Binary Serialization Tutorial

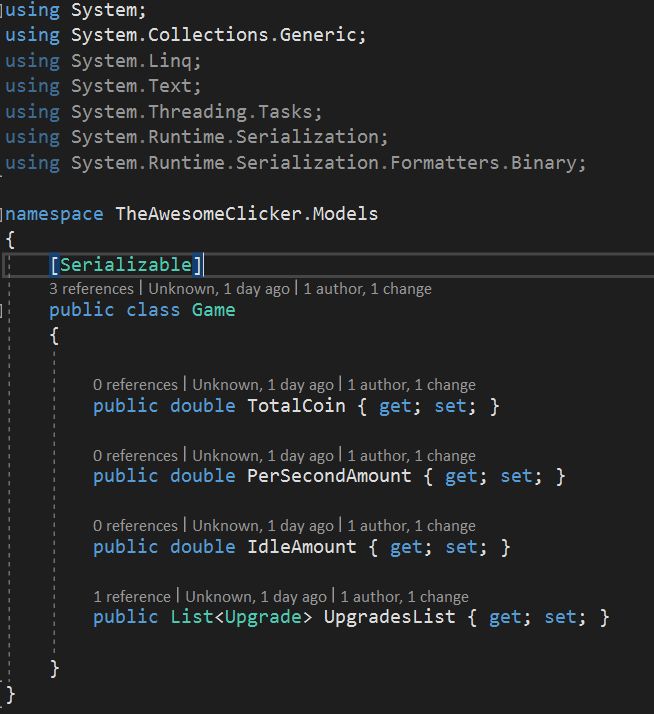
Zachary T. Andersen  
Joshua Carpenter  
Jacky Luo  
Walker Price

Dylan McClain

GitHub link: <https://github.com/JWPeace/SerializationTutorial.git>

# How to make custom types serializable

To make a custom type serializable, you must first reference the namespace System.Runtime.Serialization and add the [Serializable()] attribute above the class declaration you wish to persist. All types of serialization should stem from System.Runtime.Serialization, however the specific namespace that contains the source code for whichever type you want may not be installed (Such as the DataContractJsonSerializer from System.Runtime.Serialization.Json). You may be able to install these libraries with NuGet. To prevent a field in a custom type from interacting with serialization you must add the [NonSerialized] attribute behind the declaration of that field. Properties may not be assigned this attribute. Instances of a class with the [Serializable] attribute will now be serializable.



# How to serialize and deserialize objects

When serializing and deserializing objects you have 3 requirements

* The object wanting to serialize/deserialize
* A stream to send/retrieve the serialized information with a filepath
* A formatter to determine how to serialize the object i.e. BinaryFormatter, SoapFormatter, XMLSerializer, DataContractJsonSerializer, etc.

static void Serialize()   
 {  
 // Create a object with values that will eventually be serialized.  
 Object objectInstance = new Object(){ Num = 1, Name = Janet };  
  
 // To serialize object and values,   
 // you must first open a stream for writing.   
 // In this case, use a file stream.  
 FileStream fs = new FileStream("DataFile.dat", FileMode.Create);  
  
 // Construct a BinaryFormatter and use it to serialize the data to the stream.

// You can also use the same format with SoapFormatter

//XMLSeriallizer uses a similar format

//XmlSerializer mySerializer = new XmlSerializer(typeof(Object))

//mySerializer.Serialize(fs, objectInstance)

BinaryFormatter formatter = new BinaryFormatter();  
 try   
 {  
 formatter.Serialize(fs, objectInstance);  
 }  
 catch (SerializationException e)   
 {  
 Console.WriteLine("Failed to serialize. Reason: " + e.Message);  
 throw;  
 }  
 finally   
 {  
 fs.Close();  
 }  
 }

static void Deserialize()   
 {  
 // Declare the object reference.  
 Object objectInstance = null;  
  
 // Open the file containing the data that you want to deserialize.  
 FileStream fs = new FileStream("DataFile.dat", FileMode.Open);  
 try   
 {  
 BinaryFormatter formatter = new BinaryFormatter();  
  
 // Deserialize the object from the file and   
 // assign the reference to the local variable.

// Format is the same as SoapFormatter and XMLSerializer.  
 objectInstance = (Object) formatter.Deserialize(fs);  
 }  
 catch (SerializationException e)   
 {  
 Console.WriteLine("Failed to deserialize. Reason: " + e.Message);  
 throw;  
 }  
 finally   
 {  
 fs.Close();  
 }  
  
 // To prove that the object deserialized correctly,   
 // display value.  
   
 Console.WriteLine( object.toString() );  
 }

# How to persist the serialized objects to a file (save)

# How to deserialize objects from a file (load)

# 

# How to create a custom file extension

The first step is to just save the file with the extension you want. When you save it just type in the extension you want and then it will make it.

Next you need to associate it to your program by creating an instance of the FileAssociationInfo class and specifying the extension you wish to deal with into the constructor. Next we see if the extension already exists and if it doesn't, we create it with the specified ProgID (MyProgramName), and then set up the optional ContentType and OpenWithList properties.

FileAssociationInfo fai = new FileAssociationInfo(".bob");  
 if (!fai.Exists)  
 {  
 fai.Create("MyProgramName");  
  
 *//Specify MIME type (optional)*  
 fai.ContentType = "application/myfile";  
  
 *//Programs automatically displayed in open with list*  
 fai.OpenWithList = new string[]  
 { "notepad.exe", "wordpad.exe", "someotherapp.exe" };  
 }

Finally, we create an instance of the ProgramAssociationInfo class and specify the ProgID we wish to deal with in its constructor. Should this ProgID not exist, we create it and specify both a description for the program type (shared between all files using this ProgID) and the command verb that is used in selecting different ways to load the file.

ProgramAssociationInfo pai = new ProgramAssociationInfo(fai.ProgID);  
 if (!pai.Exists)  
 {  
 pai.Create  
 (  
 *//Description of program/file type*  
 "My Program's File Type",  
  
 new ProgramVerb  
 (  
 *//Verb name*  
 "Open",  
 *//Path and arguments to use*  
 @"C:\SomePath\MyApp.exe %1"  
 )  
 );  
  
 *//optional*  
 pai.DefaultIcon = new ProgramIcon(@"C:\SomePath\SomeIcon.ico");  
 }

# How to make the file dialogue box (used for save and load) look for and only use your file extension