**La sérialisation**

**Introduction**

La sérialisation est un procédé qui consiste à sauvegarder l'état (ou les données) d'un objet dans un fichier plutôt que de le garder en mémoire. Ce mécanisme se montre nécessaire dès que les objets doivent être accessibles à partir d'un environnement étranger (service Web, par exemple) : un objet peut ainsi exister entre deux programmes, ou deux exécutions d'un programme (c'est la [persistance objet](http://www.journaldunet.com/developpeur/tutoriel/theo/030408theo_persistence.shtml)).

.NET Framework offrent deux technologies de sérialisation :

* La sérialisation XML
* La sérialisation binaire

**Sérialisation XML**

**XML*(Extensible Markup Language***) est un [langage informatique](http://fr.wikipedia.org/wiki/Langage_informatique) de [balisage](http://fr.wikipedia.org/wiki/Langage_de_balisage) *générique.* Son objectif initial est de faciliter l'échange automatisé de contenus entre [systèmes d'informations](http://fr.wikipedia.org/wiki/Syst%C3%A8me_d%27information) hétérogènes. []La sérialisation XML peut être utile si vous souhaitez par exemple échanger des informations avec des applications qui ne sont pas créés avec les technologies .NET.

En contrepartie, la sérialisation XML ne peut sérialiser que des éléments dont la portée est définie en Publique. Il est également impossible de sérialiser des objets graphiques (Des images par exemples) ; seuls les objets standards sont sérialisables.

**Rendre ses classes sérialisable**

Il faut utiliser un attribut spécial au dessus des classes que nous voulons sérialiser. Cet attribut indique par défaut au Runtime que l’on souhaite sérialiser toute la classe.

*[Serializable]*

*Public Class nomClasse{}*

Avant d'aller plus loin, sachez qu'il existe cependant des contraintes à respecter pour que vos objets soient sérialisables :

* Les classes, structures, et énumérations sont sérialisables. Les interfaces **ne sont pas** sérialisables
* Les classes à sérialiser doivent être **publics**
* La classe à sérialiser doit posséder un **constructeur par défaut** (public et sans paramètres) : cela est nécessaire pour que la désérialisation puisse créer une instance de ce type.
* Seuls les champs et propriétés **publics** sont sérialisés (les champs privés ne sont pas pris en considération lors de la sérialisation)
* Les propriétés à sérialiser ne doivent pas être en lecture seule (les propriétés qui renvoient une collection sont une exception à cette règle)

**Sérialisation/Désérialisation d'objets en XML**

Le code ci-dessous montre l’exemple de la sérialisation de la classe Personne

[Serializable]

public class Personne

{

private string nom;

private string prenom;

private int age;

private static int nbPersonnes;

public Personne()

{

nbPersonnes++;

}

public Personne(string nom,string prenom, int age)

{

this.nom = nom;

this.prenom = prenom;

this.age = age;

nbPersonnes++;

}

public int Age

{

get { return this.age; }

set { age = value; }

}

public string Nom

{

get { return this.nom; }

set { this.nom=value; }

}

public string Prenom

{

get { return this.prenom; }

set { this.prenom = value; }

}

public static int NbPersonnes

{

get { return nbPersonnes; }

}

public void afficher()

{

Console.WriteLine("nom: " + nom + "\tprenom: " + prenom + "\tage: " + age);

}

public void serialisation(string nomfichier)

{

FileStream fichier = File.Open(nomfichier, FileMode.Create);

XmlSerializer s=new XmlSerializer(typeof(Personne));

s.Serialize(fichier,this);

fichier.Close();

}

public static Personne deserialisation(string nomfichier)

{

FileStream fichier = File.Open(nomfichier, FileMode.OpenOrCreate);

XmlSerializer s=new XmlSerializer(typeof(Personne));

Personne p = (Personne)s.Deserialize(fichier);

fichier.Close();

return p;

}

}

Le principe de sérialisation et de désérialisation des données consiste à :

1. Créer un flux (objet de la classe FileStream) vers un fichier qui contiendra nos données. Pour cela, on utilise la méthode Open de la classe File qui prend en paramètre une chaine de caractères représentant le nom du fichier, et le mode d’ouverture.

Parmi les modes d’ouverture les plus utilisés, on cite :

* Create : Spécifie que le système d'exploitation doit créer un fichier. Si le fichier existe, il est remplacé.
* Open : Spécifie que le système d'exploitation doit ouvrir un fichier existant. L’exception [System.IO.FileNotFoundException](http://msdn.microsoft.com/fr-fr/library/system.io.filenotfoundexception.aspx) est levée si ce fichier n'existe pas.
* OpenOrCreate : Spécifie que le système d'exploitation doit ouvrir un fichier s'il existe ; sinon, un nouveau fichier doit être créé.
* Append : Ouvre le fichier s'il existe et accède à la fin du fichier, ou crée un nouveau fichier si le fichier spécifié n’existe pas.

La méthode Open retourne un objet de la classe FileStream qui prendra en charge les opérations de lecture et écriture dans le fichier spécifié.

**Note :** La classe File Fournit des méthodes statiques pour créer, copier, supprimer, déplacer et ouvrir des fichiers et facilite la création d'objets [FileStream](http://msdn.microsoft.com/fr-fr/library/system.io.filestream.aspx).

1. Instancier la classe de sérialisation adaptée. La classe utilisée pour sérialiser ou désérialiser en XML s'appelle XmlSerializer.
2. Utiliser la méthode Serialise pour sérialiser ou Deserialise pour effectuer l'opération inverse.

Pour pouvoir utiliser la classe FileStream et la classe XmlSerializer, il faut ajouter les deux espaces de noms suivants :

using System.IO;

using System.Xml.Serialization;

Ci-dessous, un programme de test de la sérialisation et la désérialisation d’un objet de la classe personne.

static void Main(string[] args)

{

Personne p1 = new Personne("aloui", "ilham", 25);

p1.serialisation("person.xml");

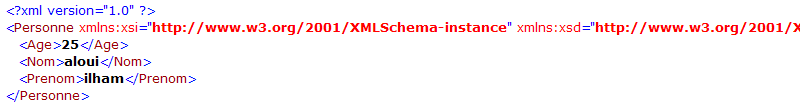
Personne pers = Personne.deserialisation("person.xml");

pers.afficher();

Console.Read();

}

Voici le contenu du fichier personne.xml après exécution du programme:



**Sérialisation Binaire**

La sérialisation d’un objet en binaire est la plus basique. Elle permet de convertir un objet en un flux binaire peu lisibles (voir pas du tout) puis de le stocker dans un fichier. La sérialisation binaire est la seule à pouvoir sérialiser des objets courant comme des objets de type de graphique.

Parmi les différences entre la sérialisation binaire et XML, on peut citer que dans la sérialisation binaire :

* La classe est sérialisée même s’elle privée.
* Tous les champs sont sérialisés (même ceux qui sont déclarés private)
* La classe n’est pas censée avoir un constructeur sans paramètres.

Le code suivant illustre l’exemple de la sérialisation d’un objet de la classe Personne

[Serializable]

class Personne

{

private string nom;

private string prenom;

private int age;

private static int nbPersonnes;

public Personne(string nom,string prenom, int age)

{

this.nom = nom;

this.prenom = prenom;

this.age = age;

nbPersonnes++;

}

public string Nom

{

get { return this.nom; }

set { this.nom=value; }

}

public string Prenom

{

get { return this.prenom; }

set { this.prenom = value; }

}

public static int NbPersonnes

{

get { return nbPersonnes; }

}

public void afficher()

{

Console.WriteLine("nom: " + nom + "\tprenom: " + prenom + "\tage: " + age);

}

public void serialisation(string nomfichier)

{

FileStream fichier = File.Open(nomfichier, FileMode.OpenOrCreate);

BinaryFormatter s=new BinaryFormatter();

s.Serialize(fichier,this);

fichier.Close();

}

public static Personne deserialisation(string nomfichier)

{

FileStream fichier = File.Open(nomfichier, FileMode.Open);

BinaryFormatter s = new BinaryFormatter();

Personne p = (Personne)s.Deserialize(fichier);

fichier.Close();

return p;

}

}

Vous remarquez que le principe de la sérialisation binaire reste le même que celle en XML. La seule différence est qu’il faut utiliser un objet de classe BinaryFormatter (au lieu de XmlSerializer) qui prendra en charge la sérialisation ou la désérialisation binaire d’un objet.

De même, pour utiliser la classe FileStream et la classe BinaryFormatter, il faut ajouter les deux espaces de nom suivants :

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

Ci-dessous, un programme de test de la sérialisation et la désérialisation binaire d’un objet de la classe personne.

static void Main(string[] args)

{

Personne p1 = new Personne("aloui", "ilham", 25);

p1.serialisation("person.txt");

Personne pers = Personne.deserialisation("person.txt");

pers.afficher();

Console.Read(); }

Si vous ouvrez le fichier personne.txt, vous remarquez que son contenu est illisible, contrairement au cas de la sérialisation XML.