Cours .Net
XML

Lemettre Arnaud
Arnaud.lemettre@gmail.com

Sommaire

- Introduction
- Les différents parseurs
- Les parseurs XmlDocument
- Les parseurs Linq
- Les validations XML
- La signature de fichiers

Introduction

Pourquoi utiliser du xml en .Net ?

- Fichier d'échanges

  (cross plateform) : inter bancaire, financier …
- Web Services

  (permet l'interop) : Mélange de technologies, API, …
- Sérialisation

  (sauvegarde d'objets, envoi d'objets à travers les web services)
- Fichier de configuration

  app.config, web.config
- …

Introduction

- Le XML vient du SGML
- On utilise le XML pour structurer les données dans un fichier.
- Beaucoup plus simple, à traiter de façon informatique
- Chaque balise doit être fermée, respect de la case …

Les différents parseurs

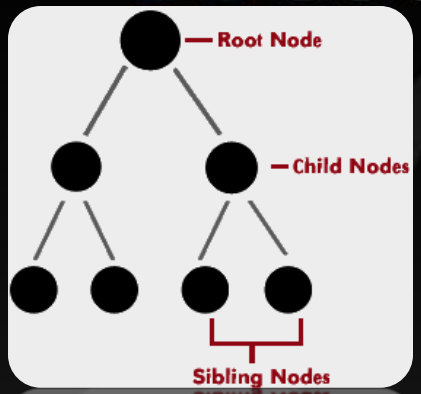
- Quels sont les différents parseurs ?

Les différents parseurs

- 2 grandes familles :
  - Les parseurs SAX
  - Les parseurs DOM

Les parseurs SAX

Les parseurs SAX

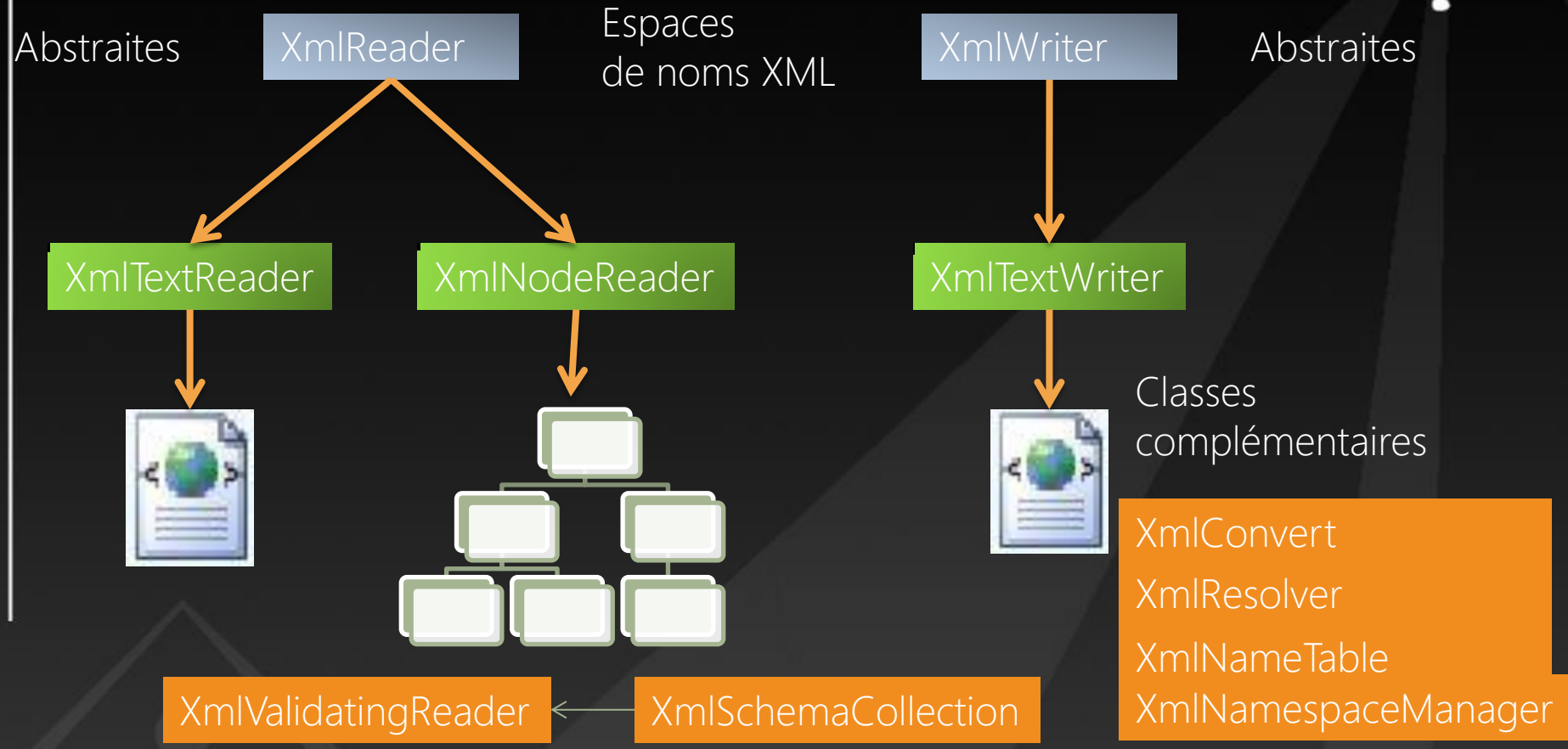


- Lit nœud par nœud
- Econome en mémoire
- On ne peut pas revenir en arrière,

à partir d'un nœud on ne peut pas remonter à l'élément racine

- Permet de gérer de gros documents
- accès en avant uniquement et en lecture seule à un flux de données XML

Les parseurs SAX



Utilisation dans les versions 2.0 à 3.5
Compatible avec le Compact Framework

Les parseurs SAX

Déclaration :

```
XmlTextReader txtReader = new XmlTextReader("golfers.xml");
XmlReaderSettings settings = new XmlReaderSettings();
settings.Schemas.Add("urn:po-schema", "PO.xsd");
settings.ValidationType = ValidationType.Schema;
XmlReader reader = XmlReader.Create(txtReader, settings);
```

Les parseurs SAX

Lecture :

```
while (reader.Read())
{
    if (reader.IsStartElement())
    {
        if (reader.IsEmptyElement)
            Console.WriteLine("<{0}/>", reader.Name);
        else
            Console.WriteLine("<{0}> ", reader.Name);
    }
    else
    {
        if (reader.HasValue)
            Console.WriteLine(reader.Value);
        else
            Console.WriteLine("</{0}>", reader.Name);
    }
}
```

Les parseurs SAX

Ecriture :

```
static void WriteQuote(XmlWriter writer, int id, string name)
{
    writer.WriteStartElement("golfer");
    writer.WriteAttributeString("id",  XmlConvert.ToString(id));
    writer.WriteElementString("name",name);
    writer.WriteEndElement();
}
public static void Main()
{
    XmlTextWriter writer = new XmlTextWriter(Console.Out);
    writer.Formatting = Formatting.Indented;
    WriteQuote(writer, 1,"Will");
    writer.Close();
}
```

Les parseurs DOM

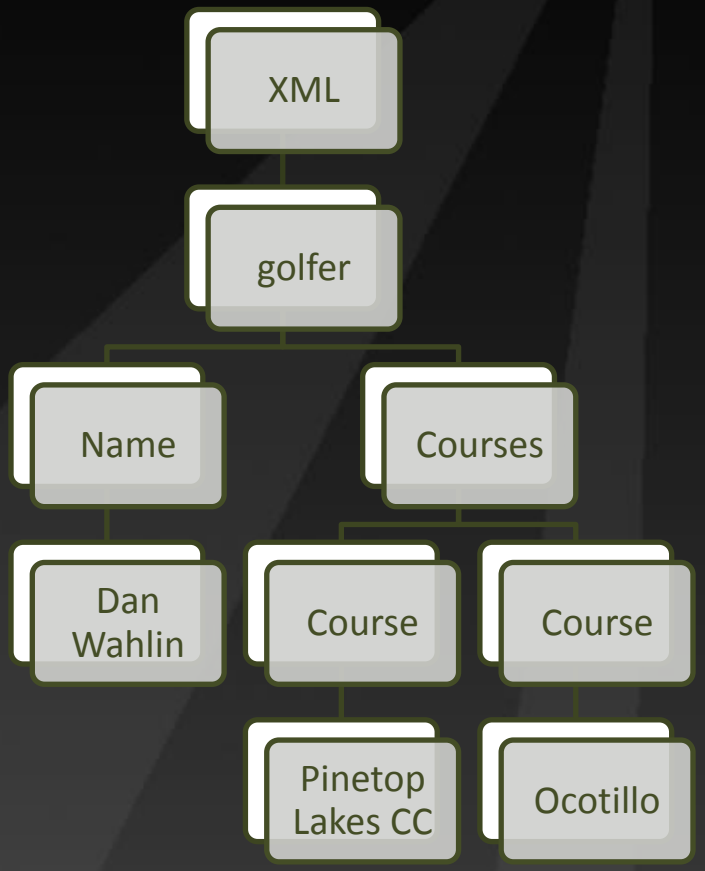
Les parseurs DOM

- Tout l'arbre XML est en mémoire.
- On peut manipuler le XML
- Ajouts, modifications, suppressions, déplacements des nœuds
- Coûteux en mémoire
- Accès direct à certains nœuds (XPath)

Les parseurs DOM

Exemple DOM :

```
<?xml version="1.0" encoding="utf-8" ?>
<golfer>
  <name>Dan Wahlin</name>
  <courses>
    <course>Pinetop Lakes CC</course>
    <course>Ocotillo</course>
  </courses>
</golfer>
```



Les parseurs DOM

- Les classes utiles :
  - XmlDocument
  - XmlAttribute
  - XmlElement
  - XmlNode
  - XmlNodeList
- Utilisation dans les versions 2.0 à 3.5
- Compatible avec le Compact Framework

Les parseurs DOM

Chargement d'un fichier

```
XmlDocument tmpDoc = new XmlDocument();

tmpDoc.Load("fichier.xml");
```

Les parseurs DOM

Sauvegarde d'un fichier

```
XmlDocument tmpDoc = new XmlDocument();

tmpDoc.Save("fichier.xml");
```

Les parseurs DOM

Parcours d'un fichier

Contrairement aux parseurs SAX, nous pouvons sélectionner directement le ou les nœuds sur lesquels travailler.

Pour cela, nous avons deux méthodes :

```
SelectSingleNode() ;
SelectNodes () ;
```

Ces 2 méthodes prennent en paramètre des expressions XPATH pour sélectionner les nœuds.

Les parseurs DOM

Parcours d'un fichier : utilisation de XPath

Sélectionner un nœud n'importe où dans l'arbre XML :

```
//nom_du_noeud
```

Sélectionner un attribut d'un nœud :

```
@nom_attribut
```

Position d'un nœud dans un document :

Position() = 2

Exemple :

```
Child::customer[position() = 2]
```

Permet de sélectionner le 2ème nœud customer de l'arbre XML

Les parseurs DOM

Parcours d'un fichier : utilisation de XPath

```
<?xml version="1.0" encoding="utf-8"?>
<golfers>
  <golfer skill="excellent" handicap="4" clubs="Taylor" id="1">
    <name>
      <lastname>Wahlin</lastname>
      <firstname>Dan</firstname>
    </name>
    <courses>
      <course state="AZ">Pinetop Lakes CC</course>
      <course state="AZ">Ocotillo</course>
    </courses>
  </golfer>
  <golfer skill="moderate" handicap="8" clubs="Taylor" id="2">
    <name>
      <lastname>Wahlin</lastname>
      <firstname>Heedy</firstname>
    </name>
    <courses>
      <course state="AZ">White Mountain</course>
      <course state="AZ">Hobble</course>
    </courses>
  </golfer>
</golfers>
```

Sélection de l'attribut id du 1er golfer :

Tous les descendants de golfer :

Attribut state de la 2nd course du 2nd golfer :

Les parseurs DOM

Parcours d'un fichier : utilisation de XPath

```
<?xml version="1.0" encoding="utf-8"?>
<golfers>
  <golfer skill="excellent" handicap="4" clubs="Taylor" id="1">
    <name>
      <lastname>Wahlin</lastname>
      <firstname>Dan</firstname>
    </name>
    <courses>
      <course state="AZ">Pinetop Lakes CC</course>
      <course state="AZ">Ocotillo</course>
    </courses>
  </golfer>
  <golfer skill="moderate" handicap="8" clubs="Taylor" id="2">
    <name>
      <lastname>Wahlin</lastname>
      <firstname>Heedy</firstname>
    </name>
    <courses>
      <course state="AZ">White Mountain</course>
      <course state="AZ">Hobble</course>
    </courses>
  </golfer>
</golfers>
```

Sélection de l'attribut id du 1er golfer :

```
/golfers/golfer[1]/@id
```

Tous les descendants de golfer

```
/golfers/golfer/*
//golfer/*
```

Attribut state de la 2nd course du 2nd golfer :

```
//golfer[2]/courses/course[2]/@state
```

Les parseurs DOM

Parcours d'un fichier : utilisation de XPath

Exemples de sélections de nœud en C# :

```
XmlDocument doc = new XmlDocument();
//chargement du fichier
doc.Load("XMLFile1.xml");
//on veut un seul noeud ; innerText permet d'accéder à la valeur
string id = doc.SelectSingleNode("/golfers/golfer[1]/@id").InnerText;
//on sélectionne une multitude de noeuds
XmlNodeList list = doc.SelectNodes("//golfer/*");
string state =
   doc.SelectSingleNode("//golfer[2]/courses/course[2]/@state").InnerText;
```

Les parseurs DOM

Création d'un arbre XML

```
XmlDocument xmldoc = new XmlDocument();
//déclaration permettant de spécifier l'encodage
XmlDeclaration declaration = xmldoc.CreateXmlDeclaration("1.0", "utf-8", null);
//on l'insère au début
xmldoc.InsertBefore(declaration, xmldoc.DocumentElement);
//élément principal de l'arbre
XmlElement root = xmldoc.CreateElement("golfers");
if (list.Count != 0)
{
    root.SetAttribute("IDLibrary", list[0].IDLibrary);
}
//on l'attache à un nœud
XmlNode rootNode = xmldoc.AppendChild(root);
//on crée l'élément golfer
XmlElement elt = xmldoc.CreateElement("golfer");
XmlNode cmdNode = rootNode.AppendChild(elt);
```

Les parseurs DOM

Suppression d'un nœud

```
doc.SelectSingleNode("/golfers/golfer[2]").RemoveAll();
```

Il existe d'autres méthodes de suppression qui permettent de supprimer uniquement tous les fils d'un nœud ou juste un nœud fils.

Les parseurs DOM

Une autre technologie ….

… Linq To XML

Disponible uniquement avec la version de C# 3.0
Linq signifie Language Integrated Query (Requête intégrée au langage)

Comme Linq utilise le DOM, l'ensemble du fichier est stocké en mémoire.
On peut effectuer des requêtes comme sur une base SQL

Les parseurs DOM

- Les classes utiles :
  - XDocument
  - XAttribute
  - XElement
- Utilisation dans la version 3.5 du Framework .Net
- Compatible avec le Compact Framework

Les parseurs DOM

Création d'un XML avec Linq

2 manières de faire, en mettant le XML sous forme de chaîne :

```
XDocument docXml = XDocument.Parse("<golfers><golfer/></golfers>");
```

En utilisant les classes de Linq :

```
XElement xml = new XElement("golfers",
    new XElement("golfer",
        new XAttribute("id", "1"),
        new XElement("name",
            new XElement ("firstname","Dan")
   )));

XDocument docXml = new XDocument(xml);
```

```
<golfers>
 <golfer id="1">
  <name>
   <firstname>Dan</firstname>
  </name>
 </golfer>
</golfers>
```

Les parseurs DOM

Création d'un XML avec Linq

Passer des paramètres au constructeur pour spécifier les encodages, …

```
XDocument docXml = new XDocument(
                new XDeclaration("1.0", "utf-8", "yes"),
                new XComment("Exemple golfers"),
                xml);
```

Ceci donnera :

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!—Exemple golfers-->
```

Les parseurs DOM

Requête sur un XDocument

Syntaxe d'une requête :

```
from nom_var_local in nœuds_concernés_par_la_requête

where condition_de_validation

select résultat_à_retourner
```

Exemple :

```
 //selection des nœuds
var q = from c in docXML.Descendants("golfer")
        where (string)c.Attribut("id") == "1"
        select c;

 //affichage des nœuds correspondant à la requête.
foreach (XElement item in q)
{
    Console.WriteLine(item);
}
```

Les parseurs DOM

Requête sur un XDocument

Tout comme en SQL, on peut rajouter des clauses dans le but de paramétrer les requêtes.

- Group, groupe les résultats par rapport à une clé
- Into, identificateur pour les requêtes join, group, select
- Orderby, permet d'ordonner selon une clé
- Join, joint 2 sources
- Let, pour stocker des résultats de sous requêtes
- In, on, equals, mots clés dans une clause Join
- By, mot clé dans une requête group
- Ascending, descending, mots clés dans une requête Orderby

Les parseurs DOM

Insertion d'un nœud dans un XDocument

```
XElement elt =  new XElement("golfer",
        new XAttribute("id", "2"),
        new XElement("name",
            new XElement ("firstname","Will")
            ));

docXml.Element("golfers").Add(elt);
```

```
<golfers>
 <golfer id="1">
  <name>
   <firstname>Dan</firstname>
  </name>
 </golfer>
 <golfer id="2">
  <name>
   <firstname>Will</firstname>
  </name>
 </golfer>
</golfers>
```

Les parseurs DOM

Suppression d'un nœud dans un XDocument

```
docXML.Descendants("golfer").Where(x => (int)x.Attribute("id") == 2).Remove();
```

```
<golfers>
 <golfer id="1">
  <name>
   <firstname>Dan</firstname>
  </name>
 </golfer>
</golfers>
```

Les parseurs DOM

Modification d'un nœud dans un XDocument

```
var node = from XElement e in docXml.Descendants("golfer")
           where e.Attribute("id").Value == "1"
           select e;

if (node.Any())
{
    node.FirstOrDefault().Value = "Will";
}
```

```
<golfers>
 <golfer id="1">
  <name>
   <firstname>Will</firstname>
  </name>
 </golfer>
</golfers>
```

FirstOrDefault peut renvoyer Null dans certains cas
Le .Any() permet de savoir s'il y a au moins un élément, plus optimisé que le .Count() dans ce cas de figure

Les parseurs DOM

Sauvegarde d'un XDocument

```
docXML.Save("golfers.xml");
```

Chargement d'un XDocument

```
XDocument.Load("livres.xml");
```

XML AVANCÉ

XML Avancé

- Validation d'un XML :
  - Grâce aux schémas XSD

Exemple :

XML Avancé

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="golfers">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="golfer">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="lastname" type="xs:string" />
                    <xs:element name="firstname" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xs:unsignedByte" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML Avancé

Chargement du fichier schéma :

```
//récupération du schéma de validation
XmlReader fileSchema = DataAccess.Echange.GetSchema("echange.xsd");
//ajout du schéma
doc.Schemas.Add("", fileSchema);
```

XML Avancé

Inscription à un événement pour la validation :

```
ValidationEventHandler eventHandler = new
  ValidationEventHandler(ValidationEventHandler);
doc.Validate(eventHandler);
```


XML Avancé

Méthode qui sera appelée s'il y a une erreur lors de la validation :

```
void ValidationEventHandler(object sender, ValidationEventArgs e)
{
    switch (e.Severity)
     {
          case XmlSeverityType.Error:
               Debug.WriteLine(e.Message);
               break;
          case XmlSeverityType.Warning:
               Debug.WriteLine( e.Message);
               break;
     }
}
```


XML Avancé

Signature d'un XML

- Pourquoi signer un fichier ?
- Peut on signer autre chose qu'un fichier en entier ? (Pour le XML)
- Quelle est la différence entre un hash md5 et signer avec un certificat ?

XML Avancé

Signature d'un XML

- Pourquoi signer un fichier ?

  Permet de savoir si un fichier a été corrompu, permet la non répudiation d'un message, garantit son intégrité.

- Peut on signer autre chose qu'un fichier en entier ? (Pour le XML)

  Oui, on peut ne signer qu'un seul nœud du fichier

- Quelle est la différence entre un hash md5 et signer avec un certificat ?

  Un hash permet juste de vérifier si les données ont été modifiées. De plus, cela tient compte de tous les caractères et ne permet pas d'authentifier la personne qui à fait l'empreinte.

XML Avancé

Signature d'un XML : comment ça marche ?

- Il faut disposer d'un certificat SSL.
  - Une clé privée pour nous.
  - Partager ensuite le certificat (clé public) avec les autres utilisateurs.
- La signature peut être enveloppante, enveloppée, ou détachée.

XML Avancé

Signature d'un XML : les 3 types de signature

Signature enveloppante :

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

Signature enveloppée :

```
<?xml version="1.0" encoding="utf-8"?>
<root>
        <data/>
        <signature/>
</root>
```

Signature détachée:

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

```
        <signature>
        </signature>
```

XML Avancé

Signature d'un XML : méthode enveloppée en .Net

Le fait de ne pas donner de code est volontaire => Exo bonus ;)
Une correction sera fournie, avec tout le code nécessaire.

- Récupération du certificat :
  - Récupération du certificat : X509Store, X509Certificate2
  - Génération de la signature : SignedXml, KeyInfo
  - Ajout de la signature au document XML

XML Avancé

Signature d'un XML : méthode enveloppée en .Net

Pour vérifier la signature :

- Lecture du fichier XML
- Vérification de la signature : SignedXml

QUESTIONS ?