



DeepCopy.zip

Download Free .NET & JAVA Files API

## Introduction

System.Object is base class of all classes, structures, enumeration and delegates. We can say it is the root of the type hierarchy. System.Object has a method called MemberwiseClone that helps to create a clone of the current object instance.

### Problem Statement

The MemberwiseClone method of System.Object creates a shallow copy of a new object and it copies the non-static fields of the current object instance to a new object. Copy Object is performed property by property, if property is a value type then it copies data bit by bit and if a property is a reference type then it copies the reference of the original object, in other words the clone object refers to the same object. This means that the MemberwiseClone method does not create a deep copy of the object.

### Solution

There are numerous ways to implement a deep copy class object. Two of them I have described here with examples.

1. Implement Deep Cloning using Serializing Deserializing objects
2. Implement Deep Cloning using Reflection

## 1. Implement Deep Cloning using Serializing Deserializing objects

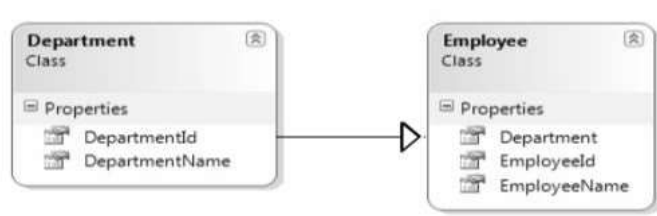
The ICloneable interface enables us to provide customized implementation to create a copy of the existing object using the "Clone" method. Generally the "Object.MemberwiseClone" method helps us to create a copy of an existing object, but it creates a shallow copy of the object.

Serialization is the process of storing the state of an object to the stream of bytes. Deserialization is the process of converting from a byte stream to an original object. In .NET there are many ways to perform serialization and deserialization like Binary serialization, XML serialization, data contract serialization and so on. Binary serialization is much faster than XML serialization and binary serialization uses private and public fields to serialize so binary serialization is a good option to perform serialization and deserialization.

Using serialization and deserialization, we can create a deep copy of any object. Note that to perform serialization and deserialization all types must be marked as "serializable".

### Example

Suppose I have an Employee class and it contains the Department class type as a property. Now I implemented the Employee class with the ICloneable interface and I have implemented the "Clone" method of the ICloneable interface. Using a binary formatter, I just serialized the current object and deserialized it into a new object.



```
1 [Serializable]
2 public class Department
3 {
4     public int DepartmentId { get; set; }
5     public string DepartmentName { get; set; }
6 }
7 [Serializable]
8 public class Employee : ICloneable
```

```

11 public string EmployeeName { get; set; }
12 public Department Department { get; set; }
13
14 {
15     using (MemoryStream stream = new MemoryStream())
16     {
17         if (this.GetType().IsSerializable)
18         {
19             BinaryFormatter formatter = new BinaryFormatter();
20             formatter.Serialize(stream, this);
21             stream.Position = 0;
22             return formatter.Deserialize(stream);
23         }
24         return null;
25     }
26 }
27 }

```

We can also implement this using an Extension method.

```

1 public static class ObjectExtension
2 {
3     public static T CopyObject<T>(this object objSource)
4     {
5         using (MemoryStream stream = new MemoryStream())
6         {
7             BinaryFormatter formatter = new BinaryFormatter();
8             formatter.Serialize(stream, objSource);
9             stream.Position = 0;
10            return (T)formatter.Deserialize(stream);
11        }
12    }
13 }

```

## 2. Implement Deep Cloning using Reflection

Reflection is used for obtaining Meta information of an object at runtime. The System.Reflection namespace has classes that allow us to obtain object information at runtime as well as we can create an instance of a type of object from the existing object and also accessing its properties and invoke its methods. To do a deep copy of an object, we can use reflection. Consider the following code. Here I have created one static method that accepts any object and it returns the same type of object with a new reference.

```

2 public class Utility
3 {
4     public static object CloneObject(object objSource)
5     {
6         //step : 1 Get the type of source object and create a new instance of that type
7         Type typeSource = objSource.GetType();
8         object objTarget = Activator.CreateInstance(typeSource);
9         //Step2 : Get all the properties of source object type
10        PropertyInfo[] propertyInfo = typeSource.GetProperties(BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Instance);
11        //Step : 3 Assign all source property to target object 's properties
12        foreach (PropertyInfo property in propertyInfo)
13        {
14            //Check whether property can be written to
15            if (property.CanWrite)
16            {
17                //Step : 4 check whether property type is value type, enum or string type
18                if (property.PropertyType.IsValueType || property.PropertyType.IsEnum || property.PropertyType.IsString)
19                {
20                    property.SetValue(objTarget, property.GetValue(objSource, null), null);
21                }
22                //else property type is object/complex types, so need to recursively call this method until the end
23                else
24                {

```

```
27         property.SetValue(objTarget, null, null);
28     }
29
30     else
31     {
32         property.SetValue(objTarget, CloneObject(objPropertyValue), null);
33     }
34 }
35 }
36 }
37 return objTarget;
38 }
```

This can be also be achieved by an Object extension method that is described below.

## Using Object Extension Method

```
1 public static class ObjectExtension
2 {
3     public static object CloneObject(this object objSource)
4     {
5         //Get the type of source object and create a new instance of that type
6         Type typeSource = objSource.GetType();
7         object objTarget = Activator.CreateInstance(typeSource);
8         //Get all the properties of source object type
9         PropertyInfo[] propertyInfo = typeSource.GetProperties(BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Instance);
10        //Assign all source property to target object 's properties
11        foreach (PropertyInfo property in propertyInfo)
12        {
13            //Check whether property can be written to
14            if (property.CanWrite)
15            {
16                //check whether property type is value type, enum or string type
17                if (property.PropertyType.IsValueType || property.PropertyType.IsEnum || property.PropertyType.IsString)
18                {
19                    property.SetValue(objTarget, property.GetValue(objSource, null), null);
20                }
21                //else property type is object/complex types, so need to recursively call this method until the end
22                else
23                {
24                    object objPropertyValue = property.GetValue(objSource, null);
25                    if (objPropertyValue == null)
26                    {
27                        property.SetValue(objTarget, null, null);
28                    }
29                    else
30                    {
31                        property.SetValue(objTarget, objPropertyValue.CloneObject(), null);
32                    }
33                }
34            }
35        }
36        return objTarget;
37    }
38 }
```

## Sample code and output

```
2 class Program
3 {
4     static void Main(string[] args)
5     {
```

```

8      emp.EmployeeName = "Jignesh";
9      emp.Department = new Department { DepartmentId = 1, DepartmentName = "Engineering" };

11     Employee empClone1 = Utility.CloneObject(emp) as Employee;
12     //now Change Original Object Value
13     emp.EmployeeName = "Tejas";
14     emp.Department.DepartmentName = "Admin";
15     //Print original as well as clone object properties value.
16     Console.WriteLine("Original Employee Name : " + emp.EmployeeName);
17     Console.WriteLine("Original Department Name : " + emp.Department.DepartmentName);
18     Console.WriteLine("");
19     Console.WriteLine("Clone Object Employee Name (Clone Method) : " + empClone.EmployeeName);
20     Console.WriteLine("Clone Object Department Name (Clone Method) : " + empClone.Department.DepartmentName);
21     Console.WriteLine("");
22     Console.WriteLine("Clone Object Employee Name (Static Method) : " + empClone1.EmployeeName);
23     Console.WriteLine("Clone Object Department Name (Static Method) : " + empClone1.Department.DepartmentName);
24     Console.WriteLine("");
25     Console.WriteLine("Clone Object Employee Name (Extension Method) : " + empClone2.EmployeeName);
26     Console.WriteLine("Clone Object Department Name (Extension Method) : " + empClone2.Department.DepartmentName);
27     Console.ReadKey();
28 }
}

```

Deep-Copy-of-Object-in-Csharp-2.jpg

## Conclusion

Using Serialization and Reflection we can create a deep copy of an object. The only disadvantage of a deep copy using serialization is that we must mark our object as "serializable".

C# C# Deep Copy of Object Create deep copy of an object Reflection serialization

## OUR BOOKS



Jignesh Trivedi *TOP 50*

Jignesh Trivedi is a Developer, C# Corner MVP, Microsoft MVP, Author, Blogger, eager to learn new technologies

<https://www.c-sharpcorner.com/members/jignesh-trivedi>

6 40.8m 10 2

7 14



Type your comment here and press Enter Key (Minimum 10 characters)

Follow Comments



Good one! But usage of Binary formatter is deprecated as of oct 28 2022 due to deserialization security issues.

Arulvel Kumarasamy

2153 2 0

Nov 26, 2022

0 0 Reply




Hi, using reflection code to clone an object who has a Dictionary<Guid, Guid> property raise an exception of "Parameter count mismatch". it's a bug?

Gambero

2153 2 0

May 10, 2022

0 0 Reply

 1369 817 0 2152 3 0

0 1 Reply



No it does not work only with same type object but you can achieve this using reflection method (that required some modification) explain in this article

Jignesh Trivedi  
6 65k 40.8m

Mar 04, 2019  
0



I had the similar problem and used the first approach you have mentioned in 'Implement Deep Cloning using Serializing Deserializing objects'. Thanks :)

Manthan Vyas  
2152 3 0

Jul 12, 2018  
1 1 Reply



Thanks Manthan for your kind word

Jignesh Trivedi  
6 65k 40.8m

Jul 15, 2018  
0



Superb !!!

mewans  
2031 124 0

Jun 20, 2018  
1 0 Reply



hi jignesh.. i wish to copy a type of object into another .. wherein both are similar.. dont wish to use automapper .. wish to do it manually .. so we have the hold of it.. ne help wud be good.. thks :)

yogaa kapadia  
2151 4 0

Jul 28, 2015  
0 2 Reply



I did not understand question end to end, you can use any of above method to map one object with another.

Jignesh Trivedi  
6 65k 40.8m

Aug 04, 2015  
0



Yoga, Reflection is best suitable for your case.

Hemant Mahajan  
1194 1.1k 151.7k

Jun 07, 2016  
0



Found the code very useful but was wondering how I would modify it to cater for a collection in the class.

Raymond Brack  
2151 4 0

May 22, 2014  
0 0 Reply



Hi Jignesh,

Raymond Brack  
2151 4 0

May 22, 2014  
0 0 Reply



nice article.

Sourav Kayal  
70 27.7k 22.6m

Jul 15, 2013  
0 0 Reply

10 Productivity Tips for Software Developers

Examining The ConfigurationManager In .NET 6

How To Prevent XSS(Cross Site Scripting) Attacks In Angular

Apache Kafka Introduction, Installation, And Implementation Using .NET Core 6

Unit Of Work With Generic Repository Implementation Using .NET Core 6 Web API



#### CHALLENGE YOURSELF



C# Skill

#### GET CERTIFIED



Java Developer

