

# Utiliser un document DOM via MsXml avec Delphi.

par [BIG](#)

Date de publication : 03/08/2006

Dernière mise à jour :

Le but de cet article est de montrer comment utiliser les fonctionnalités d'un document DOM (Document Object Model) en utilisant l'API MsXml.

- I - Introduction
- II - Lire un XML avec MsXml et le parser DOM
- III - Ecrire un XML avec MsXml
- IV - Utilisation de Xpath
- V - Utilisation de XSLT
- VI - Conclusion
- VII - Téléchargement

## I - Introduction

Le but de cet article est de montrer comment utiliser les fonctionnalités d'un document DOM (Document Object Model) en utilisant l'API MSXML. Borland a porté la DLL de Microsoft (MSXML.dll) en unité du même nom, **MSXML.pas**. Ce document sert aussi à la compréhension du DOM pour gérer un fichier XML dans vos applications Delphi. Ainsi, vous pourrez utiliser facilement des fichiers XML comme support de base de données, support de communication entre vos applications et/ou internet, remplacer les fichiers ini, ...

MSXML est un composant COM de Microsoft, implémentant les interfaces DOM, Xpath, WXS (XML-Schema) et XSLT.

MSXML est disponible en version 4. Pour ce tutoriel, j'ai utilisé Delphi 7 qui implémente MSXML v3.

DOM est une recommandation du W3C (World Wide Web Consortium), décrivant une interface totalement indépendante du langage de programmation et de la plate-forme utilisée. De ce fait, l'utilisation de DOM est quasi identique quel que soit le langage utilisé.

DOM-1 (Level 1) a été publié en 1998 pour standardiser les documents. Le W3C a défini une manière précise de représenter un document (XML) sous forme d'un arbre. Le consortium a également proposé des méthodes de navigation standard, notamment pour la gestion des formulaires HTML.

En 2000, le W3C publie DOM-2, qui est le standard actuel. Depuis 2004, DOM existe aussi en [version 3](#), apportant 16 modules différents .

Le parser DOM de MSXML est un parser DOM-2.

## II - Lire un XML avec MsXml et le parser DOM

Soit le document XML suivant (adresses.xml):

```
adresses.xml
<?xml version="1.0"?>
<Carnet>
  <Personne>
    <Nom>Milbal</Nom>
    <Prenom>Jessie</Prenom>
    <Adresse>Rue Sax, 43</Adresse>
    <Localite CP='1000'>Bruxelles</Localite>
  </Personne>
  <Personne>
    <Nom>Hainerve</Nom>
    <Prenom>Kim</Prenom>
    <Adresse>Avenue du Roy,17</Adresse>
    <Localite CP='4000'>Liege</Localite>
  </Personne>
</Carnet>
```

Pour lire ce document, il faut initialiser le parser DOM. Plaçons sur une fiche un bouton, qui va charger le document XML, et un mémo qui permettra d'afficher le document.

La création du parser se fait via l'interface **IXMLDOMDocument** et l'objet associé, **CoDOMDocument**.

```
procedure TForm1.LoadButtonClick(Sender: TObject);
var DOMDoc : IXMLDOMDocument; //l'interface d'un document DOM
begin
  DOMDoc := CoDOMDocument.Create; //création de l'objet correspondant.
  with DOMDoc do begin
    async := False; //Permet de charger un document XML en mode synchrone
    if load('adresses.xml') then begin
      MemoResult.Lines.Add(xml); //affiche le document xml en entier dans un mémo
      ShowMessage(documentElement.nodeName); //affiche la racine du document
    end;
  end;
end;
```

Expliquons un peu ce code :

L'objet document XML est créé via **CoDOMDoc**, qui renvoie une interface de type **IXMLDOMDocument**.

La propriété **async** permet de charger le document xml en mode synchrone ou asynchrone.

Par défaut, le chargement se fait toujours en mode synchrone. La différence résulte dans le mode de fonctionnement : en mode synchrone, le document va être chargé en mémoire, puis le parser va rendre la main au programme pour continuer le traitement. En mode asynchrone, un thread va être créé pour charger le document au fur et à mesure, permettant ainsi au programme de continuer son exécution.

Il faut alors tester la propriété **readyState** afin de savoir si le document est chargé en mémoire. **ReadyState** vaut 'S\_OK' si le chargement a été correctement effectué, sinon il renvoie un code d'erreur.

La méthode asynchrone est conseillée pour charger de gros documents en mémoire sans bloquer l'application.

On préférera donc utiliser le mode asynchrone pour le pré chargement des données ou leur lecture. On préférera le mode synchrone pour travailler directement sur le document.

Le chargement d'un document XML se fait par plusieurs méthodes. La plus courante est l'appel de **load(xmlSource: OLEVariant)**. Cette méthode va charger le document en mémoire, puis l'analyser avec le parser DOM.

Il existe une variante pour charger un document XML en mémoire, à partir d'une String : **loadXML(bstrXML: WideString)**. Cette méthode est généralement utilisée pour créer un document XML avec une racine, ou pour charger un document XML contenu dans un flux, par exemple. La string entrée en paramètre doit posséder la syntaxe correcte du XML.

```
var DOMDoc: IXMLDOMDocument;  
begin  
  DOMDoc := CoDOMDocument.Create;  
  DOMDoc.loadXML('<racine/>'); //crée un document XML à partir de la phrase  
end;
```

*Il est également possible de déclencher une notification en assignant l'événement **onreadystatechange** de l'interface. Cet événement est très utile pour les chargements asynchrones.*

La navigation dans ce document se fait à partir de la racine, via les méthodes **childNodes** pour les noeuds enfants, et **attributes**, pour les attributs situé dans le noeud courant.

La lecture des noeuds se fait via les propriétés **nodeValue** et **text** de l'interface **IXMLDOMNode**. La méthode **nodeName** renvoie quant à elle le nom du noeud sélectionné.

```
var root, node : IXMLDOMNode;  
begin  
  root := DOMDoc.DocumentElement; //recherche de l'élément racine;  
  node := root.childNodes[0]; //sélection du premier enfant  
  ShowMessage(node.NodeValue); //affichage du nom de la racine, employes dans l'exemple  
end;
```

Comment savoir quelle propriété utiliser pour obtenir les données ? DOM traite les noeuds selon des types différents. Le type le plus versatile est le type *element*. Il regroupe les autres types. Il existe aussi des noeuds texte, attributs, ...

En règle générale, la propriété **nodeValue** permet de lire la valeur d'un noeud de type *element* (et de ses dérivés). Or, selon la MSDN (Microsoft Developer Network ), un noeud de type *element* renvoie toujours Null. Et le type de noeud par défaut est un noeud de type *element*. On utilisera alors la propriété **text**, spécifique aux noeuds de type *texte*, qui renverra le contenu entre les balises du document.

Ce système de navigation est fastidieux pour les grands fichiers xml, aussi il est conseillé d'utiliser Xpath pour atteindre directement l'information souhaitée. Nous verrons Xpath plus loin.

Voici les différentes interfaces de MsXml. Nous n'utiliserons que celles marquées en gras :

La documentation complète est disponible dans la [MSDN](#).

### III - Ecrire un XML avec MsXml

L'écriture d'un fichier XML via DOM est très simple. La création d'un noeud se fait par la méthode **CreateNode**, ou un de ses dérivés : **CreateElement**, **CreateText**, **CreateAttribute**, **CreateComment** et **CreateCDATASection**.

La méthode **CreateNode** est rarement utilisée, on préférera utiliser les autres méthodes pour créer le type de noeud requis.

```

procedure TForm1.SaveButtonClick(Sender: TObject);
var
  DOMDoc: IXMLDOMDocument;
  node: IXMLDOMNode;
  element: IXMLDOMELEMENT;
  attribute: IXMLDOMATTRIBUTE;
  texte: IXMLDOMTEXT;
  comment: IXMLDOMCOMMENT;
  CData : IXMLDOMCDATASECTION;
begin
  DOMDoc := CoDOMDocument.Create;
  with DOMDoc do begin
    async :=false;
    documentElement := createElement('root'); //création de l'élément racine
    node := createNode('element','node',''); //méthode globale, le premier argument peut être
    (*
     * element : identique à la méthode createElement
     * text : identique à la méthode createText
     * cdata: identique à la méthode createComment
     * comment: identique à la méthode createCDATASection
     * attribute: identique à la méthode createAttribute
    *)
    documentElement.appendChild(node);
    (*appendChild ajoute le noeud à l'élément parent*)
  end;
end;

```

On utilisera plus fréquemment la méthode **CreateElement**, la plus versatile, car elle dispose de fonctions pour ajouter une série d'attributs, ainsi que du texte.

L'élément étant indépendant, il est nécessaire de l'attacher à un noeud "ancêtre", par la méthode **appendChild** du noeud parent.

```

element := createElement('element');
element.setAttribute('attribute','setAttribute method');
element.text := 'Text Element';
documentElement.appendChild(element);
(*La création d'un élément est une des manières les plus simples à
utiliser, car elle permet de créer facilement des noeuds textes et des
attributs, sans passer par la méthode décrite ci-dessous.
La propriété text de l'élément revient à assigner un noeud de type
texte à cet élément*)

```

L'écriture d'un attribut sur un élément est très simple ; on utilise la méthode **setAttribute([nom de l'attribut], [valeur de l'attribut])**. De même, la méthode **getAttribute([nom de l'attribut])** permet d'atteindre directement la valeur de l'attribut sélectionné.

La méthode **createAttribute** permet de créer un attribut sur un noeud xml. Un attribut est une valeur non textuelle.

#### exemple :

```
<salaire unite= 'USD ' '>55000</salaires>
```

L'attribut étant créé de manière indépendante, il faut l'affecter à un noeud existant, par la méthode **setNamedItem** de l'objet **attribute** dépendant du noeud.

Cette méthode est plus longue, mais explique le fonctionnement interne de la méthode **setAttribute** sur un élément DOM.

On peut lire un attribut via la méthode **getNamedItem** de l'objet **attribute** associé à un noeud. Une méthode plus simple, consiste à transtyper un noeud en élément, afin d'utiliser la méthode **getAttribute**.

```
attribute := createAttribute('attrib');
attribute.value:= 'attribute method';
documentElement.attributes.setNamedItem(attribute);
(*Cette méthode est plus longue, mais permet d'assigner des attributs aux
divers types de noeuds existants*)
```

La méthode **createTextNode** va créer un noeud texte, non attaché au document. On attache le noeud à son ancêtre par la méthode **appendChild** du noeud parent.

Il est possible d'arriver au même résultat avec un élément, en assignant la propriété **text** de l'élément (voir plus haut).

```
texte := createTextNode('text node');
documentElement.appendChild(texte);
(*Cette méthode permet de créer un noeud texte, et de l'assigner à un
noeud quelconque*)
```

La méthode **createComment** permet de créer un commentaire xml. C'est une des seules fonctions qu'il est impossible de créer via la méthode **createElement**.

```
comment := createComment('comments');
documentElement.insertBefore(comment,node);
(*créé un élément de type commentaire.
La méthode insertBefore permet de définir l'emplacement du commentaire
dans le noeud sélectionné*)
```

La méthode **createCDATASection** permet de créer une zone CDATA (Character data). Une section CDATA est utilisée pour délimiter des blocs de texte possédant des caractères qui seraient reconnus comme du balisage par le parser.

Une section CDATA se reconnaît par la forme **<![CDATA[ bloc de texte ]>** . Le parser rencontrant cette balise va simplement retransmettre le contenu tel quel, sans l'analyser. On utilise généralement les balises CDATA dans les documents XHTML pour définir des zones de scripts.

```
CDATA := createCDATASection('CDATA Section');
documentElement.insertBefore(CDATA,comment);
(*Permet de créer une section CDATA*)
save('test-xml.xml');
MemoResult.Lines.Add(xml);
end;
end;
```

Grâce à toutes ces méthodes, vous serez capable de générer un document XML bien formé.

## IV - Utilisation de Xpath

Reprenons le document XML du début. Grâce aux méthodes de lecture décrites précédemment, essayons d'afficher dans un label l'adresse Kim Hainerve, à savoir Avenue du Roy, 17 4000 Liege.

Créons pour cela un nouveau projet, posons un mémo et un bouton. Dans la clause **uses**, ajoutons l'unité **msxml**.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  xml : IXMLDOMDocument;
  node, attrib: IXMLDOMNode;
begin
  xml := CoDOMDocument.Create;
  xml.async := false;
  xml.load('adresses.xml');
  node := xml.documentElement.childNodes.item[1].childNodes.item[3];
  attrib := node.attributes.getNamedItem('CP');
  // ou attrib := node.attributes.item[0];
  Memo1.Lines.Add(node.text + ' ' + attrib.nodeValue);
end;
```

On remarque tout de suite les contraintes liées à ce mode de lecture. Il faut en effet connaître la position exacte de chaque élément, ainsi que son type.

Le problème se situe au niveau de l'arborescence. Imaginez de vouloir récupérer une information se trouvant au 10ème niveau. Les lignes de code deviennent vite longues.

Les plus courageux verront un système complexe de boucles et de fonctions récursives, afin de pouvoir récupérer les informations souhaitées. Mais il existe une solution simple et efficace : **Xpath** !

**Xpath** est à XML ce que SQL est aux bases de données. **Xpath** permet de sélectionner un ou plusieurs noeuds selon une requête. Le principe est simple : on imagine le document XML comme un disque dur, contenant divers dossiers et fichiers. Une balise correspond à un dossier, et l'information correspond à un fichier.

Sous Windows, on peut accéder à un fichier en entrant une adresse : *c:\documents and settings\Big\documents\msxml.odt*

Avec Xpath, on utilisera une syntaxe similaire : ***/Personne[2]/Prenom***

C'est quand même beaucoup plus simple que d'utiliser une série de **childNodes.item** pour retrouver l'information, non ? Mais **Xpath** ne se limite pas à ça : si on ne connaît pas l'emplacement exact du noeud, on peut donner un prédicat. Un prédicat est un critère de recherche. Le prédicat est toujours signalé entre crochets, et s'applique au noeud courant, sauf en cas de présence des modificateurs d'accès (.. , /, @, etc)" à la requête. ***/Personne[Nom = 'Hainerve']/Prenom*** sélectionne le même élément. L'avantage est visible : plus besoin de connaître la position exacte du noeud !

Nous pouvons également sélectionner plusieurs noeuds en même temps, ainsi que des attributs :

***//adresse*** sélectionne tous les noeuds 'adresse' dans une liste de noeuds (NodeList);

***/Personne[2]/Localite/@CP*** sélectionne l'attribut 'CP' (@ est le symbole pour sélectionner un attribut)

Le but de ce tutoriel n'étant pas d'expliquer Xpath en détails, je vous conseille de vous familiariser un peu avec les diverses requêtes, en visitant les liens suivants :

<http://jerome.developpez.com/xml/xsl/xpath/>

ou [http://www.zvon.org/xxl/XPathTutorial/General\\_fre/examples.html](http://www.zvon.org/xxl/XPathTutorial/General_fre/examples.html)

Une fois aguerri aux requêtes **Xpath**, utilisons-les avec **MsXml**.

L'unité **MsXml** fournit deux méthodes pour sélectionner des noeuds avec des requêtes Xpath :

- **selectSingleNode(QueryString: String): IXMLDOMNode** permet de sélectionner un élément.
- **SelectNodes(QueryString: String): IXMLDOMNodeList** permet de sélectionner plusieurs éléments.

Nous allons refaire le même projet que tout à l'heure, mais en utilisant une requête **Xpath**. Modifions la procédure du bouton afin d'utiliser **selectSingleNode**.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  xml : IXMLDOMDocument;
  node, attrib: IXMLDOMNode;
begin
  xml := CoDOMDocument.Create;
  xml.async := false;
  xml.load('adresses.xml');
  //sélectionne le noeud Prenom du noeud Personne ayant comme nom Hainerve
  node := xml.selectSingleNode('//Personne[Nom="Hainerve"]/Prenom');
  //sélectionne l'attribut CP du noeud Adresse
  attrib := node.selectSingleNode('//Personne[Nom="Hainerve"]/Localite/@CP');
  Memo1.Lines.Add(node.text + ' ' + attrib.nodeValue);
end;
```

Je rappelle que les requêtes **Xpath** peuvent débuter de la racine (/) ou du noeud actuel (./)

Le code est plus compréhensible, non ? Grâce au prédicat, nous effectuons un filtre de recherche. Nous aurions obtenu le même résultat avec d'autres prédicats, tels que:

- **//Personne[1]/Prenom**, qui sélectionne le noeud 'Prenom' du premier noeud 'Personne' ;
- **//Localite[@CP="4000"]**, qui sélectionne le noeud 'Localite' du premier noeud qui possède l'attribut CP=4000 ;
- **//Personne[prenom="Kim"]/Adresse**, qui sélectionne le noeud 'Adresse' du premier noeud 'Personne' qui possède Kim comme prénom;
- ...

Les possibilités sont beaucoup plus grandes avec **Xpath**, comparé au code de lecture basique.

Si nous voulons sélectionner plusieurs noeuds à la fois, nous pouvons utiliser la méthode **selectNodes**. Essayons d'afficher un listing de toutes les adresses. Modifions le code comme ceci :

```
procedure TForm1.Button1Click(Sender: TObject);
var
  xml : IXMLDOMDocument;
  nodes : IXMLDOMNodeList;
  attrib : IXMLDOMNode;
  i: integer;
begin
  xml := CoDOMDocument.Create;
```

```
xml.async :=false;
xml.load('adresses.xml');
//sélectionne tous les noeuds Adresse
nodes := xml.selectNodes('//Personne/Adresse');
for i:=0 to nodes.length-1 do begin
  attrib:= nodes.item[i].selectSingleNode('../Localite/@CP');
  Memol.Lines.Add(nodes.item[i].text+' '+attrib.nodeValue);
end;
end;
```

Essayez la requête **//Adresse** et comparez le résultat. Nous voyons que la requête peut être simplifiée. Ajoutez ensuite quelques noeuds 'Personne' au document, et faites les petits exercices suivants :

- afficher toutes les adresses avec un CP supérieurs à 5000
- afficher les noms, prénoms et adresse par ordre de CP croissant.

Grâce à **Xpath**, vous pouvez gérer facilement une petite base de données !

## V - Utilisation de XSLT

XSLT (eXtended Stylesheet Language Transformations). Le but de XSLT est de transformer un document XML en un autre document texte (texte, html, xml). XSLT se base sur Xpath pour naviguer dans le document source.

Pour se familiariser avec XSLT, je vous conseille les sites suivants :

<http://haypo.developpez.com/tutoriel/xml/xslt/>

et <http://haypo.developpez.com/tutoriel/xml/xslt/programmation/>

Nous utiliserons, pour ce tutoriel, le document XSLT suivant :

```
tohtml.xsl
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output omit-xml-declaration="yes" />
  <xsl:template match="Carnet">
    <HTML>
      <BODY>
        <H1>LISTING DES CONTACTS</H1>
        <TABLE>
          <TR>
            <TD>Nom</TD>
            <TD>Prenom</TD>
            <TD>Adresse</TD>
            <TD colspan='2'>Localite</TD>
          </TR>
          <xsl:for-each select="Personne">
            <xsl:sort select="Nom" order="ascending" />
            <TR>
              <TD><xsl:value-of select="Nom" /></TD>
              <TD><xsl:value-of select="Prenom" /></TD>
              <TD><xsl:value-of select="Adresse" /></TD>
              <TD><xsl:value-of select="Localite/@CP"/></TD>
              <TD><xsl:value-of select="Localite" /></TD>
            </TR>
          </xsl:for-each>
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

Ce fichier va nous permettre de créer une page html basée sur le document xml adresses.xml.

MsXml dispose de la fonction **transformNode(stylesheet: IXMLDOMDocument) : IXMLDOMDocument**.

Créons un dernier projet, toujours avec un mémo et un bouton.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  xml,xsl : IXMLDOMDocument;
begin
  xml := CoDOMDocument.Create;
  xsl := CoDOMDocument.Create;
  xml.load('adresses.xml');
  xsl.load('tohtml.xsl');
  memo1.Text := xml.transformNode(xsl);
end;
```

C'est donc très facile d'appliquer une feuille de style XSL à un document. Grâce à XSL, vous pouvez exporter vos données dans d'autres formats de texte !

## VI - Conclusion

L'utilisation de MSXML est très simple d'emploi pour gérer des documents XML orientés données.

Comparé aux autres parsers que sont Xerces et Open XML, ses **avantages** sont :

- Légèreté du code;
- Documentation complète (sur la MSDN);
- Puissance de recherche (via Xpath) et de traitement;
- Possibilité d'utiliser des feuilles de style XSL.

Ses principaux **inconvenients** sont :

- Il n'est disponible que pour la plate forme Windows;
- Il n'est pas prévu pour un affichage direct (les noeuds ne sont pas indentés). Ce problème peut néanmoins être corrigé via une feuille de style XSL.

*Il est à noter que le composant OmniXML permet de travailler plus facilement avec l'implémentation de MsXml, sur lequel il est basé.*

## VII - Téléchargement

Version PDF de cet article :

Miroir 1 : [Version PDF](#)

Dans le cas où le miroir 1 ne fonctionne pas :

Miroir 2 : [Version PDF](#)