**Object Serialization using C#**

**Introduction**

Writing crucial data to the disk as TEXT is always dangerous. Any anonymous user can open the text file and easily read your data. With Object Serialization, you can reduce this danger to a certain extent. You can write any complex object directly to a filestream without converting values of individual properties into a text. You can make the data written, to the disk, atleast not human readable. In order for the users to read your data files, they have to use your program. Like a File Open command which you may provide in your application.

**.NET and Object Serialization**

Now what does .NET framework provide us to do Serialization of Custom-build class objects? With the advent of any programming language, developers would first look whether it is an OOPL. If so, how to create my own classes with my own properties and functions. And then comes inheritance and all others. Explaining how to create user-defined classes is out of scope of this article. But along the way you will get to know about what a user-defined class is.

**Serialization explained**

*Serialization is the process of converting complex objects into stream of bytes for storage. Deserialization is its reverse process, that is unpacking stream of bytes to their original form.* The namespace which is used to read and write files is System.IO. For Serialization we are going to look at the System.Runtime.Serialization namespace. The ISerializable interface allows you to make any class Serializable.

Here are the following steps that we are going to do to create a serializable class and test it.

Create a custom class named **Employee** and assign properties.

Define the serialization functions.

Create a main class and instantiate our **Employee** class.

Serialize the object to a sample file.

Deserialize the values by reading it from the file.

**Defining Employee class and properties**

Our custom class **Employee** should be derived from the ISerializable interface and should hold the Serializable attribute. Here is the code snippet. Collapse | Copy Code

using System;

using System.IO;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

namespace MyObjSerial

{

[Serializable()] //Set this attribute to all the classes that want to serialize

public class Employee : ISerializable //derive your class from ISerializable

{

public int EmpId;

public string EmpName;

//Default constructor

public Employee()

{

EmpId = 0;

EmpName = null;

}

}

}

**Define Serialization functions**

Now we need two functions: One to say how to serialize **Employee** objects and another to say how to deserialize them. For serialization we override the GetObjectData() function provided by the ISerializable interface. For deserialization we provide a special constructor with the serialization parameters as arguments. This constructor will be called when we deserialize our file to object (which will be shown later).

One of the important parameters is the SerializationInfo object. This object holds a name-value pair for the properties to be serialized. You can decide which property should be serialized and which not in the GetObjectData() function. All the properties that are added to this SerializationInfo parameter will be serialized. Here are the codes for the two functions. Add them to our **Employee** class. Collapse | Copy Code

//Deserialization constructor.

public Employee(SerializationInfo info, StreamingContext ctxt)

{

//Get the values from info and assign them to the appropriate properties

EmpId = (int)info.GetValue("EmployeeId", typeof(int));

EmpName = (String)info.GetValue("EmployeeName", typeof(string));

}

//Serialization function.

public void GetObjectData(SerializationInfo info, StreamingContext ctxt)

{

//You can use any custom name for your name-value pair. But make sure you

// read the values with the same name. For ex:- If you write EmpId as "EmployeeId"

// then you should read the same with "EmployeeId"

info.AddValue("EmployeeId", EmpId);

info.AddValue("EmployeeName", EmpName);

}

Thats it. You have created your own class which is now serializable. Now lets see how to write an instance of **Employee** to a special file with a custom .osl extension. And we also see how to read back **Employee** object from the file.

**Create a main class and instantiate our Employee class**

Following is the code snippet for **ObjSerial** class which holds our application's main entry point. Collapse | Copy Code

//Main class

public class ObjSerial

{

public static void Main(String[] args)

{

//Create a new Employee object

Employee mp = new Employee();

mp.EmpId = 10;

mp.EmpName = "Omkumar";

//Add code below for serialization

}

}

**Serialize the object to a sample file**

For serializing, lets open a stream object and give a sample file name EmployeeInfo.osl. Note, the demo exe file has this same name. So when you run ObjSerial.exe, the EmployeeInfo.osl file will be created under the folder where you copied the exe file. Add the following code to our ObjSerial class. Once a stream is open we create a BinaryFormatter and use the Serialize method to serialize our object to the stream. What Serialize method would do? It converts our object into binary format and streams it in. Collapse | Copy Code

// Open a file and serialize the object into it in binary format.

// EmployeeInfo.osl is the file that we are creating.

// Note:- you can give any extension you want for your file

// If you use custom extensions, then the user will now

// that the file is associated with your program.

Stream stream = File.Open("EmployeeInfo.osl", FileMode.Create);

BinaryFormatter bformatter = new BinaryFormatter();

Console.WriteLine("Writing Employee Information");

bformatter.Serialize(stream, mp);

stream.Close();

**Deserialize the values by reading it from the file**

Now we read the created file and cast the return value to our **Employee** class for further usage. For reading we again create a BinaryFormatter to read the object in binary form. We then use the Deserialize method which converts the stream of bytes to an Object object. This object can then be easily casted to our **Employee** class. Collapse | Copy Code

//Clear mp for further usage.

mp = null;

//Open the file written above and read values from it.

stream = File.Open("EmployeeInfo.osl", FileMode.Open);

bformatter = new BinaryFormatter();

Console.WriteLine("Reading Employee Information");

mp = (Employee)bformatter.Deserialize(stream);

stream.Close();

Console.WriteLine("Employee Id: {0}",mp.EmpId.ToString());

Console.WriteLine("Employee Name: {0}",mp.EmpName);

**Conclusion**

This sample application explains the core part of serialization only. Actually you can do a lot with the objects while serializing and deserializing. Now you can create a Mulitple document application with .NET Windows Forms and go ahead Serialize the world into your disk.

Bottom of Form 1