# La sérialisation

*La sérialisation* est l'opération qui transforme la description de l'état d'un objet en une séquence de valeurs. La *désérialisation* est l'opération inverse qui permet à un objet d'obtenir son état à partir d'une séquence de données.
Avec le framework .NET, il y a plusieurs moteurs de sérialisation. On peut réaliser notamment une sérialisation binaire (l'état de l'objet est décrit par une séquence d'octets) ou une sérialisation XML (l'état de l'objet est décrit au format XML).

**A quoi sert la sérialisation ?**
Une fois sérialisé, l'état d'un objet peut être soit sauvegardé en vue d'être reconstitué plus tard (sérialisation dans un fichier), soit transmis via un flux de donnée pour que l'objet soit reconstitué dans une application distante (sérialisation dans un flux réseau).

Le mécanisme de sérialisation lorsqu'il est associé à une sauvegarde en fichier, permet également d'avoir des objets **persistants** : les objets peuvent être régénérés à partir d'un fichier et peuvent donc exister au-delà de l'arrêt de l'application (les bases de données peuvent également servir à rendre les objets **persistants** !).

## I. Sérialisation binaire

Le premier moteur de sérialisation, dans les cas généraux où l'on ne souhaite pas intervenir dans le processus de sérialisation, est très simple à utiliser.

En C#, il suffit de marquer la classe avec l'attribut [Serializable].

```
[Serializable]
 class Personne
 {
     public string Nom;
     public int Age;
     public Personne(string nom, int age){    Nom = nom;    Age = age; }
     public override string ToString() {
         return Nom + ":" + Age.ToString() + " ans";
     }

 }
```

Dès lors, il est possible de sérialiser/désérialiser un objet **`Personne`** à partir d'un flux de données.
L'appel du mécanisme de sérialisation nécessite un objet *formatter*. Seuls deux formateurs sont disponibles : `BinaryFormatter et SoapFormatter`

Les namespaces utilisés pour ces différentes classes sont

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.Runtime.Serialization.Formatters.Soap;
using System.IO; // flux IO
```

### Ex : Sérialisation vers un flux FileStream (avec `BinaryFormatter`):

```
Personne p1 = new Personne("Dupont", 24);
BinaryFormatter format = new BinaryFormatter();
using(FileStream fs = new FileStream("ObjetPers.dat",  FileMode.OpenOrCreate,
                                                        FileAccess.Write))
  {
         format.Serialize(fs,p1);  // sérialise l'objet p -> fichier ObjetPers.dat
  }
```

**Ex: Désérialisation depuis un flux FileStream (avec `BinaryFormatter`):**

```
BinaryFormatter format = new BinaryFormatter();
using(FileStream fs = new FileStream("ObjetPers.dat",
                                     FileMode.Open,
                                     FileAccess.Read))
{
        // reconstitue l'objet p2 depuis ObjetPers.dat
        Personne p2 = (Personne) format.Deserialize(fs);
}
```

**Remarque : Les collections (`Array`,`ArrayList`,`List<>`,...) de type sérialisables sont sérialisables et les types primitifs .NET sont sérialisables.**

**Ex : Sérialisation d'un tableau d'objets sérialisables :**

```
Personne[] tab= new Personne[4];

tab[0]=new Personne("Dupont", 24);
tab[1]=new Personne("Durant", 18);
tab[2]=new Personne("Martin", 13);
tab[3]=new Personne("Hardouin", 13);

BinaryFormatter format = new BinaryFormatter();
using(FileStream fs = new FileStream("ObjetPers.dat", FileMode.OpenOrCreate,
                                                      FileAccess.Write))
{
        format.Serialize(fs,tab);
}
```

**Ex: Désérialisation d'un tableau d'objets**

```
BinaryFormatter format = new BinaryFormatter();
using(FileStream fs = new FileStream("ObjetPers.dat",
                                     FileMode.Open, FileAccess.Read))
{
        Personne[] tab = (Personne[]) format.Deserialize(fs);
}
```

## II. Sérialisation XML (namespace `System.Xml.Serialization`)

**NA** : *ce moteur de sérialisation est moins simple à mettre en oeuvre. En outre, une alternative est d'exploiter les données XML à travers des adaptateurs de flux XML.*

Le moteur de sérialisation fourni via la classe **`XmlSerializer`** permet de décrire l'état d'un objet au format XML. Ce moteur ne nécessite pas de marquer les membres par des attributs, en revanche, seuls les champs publics sont sérialisés (ce qui limite un peu l'utilisation de ce moteur).

```
public class Personne
 {
   public string Nom; public int Age;

 public Personne(){  }

 public Personne(string N, int a){ Nom=N; Age = a; }

 public override string ToString(){
     return "Nom=" + Nom + " \tAge=" + Age.ToString();
 }
 }
```

La sérialisation s'effectue à l'aide d'un objet **`XmlSerializer`**.

```
try
{
    XmlSerializer xSerial = new XmlSerializer(typeof(Personne));

  using (Stream fs =File.Open("personne.xml",FileMode.OpenOrCreate,
                                                    FileAccess.Write))

  {
      Personne p = new Personne("toto",12);
      xSerial.Serialize(fs,p);

   }
}
catch (Exception exc)
{
    // ...
}
```

Le résultat produit au format XML est

```
<?xml version="1.0"?>
<Personne xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Nom>toto</Nom>
  <Age>12</Age>
</Personne>
```

La désérialisation reconstitue un objet depuis un flux XML

```
      XmlSerializer xSerial = new XmlSerializer(typeof(Personne));
      using (Stream fs = File.OpenRead("personne.xml"))
      {
        Personne p =(Personne)xSerial.Deserialize(fs);
       }
```