

La sérialisation

Sommaire

La sérialisation	1
1 Introduction.....	2
2 Les bases de la sérialisation.....	3
2.1 Sérialisation Binaire.....	3
2.2 Dé-sérialiser un objet	5
2.3 Rendre ses classes sérialisable	6
2.3.1 Empêcher la sérialisation d'un membre.....	6
2.3.2 Gestion de version d'objets sérialisable	8
2.4 Sérialisation SOAP	9
2.4.1 Avoir accès à SoapFormatter.....	9
2.4.2 Utilisation de SoapFormatter	11
2.5 Sérialisation personnalisée.....	13
2.5.1 Les évènements de sérialisation	15
2.5.2 Obtenir un contexte de sérialisation.....	18
3 Sérialisation XML	19
3.1 Sérialisation/Dé-sérialisation d'objets en XML.....	19
3.2 Contrôler la sérialisation	23
3.3 Utilisation de schémas XML	26
4 Conclusion	27

1 Introduction

Dans les chapitres précédents, nous avons vu comment accéder aux données via les flux, comment utiliser le bon encodage et comment stocker ces données dans des tableaux dynamiques.



Dans cette partie, nous allons aborder un aspect important du stockage de données : La sérialisation. La sérialisation, c'est le fait de faire une copie des données d'un objet en mémoire à un instant T dans un fichier, un flux réseaux etc. (Pour envoyer des objets à travers un réseau (utilisation de Webservices, Remoting...) ou enregistrer le travail d'une application, d'un jeu vidéo...).

Dans un premier temps, nous verrons comment sérialiser nos données lorsque celles-ci sont destinées à d'autres applications .NET ; puis nous verrons comment sérialiser en XML. Pour finir, nous verrons comment créer nos propres formats de sérialisation.

L'espace de nom `System.Runtime.Serialization` contient la majorité des outils donnant accès à la sérialisation des objets.

2 Les bases de la sérialisation

2.1 Sérialisation Binaire

La sérialisation d'un objet en binaire est la plus basique. Elle convertit un objet en une suite d'octets peu lisibles (voir pas du tout). La sérialisation binaire est la seule à pouvoir sérialiser des objets courant comme des objets de type de graphique.

Elle se fait très facilement en utilisant les deux espaces de nom `System.Runtime.Serialization` et `System.Runtime.Serialization.Formatters.Binary`

Nous allons également utiliser un flux pour enregistrer l'objet sérialisé dans un fichier:

```
'VB
<Serializable()> _
Class Personnage
    Private _nom As String
    Private _age As Integer
    Private _infos As String
    Public Property infos() As String
        Get
            Return _infos
        End Get
        Set(ByVal value As String)
            _infos = value
        End Set
    End Property
    Public Property nom() As String
        Get
            Return _nom
        End Get
        Set(ByVal value As String)
            _nom = value
        End Set
    End Property
    Public Property age() As Integer
        Get
            Return _age
        End Get
        Set(ByVal value As Integer)
            _age = value
        End Set
    End Property
End Clas
Sub Main()
    Dim perso As Personnage = New Personnage()
    perso.infos = "Informations sur votre personnage"
    Dim fichierIn As FileStream = New FileStream("c:\serialized.txt",
    FileMode.Create)
    Dim bf As BinaryFormatter = New BinaryFormatter()
    bf.Serialize(fichierIn, perso)
    fichierIn.Close()
End Sub
```

```
//C#
[Serializable]
class Personnage
{
    private string _nom;
    private int _age;
    private string _infos;

    public string infos { get { return _infos; } set{ _infos = value; } }
    public string nom { get { return _nom; } set{ _nom = value; } }
    public int age { get { return _age; } set{ _age = value; } }
}

static void Main(string[] args)
{
    string infos = "Informations sur votre personnage";
    Personnage perso = new Personnage();
    perso.infos = infos;

    FileStream fichierIn = new FileStream(@"c:\serialized.txt",
    FileMode.Create);
    BinaryFormatter bf = new BinaryFormatter();

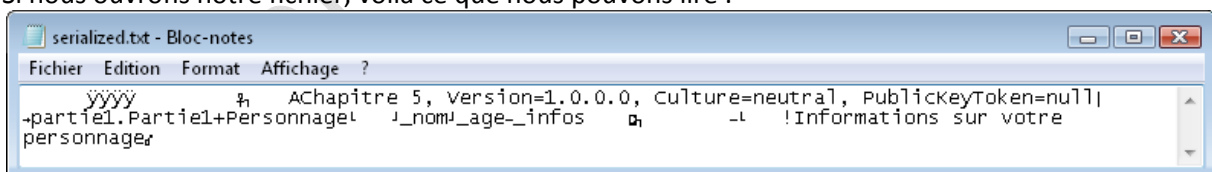
    bf.Serialize(fichierIn, perso);
    fichier.Close();
}
```

Nous avons donc cr e une classe, qui est s rialisable (elle est pr ced e par l'attribut `Serializable`, nous reviendrons sur cette notion plus bas) et qui poss de trois attributs et trois accesseurs.

Dans notre `Main`, nous cr ons une chaine de caract re, nousinstancions un objet de type `Personnage` et nous lui assignons la chaine d'information.

Ensuite nous cr ons un fichier qui va contenir les donn es et le `BinaryFormatter` qui va formater nos donn es en binaire. Enfin nous utilisons la m thode `Serialize` de notre `BinaryFormatter` qui prend le fichier en premier param tre et l'objet   s rialiser en second. On n'oublie pas de fermer le flux sur le fichier.

Si nous ouvrons notre fichier, voila ce que nous pouvons lire :



Nous devinons les divers membres et donn es enregistr es sous forme ASCII ainsi que des caract res  tranges qui d crivent les donn es en vue de les d -s rialiser.

2.2 Dé-sérialiser un objet

Nous avons vu comment sérialiser un objet, nous allons maintenant voir comment le dé-sérialiser. La manipulation n'est guère plus compliquée, en effet le processus est similaire :

```
'VB
Sub Main()
    Dim bf As BinaryFormatter = New BinaryFormatter()
    Dim fichierOut As FileStream = New FileStream("c:\serialized.txt",
    FileMode.Open)
    Dim NewPerso As Personnage = CType(bf.Deserialize(fichierOut),
    Personnage)
    fichierOut.Close()
    Console.WriteLine(NewPerso.infos)
    Console.Read()
End Sub

//C#
static void Main(string[] args)
{
    FileStream fichierOut = new FileStream(@"c:\serialized.txt",
    FileMode.Open);

    BinaryFormatter bf = new BinaryFormatter();

    Personnage NewPerso = (Personnage) bf.Deserialize(fichierOut);
    fichierOut.Close();

    Console.WriteLine(NewPerso.infos);
    Console.Read();
}
```

Informations sur votre personnage

On ouvre donc le fichier à dé-sérialiser, et on crée notre objet BinaryFormatter qui va nous permettre de dé-sérialiser (on peut aussi réutiliser le précédent dans notre exemple).

Ensuite nous créons un objet "NewPerso" de type Personnage et on lui assigne le contenu du fichier, dé-sérialisé par la méthode `Deserialize` et casté en type Personnage. Le cast est important car l'objet dé-sérialisé est de type Object. Si vous omettez le cast, il y aura une erreur à la compilation.

Enfin on affiche le champ infos de "NewPerso", et on voit qu'il est identique à celui qui a été sérialisé.

2.3 Rendre ses classes sérialisable

Nous avons vu dans la [partie 2.1](#) qu'il fallait 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, y compris les membres « private ».

```
'VB
<Serializable()>

//C#
[Serializable]
```

Si vous ne savez pas si vous allez sérialiser votre classe plus tard, ou si un développeur aura besoin de sérialiser une classe héritant de la votre, ajoutez l'attribut dans le doute.

Vous pouvez également configurer votre classe pour indiquer des champs à ne pas sérialiser ou gérer la comptabilité entre deux versions d'un logiciel.

2.3.1 Empêcher la sérialisation d'un membre

Vous pouvez avoir besoin, dans vos classes, de créer des variables qui n'ont pas besoin d'être sérialisées (variable contenant le résultat de calculs en général). Aussi, pour limiter la taille des objets sérialisés, nous pouvons utiliser l'attribut `NonSerialized` placé avant chaque membre qui ne doit pas nécessairement être sérialisé (Les accesseurs ne sont pas indiqués ici mais doivent toujours être placés dans vos codes):

```
'VB
<Serializable()> _
Class Personnage
    Private _nom As String
    <NonSerialized()> Private _age As Integer
    Private _infos As String
End Class

//C#
[Serializable]
class Personnage
{
    private string _nom;
    [NonSerialized] private int _age;
    private string _infos;
}
```

Après la dé-sérialisation, le membre `_age` n'existe pas, il n'est donc pas instancié. Voici ce qui est retourné si on affiche `_age` :

```
0
```

Si vous utilisez `_age` dans une méthode qui sera sérialisé, aucune exception ou erreur ne sera renvoyée, en revanche, la valeur qui aura été assigné à `_age` sera perdue.

Il est possible, en implémentant l'interface `IDeserializationCallback`, d'utiliser la méthode `OnDeserialization` qui va vous permettre de définir le comportement de la classe lors d'une dé-sérialisation. Cela permet d'effectuer quelques opérations pour, par exemple, retrouver la valeur des membres qui n'ont pas été sérialisés.

Afin de mieux comprendre, voici un exemple :

```
'VB
<Serializable()> _
Class Personnage
    Implements IDeserializationCallback

    Private _nom As String
    <NonSerialized()> Private _age As Integer
    Private _infos As String

    Public Sub New()
    End Sub
    Public Sub OnDeserialization(ByVal sender As Object) Implements
System.Runtime.Serialization.IDeserializationCallback.OnDeserialization
        _age = 42
    End Sub
End Class

//C#
[Serializable]
class Personnage : IDeserializationCallback
{
    private string _nom;
    [NonSerialized] private int _age;
    private string _infos;

    public Personnage()
    {;}

    public void OnDeserialization(object sender)
    {
        _age = 42;
    }
}
```

Si nous affichons la valeur de `_age` après dé-sérialisation :

42

La valeur de `_age` a été assignée à la dé-sérialisation, on ne retrouve aucunes traces de `_age` dans le fichier `serialized.txt`.

2.3.2 Gestion de version d'objets sérialisable

Il se peut que lorsque vous développez votre programme, vous soyez amené à modifier une classe, pour lui rajouter des membres par exemple. Si vous aviez sérialisé un objet de la version précédente de votre classe et que vous tentez de le dé-sérialiser sur la nouvelle version, il est possible que des erreurs inattendues apparaissent. Aussi, le Framework .NET fournit également des outils permettant de gérer la compatibilité d'un objet sérialisé entre plusieurs versions différentes d'un programme.

Pour cela, nous pouvons :

- Créer un sérialiseur personnalisé qui peut gérer les multiples versions (traité dans la [partie 2.5](#)).
- Utiliser l'attribut `OptionalField` pour rendre optionnels certains membres d'une classe.

Voyons un exemple de l'utilisation d'`OptionalField` :

```
'VB
<Serializable()> _
Class Personnage
    Implements IDeserializationCallback

    Private _nom As String
    <NonSerialized()> Private _age As Integer
    Private _infos As String
    <OptionalField()> Private _region As String
    Private _majeur As Boolean

    Public Sub New()
        _age = 18
    End Sub

    Public Sub OnDeserialization(ByVal sender As Object) Implements
System.Runtime.Serialization.IDeserializationCallback.OnDeserialization
        _majeur = _age > 18
        _region = "Groland"
    End Sub
End Class
```

```
//C#
[Serializable]
class Personnage : IDeserializationCallback
{
    private string _nom;
    [NonSerialized] private int _age;
    private string _infos;
    [NonSerialized] private string _region;

    public Personnage()
    {
        _age = 18;
    }

    public void OnDeserialization(object sender)
    {
        _majeur = _age > 18;
        _region = "Groland";
    }
}
```

Nous avons donc notre champ `region` qui est optionnel, c'est-à-dire que si nous souhaitons dé-sérialiser un objet qui ne possédait pas encore ce membre, la région sera retrouvée à la dé-sérialisation et mise par défaut à la valeur "Groland".

2.4 Sérialisation SOAP

Nous avons utilisé jusque là un format binaire pour sérialiser nos objets. Sachez qu'il existe également dans l'espace de nom `System.Runtime.Serialization.Formatters.Soap` un autre format pour sérialiser les objets : `SoapFormatter`.

`SoapFormatter` permet de sérialiser les objets dans un format dérivé du XML dans le but d'utiliser votre application avec des WebServices SOAP (Simple Object Access Protocol). Il est possible d'utiliser `SoapFormatter` pour envoyer des données vers des applications capable de lire du XML, mais préférez l'utilisation de la sérialisation XML standard (sera abordée plus bas).

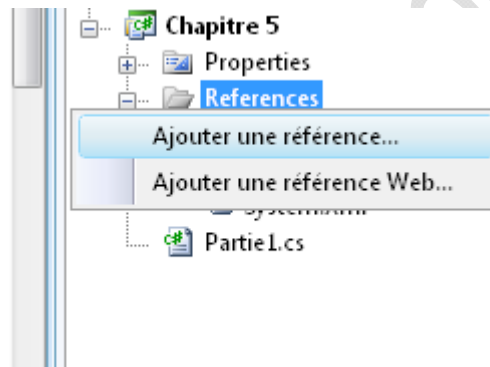
Dans tous les cas, `SoapFormatter` est plus portable que `BinaryFormatter`, mais plus lourd.

2.4.1 Avoir accès à `SoapFormatter`

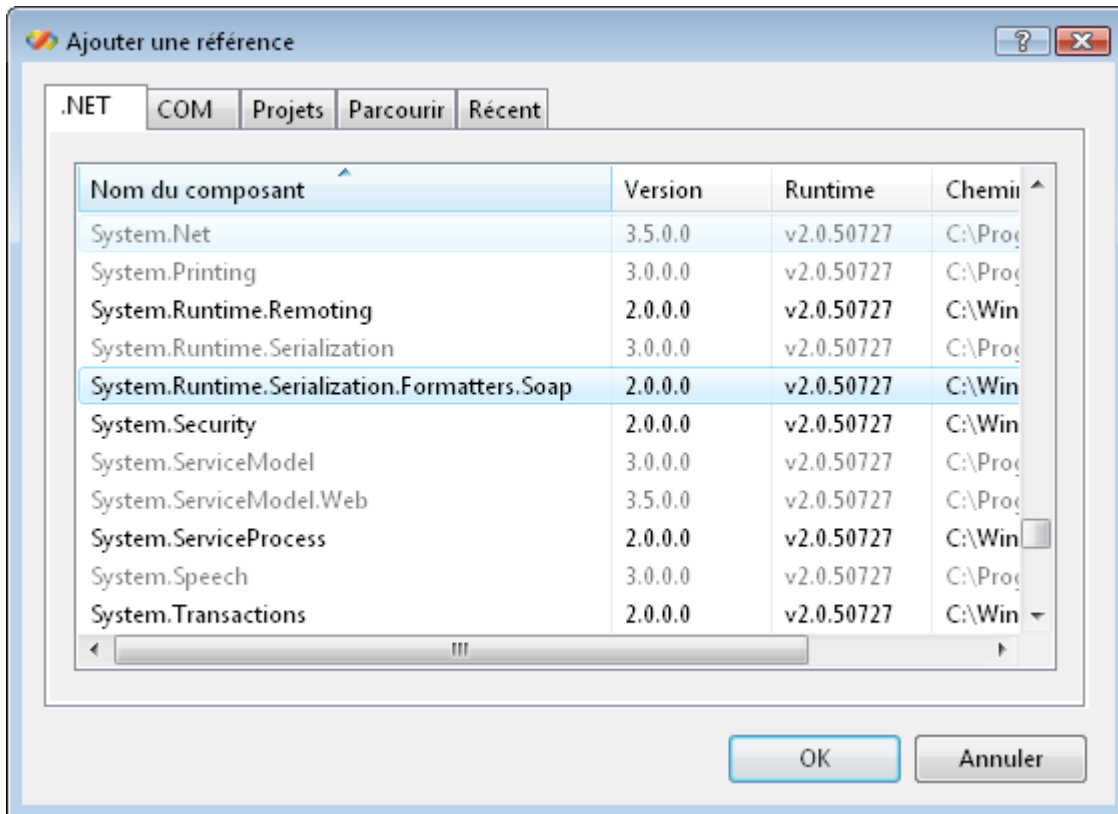
Alors que `BinaryFormatter` est disponible par défaut dans votre projet, il vous faudra rajouter manuellement l'assembly contenant `SoapFormatter`.

- Ajouter la référence en C#

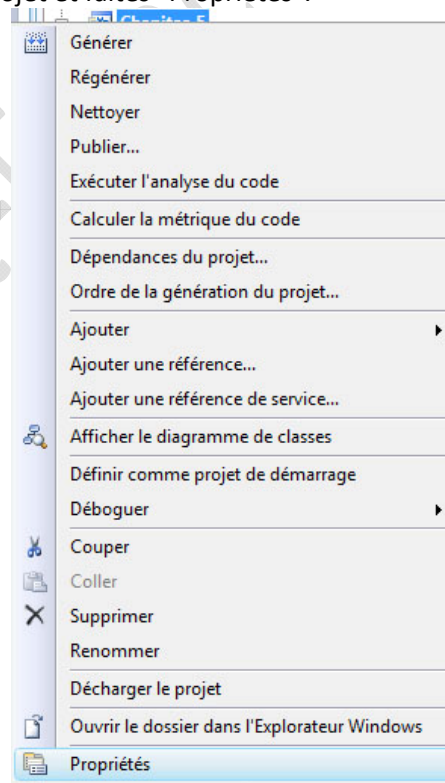
Pour se faire, cliquez avec le bouton droit sur `References` dans l'explorateur de solutions puis faites "Ajouter une référence..."



Ensuite, cherchez dans la liste `System.Runtime.Serialization.Formatters.Soap` et faites "Ok" :



- Ajouter la référence en VB.NET
Cliquez droit sur votre projet et faites "Propriétés".



Une fois dans les propriétés, allez sur l'onglet "Références" et cliquez sur le bouton "Ajouter". Vous devriez avoir la même fenêtre de sélection des références qu'en C#. Suivez les mêmes opérations et validez.

Il ne vous reste plus qu'à rajouter au début de votre projet :

```
'VB
Imports System.Runtime.Serialization.Formatters.Soap

//C#
using System.Runtime.Serialization.Formatters.Soap;
```

2.4.2 Utilisation de SoapFormatter

L'utilisation de SoapFormatter est en fait totalement identique à celle de BinaryFormatter. Si vous changez le code précédent en utilisant la classe SoapFormatter, voilà ce que vous obtenez si vous sérialisez la classe Personnage et que vous essayez de le lire avec un éditeur de texte :

```
1 <SOAP-ENV:Envelope
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
5   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
6   xmlns:clr="http://schemas.microsoft.com/soap/encoding clr/1.0"
7   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
8   <SOAP-ENV:Body>
9     <a1:Partie12_x002B_Personnage id="ref-1"
10    xmlns:a1="http://schemas.microsoft.com/clr/nsassem/partie12/Chapit
11      <_nom xsi:null="1"/>
12      <_age>0</_age>
13      <_infos id="ref-3">Informations sur votre personnage</_infos>
14    </a1:Partie12_x002B_Personnage>
15  </SOAP-ENV:Body>
16 </SOAP-ENV:Envelope>
```

Nous avons indenté un petit peu le code et coupé une partie pour que vous vous rendiez mieux compte. Nous retrouvons donc nos variables `_nom` et `_age` à null car on ne leur assigne aucune valeur et `_infos` contient la chaîne "Informations sur votre personnage".

Vous pouvez également configurer la façon dont les données sont sérialisés grâce à quelques attributs spécifiques à SoapFormatter. Vous en apprendrez plus sur l'utilisation de ces attributs dans la partie suivante. Sachez que les attributs de Soap se trouvent dans l'espace de nom `System.Xml.Serialization` :

Attribut	S'applique sur	Description
<code>SoapAttribute</code>	Champ publique, propriété, paramètres, valeurs de retour	Spécifie à <code>XmlSerializer</code> que l'élément sera sérialisé en tant qu'attribut.
<code>SoapElement</code>	Champ publique, propriété, paramètres, valeurs de retour	Spécifie à <code>XmlSerializer</code> que le membre sera sérialisé en tant qu'élément.
<code>SoapEnum</code>	Valeurs d'énumérations	Indique des propriétés de sérialisation des valeurs à l'intérieur d'une énumération.
<code>SoapIgnore</code>	Propriété et champs publics	La propriété ou le champ est ignoré au moment de la sérialisation
<code>SoapInclude</code>	Classe parente de classe dérivée ou valeur de retour de méthodes dans des documents WSDL	Permet à l'outil de sérialisation d'inclure un type particulier lors d'une sérialisation ou d'une dé-sérialisation.

2.5 Sérialisation personnalisée

Bien que les méthodes de sérialisation vues précédemment couvrent un bon nombre de besoins, il se peut que vous ayez besoin de customiser le processus de sérialisation.

Pour cela, vous devez remplacer le processus de sérialisation du Framework .NET en implémentant l'interface `ISerializable` dans vos classes.

Cette interface propose la méthode `GetObjectData` qui sera appelée au moment de la sérialisation et vous devez créer un constructeur qui sera appelé quand la classe sera dé-sérialisée.

ATTENTION : Seul la méthode `GetObjectData` est à implémenter obligatoirement ! Le constructeur de dé-sérialisation n'est pas obligatoire mais si vous ne le mettez pas, vous ne pourrez pas dé-sérialiser votre objet.

Dans la méthode `GetObjectData` vous allez devoir remplir un objet `SerializationInfo` en ajoutant tous les attributs que vous souhaitez sérialiser grâce à la méthode `AddValue` qui prend en premier paramètre une clé et en second votre variable.

Le constructeur dédié à la dé-sérialisation va pouvoir récupérer les valeurs grâce à leurs clés.

Voici un exemple en reprenant notre classe calcul :

```
'VB
<Serializable()> _
Class Personnage
    Implements IDeserializationCallback, ISerializable

    Private _nom As String
    <NonSerialized()> Private _age As Integer
    Private _infos As String
    <OptionalField()> Private _region As String
    Private _majeur As Boolean
    Public Sub New()
        _age = 18
    End Sub
    Public Sub GetObjectData(ByVal info As
System.Runtime.Serialization.SerializationInfo, ByVal context As
System.Runtime.Serialization.StreamingContext) Implements
System.Runtime.Serialization.ISerializable.GetObjectData
        info.AddValue("Nom", _nom)
        info.AddValue("Info", _infos)
        info.AddValue("Majeur", _majeur)
    End Sub
    Public Sub New(ByVal info As
System.Runtime.Serialization.SerializationInfo, ByVal context As
System.Runtime.Serialization.StreamingContext)
        _majeur = info.GetBoolean("Majeur")
        _infos = info.GetString("Info")
        _nom = info.GetString("Nom")
    End Sub
    Public Sub OnDeserialization(ByVal sender As Object) Implements
System.Runtime.Serialization.IDeserializationCallback.OnDeserialization
        _majeur = _age > 18
        _region = "Groland"
    End Sub
End Class
```

```
//C#
[Serializable]
class Personnage : IDeserializationCallback, ISerializable
{
    private string _nom;
    [NonSerialized] private int _age;
    private string _infos;
    [OptionalField] private string _region;
    private bool _majeur;

    public Personnage()
    {
        _age = 18;
    }

    public void OnDeserialization(object sender)
    {
        _majeur = _age > 18;
        _region = "Groland";
    }

    public void GetObjectData(SerializationInfo info, StreamingContext
context)
    {
        info.AddValue("Nom", nom);
        info.AddValue("Majeur", majeure);
        info.AddValue("Info", infos);
    }
    public Personnage(SerializationInfo info, StreamingContext context)
    {
        majeure = info.GetBoolean("Majeur");
        nom = info.GetString("Nom");
        infos = info.GetString("Info");
    }
}
```

On a donc deux constructeurs : un ne prenant aucun argument et celui appelé automatiquement à la dé-sérialisation. A l'intérieur on récupère les valeurs avec les clés, on spécifie que ce sont des booléen et des chaînes de caractères, mais nous aurions pu récupérer tous les types de base du Framework. Cela dépend du type de donnée qui a été sérialisée.

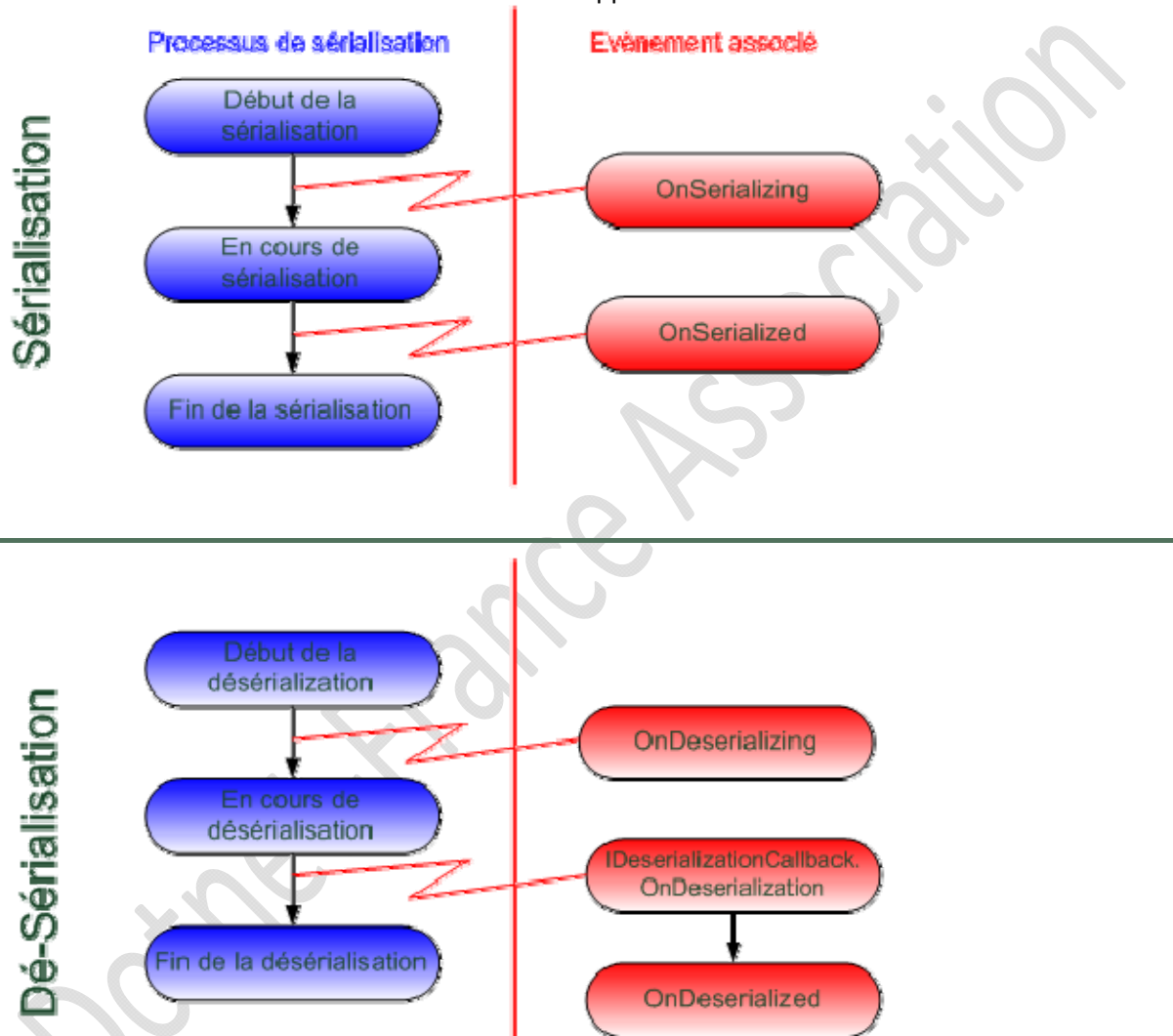
La méthode `GetObjectData` ajoute à l'objet `SerializationInfo` les valeurs que l'on souhaite sérialiser, référencés par les clés "Nom", "Info" et "Majeur".

2.5.1 Les évènements de sérialisation

Quand vous utilisez le formateur `BinaryFormatter` pour sérialiser vos données, vous pouvez utiliser des évènements sous forme d'attributs afin d'agir pendant les processus de sérialisation et désérialisation.

- `Serializing`, généré au cours du processus de sérialisation.
- `Serialized`, généré une fois que l'objet a été sérialisé.
- `Deserializing`, généré pendant la dé-sérialisation.
- `Deserialized`, généré quand l'objet a été dé-sérialisé.

Le schéma ci-dessous nous montre l'ordre d'appel des évènements:



Ces évènements sont un peu particuliers. En effet, nous avons vu qu'il était possible de créer des évènements en utilisant les délégués. Dans ce cas-ci, les évènements sont des attributs à placer avant la ou les méthodes qui seront appelée lors de l'évènement indiqué. Chaque méthode doit prendre en argument un objet de type `StreamingContext`:

```
'VB
<OnDeserialized()> _
Public Sub SetAge(ByVal context As StreamingContext)
End Sub

//C#
[OnDeserialized]
public void SetAge(StreamingContext context){}
```

Il faut savoir qu'il est tout à fait possible de faire en sorte qu'une seule méthode soit appelée dans plusieurs évènements. Pour cela, il suffit d'ajouter plusieurs attributs au dessus de la méthode à exécuter.

L'exemple ci-dessous présente comment utiliser l'évènement `OnDeserialized` :

```
'VB
<Serializable()> _
Class Personnage
    Implements IDeserializationCallback

    Private _nom As String
    <NonSerialized()> Private _age As Integer
    Private _infos As String
    <OptionalField()> Private _region As String
    Private _majeur As Boolean
    Public Sub New()
        _age = 18
    End Sub
    <OnDeserialized()> _
    Public Sub SetAge(ByVal context As StreamingContext)
        _age = 42
    End Sub
    Public Sub GetObjectData(ByVal info As
System.Runtime.Serialization.SerializationInfo, ByVal context As
System.Runtime.Serialization.StreamingContext) Implements
System.Runtime.Serialization.ISerializable.GetObjectData
        info.AddValue("Nom", _nom)
        info.AddValue("Info", _infos)
        info.AddValue("Majeur", _majeur)
    End Sub
    Public Sub New(ByVal info As
System.Runtime.Serialization.SerializationInfo, ByVal context As
System.Runtime.Serialization.StreamingContext)
        _majeur = info.GetBoolean("Majeur")
        _infos = info.GetString("Info")
        _nom = info.GetString("Nom")
    End Sub
    Public Sub OnDeserialization(ByVal sender As Object) Implements
System.Runtime.Serialization.IDeserializationCallback.OnDeserialization
        _majeur = _age > 18
        _region = "Groland"
    End Sub
End Class
```

```
[Serializable]
class Personnage : IDeserializationCallback, ISerializable
{
    private string _nom;
    [NonSerialized] private int _age;
    private string _infos;
    [OptionalField] private string _region;
    private bool _majeur;

    public Personnage()
    {
        _age = 18;
    }

    [OnDeserialized]
    public void SetAge(StreamingContext context)
    {
        _age = 42;
    }

    public void OnDeserialization(object sender)
    {
        _majeur = _age > 18;
        _region = "Groland";
    }

    public void GetObjectData(SerializationInfo info, StreamingContext
context)
    {
        info.AddValue("Nom", nom);
        info.AddValue("Majeur", majeure);
        info.AddValue("Info", infos);
    }
    public Personnage(SerializationInfo info, StreamingContext context)
    {
        majeure = info.GetBoolean("Majeur");
        nom = info.GetString("Nom");
        infos = info.GetString("Info");
    }
}
```

Comme vous pouvez le voir, nous avons utilisé l'attribut `OnDeserialized` au dessus de notre méthode "SetAge" afin de calculer le total après la dé-sérialisation des membres.

Notre méthode "SetAge" prend en paramètre un objet `StreamingContext` et retourne un `void`.

Utiliser les attributs de `BinaryFormatter` simplifie le développement, préférez les à `ISerializable` si vous utilisez `BinaryFormatter`.

2.5.2 Obtenir un contexte de sérialisation

La classe `StreamingContext` permet d'obtenir le contexte dans lequel l'objet a été sérialisé. Ce contexte peut contenir n'importe quelles informations, comme par exemple l'origine ou la destination de l'objet. La classe compte deux propriétés `Context` et `State` :

- La propriété `Context` contient une référence vers les informations du contexte.
- La propriété `State` permet de définir et de récupérer l'origine et la destination des objets sérialisés et dé-sérialisés. Vous devez utiliser l'énumération `StreamingContextStates` pour définir les valeurs.

Voici les différents membres de l'énumération `StreamingContextStates` :

Membre	Description
<code>CrossProcess</code>	Indique que la source ou la destination est un processus différent sur une même machine.
<code>CrossMachine</code>	Spécifie que la source ou la destination est sur une machine différente.
<code>File</code>	Indique que la source ou la destination est un fichier.
<code>Persistence</code>	Indique que la source ou la destination est un fichier, une base de donnée ou autre stockage dit persistant (ou non-volatile)
<code>Remoting</code>	Réfère à une source ou une destination distante inconnue. On ne peut savoir si c'est sur la même machine ou une autre machine.
<code>Other</code>	La source ou la destination est inconnue.
<code>Clone</code>	Clone le contexte de l'objet, la dé-sérialisation devra se faire dans le même contexte que la sérialisation.
<code>CrossAppDomain</code>	La source ou la destination sont sur un domaine d'application différent.
<code>All</code>	La source ou la destination peuvent être n'importe lequel des contextes précédents. C'est la valeur par défaut.

Si vous utilisez `BinaryFormatter` ou `SoapFormatter`, vous ne pourrez pas assigner un contexte personnalisé, celui-ci sera toujours null pour la propriété `Context` et `All` pour la propriété `State`.

Pour pouvoir modifier le contexte, il vous faudra créer votre propre formateur de sérialisation.

Afin de créer un formateur personnalisé, vous devez implémenter l'interface `IFormatter` pour un formateur simple ou bien `IGenericFormatter` si vous souhaitez sérialiser des types génériques.

Sachez que `BinaryFormatter` et `SoapFormatter` implémentent l'interface `IFormatter`.

La classe `FormatterServices` met à disposition des méthodes statiques (dont `GetObjectData`) afin de faciliter la création de votre formateur.

La création d'un formateur sort du cadre de ce cours.

3 Sérialisation XML

Le .NET Framework fourni, en plus d'une sérialisation Binaire et SOAP, une sérialisation XML. Ce type de sérialisation peut être utile si vous souhaitez par exemple échanger des informations avec des applications qui ne sont pas créés avec les technologies .NET, ou encore permettre à l'utilisateur de personnaliser l'application en modifiant les paramètres de l'objet sérialisé en XML.

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 sérialisable.

3.1 Sérialisation/Dé-sérialisation d'objets en XML

Le principe de sérialisation et de dé-sérialisation des données reste grossièrement le même que ce qui a été vu dans la [partie 2](#) :

- On crée un flux vers un fichier qui contiendra nos données
- On instancie la classe de sérialisation adaptée
- On utilise la méthode `Serialize` pour sérialiser ou `Deserialize` pour effectuer l'opération inverse.

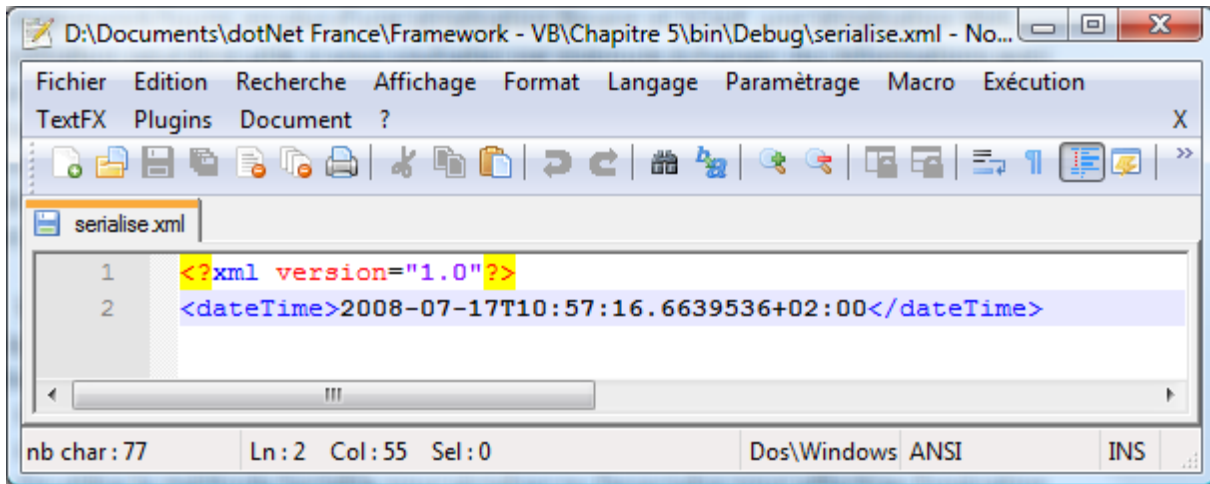
Les outils nécessaires à la sérialisation XML se trouvent dans l'espace de nom `System.Xml.Serialization` et la classe utilisée pour sérialiser en XML s'appelle `XmlSerializer`.

L'exemple ci-dessous se contente de sérialiser un objet `DateTime` dans le fichier `serialize.xml`. Ensuite, nous remettons le curseur du flux au début du flux et nous dé-sérialisons l'objet sérialisé.

```
'VB
Sub Main()
    Dim fichier As FileStream = New FileStream("serialize.xml",
    FileMode.Truncate, FileAccess.ReadWrite)
    Dim serializer As XmlSerializer = New
    XmlSerializer(GetType(DateTime))
    Dim time As DateTime = DateTime.Now
    serializer.Serialize(fichier, time)
    fichier.Seek(0, SeekOrigin.Begin)
    Dim timedeserial As DateTime =
    CType(serializer.Deserialize(fichier), DateTime)
    Console.WriteLine(timedeserial.ToString())
    Console.Read()
End Sub

//C#
static void Main(string[] args)
{
    FileStream fichier = new FileStream(@"serialize.xml",
    FileMode.Truncate, FileAccess.ReadWrite);
    XmlSerializer serializer = new XmlSerializer(typeof(DateTime));
    DateTime time = DateTime.Now;
    serializer.Serialize(fichier, time);
    fichier.Seek(0, SeekOrigin.Begin);
    DateTime timedeserial = (DateTime) serializer.Deserialize(fichier);
    Console.WriteLine(timedeserial.ToString());
    Console.Read();
}
```

Voici le contenu du fichier `serialize.xml` après exécution du programme:



```
D:\Documents\dotNet France\Framework - VB\Chapitre 5\bin\Debug\serialize.xml - No...
Fichier Edition Recherche Affichage Format Langage Paramètre Macro Exécution
TextFX Plugins Document ?
serialize.xml
1 <?xml version="1.0"?>
2 <dateTime>2008-07-17T10:57:16.6639536+02:00</dateTime>
nb char: 77 Ln: 2 Col: 55 Sel: 0 Dos\Windows ANSI INS
```

Et ici, le résultat affiché dans la console:

```
17/07/2008 10:57:16
```

Nous pouvons également sérialiser nos propres objets. La différence avec les autres méthodes de sérialisation est que cette fois-ci, il n'est plus nécessaire de spécifier l'attribut `Serializable` avant les classes qui doivent être sérialisées.

Retenez juste que la classe de sérialisation XML procède comme suit:

- Elle sérialise tous les éléments déclarés en public
- Elle ignore tous les autres éléments déclarés `private` ou `protected`.
- Si dans votre classe, vous créez un constructeur qui prend au moins un paramètre, vous êtes également obligé de créer un constructeur qui n'en prend aucun!

Note : S'il est impossible d'effectuer ces opérations sur des objets de type graphique, il est possible de sérialiser des objets `DataSet` utilisés pour créer des mini-bases de données en mémoire vive. Un exemple rapide est fourni dans le projet-type fourni sur le site DotnetFrance. Sachez juste que sérialiser un `DataSet` est exactement pareil que sérialiser un objet courant.

Dans l'exemple suivant, nous créons une classe Personnage que nous instancions et sérialisons dans le même fichier que précédemment:

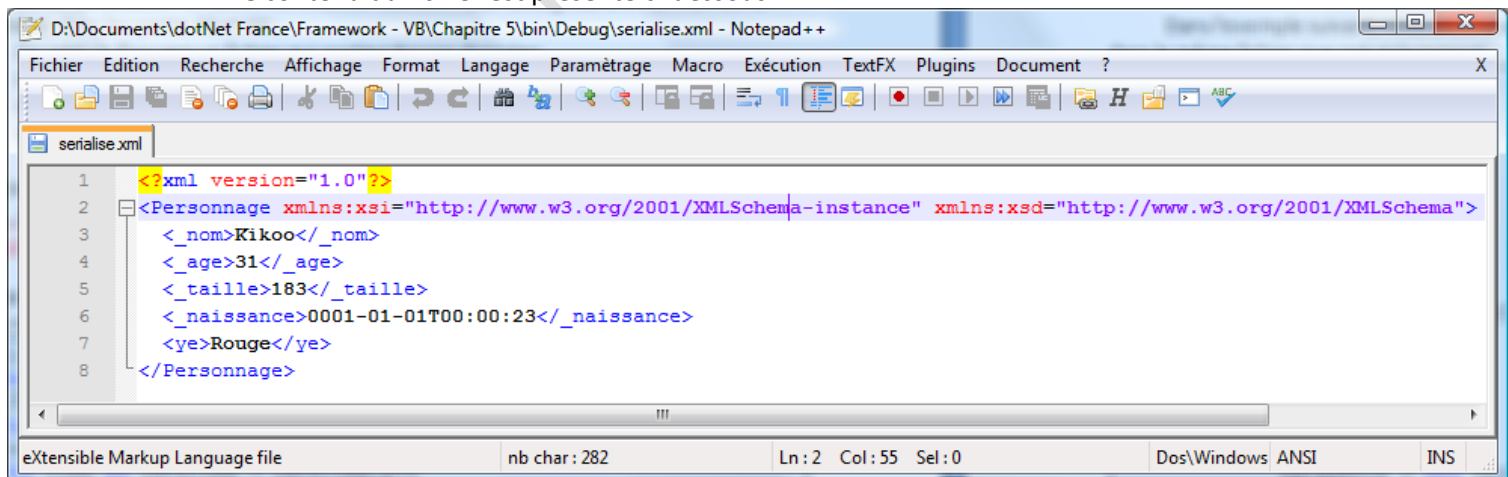
```
'VB
Public Enum Yeux
    Bleu
    Vert
    Rouge
    Marron
End Enum
Public Class Personnage
    Public _nom As String
    Public _age As Integer
    Public _taille As Short
    Public _naissance As DateTime
    Public ye As Yeux
    Public Sub New()
        _nom = "Kikoo"
        _age = 31
        _taille = 183
        _naissance = New DateTime(1, 1, 1, 0, 0, 23)
        ye = Yeux.Rouge
    End Sub
End Class
Sub Main()
    Dim fichier As FileStream = New FileStream("serialise.xml",
    FileMode.Truncate, FileAccess.ReadWrite)
    Dim serializer As XmlSerializer = New
    XmlSerializer(GetType(Personnage))
    Dim toserial As Personnage = New Personnage()
    serializer.Serialize(fichier, toserial)
    fichier.Close()
End Sub
```

Dotnet-France

```
//C#
public enum yeux
{
    Bleu,
    Vert,
    Rouge,
    Marron
}
public class Personnage
{
    public string _nom;
    public int _age;
    public short _taille;
    public DateTime _naissance;
    public yeux ye;
    public string[] _infos;

    public Personnage()
    {
        _infos = new String[4]{"kikoo", null, "lol", "juloe"};
        _nom = "Kikoo";
        _age = 31;
        _taille = 183;
        _naissance = new DateTime(1, 1, 1, 0, 0, 23);
        ye = yeux.Rouge;
    }
}
static void Main(string[] args)
{
    FileStream fichier = new FileStream(@"serialize.xml",
    FileMode.Truncate, FileAccess.ReadWrite);
    serializer = new XmlSerializer(typeof(Personnage));
    Personnage toserial = new Personnage();
    serializer.Serialize(fichier, toserial);
    fichier.Close();
}
```

Le contenu du fichier est présenté ci-dessous :



```
<?xml version="1.0"?>
<Personnage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <_nom>Kikoo</_nom>
  <_age>31</_age>
  <_taille>183</_taille>
  <_naissance>0001-01-01T00:00:23</_naissance>
  <ye>Rouge</ye>
</Personnage>
```

eXtensible Markup Language file nb char : 282 Ln : 2 Col : 55 Sel : 0 Dos\Windows ANSI INS

3.2 Contrôler la sérialisation

Tout comme la sérialisation SOAP, il est possible de contrôler la sérialisation XML grâce à certains attributs indiqués ci-dessous avec leur description:

Attributs	Utilisable avec	Description
<code>XmlAnyAttribute</code>	Propriété, champ public, paramètre ou valeur de retour contenant un tableau d'objets de type <code>XmlAttribute</code> .	Indique que le membre contient des attributs XML qui n'ont pas d'équivalent dans la classe en cours de sérialisation.
<code>XmlAnyElement</code>	Propriété, champ public, paramètre ou valeur de retour contenant un tableau d'objets de type <code>XmlElement</code> .	Indique que le membre contient des éléments XML (<code>XmlElement</code> ou <code>XmlNode</code>) qui n'ont pas d'équivalent dans la classe en cours de sérialisation.
<code>XmlAttribute</code>	Propriété, champ public, paramètre ou valeur de retour.	L'élément sera sérialisé en tant qu'attribut.
<code>XmlElement</code>	Propriété, champ public, paramètre ou valeur de retour.	L'élément sera sérialisé en tant qu'élément XML.
<code>XmlArray</code>	Propriété, champ public, paramètre ou valeur de retour de type <code>Array</code> .	L'élément sera sérialisé en tant que tableau d'éléments XML.
<code>XmlArrayItem</code>	Propriété, champ public, paramètre ou valeur de retour de type <code>Array</code> .	Spécifie les types qui peuvent être insérés dans un tableau d'objets.
<code>XmlChoiceIdentifier</code>	Propriété, champ public, paramètre ou valeur de retour.	Indique à l'outil de sérialisation qu'on peut identifier des membres avec des valeurs d'une énumération.
<code>XmlAttribute</code>	Membre d'une énumération	Spécifie la manière dont l'outil de sérialisation va sérialiser un élément d'une énumération.
<code>XmlAttribute</code>	Champs et propriétés publiques	Permet d'ignorer un élément lors de la sérialisation.
<code>XmlAttribute</code>	Classe parente de classe dérivée ou valeur de retour de méthodes dans des documents WSDL	Permet à l'outil de sérialisation d'inclure un type particulier lors d'une sérialisation ou d'une dé-sérialisation.
<code>XmlAttribute</code>	Classe publique	Permet de modifier les propriétés de l'élément XML racine.
<code>XmlAttribute</code>	Classe publique	Permet de modifier le nom et l'espace de nom du document XML.
<code>XmlAttribute</code>	Champs et propriétés publiques	L'élément sera sérialisé sous forme de texte.

Si nous reprenons l'exemple précédent, nous pouvons modifier quelques données en ajoutant quelques-uns de ces attributs (Le code de sérialisation restant inchangé, il n'est pas repris dans cet exemple):

```
'VB
Public Enum Yeux
    <XmlAttribute("BlueEyes")> Bleu
    <XmlAttribute("GreenEyes")> Vert
    <XmlAttribute("RedEyes")> Rouge
    <XmlAttribute("BrownEyes")> Marron
End Enum

<XmlRoot("RoutedCharacter"), XmlType(AnonymousType:=False,
IncludeInSchema:=False, Namespace:="Personnage")> _
Public Class Personnage
    <XmlAttribute("Nom")> Public _nom As String
    <XmlElement("Age")> Public _age As Integer
    <XmlIgnore()> Public _taille As Short
    <XmlElement("Naissance")> Public _naissance As DateTime
    <XmlElement("Yeux")> Public ye As Yeux
    <XmlArray("Informations")> Public _info(4) As String

    Public Sub New()
        _info = New String() {"kikoo", Nothing, "lol", "juloe"}
        _nom = "Kikoo"
        _age = 31
        _taille = 183
        _naissance = New DateTime(1, 1, 1, 0, 0, 23)
        ye = Yeux.Rouge
    End Sub
End Class
```

Dotnet-France

```
//C#
public enum yeux
{
    [XmlAttribute("BlueEyes")] Bleu,
    [XmlAttribute("GreenEyes")] Vert,
    [XmlAttribute("RedEyes")] Rouge,
    [XmlAttribute("BrownEyes")] Marron
}

[XmlRoot("RoutedCharacter"), XmlType(AnonymousType=false,
IncludeInSchema=false, Namespace="Personnage")]
public class Personnage
{
    [XmlAttribute("Nom")] public string _nom;
    [XmlElement("Age")] public int _age;
    [XmlElement("Naissance")] public short _taille;
    [XmlElement("Yeux")] public DateTime _naissance;
    [XmlElement("Informations")] public string[] _infos;

    public Personnage()
    {
        _infos = new String[4]{ "kikoo", null, "lol", "juloe" };
        _nom = "Kikoo";
        _age = 31;
        _taille = 183;
        _naissance = new DateTime(1, 1, 1, 0, 0, 23);
        ye = yeux.Rouge;
    }
}
```

Le fichier contient à présent :

```
<?xml version="1.0"?>
<RoutedCharacter xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" Nom="Kikoo">
  <Age xmlns="Personnage">31</Age>
  <Naissance xmlns="Personnage">0001-01-01T00:00:23</Naissance>
  <Yeux xmlns="Personnage">RedEyes</Yeux>
  <Informations xmlns="Personnage">
    <string>kikoo</string>
    <string xsi:nil="true" />
    <string>lol</string>
    <string>juloe</string>
  </Informations>
</RoutedCharacter>
```

3.3 Utilisation de schémas XML

L'XML permet notamment l'interopérabilité des informations sérialisées. Seulement, ce procédé ne saurait être efficace si le schéma XML est différent d'une application à l'autre.

Pour assurer que tous les documents XML générés auront la même forme, on utilise des schémas XML qui dictent comment sont agencés les éléments XML, quels attributs ils peuvent avoir...

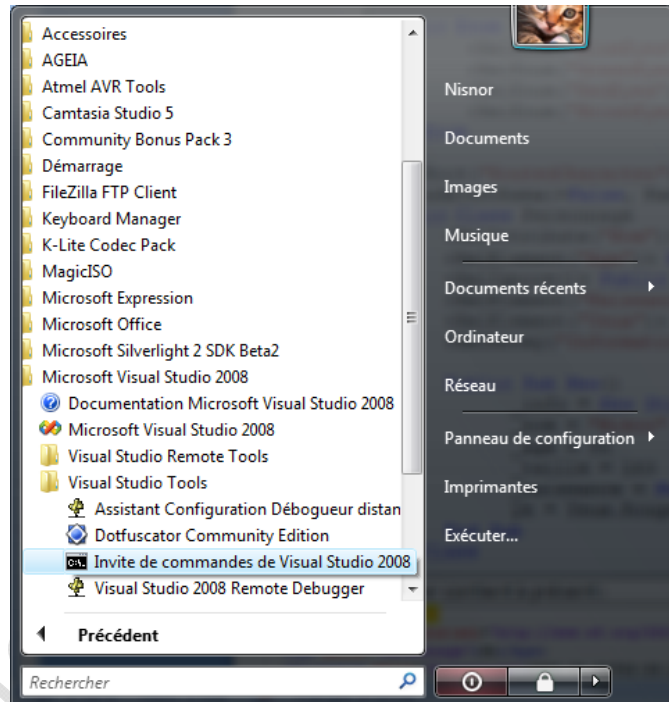
Ces schémas de description sont contenus dans des fichiers et sont eux-mêmes des documents XML. Pour plus de détails sur les schémas XML, je vous invite à visiter [Wikipedia](#) ou [le W3C](#).

Une fois que vous possédez un schéma XML dans un fichier .xsd, vous devez le transformer en classe exploitable dans votre code. Pour ce faire, vous ouvrez la console Visual Studio.

Dans cette console, vous utiliserez la commande suivante :

```
Xsd.exe
<chemin_vers_votre_schéma.xsd> /classes
/language:<CS ou VB>
```

Une fois cette commande exécutée, un fichier .cs ou .vb sera généré (le chemin d'accès du fichier généré est indiqué dans la console). Il vous suffira ensuite d'importer ce code source dans votre solution. La classe sera prise en compte automatiquement à chaque fois que vous sérialisez un nouvel objet.



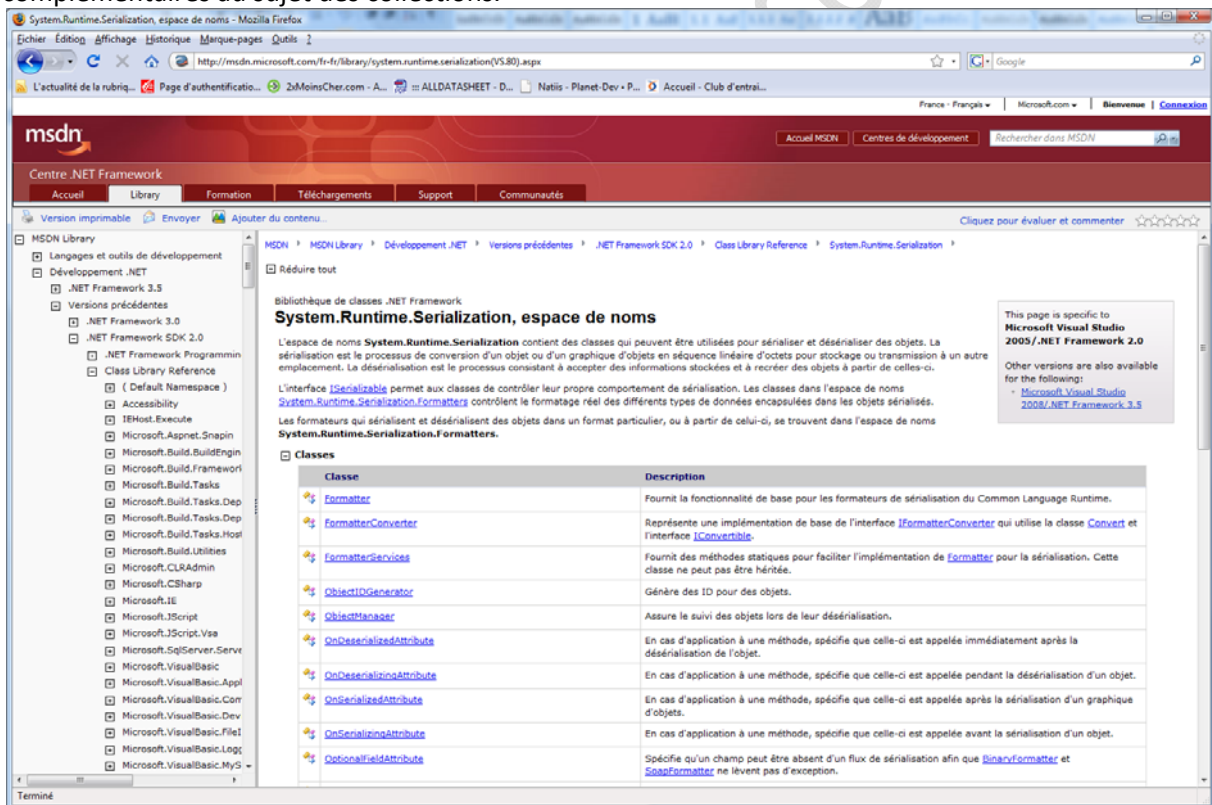
4 Conclusion

Vous aurez rapidement constaté que la sérialisation est un moyen de stockage ou d'échange de données rapide et facile à utiliser. En effet, seules quelques classes et attributs sont nécessaires pour pouvoir enregistrer des objets dans des fichiers. De plus, il vous est tout à fait possible de créer votre propre outil de sérialisation.

A la fin de ce chapitre, vous devez être capable de :

- Savoir ce qu'est la sérialisation et la dé-sérialisation.
- Quels sont les types de sérialisation fournis par défaut dans le .NET Framework ainsi que leurs classes associées (BinaryFormatter, SoapFormatter, XmlSerializer).
- Savoir comment modifier le processus de sérialisation (Attributs placés devant les membres).
- Savoir comment utiliser un schéma XML lors d'une sérialisation.
- Savoir comment créer son propre outil de sérialisation.

Dans tous les cas, vous pouvez également vous aider du [MSDN](http://msdn.microsoft.com) pour avoir des informations complémentaires au sujet des collections.



The screenshot shows the MSDN website page for the `System.Runtime.Serialization` namespace. The page title is "Bibliothèque de classes .NET Framework System.Runtime.Serialization, espace de noms". The main content area contains an introduction to the namespace, explaining that it contains classes for serializing and deserializing objects. It mentions the `ISerializable` interface and the `System.Runtime.Serialization.Formatters` namespace. Below the text is a table of classes with their descriptions.

Classe	Description
Formatter	Fournit la fonctionnalité de base pour les formateurs de sérialisation du Common Language Runtime.
FormatterConverter	Représente une implémentation de base de l'interface <code>IFormatterConverter</code> qui utilise la classe <code>Convert</code> et l'interface <code>IConvertible</code> .
FormatterServices	Fournit des méthodes statiques pour faciliter l'implémentation de <code>Formatter</code> pour la sérialisation. Cette classe ne peut pas être héritée.
ObjectIDGenerator	Génère des ID pour des objets.
ObjectManager	Assure le suivi des objets lors de leur désérialisation.
OnDeserializingAttribute	En cas d'application à une méthode, spécifie que celle-ci est appelée immédiatement après la désérialisation de l'objet.
OnDeserializingAttribute	En cas d'application à une méthode, spécifie que celle-ci est appelée pendant la désérialisation d'un objet.
OnSerializedAttribute	En cas d'application à une méthode, spécifie que celle-ci est appelée après la sérialisation d'un graphique d'objets.
OnSerializingAttribute	En cas d'application à une méthode, spécifie que celle-ci est appelée avant la sérialisation d'un objet.
OptionalFieldAttribute	Spécifie qu'un champ peut être absent d'un flux de sérialisation afin que <code>BinaryFormatter</code> et <code>SoapFormatter</code> ne lèvent pas d'exception.