# Sérialisation en .Net

par Olivier DELMOTTE

Date de publication : 27/01/2006

Dernière mise à jour : 27/01/2006

La sérialisation, qu'est-ce que c'est et à quoi ça sert ? Nous étudierons dans ce tutoriel les 3 différentes méthodes de sérialisation que Microsoft a mis à notre disposition dans le Framework .NET et comment les utiliser. Pour terminer, je vous fournirais un petit récapitulatif sur les 3 méthodes. Bonne lecture.

## Introduction

> *La sérialisation est le processus de conversion de l'état d'un objet en une forme enregistrable ou transportable. Le complément de la sérialisation est la désérialisation, qui convertit un flux en objet. Ensemble, ces processus facilitent le stockage et le transfert des données.*

Cet extrait de la MSDN résume parfaitement le rôle de la sérialisation. Vous comprenez très vite les avantages que vous pouvez en tirer. Plus besoin d'écrire vous-même des pages de code énormes pour sauvegarder vos données dans des fichiers, la sérialisation le fait pour vous. La communication entre applications devient également un jeu d'enfant. Les Services Web sont, entre autres, fondés sur ce processus de sérialisation / désérialisation.

Afin de vous faciliter encore plus le travail, Microsoft a inclus dans le Framework .NET les outils pour la sérialisation sous 3 formes : XML, SOAP et binaire. Chacune de ces techniques de sérialisation présente ses avantages et ses inconvénients. Nous verrons les possibilités qu'apportent chacune d'entre elles.

Afin d'avoir une base pour la mise en pratique ce que nous allons voir, nous utiliserons un objet type. Vous pouvez télécharger les sources complètes à la fin de l'article.

Afin de simplifier le code au maximum et ne pas le surcharger afin de ne pas perdre sa clarté, j'ai volontairement omis tout ce qui concerne la gestion des erreurs (dans la mesure du possible). Etant donné que dans ce tutoriel:c:, nous travaillerons avec des fichiers, n'oubliez pas la gestion des exceptions relatives aux fichiers.

> *Ce tutoriel a été écrit en se basant sur la version 2 du Framework .Net.*

> *Nous ne nous attarderons pas non plus sur la gestion des exceptions. Pensez quand même que dans ces exemples nous travaillons avec des fichiers et que vous pourriez vous retrouver avec des exceptions de type FileNotFoundException ou du style.*

# 1 - Les attributs de la sérialisation

Pour effectuer une sérialisation d'un objet, il est nécessaire que lui et ses propriétés possèdent certains attributs.

## 1.1 - SerializableAttribute

Cet attribut spécifie que l'objet est sérialisable. Si vous tentez de sérialiser un objet ne possédant pas cet attribut, une SerializationException sera levée.

*La sérialisation XML ne nécessite pas cet attribut.*

## 1.2 - NonSerializedAttribute

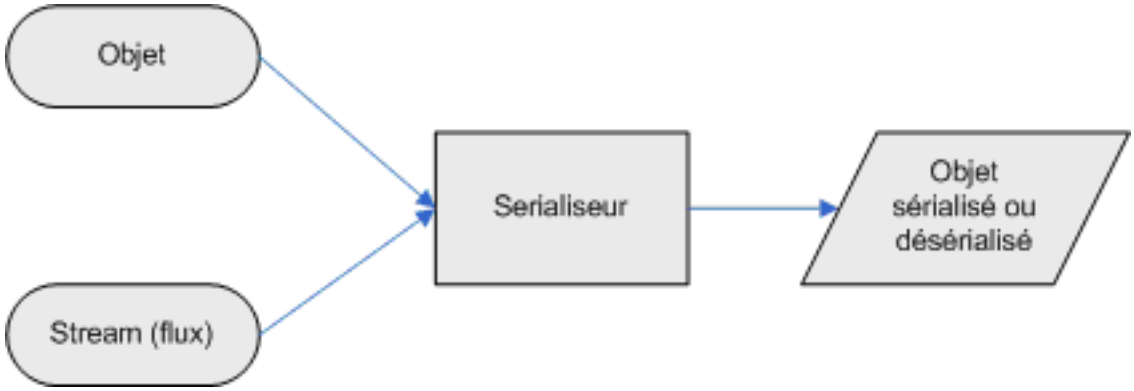
Il s'utilise devant un champ d'un objet pour spécifier qu'il ne doit pas être sérialisé. Etant donné que l'AttributeUsage du NonSerializedAttribute ne prévoit pas les méthodes, n'essayez pas de l'utiliser sur des propriétés et encore moins sur des méthodes.

## 2 - Idée générale

Bien le fait que le Framework .NET fournisse 3 méthodes différentes pour sérialiser les objets, le principe de mise en #uvre reste sensiblement le même.

Comme vous le constaterez dans les exemples de code ci-après, les différences sont minimes (espace de noms, types de sérialiseur).

Je vais donc expliquer ici le principe de fonctionnement général.




*Principe de la sérialisation*

Le schéma peut vous paraître assez simpliste, mais la réalité l'est tout autant.

Pour sérialiser un objet, vous devez fournir au sérialiseur un flux (FileStream, MemoryStream, #) et l'objet à sérialiser, bien entendu.

Bien sur, il existe des spécificités pour chacune des 3 méthodes, regardons-les de plus près.

Pour tous les exemples et les résultats, nous utiliserons cette classe comme objet à sérialiser :

```
[Serializable()]
public class ObjSerialization
{
        public ObjSerialization()
        {
        }

        private string _Nom = "Exemple de sérialisation SOAP";
        public string Nom
        {
                get
                {
                        return _Nom;
                }
                set
                {
                        _Nom = value;
                }
        }

        public System.Drawing.Color _Couleur = System.Drawing.Color.AliceBlue;
        public System.Drawing.Color Couleur
        {
                get
                {
                        return _Couleur;
                }
```

```
                set
                {
                        _Couleur = value;
                }
        }
}
```

# 3 - La sérialisation XML

Pour pouvoir utiliser la sérialisation XML, faites référence au namespace System.XML.Serialization contenu dans l'assembly System.XML.dll

La première chose à savoir sur le sérialiseur XML, c'est qu'il ne tient pas compte des champs et propriétés privés d'un type. De plus, les attributs SerializableAttribute et NonSerializedAttribute n'ont aucun effet sur le sérialiseur XML, donc n'importe quel champ ou propriété publiques de n'importe quelle classe sont sérialisables.

Pourquoi un tel comportement ? Le sérialiseur XML n'utilise pas les attributs standards de sérialisation (SerializableAttribute et NonSerializedAttribute). Pour ignorer un champs lors de la sérialisation, utilisez le XmlIgnoreAttribute.

## 3.1 - Sérialisation

```
public static bool Serialize(object objet, string fichier, System.Type mytype)
{
	if (objet == null)
		return false;

	StreamWriter stream = new StreamWriter(fichier);
	XmlSerializer serializer = new XmlSerializer(mytype);
	serializer.Serialize(stream, objet);
	stream.Close();

	return true;
}
```

Exemple de flux obtenu par sérialisation XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ObjSerialization xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
	<_Couleur />
	<Nom>Exemple de sérialisation SOAP</Nom>
	<Couleur />
</ObjSerialization>
```

## 3.2 - Déserialisation

```
public static object Deserialize(string fichier, System.Type objtype)
{
	if (!File.Exists(fichier))
		return null;

	XmlSerializer deserializer = new XmlSerializer(objtype);

	StreamReader stream = new StreamReader(fichier);

	object result = null;
	result = deserializer.Deserialize(stream);
	stream.Close();

	return result;
}
```

# 4 - Sérialisation SOAP

La sérialisation SOAP est similaire à la sérialisation XML quant au format de sortie mais diffère en bien des points.

> *SOAP est un dérivé de XML. En plus des noms des champs, il fournit d'autres informations complémentaires, mais nous n'allons pas nous attarder dessus, nous dépasserions le cadre de ce tutoriel.*

Avant toutes choses, la classe que vous allez sérialiser doit porter l'attribut SerializableAttribute, sans quoi vous vous retrouverez avec une jolie exception de type SerializationException vous indiquant que l'objet XXX n'est pas marqué comme sérialisable.

Ensuite, il vous faut ajouter la référence à la libraire du .Net Framework System.Runtime.Serialization.Formatters.Soap et importer le namespace du même nom.

## 4.1 - Sérialisation

```
public static bool Serialize(object objet, string fichier)
{
        if (objet == null)
                return false;

        FileStream stream = new FileStream(fichier, FileMode.Create);

        SoapFormatter serializer = new SoapFormatter();
        serializer.Serialize(stream, objet);
        stream.Close();

        return true;
}
```

Exemple de flux obtenu par sérialisation SOAP

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <SOAP-ENV:Body>
                <a1:ObjSerialization id="ref-1"
xmlns:a1="http://schemas.microsoft.com/clr/nsassem/TestSerializationVB/TestSerializationVB%2C%20
                Version%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
                        <_Nom id="ref-4">Exemple de sÃ©rialisation SOAP</_Nom>
                        <_Couleur>
                                <name xsi:null="1"/>
                                <value>0</value>
                                <knownColor>28</knownColor>
                                <state>1</state>
                        </_Couleur>
                </a1:ObjSerialization>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## 4.2 - Désérialisation

```
public static object Deserialize(string fichier)
{
        if (!File.Exists(fichier))
                return null;

        SoapFormatter deserializer = new SoapFormatter();

        FileStream stream = new FileStream(fichier, FileMode.Open);
```

```
            object result = null;
            result = deserializer.Deserialize(stream);
            stream.Close();

            return result;
    }
```

# 5 - Sérialisation Binaire

La sérialisation binaire n'a rien à voir avec les deux types de sérialisation que nous venons de voir au niveau du format de sortie.

Pour pouvoir utiliser la sérialisation binaire, importez le namespace System.Runtime.Serialization.Formatters.Binary.BinaryFormatter.

## 5.1 - Sérialisation

```
public static bool Serialize(object objet, string fichier)
{
        if (objet == null)
            return false;

        FileStream stream = new FileStream(fichier, FileMode.Create);

        System.Runtime.Serialization.Formatters.Binary.BinaryFormatter serializer =
                new System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();
        serializer.Serialize(stream, objet);
        stream.Close();

        return true;
}
```

## 5.2 - Désérialisation

```
public static object Deserialize(string fichier)
{
        if (!File.Exists(fichier))
                return null;

        System.Runtime.Serialization.Formatters.Binary.BinaryFormatter deserializer =
                new System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();

        FileStream stream = new FileStream(fichier, FileMode.Open);

        object result = null;
        result = deserializer.Deserialize(stream);
        stream.Close();

        return result;
}
```

## 6 - Récapitulatif

Pour clarifier un peu les avantages et les inconvénients de ces différentes méthodes, voici un petit récapitulatif sous forme de tableau pour comparer les différentes possibilités proposées par les sérialiseurs du Framework.NET :

| | XML | SOAP | Binaire |
|---|---|---|---|
| "Human Readable" | Oui | Oui | Non |
| Communication avec Mono sous Linux | Oui | Oui | Non |
| Sérialisation de types non standards | Non | Oui | Oui |
| Sérialisation des éléments privés | Non | Oui | Oui |
| Sérialisation des champs | Non | Oui | Oui |
| Sérialisation des propriétés | Oui | Non | Non |
| Utilisation faite dans la plate-forme .Net | - | Services Web | Remoting |

Pour tester un peu les performances de ces différentes méthodes, j'ai sérialisé et désérialisé dans la foulée notre objet ObjSerialization 50 000 fois.

La machine utilisée pour ces tests est un Dell Inspiron 8200 doté d'un Pentium IV à 1,7GHz et de 1Go de Ram sous Windows XP SP2 avec le Framework v2.0.50727.

| Essai | XML | SOAP | Binaire |
|---|---|---|---|
| Essai n°1 | 1:06.4 | 1:21.1 | 0:50.4 |
| Essai n°2 | 1:03.7 | 1:21.8 | 0:49.5 |
| Essai n°3 | 1:03.4 | 1:21.2 | 0:49.5 |
| Moyenne | 1:04.5 | 1:21.4 | 0:49.8 |

Comme vous pouvez le constater, malgré qu'elle soit complète la sérialisation binaire est celle qui offre les meilleures performances. La sérialisation XML et SOAP étant similaire, mais la première n'étant pas aussi précise que la seconde, les performances sont donc meilleures avec XML qu'avec SOAP mais au prix d'une perte d'information dans certains cas (ici par exemple).

Sachez que la sérialisation binaire est utilisée en remoting du fait de sa rapidité par rapport aux deux autres méthodes fournies par le Framework .Net.

La sérialisation SOAP, elle, est utilisée pour les Services Web puisque son format dérivé de XML (SOAP) permet l'interoperabilité entre les plate-formes (java, delphi, php, ...) et passe plus facilement au travers des firewalls.

Je pense que ce petit récapitulatif vous aidera à vous orienter vers la méthode de sérialisation la plus adaptée à vos besoins.

## Conclusion

Avec la sérialisation, vous voyez qu'il est possible de sauvegarder et surtout de récupérer vos informations en quelques lignes en utilisant les outils mis à votre disposition par le Framework .NET. Si toutefois vous n'étiez pas satisfait, vous pouvez toujours repartir des sérialiseurs mis à votre disposition pour créer le votre sans trop de travail, mais cette application fera l'objet d'un prochain tutoriel.

Un grand merci à Jean-Marc Rabilloud pour ses explications complémentaires sur la sérialisation XML et à Xo et à Cécile Muno pour leurs remarques et corrections.

## Téléchargements