    else
    {
        Customer.NoMatches = true;
    }
    /* Renvoi de toutes les données dans une instance de la classe "Client" */
    return Customer;
}
```

Cette fonction renvoie une classe Client et permet à partir de son seul argument (qui est l'ID du client) de récupérer toutes les données relatives à ce client.

Premièrement, une instance de la classe Client définie dans Structures.cs est créée, cette classe permet de regrouper en un même objet toutes les données sur un client.

Les propriétés de cette classe sont claires et suffisamment commentées dans le code source, elles ne seront donc pas expliquées ici.

Ensuite, une instance de la classe XPathDocument est créée, cette classe permet le chargement (la mise en mémoire) du document XML dans lequel se trouvent les données sur les clients.

Le constructeur d'un XPathDocument prend un argument: le chemin d'accès au fichier XML qui sera analysé.

La classe XPathNavigator permet de se déplacer dans les noeuds et d'en retirer le contenu.

L'expression XPath fonctionne de la manière suivante:

tout en ignorant les noeuds déclarés comme supprimés (Status != 'Deleted'), elle va rechercher l'ensemble de tous les noeuds "Client" du document ayant comme valeur pour l'attribut ID celle passée en argument à la méthode SearchCustomerByID.

Une fois l'expression XPath créée, la recherche est lancée et tous les noeuds répondant aux conditions de recherche (si il y en a) sont placés dans un XPathNodeIterator, classe facilitant la navigation dans les noeuds trouvés de cette façon.

La recherche s'effectue avec la méthode Select d'un XPathNavigator.

De plus, l'expression est compilée pour la rendre plus rapide grâce à XPathNavigator.Compile(string XPathExp).

Le nombre de noeuds trouvés est donné grâce à la propriété Count d'un XPathNode Iterator.

Si aucun noeud n'a été trouvé, la valeur True est affectée à Client.NoMatches.

Si au moins un noeud a été trouvé, le XPathNodeIterator est placé dessus grâce à la méthode MoveNext (attention, oublier d'utiliser cette méthode renvoie des résultats erronés lors de l'extraction ou de la modification de données).

Une fois positionné sur le noeud Client, la méthode MoveToFirstChild() est invoquée, celle-ci place le XPathNodeIterator sur le premier noeud enfant (l'ordre correspond aux déclarations des noeuds dans le document).

Ensuite, la valeur du noeud sur lequel le XPathNodeIterator est positionné est encodée (ici "Prenom").

Pour naviguer dans les noeuds se situant au même niveau (hiérarchique) dans le document, il convient d'utiliser la méthode MoveToNext(), encore une fois, le XPathNodeIterator se place sur le noeud suivant en se basant sur l'ordre des déclarations des noeuds dans le document.

Quand toutes les données sont encodées, la fonction SearchCustomerByID renvoie une classe Client qui contient toutes les données sur le client recherché.

6.3 - Rechercher un noeud à partir d'un noeud enfant

Rechercher un noeud à partir d'un de ses noeuds enfants

```
public static StructureLibrary.Client SearchCustomerByNode(string Criteria, string Node)
{
    /* Pour les commentaires, voir SearchCustomerByID */
    StructureLibrary.Client Customer = new StructureLibrary.Client();
    XPathDocument XPathDocu = new XPathDocument("Clients.xml");
    XPathNavigator Navigator;
    XPathNodeIterator Nodes;
    Customer.NoMatches = false;
    Navigator = XPathDocu.CreateNavigator();
    ExpXPath = "Clients/Client["+Criteria+"=" + Node + " and Status != 'Deleted'"]";
}
```

Rechercher un noeud à partir d'un de ses noeuds enfants

```
Nodes = Navigator.Select(Navigator.Compile(ExpXPath));
if (Nodes.Count != 0)
{
    Nodes.MoveNext();
    Customer.ID = Nodes.Current.GetAttribute("id", "");
    Nodes.Current.MoveToFirstChild();
    Customer.FirstName = Nodes.Current.Value;
    Nodes.Current.MoveToNext();
    Customer.Name = Nodes.Current.Value;
    Nodes.Current.MoveToNext();
    Customer.PhoneNumber = Nodes.Current.Value;
    Nodes.Current.MoveToNext();
    Customer.RegistrationDate = Nodes.Current.Value;
}
else
{
    Customer.NoMatches = true;
}
return Customer;
}
```

Ce code permet de trouver un noeud à partir de n'importe lequel de ses noeuds enfants.

La méthode SearchCustomerByNode prend deux arguments, respectivement le nom du noeud enfant sur lequel la recherche va se baser pour la et la valeur que ce dernier doit avoir.

NOTE/RAPPEL: La condition de l'expression XPath n'est plus précédée du symbole @, ce qui indique que la condition porte sur un noeud.

Ex. : Si le nom du noeud enfant est 'Prenom' et sa valeur attendue 'Toto' alors l'expression prendra cette forme : "Clients/Client[Prenom='Toto' and Status != 'Deleted']"

L'encodage des données dans une instance de la classe client est similaire à la fonction SearchCustomerByID si ce n'est qu'il faut trouver l'attribut ID avant de l'encoder car ce n'est plus le critère de recherche.

Pour se faire, il convient d'invoquer la procédure GetAttribute qui prend deux arguments, le premier étant le nom de l'attribut recherché et le deuxième étant l'espace-nom (namespace) du noeud (si aucun espace-nom n'est déclaré dans le fichier XML, il convient de laisser le second argument vide).

7 - Modification de données XML

7.1 - Edition de données XML

Modifier des données XML déjà existantes

```
public static Boolean EditXMLData(StructureLibrary.Client Customer)
{
    /* On utilise un XmlDocument et non un XPathDocument car ce dernier ne permet
    * pas l'édition des données XML. */
    XmlDocument XmlDoc = new XmlDocument();
    XPathNavigator Navigator;
    XPathNodeIterator Nodes;
    XmlDoc.Load("Clients.xml");
    Navigator = XmlDoc.CreateNavigator();
    ExpXPath = "//Client[@id='" + Customer.ID + "' and Status != 'Deleted']";
    Nodes = Navigator.Select(Navigator.Compile(ExpXPath));
    if (Nodes.Count != 0)
    {
        /* Encodage des nouvelles données */
        Nodes.MoveNext();
        Nodes.Current.MoveToFirstChild();
        Nodes.Current.SetValue(Customer.FirstName);
        Nodes.Current.MoveNext(XPathNodeType.Element);
        Nodes.Current.SetValue(Customer.Name);
        Nodes.Current.MoveNext(XPathNodeType.Element);
        Nodes.Current.SetValue(Customer.PhoneNumber);
        Nodes.Current.MoveNext(XPathNodeType.Element);
        Nodes.Current.SetValue(Customer.RegistrationDate);
        XmlDoc.Save("Clients.xml");
        return true;
    }
    else
    {
        return false;
    }
}
```

Pour éditer des données et non simplement les rechercher, il faut utiliser un XmlDocument à la place d'un XPathDocument.

En effet, l'objet XPathDocument ne permet que de rechercher des données; c'est pourquoi l'utilisation d'un XmlDocument s'avère nécessaire car ce dernier permet la modification des données.

Le processus qui instancie un XmlDocument diffère légèrement de celui qui instancie un XPathDocument.

En effet, le constructeur du XmlDocument ne prend aucun argument, le document est chargé à l'aide de la méthode Load du XmlDocument.

Cependant, le processus de recherche est tout à fait identique.

Une fois positionné sur un noeud, le XPathNodeIterator se déplace sur le premier noeud-enfant grâce à la méthode MoveToFirstChild().

La modification de la valeur d'un noeud s'exécute à l'aide de la méthode SetValue(string NewValue) dont le seul argument est la nouvelle valeur qui sera affectée au noeud.

Une fois toutes les données éditées, il suffit de sauver le document avec la méthode `Save(string FilePath)` du `XmlDocument` qui prend comme argument le chemin où sera créé le nouveau fichier (en l'occurrence, le programme écrase le dernier fichier).

7.2 - Insertion de nouveaux noeuds/attributs dans un document XML

Insertion de nouvelles données dans un document XML

```
public static Boolean InsertNewCustomer(StructureLibrary.Client Customer)
{
    XmlDocument XmlDoc = new XmlDocument();
    XPathNavigator Navigator;
    XPathNodeIterator Nodes;
    Int32 ID; /* Variable utilisée pour savoir quel est l'ID qu'il faut affecter au nouveau
              * noeud créé */
    XmlDoc.Load("Clients.xml");
    Navigator = XmlDoc.CreateNavigator();
    /* Recherche du noeud MaxID pour déterminer quelle sera l'ID du nouveau
     * client. */
    ExpXPath = "//MaxID";
    Nodes = Navigator.Select(Navigator.Compile(ExpXPath));
    Nodes.MoveNext();
    /* On place l'ID le plus élevé du document dans la variable ID */
    ID = Nodes.Current.ValueAsInt;
    /* On incrémente la valeur du noeud MaxID car une fois notre nouveau noeud
     * créé, l'ID le plus élevé du document sera aussi incrémenté */
    Nodes.Current.SetValue((ID+1).ToString());
    /* On se place sur le noeud ayant l'ID le plus élevé */
    ExpXPath = "//Client[@id='"+ID.ToString()+"']";
    Nodes = Navigator.Select(Navigator.Compile(ExpXPath));
    if (Nodes.Count != 0)
    {
        Nodes.MoveNext();
        /* On crée le noeud principal (Client). */
        Nodes.Current.InsertElementAfter("", "Client", "", "");
        /* On se place sur le noeud ainsi créé. */
        Nodes.Current.MoveToNext(XPathNodeType.Element);
        ID++; /* On incrémente ID pour que sa valeur soit identique à celle se
              * trouvant dans le noeud MaxID. */
        /* Encodage des données */
        Nodes.Current.CreateAttribute("", "id", "", ID.ToString());
        Nodes.Current.AppendChildElement("", "Prenom", "", Customer.FirstName);
        Nodes.Current.AppendChildElement("", "Nom", "", Customer.Name);
        Nodes.Current.AppendChildElement("", "Telephone", "", Customer.PhoneNumber);
        Nodes.Current.AppendChildElement("", "DateInscription", "",
Customer.RegistrationDate);
        Nodes.Current.AppendChildElement("", "Status", "", "Activated");
        XmlDoc.Save("Clients.xml");
        return true;
    }
    else
    {
        return false;
    }
}
```

La première opération consiste à récupérer la valeur du noeud `MaxID` afin de savoir quel ID affecter au futur noeud.

Pour faire cela, le `XPathNodeIterator` se place sur ce noeud et en extrait la valeur qui est affectée à la variable `ID`.


La valeur du noeud MaxID est incrémentée de façon à la faire correspondre avec l'ID du noeud qui sera créé par la suite.

Ensuite, l'XPathNodelterator se place sur le noeud ayant l'attribut ID le plus élevé (celui dont la valeur correspond avec celle de la variable ID).

Une fois le XPathNodelterator positionné dessus, un nouveau noeud Client est créé; ce dernier se trouve après tous les autres noeuds.

La création d'un nouveau noeud se fait à l'aide de la méthode InsertElementAfter(string prefix, string LocalName, string NameSpaceURI, string Value).

En l'occurrence, seul l'argument LocalName est intéressant car c'est cet argument qui détermine le nom du futur noeud.

 *La méthode `InsertElementAfter` insère le nouveau noeud après le noeud sur lequel le XPathNodelterator est actuellement positionné.*

Pour insérer le nouveau noeud avant le noeud actuel, il faut utiliser la méthode `InsertElementBefore` qui prend naturellement les mêmes arguments que son homologue.

La prochaine étape consiste à placer le XPathNodelterator sur le nouveau noeud créé avec la méthode MoveToNext(XPathNodeType NodeType).

La variable ID est incrémentée de façon à la faire correspondre avec celle du noeud MaxID.

Enfin, L'attribut ID et les noeuds enfants sont encodés avec les méthodes CreateAttribute et AppendChildElement.

Le méthode CreateAttribute crée un attribut et prend les mêmes arguments que la méthode InsertElementAfter.

La méthode AppendChildElement crée un noeud-enfant et prend également les même arguments que la méthode InsertElementAfter.

Une fois un attribut ou un noeud enfant créé, le XPathNodelterator ne se place pas automatiquement dessus.

7.3 - Suppression d'un noeud et de ses noeuds-enfants.

Il n'y a pas de fonction dans le programme-exemple qui permet de supprimer définitivement un noeud.

Pourquoi? Simplement car lorsque un noeud est définitivement détruit, il n'y a pas moyen de faire une restauration (un rollback).

Le programme ne supprime donc pas le noeud en question mais affecte la valeur "Deleted" à son noeud Status.

Ainsi, ce client est ignoré dans toutes les recherches et apparait comme supprimé mais il y a toujours moyen de réactiver son compte.

Cependant, il existe évidemment un moyen de supprimer définitivement un noeud ou un attribut.

La méthode de suppression d'un noeud et de ses noeuds-enfants est très simple: il suffit de placer un XPathNodelterator sur le noeud à supprimer et d'appeler la méthode DeleteSelf qui ne prend aucun argument.

Si l'opération est un succès, le XPathNodelterator (ou le XpathNavigator) est automatiquement positionné sur l'ancêtre du noeud supprimé.

8 - Améliorations possibles du programme-exemple

Une série d'améliorations du programme-exemple est proposée en guise d'exercice:

- Lorsque le programme recherche un client par son prénom, il ne prend en compte que le premier résultat, or plusieurs clients peuvent avoir le même prénom.

Il est donc préférable que le programme affiche le nombre total de résultats et permette à l'utilisateur de naviguer parmi ceux-ci (par exemple en renvoyant une array de classes "Client").

- Créer une fonction permettant de réactiver le compte d'un client en modifiant la valeur de son noeud Status.

9 - Conclusion

La technologie XPath permet d'écrire un code clair et concis tout en en gardant une application performante.

De plus, l'apprentissage de XPath ne demande pas beaucoup d'investissement.

Enfin, cette technologie est intégrée sur d'autres plateformes telles que Java.

10 - Remerciements

Merci à Erwy pour ses commentaires sur le fond de l'article.

Merci à Cardi pour ses remarques.

Merci à Skalp pour ses remarques sur l'orthographe.

Merci à Bernard Determe pour ses remarques sur le style de l'article.

Et merci à toute l'équipe de developpez.com qui m'a permis d'héberger cet article.

