SLAM-2

matthieu.planchot@stjosup.com

Résumé

Langage standard de balisage : XML  
Programmation DOM  
Sérialisation : mise à plat d’un objet dans un fichier  
Désérialisation : reconstruction en mémoire de l’objet

Serialisation Deserialisation

Matthieu Planchot

# Introduction

Un document XML se compose d’éléments imbriqués. Un élément est représenté par une balise ouvrante et une fermante. Il n’y a pas de balisage prédéfini comme le Html par exemple. Les noms des éléments sont sensibles à la casse.

<?xml version="1.0" encoding="ISO-8859-1"?>

[<voitures>](file:///D:\source\repos\xml\voitures1.xml)

[<voiture>](file:///D:\source\repos\xml\voitures1.xml)

<immatriculation>1245TR93</immatriculation>

<modele>Laguna</modele>

<couleur>blanche</couleur>

<nbKm>125000</nbKm>

<dateSortie>12/12/1996</dateSortie>

</voiture>

[<voiture>](file:///D:\source\repos\xml\voitures1.xml)

<immatriculation>4578GH93</immatriculation>

<modele>Golf</modele>

<couleur>bleue</couleur>

<nbKm>12450</nbKm>

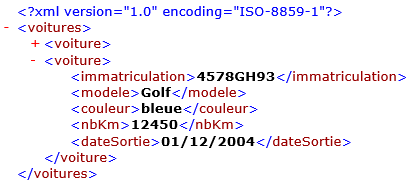
<dateSortie>01/12/2004</dateSortie>

</voiture>

</voitures>

En haut de document on voit la version xml utilisé et l’encodage du document. Il existe 2 éléments « voiture » dans « voitures », chacune comportant 5 éléments.

Dans un navigateur web, il est possible de rétracter un élément qui en contient plusieurs.



# Attributs

Il est possible de mettre des attributs au sein d’un élément.

<?xml version="1.0" encoding="ISO-8859-1"?>

[<voitures>](file:///D:\source\repos\xml\voitures3.xml)

<voiture dateSortie="**12/12/1996**" nbKm="**125000**" couleur="**blanche**" modele="**Laguna**" imatriculation="**1245TR93**"/>

<voiture dateSortie="**01/12/2004**" nbKm="**12450**" couleur="**bleue**" modele="**Golf**" imatriculation="**4578GH93**"/>

</voitures>

Une structure avec des éléments sera plus souple qu’avec les attributs mais l’attribut peut avoir son intérêt pour caractériser des valeurs stables comme les identifiants.

# Règles de nommage

Les éléments ne doivent pas contenir contenir les caractères \_ - . et ne commence pas par un chiffre, certains caractères doivent être remplacé pour être utilisé, il s’agit de < & > " et '

Il est possible de déclarer une section de donnée pour admettre tous les caractères : < ![CDATA[ n’importe quoi ]]>

# Commentaire

< !--pareil qu’en HTML -->

# Définition de type de document, la DTD

La DTD permet de définir la structure d’un document XML, s’il est bien formé en accord avec les éléments, le document XML sera dit valide.

<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE voitures [

<!ELEMENT voiture (immatriculation, modele, couleur, nbKm, dateSortie)>

<!ELEMENT immatriculation (#PCDATA)>

<!ELEMENT modele (#PCDATA)>

<!ELEMENT couleur (#PCDATA)>

<!ELEMENT nbKm (#PCDATA)>

<!ELEMENT dateSortie (#PCDATA)>

]>

<voitures>

<voiture>

<immatriculation>1245TR93</immatriculation>

<modele>Laguna</modele>

<couleur>blanche</couleur>

<nbKm>125000</nbKm>

<dateSortie>12/12/1996</dateSortie>

</voiture>

<voiture>

<immatriculation>4578GH93</immatriculation>

<modele>Golf</modele>

<couleur>bleue</couleur>

<nbKm>12450</nbKm>

<dateSortie>01/12/2004</dateSortie>

</voiture>

</voitures>

Il peut être placé dans le document ou dans un fichier séparé et référencé ainsi : <!DOCTYPE voitures SYSTEM "voitures.dtd">

Pour déclarer des attributs associés à des éléments, on a le format :

< !ATTLIST voiture immatriculation CDATA déclaration>

<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE voitures [

<!ELEMENT voiture (immatriculation, modele, couleur, nbKm, dateSortie)>

<!ATTLIST voiture immatriculation CDATA #REQUIRED modele CDATA #IMPLIED>

<!-- REQUIRED : obligatoire IMPLIED : optionnel -->

<!ELEMENT couleur (#PCDATA)>

<!ELEMENT nbKm (#PCDATA)>

<!ELEMENT dateSortie (#PCDATA)>

]>

<voitures>

<voiture immatriculation="1245TR93" modele="Laguna">

<couleur>blanche</couleur>

<nbKm>125000</nbKm>

<dateSortie>12/12/1996</dateSortie>

</voiture>

<voiture immatriculation="4578GH93" modele="Golf">

<couleur>bleue</couleur>

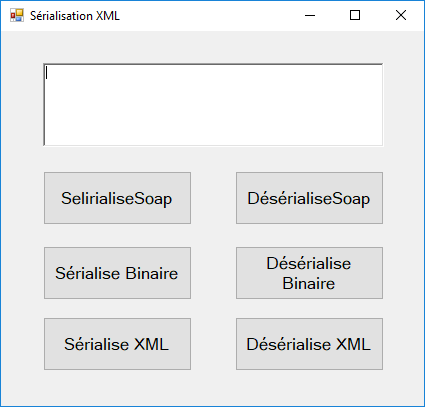
<nbKm>12450</nbKm>

<dateSortie>01/12/2004</dateSortie>

</voiture>

</voitures>

# Serialisation / Deserialisation sur Visual Studio



La classe Voiture est Serializable, cela permet le passage à plat (la sérialisation) ou la reconstruction de l’objet (la désérialisation).

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Serialization;

namespace serial

{

[Serializable]

public class Voiture

{

public string immatriculation { get; set; }

public string couleur { get; set; }

public string dateSortie { get; set; }

public DateTime dateS { get; set; }

public string modele { get; set; }

public int nbKm { get; set; }

public Voiture() { }

public Voiture(string immatriculation, string modele, string couleur, int nbKm, string dateSortie)

{

this.immatriculation = immatriculation;

this.modele = modele;

this.couleur = couleur;

this.nbKm = nbKm;

this.dateS = Convert.ToDateTime(dateSortie);

}

public void roule(int km)

{

this.nbKm += km;

}

}

}

La classe Garage définit les méthodes de sérialisation et désérialisation. La sérialisation a pour effet de créer des fichiers soap, binaire ou xml. Pour les désérialisation, on affiche l’immatriculation de la deuxième voiture sauf pour Xml où on affiche l’immatriculation de la voiture.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Serialization;

using System.IO;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Soap;

using System.Collections;

using System.Windows.Forms;

using System.Runtime.Serialization.Formatters.Binary;

namespace serial

{

[Serializable]

public class Garage

{

public Garage()

{

ArrayList mesVoitures = new ArrayList();

}

private ArrayList mesVoitures = new ArrayList();

public void AddVoiture(Voiture v)

{

mesVoitures.Add(v);

}

public Voiture GetVoiture(int n)

{

return (Voiture)mesVoitures[n];

}

public void Soap()

{

Voiture v = new Voiture("1234YT93", "laguna", "bleue", 1250, "12/1/2005");

Voiture v1 = new Voiture("4444GT93", "twingo", "verte", 10400, "1/11/2004");

Garage g = new Garage();

g.AddVoiture(v);

g.AddVoiture(v1);

FileStream fichier = new FileStream("D:\\source\\repos\\xml\\serial\\serialisation\\garage.sr", FileMode.Create);

SoapFormatter sf = new SoapFormatter();

sf.Serialize(fichier, g);

fichier.Close();

}

public string DSoap()

{

FileStream fichier = new FileStream("D:\\source\\repos\\xml\\serial\\serialisation\\garage.sr", FileMode.Open);

SoapFormatter sf = new SoapFormatter();

Garage g = (Garage)sf.Deserialize(fichier);

string txt = g.GetVoiture(1).immatriculation;

fichier.Close();

return txt;

}

public void Bin()

{

Voiture v = new Voiture("1234YT93", "laguna", "bleue", 1250, "12/1/2005");

Voiture v1 = new Voiture("4444GT93", "twingo", "verte", 10400, "1/11/2004");

Garage g = new Garage();

g.AddVoiture(v);

g.AddVoiture(v1);

FileStream fichier = new FileStream("D:\\source\\repos\\xml\\serial\\serialisation\\garageBinaire.txt", FileMode.Create);

BinaryFormatter bf = new BinaryFormatter();

bf.Serialize(fichier, g);

fichier.Close();

}

public string DBin()

{

FileStream fichier = new FileStream("D:\\source\\repos\\xml\\serial\\serialisation\\garageBinaire.txt", FileMode.Open);

BinaryFormatter bf = new BinaryFormatter();

Garage g = (Garage)bf.Deserialize(fichier);

string txt = g.GetVoiture(1).immatriculation;

fichier.Close();

return txt;

}

public void Xml()

{

Voiture v = new Voiture("121GFT93", "twingo", "verte", 1235, "12/12/2004");

Garage g = new Garage();

FileStream fichier = new FileStream("D:\\source\\repos\\xml\\serial\\serialisation\\voitureXml.xml", FileMode.Create);

XmlSerializer serializer = new XmlSerializer(typeof(Voiture));

serializer.Serialize(fichier, v);

fichier.Close();

}

public string DXml()

{

FileStream fichier = new FileStream("D:\\source\\repos\\xml\\serial\\serialisation\\voitureXml.xml", FileMode.Open);

XmlSerializer serializer = new XmlSerializer(typeof(Voiture));

Voiture v = (Voiture)serializer.Deserialize(fichier);

string txt = v.immatriculation;

fichier.Close();

return txt;

}

}

}

public partial class Form1 : Form

{

Garage g = new Garage();

public Form1()

{

InitializeComponent();

}

public static void VoitureList()

{

XmlDocument docXml = new XmlDocument();

docXml.Load("voitures.xml");

XmlNodeList desNoeuds;

desNoeuds = docXml.GetElementsByTagName("voiture");

List<Voiture> lesVoitures = new List<Voiture>();

foreach (XmlNode unNoeud in desNoeuds)

{

XmlNodeList noeudsEnfants = unNoeud.ChildNodes;

string imma = noeudsEnfants[0].InnerText;

string modele = noeudsEnfants[1].InnerText;

string couleur = noeudsEnfants[2].InnerText;

int km = Convert.ToInt32(noeudsEnfants[3].InnerText);

string date = noeudsEnfants[4].InnerText;

lesVoitures.Add(new Voiture(imma, modele, couleur, km, date));

}

Voiture v = lesVoitures[1];

v.roule(100);

int n = v.nbKm;

}

private void SerialiseSoap\_Click(object sender, EventArgs e)

{ g.Soap(); }

private void DeserialiseSoap\_Click(object sender, EventArgs e)

{ txt.Text = g.DSoap(); }

private void SerialBin\_Click(object sender, EventArgs e)

{ g.Bin(); }

private void DeserialBin\_Click(object sender, EventArgs e)

{ txt.Text = g.DBin(); }

private void SerialXml\_Click(object sender, EventArgs e)

{ g.Xml(); }

private void DeserialXml\_Click(object sender, EventArgs e)

{ txt.Text = g.DXml(); }

}

Form1 est lié à l’affichage par des event. On appelle la bonne méthode selon le bouton cliqué.