Define Serilization

Serialization can be defined as the process of storing the state of an object instance to a storage medium. During this process, the public and private fields of the object and the name of the class, including the assembly containing the class, is converted to a stream of bytes, which is then written to a data stream. When the object is subsequently deserialized, an exact clone of the original object is created.

why should you use serilization when the values of obj can be stored on disk and later used whenever required?

It is often necessary to store the value of fields of an object to disk and then retrieve this data at a later stage. Although this is easy to achieve without relying on serialization, this approach is often cumbersome and error prone, and becomes progressively more complex when you need to track a hierarchy of objects. Imagine writing a large business application containing many thousands of objects and having to write code to save and restore the fields and properties to and from disk for each object. Serialization provides a convenient mechanism for achieving this objective with minimal effort.

Explain how CLR manages serilization?

The Common Language Runtime (CLR) manages how objects are laid out in memory and the .NET Framework provides an automated serialization mechanism by using reflection. When an object is serialized, the name of the class, the assembly, and all the data members of the class instance are written to storage. Objects often store references to other instances in member variables. When the class is serialized, the serialization engine keeps track of all referenced objects already serialized to ensure that the same object is not serialized more than once. The serialization architecture provided with the .NET Framework correctly handles object graphs and circular references automatically. The only requirement placed on object graphs is that all objects referenced by the object that is being serialized must also be marked as Serializable. If this is not done, an exception will be thrown when the serializer attempts to serialize the unmarked object.

Why would you want to use serialization?

The two most important reasons are to persist the state of an object to a storage medium so an exact copy can be recreated at a later stage, and to send the object by value from one application domain to another.

Give examples where serialization is used?For example, Serialization is used to save session state in ASP.NET and to copy objects to the clipboard in Windows Forms. It is also used by remoting to pass objects by value from one application domain to another.

How will you serialize a class?

The easiest way to make a class serializable is to mark it with the Serializable attribute as follows:

[Serializable]

public class MyObject {

public int n1 = 0;

public int n2 = 0;

public String str = null;

}

How will you deserialize an object?

First, create a formatter and a stream for reading, and then instruct the formatter to deserialize the object.

IFormatter formatter = new BinaryFormatter();

Stream stream = new FileStream("MyFile.bin",

FileMode.Open,

FileAccess.Read,

FileShare.Read);

MyObject obj = (MyObject) formatter.Deserialize(fromStream);

stream.Close();

Which class is responsible for binary serialization?

The BinaryFormater is the class responsible for the binary serialization and It's commonly used for the .Net Remoting.

What does it take to make my object serializable?

Your class must have the attribute SerializableAttribute and all its members must also be serializable, except if they are ignored with the attribute NonSerializedAttribute. Private and public fields are serialized by default.

What are the main advantages of binary serialization?

Smaller

Faster

More powerful (support complex objects and read only properties)

Will my read only properties be serialized?

Yes if the properties encapsulate a field. By default all the private and public fields are serialized. Binary serialization is not related to properties.

Is it possible to have circular reference?

Yes it is, you can have circular reference and the binary serialization process will work fine. .Net generate the object graph before the executing the serialization and finally generate the stream. Unlike Xml serialization process, the BinaryFormater has no problem with the circular reference.

Why my Dataset is so big when it's serialized in binary?

By default the DataSet is serialized in Xml and the binary stream only wraps the Xml Data inside it. That's mean that the size is similar to the Xml size. .Net 2.0 add a new property named RemotingFormat used to change the binary serialization format of the DataSet. SerializationFormat.Binary will generate a better result.

For previous version, it's also possible to download the DataSetSurrogate to reduce the size and increase the performance.

How can I implement a custom serialization?

If you need to control the serialization process of your class, you can implement the ISerializable interface which contains a method GetObjectData and a special constructor .

Why use custom serialization ?

By using it you will be able to handle version change in your class or get a better performance result. An exception of type SerializationException is raised if the fields does not exists.

#region ISerializable Members

//Special constructor

protected CustomInvoice(SerializationInfo info, StreamingContext context) {

clientName = info.GetString("clientName");

date = info.GetDateTime("date");

total = info.GetDouble("total");

}

[SecurityPermissionAttribute(SecurityAction.Demand, SerializationFormatter = true)]

public void GetObjectData(SerializationInfo info, StreamingContext context) {

info.AddValue("clientName", clientName);

info.AddValue("date", date);

info.AddValue("total", total);

}

#endregion

When does a change in my object break the deserialization?

Binary serialization is not tolerant to version change. There is no problem when a new field is added but if a field is missing, an exception is throw. New field will get the default value. .Net 2.0 include a new attribute named OptionalFieldAttribute. For previous version, you must implement your own custom serialization and handle manually the changes.

When does a change in my object NOT break the deserialization?

Version fault exception are only checked if the assembly is signed, all version change are ignored otherwise.

How can I make my object version tolerant?

Use the OptionalFieldAttribute or implement your own custom serialization with ISerializable.

Why set VersionAdded of OptionalFieldAttribute since it's a free text?

The parameter is only for informative purpose. .Net never checked it by default but it could be used in a custom serialization. Use the reflection to get the attribute of field and check the added version value.

Does BinaryFormatter from .Net 1.1 is compatible with the BinaryFormatter 2.0?

Absolutely, the BinaryFormatter 2.0 is 100% with other version, but it's not the case with the SoapFormatter.

How can I modify a value just before the serialization or just after the deserialization?

You can add custom attribute to some method. Your marked method will get called a the right time. This is usefull to initialize a property after the deserialization or to clean up your instance before the serialization.

OnDeserializingAttribute :This event happens before deserialization

OnDeserializedAttribute :This event happens after deserialization

OnSerializingAttribute :This event happens before serialization

OnSerializedAttribute :This even happens after serialization

[Serializable]

public class SecurityToken {

private string password;

private string userName;

private string Decrypt(string aPassword) {

// Decrypt the password here !!!

return password;

}

private string Encrypt(string aPassword) {

// Encrypt the password here !!!

return password;

}

[OnSerializing()]

internal void OnSerializingMethod(StreamingContext context) {

password = Encrypt(password);

}

[OnDeserialized()]

internal void OnDeserializedMethod(StreamingContext context) {

password = Decrypt(password);

// Set the default

if (userName == null) {

userName = Environment.UserName;

}

}

public string Password {

get {

return password;

}

set {

password = value;

}

}

public string UserName {

get {

return userName;

}

set {

userName = value;

}

}

}

How can I create a generic Binary deserialization method?

// Binary deserialization (generic version with the return value casted)

public static T DeserializeBinary(string aFileName) {

using (FileStream \_FileStream = new FileStream(aFileName, FileMode.Open)) {

BinaryFormatter \_Formatter = new BinaryFormatter();

return (T)\_Formatter.Deserialize(\_FileStream);

}

}

Which class is responsible for Xml serialization?

XmlSerializer is responsible of the Xml serialization.

What is the difference between the SoapFormatter and the XmlSerializer?

SoapFormatter is used to create a Soap envelop and use an object graph to generate the result. The XmlSerializer process use only the public data and the result is a more common xml file. The Web Service in .Net use an XmlSerializer to generate the output contained in the Soap message. The SoapFormatter and the BinaryFormatter are used in the .Net Remoting serialization process.

What does it take to make my object serializable?

Nothing, but there is some constraint :

Your object must have a public empty constructor.

Field or property must be public

Their return type must also respect serialization rules.

Property must be read write.

What are the main advantages of Xml serialization?

Based on international standard (XML).

Cross platforms.

Readable and can be edited easily.

How do I encapsulate the Xml serialization method?

public static void SerializeXml( object aObject, string aFileName) {

using (FileStream \_FileStream = new FileStream(aFileName, FileMode.Create)) {

XmlSerializer \_Serializer = new XmlSerializer ( aObject.GetType());

\_Serializer.Serialize(\_FileStream, aObject);

}

}

How do I encapsulate the Xml deserialization method?

public static object DeserializeXml( string aFileName, Type aType) {

using (FileStream \_FileStream = new FileStream(aFileName, FileMode.Create)) {

XmlSerializer \_Serializer = new XmlSerializer (aType);

return \_Serializer.Deserialize(\_FileStream);

}

}

How can I create a generic Xml deserialization method?

public static T DeserializeXml(string aFileName) {

using (FileStream \_FileStream = new FileStream(aFileName, FileMode.Open)) {

XmlSerializer \_Serializer = new XmlSerializer (typeof(T));

return (T)\_Serializer.Deserialize(\_FileStream);

}

}

How can I ignore a property in serialization?

If you use a XmlSerializer, mark your property with the custom attribute XmlIgnoreAttribute and if you use a SoapFormatter, use a SoapIgnoreAttribute instead.

How can I rename a field or a property in the Xml output?

Use the attribute XmlElementAttribute or XmlAttributeAttribute with the new name as parameter. To rename a class, use the XmlTypeAttribute.

[XmlType("city")]

public class Town {

private string name;

private string state;

[XmlElement("townname")]

public string Name {

get {

return name;

}

set {

name = value;

}

}

[XmlAttribute("state")]

public string State {

get {

return state;

}

set {

state = value;

}

}

}

How can I serialize a property as an Xml attribute?

By default properties are serialized as Xml elements, but if you add an XmlAttributeAttribute to a property, .Net will generate an attribute instead. It's must be type compatible with an Xml attribute. See example here

[XmlAttribute("state")]

public string State {

get {

return state;

}

set {

state = value;

}

}

How can I implement custom serialization?

You need to Implement the interface IXmlSerializable. This class is available in the .Net 1.X but it's was not documented. It's now official available with .Net 2.0. With custom serialization, it's possible to optimize the output and generate only what is needed. In this example, we generate only the non empty properties

public class SessionInfo : IXmlSerializable {

#region IXmlSerializable Members

public System.Xml.Schema.XmlSchema GetSchema() {

return null;

}

public void ReadXml(XmlReader reader) {

UserName = reader.GetAttribute("UserName");

while (reader.Read()) {

if (reader.IsStartElement()) {

if (!reader.IsEmptyElement) {

string \_ElementName = reader.Name;

reader.Read(); // Read the start tag.

if(\_ElementName == "MachineName") {

MachineName = reader.ReadString();

} else {

reader.Read();

}

}

}

}

}

public void WriteXml(XmlWriter writer) {

if (!String.IsNullOrEmpty(UserName))

writer.WriteAttributeString("UserName", UserName);

if (!String.IsNullOrEmpty(MachineName))

writer.WriteElementString("MachineName", MachineName);

}

#endregion

private string machineName;

private string userName;

public string MachineName {

get {

return machineName;

}

set {

machineName = value;

}

}

public string UserName {

get {

return userName;

}

set {

userName = value;

}

}

}

How can I serialize a property array?

Array are compatible with the serialization, but all elements must be of the same type.

How can I serialize an array with different types?

It's possible to tag the array with all the possible type. Use XmlInclude on the class containing the array or XmlArrayItem. All the possible types must be specified with the attributes. Array types must be known because of the Xml schema. XmlInclude could be used for property that returned differente types. It's complicate object inheritance since all types must be known. This example will fail with a message "There was an error generating the XML document." because the Cars could contain undefined types.

public class Car {}

public class Ford : Car{}

public class Honda: Car {}

public class Toyota: Car {}

public class CarSeller {

private List cars = new List();

public List Cars {

get {

return cars;

}

set {

cars = value;

}

}

}

...

Ford \_Ford = new Ford();

Honda \_Honda = new Honda();

Toyota \_Toyota = new Toyota();

CarSeller \_Seller = new CarSeller();

\_Seller.Cars.Add(\_Ford);

\_Seller.Cars.Add(\_Honda);

\_Seller.Cars.Add(\_Toyota);

SerializationHelper.SerializeXml(\_Seller, @"seller.xml");

Three possible solutions:

#1 Add XmlIncludeAttribute to the Seller class:

[XmlInclude(typeof(Ford))]

[XmlInclude(typeof(Honda))]

[XmlInclude(typeof(Toyota))]

public class CarSeller {

private List cars = new List();

public List Cars {

get {

return cars;

}

set {

cars = value;

}

}

}

#2 Add XmlArrayItem to the property named Cars:

public class CarSeller {

private List cars = new List();

[XmlArrayItem(typeof(Ford))]

[XmlArrayItem(typeof(Honda))]

[XmlArrayItem(typeof(Toyota))]

public List Cars {

get {

return cars;

}

set {

cars = value;

}

}

}

#3 Implement our own serialization with IXmlSerializable :

see items above

How can I serialize a collection?

Collection are serialized correctly, but they must contains only object of same types. Read only properties of type ArrayList, List and other collections will be serialized and deserialized correctly.

Public class Role {

private string name;

public string Name

{

get { return name; }

set { name = value; }

}

}

public class UserAccount {

private string userName;

public string UserName {

get {

return userName;

}

set {

userName = value;

}

}

// Generic version

private List roles = new List();

public List Roles {

get {

return roles;

}

}

// ArrayList version

private ArrayList roleList = new ArrayList();

public ArrayList RoleList {

get {

return roleList;

}

}

// String collection version

private StringCollection roleNames = new StringCollection();

public StringCollection RoleNames {

get {

return roleNames;

}

}

UserAccount \_UserAccount = new UserAccount();

\_UserAccount.UserName = "dhervieux";

Role \_RoleAdmin = new Role();

\_RoleAdmin.Name = "Admin";

Role \_RoleSales = new Role();

\_RoleSales.Name = "Sales";

\_UserAccount.RoleList.Add(\_RoleAdmin);

\_UserAccount.RoleList.Add(\_RoleSales);

\_UserAccount.Roles.Add(\_RoleAdmin);

\_UserAccount.Roles.Add(\_RoleSales);

\_UserAccount.RoleNames.Add("Admin");

\_UserAccount.RoleNames.Add("Sales");

SerializationHelper.SerializeXml(\_UserAccount, @"useraccount.xml");

UserAccount \_Result =

SerializationHelper.DeserializeXml(@"useraccount.xml");

Why is the first serialization of each type of object is so long?

XmlSerializer generate an assembly in memory optimized for each type. That's explain why the first call to a Web Service is so long. In .Net 2.0, there is an option in the project properties of Visual Studio to generate the Xml serialization assembly. Use it directly in the IDE or use sgen.exe, this tools come with the .Net Framework SDK.

How can I optimize the serialization process?

Pregenerate your serialization assembly with Visual Studio or sgen.exe. Implementing your own serialization could also increase the performance.

How can I serialize an array directly to stream?

Yes it possible. .Net will name the Array and save the content. All the data must be of the same type.

bool[] \_BoolArray = new bool[] { true, false, false, true };

// Serialization

SerializationHelper.SerializeXml(\_BoolArray, @"boolarray.xml");

//Deserialization

\_Result = SerializationHelper.DeserializeXml(@"directboolarray.xml");

will produce:

true

false

false

true

Which serializer is used by a Web Services?

Web Services are using SOAP to communicate, but returned objets or parameters are serialized with the XmlSerializer.

Does read only properties are serialized?

No, they are not, except for collections.

How can I serialize a multidimensional array

You need to encapsulate your array in a structure or a class an serialize it. Multidimensional array are not serializable by default.

How can I avoid serialization for an empty list or property with a default value?

There is an undocumented way of doing that, you need to create a method named ShouldSerialize where is replaced by the property name. This method should return a boolean that indicate if the property must be serialized or not. For exemple, if you have list with no item, there is no need to serialize an empty list.

public class Registration {

private string[] users = new string[0];

public bool ShouldSerializeUsers() {

return users.Length > 0;

}

public string[] Users {

get {

return users;

}

set {

users = value;

}

}

}

Result :

Without the ShouldSerializeUsers :

Why my object is marked as Serializable (like SortedList) and it's does not work?

The SerializationAttribute is only used for the binary serialization. That does not mean that it will work with an XmlSerializer. That's the case of the SortedList.

How to remove the default namespace in the serialization?

It's possible to remove the xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" and xmlns:xsd="http://www.w3.org/2001/XMLSchema" from the serialization result, it's possible to add an XmlSerializerNamespaces with an empty namespace mapping.

User \_User = new User(new string[] { "Admin", "Manager" });

using (FileStream \_FileStream = new FileStream("user.xml", FileMode.Create)) {

XmlSerializer \_Serializer = new XmlSerializer(\_User.GetType());

XmlSerializerNamespaces ns = new XmlSerializerNamespaces();

ns.Add("", "");

\_Serializer.Serialize(\_FileStream, \_User, ns);

}