How do you do a deep copy of an object in .NET (C# specifically)? [duplicate]

Ask Question

529

211

This question already has an answer here:
Deep cloning objects *41 answers*

I want a true deep copy. In Java, this was easy, but how do you do it in C#?

c# .net serialization clone

edited Feb 2 '17 at 21:38

Peter Mortensen
14k 19 87 114

asked Sep 24 '08 at 19:39

user18931
4,459 9 24 21

marked as duplicate by Sheridan, Simon Halsey, trutheality, Pranav C Balan, Chris Feb 23 '14 at 3:43

This question has been asked before and already has an answer. If those answers do not fully address your question, please ask a new question.

1 What does a Deep Copy do? Does it copy the bitstream? –

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

object. A shallow copy will only create a new object and point all the fields to the original. – swilliams Sep 24 '08 at 19:46

1 A framework for copying/cloning .NET objects: github.com/havard/copyable – jgauffin Feb 18 '11 at 13:14

2 A deep copy creates a second instance of the object with the same values. A shallow copy (oversimplified) is like creating a second reference to an object. – Michael Blackburn Mar 30 '11 at 19:49

Use a Mapper, I suggest UltraMapper github.com/maurosampietro/UltraMapper – Mauro Sampietro Apr 23 '17 at 9:15

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

Teams
Q&A for work

Learn More

11 Answers

578

I've seen a few different approaches to this, but I use a generic utility method as such:

```
public static T DeepClone<T>(T obj)
{
 using (var ms = new MemoryStream())
 {
   var formatter = new BinaryFormatter();
   formatter.Serialize(ms, obj);
   ms.Position = 0;

   return (T) formatter.Deserialize(ms);
 }
}
```

Notes:

- Your class MUST be marked as `[Serializable]` in order for this to work.
- Your source file must include the following code:

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

edited Jul 6 '12 at 7:27

Gilles
77.1k 19 165 207

answered Sep 24 '08 at 19:40

Kilhoffer
21.1k 20 89 118

4 What happen if the object have event, Do they lost everything because of the serialization? – Patrick Desjardins Sep 24 '08 at 19:56

6 Event subscribes are included into serialization graph, since BinaryFormatter uses fields via reflection, and events are just fields of delegate types plus add/remove/invoke methods. You can use [field: NonSerialized] on event to avoid this. – Ilya Ryzhenkov Sep 24 '08 at 20:16

4 @Sean87: above the class declaration, add `[Serializable]` . so `[Serializable]public class Foo { }` will make `Foo` marked as serializable. – Dan Atkinson Aug 3 '11 at 22:51

13 Recursive MemberwiseClone will do deep copy too, it works 3 times faster then BinaryFormatter, doesn't require default constructor or any attributes. See my answer: stackoverflow.com/a/11308879/235715 – Alex Burtsev Jul 12 '12 at 4:19

1 This is creating a curious exception "assembly not found" while using this Utility code within the UserControlTestContainer. Its really weird because the assembly is loaded... – v.oddou May 21 '13 at 3:13

I wrote a deep object copy extension method, based on recursive **"MemberwiseClone"**. It is fast (**three times**

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

Source code:

```
using System.Collections.Generic;
using System.Reflection;
using System.ArrayExtensions;

namespace System
{
    public static class ObjectExtensions
    {
        private static readonly MethodInfo CloneMethod =
typeof(Object).GetMethod("MemberwiseClone", BindingFlags.N
BindingFlags.Instance);

        public static bool IsPrimitive(this Type type)
        {
            if (type == typeof(String)) return true;
            return (type.IsValueType & type.IsPrimitive);
        }

        public static Object Copy(this Object originalObje
        {
            return InternalCopy(originalObject, new Dictio
ReferenceEqualityComparer()));
        }
        private static Object InternalCopy(Object original
Object> visited)
        {
            if (originalObject == null) return null;
            var typeToReflect = originalObject.GetType();
            if (IsPrimitive(typeToReflect)) return origina
            if (visited.ContainsKey(originalObject)) retur
            if (typeof(Delegate).IsAssignableFrom(typeToRe
            var cloneObject = CloneMethod.Invoke(originalO
            if (typeToReflect.IsArray)
            {
                var arrayType = typeToReflect.GetElementTy
                if (IsPrimitive(arrayType) == false)
                {
                    Array clonedArray = (Array)cloneObject
                    clonedArray.ForEach((array, indices) =
array.SetValue(InternalCopy(clonedArray.GetValue(indices),
                }

            }
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
typeToReflect);
            return cloneObject;
        }

        private static void RecursiveCopyBaseTypePrivateFi
IDictionary<object, object> visited, object cloneObject, Ty
        {
            if (typeToReflect.BaseType != null)
            {
                RecursiveCopyBaseTypePrivateFields(origina
typeToReflect.BaseType);
                CopyFields(originalObject, visited, cloneO
BindingFlags.Instance | BindingFlags.NonPublic, info => in
            }
        }

        private static void CopyFields(object originalObje
object> visited, object cloneObject, Type typeToReflect, B
BindingFlags.Instance | BindingFlags.NonPublic | BindingFl
BindingFlags.FlattenHierarchy, Func<FieldInfo, bool> filte
        {
            foreach (FieldInfo fieldInfo in typeToReflect.
            {
                if (filter != null && filter(fieldInfo) ==
                if (IsPrimitive(fieldInfo.FieldType)) cont
                var originalFieldValue = fieldInfo.GetValu
                var clonedFieldValue = InternalCopy(origin
                fieldInfo.SetValue(cloneObject, clonedFiel
            }
        }
        public static T Copy<T>(this T original)
        {
            return (T)Copy((Object)original);
        }
    }

    public class ReferenceEqualityComparer : EqualityCompa
    {
        public override bool Equals(object x, object y)
        {
            return ReferenceEquals(x, y);
        }
        public override int GetHashCode(object obj)
        {
            if (obj == null) return 0;
            return obj.GetHashCode();
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
namespace ArrayExtensions
{
    public static class ArrayExtensions
    {
        public static void ForEach(this Array array, A
        {
            if (array.LongLength == 0) return;
            ArrayTraverse walker = new ArrayTraverse(a
            do action(array, walker.Position);
            while (walker.Step());
        }
    }

    internal class ArrayTraverse
    {
        public int[] Position;
        private int[] maxLengths;

        public ArrayTraverse(Array array)
        {
            maxLengths = new int[array.Rank];
            for (int i = 0; i < array.Rank; ++i)
            {
                maxLengths[i] = array.GetLength(i) - 1
            }
            Position = new int[array.Rank];
        }

        public bool Step()
        {
            for (int i = 0; i < Position.Length; ++i)
            {
                if (Position[i] < maxLengths[i])
                {
                    Position[i]++;
                    for (int j = 0; j < i; j++)
                    {
                        Position[j] = 0;
                    }
                    return true;
                }
            }
            return false;
        }
    }
}
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

edited Dec 10 '18 at 10:36

aloisdg
10.1k 2 47 57

answered Jul 3 '12 at 10:20

Alex Burtsev
8,225 5 49 80

3 Thanks Alex, yes I needed to call copy instead and that worked! – theJerm Feb 19 '13 at 20:23

2 @MattSmith You are absolutely right about GetHashCode, checked it, yep StackOverflow exception - gist.github.com/Burtsev-Alexey/11227277 – Alex Burtsev Apr 23 '14 at 18:33

2 @MattSmith It was working for delegates, but I intently disabled it (by setting null), see github.com/Burtsev-Alexey/net-object-deep-copy/issues/7, the subscribers were cloned, in the end if you had two object A and B connected (by event subscription) you would get objects A' and B' connected, this is correct but that's not what most people want when the clone objects. – Alex Burtsev Apr 23 '14 at 18:39

2 @MattSmith Strings are treated as primitives because 1: they are immutable, 2: calling protected MemberwiseClone on string will result in memory corruption, string data will turn into random chars, and soon .NET runtime will crash with internal error saying theer is a but in .NET :-) – Alex Burtsev Apr 23 '14 at 18:45

20 Clearly people like this answer but... it is a link-only answer. Perhaps you could include the relevant code here on Stack Overflow? – Taryn East Jul 16 '14 at 0:35

Building on Kilhoffer's solution...

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
public static class ExtensionMethods
{
    // Deep clone
    public static T DeepClone<T>(this T a)
    {
        using (MemoryStream stream = new MemoryStream())
        {
            BinaryFormatter formatter = new BinaryFormatte
            formatter.Serialize(stream, a);
            stream.Position = 0;
            return (T) formatter.Deserialize(stream);
        }
    }
}
```

which extends any class that's been marked as [Serializable] with a DeepClone method

```
MyClass copy = obj.DeepClone();
```

edited May 26 '15 at 15:03

Dann
8,453 7 36 48

answered Jul 31 '09 at 16:51

Neil
3,105 1 21 21

34 To that add "public static T DeepClone<T>(this T a) where T : ISerializable" – Amir Rezaei Feb 11 '11 at 14:35

14 @Amir - it isn't necessary for the class to implement ISerializable, Marking with SerializableAttribute is sufficient. The attribute uses reflection to perform serialization, while the interface allows you to write a custom serializer – Neil Feb 14 '11 at 16:10

9 I agree with your statement, but I like Amir's suggestion b/c it provides compile-time checking. Is there any way to reconcile

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

StringBuilder("TestData"); var copy = stringbuilder.DeepClone(); Assert.IsFalse(Equals(stringbuilder,copy)); Thanks a lot. – om471987 May 6 '12 at 18:17

2 @Neil This method is 10x slower than the NestedMemberwiseClone method, see my post on this page. – Contango May 20 '12 at 18:02

49

You can use **Nested MemberwiseClone to do a deep copy**. Its almost the same speed as copying a value struct, and its an order of magnitude faster than (a) reflection or (b) serialization (as described in other answers on this page).

Note that **if** you use **Nested MemberwiseClone for a deep copy**, you have to manually implement a ShallowCopy for each nested level in the class, and a DeepCopy which calls all said ShallowCopy methods to create a complete clone. This is simple: only a few lines in total, see the demo code below.

Here is the output of the code showing the relative performance difference (4.77 seconds for deep nested MemberwiseCopy vs. 39.93 seconds for Serialization). Using nested MemberwiseCopy is almost as fast as copying a struct, and copying a struct is pretty darn close to the theoretical maximum speed .NET is capable of.

```
Demo of shallow and deep copy, using classes and Membe
  Create Bob
    Bob.Age=30, Bob.Purchase.Description=Lamborghini
  Clone Bob >> BobsSon
  Adjust BobsSon details
    BobsSon.Age=2, BobsSon.Purchase.Description=Toy ca
  Proof of deep copy: If BobsSon is a true clone, then
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
    Demo of shallow and deep copy, using structs and value
      Create Bob
        Bob.Age=30, Bob.Purchase.Description=Lamborghini
      Clone Bob >> BobsSon
      Adjust BobsSon details:
        BobsSon.Age=2, BobsSon.Purchase.Description=Toy ca
      Proof of deep copy: If BobsSon is a true clone, then
will not affect Bob:
        Bob.Age=30, Bob.Purchase.Description=Lamborghini
      Elapsed time: 00:00:01.0875454,30000000
    Demo of deep copy, using class and serialize/deseriali
      Elapsed time: 00:00:39.9339425,30000000
```

To understand how to do a deep copy using MemberwiseCopy, here is the demo project:

```
// Nested MemberwiseClone example.
// Added to demo how to deep copy a reference class.
[Serializable] // Not required if using MemberwiseClone, o
using serialization.
public class Person
{
    public Person(int age, string description)
    {
        this.Age = age;
        this.Purchase.Description = description;
    }
    [Serializable] // Not required if using MemberwiseClon
    public class PurchaseType
    {
        public string Description;
        public PurchaseType ShallowCopy()
        {
            return (PurchaseType)this.MemberwiseClone();
        }
    }
    public PurchaseType Purchase = new PurchaseType();
    public int Age;
    // Add this if using nested MemberwiseClone.
    // This is a class, which is a reference type, so clon
    public Person ShallowCopy()
    {
        return (Person)this.MemberwiseClone();
    }
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
        {
                // Clone the root ...
            Person other = (Person) this.MemberwiseClone();
                // ... then clone the nested class.
            other.Purchase = this.Purchase.ShallowCopy();
            return other;
        }
    }
    // Added to demo how to copy a value struct (this is easy
    default)
    public struct PersonStruct
    {
        public PersonStruct(int age, string description)
        {
            this.Age = age;
            this.Purchase.Description = description;
        }
        public struct PurchaseType
        {
            public string Description;
        }
        public PurchaseType Purchase;
        public int Age;
        // This is a struct, which is a value type, so everyth
        public PersonStruct ShallowCopy()
        {
            return (PersonStruct)this;
        }
        // This is a struct, which is a value type, so everyth
        public PersonStruct DeepCopy()
        {
            return (PersonStruct)this;
        }
    }
    // Added only for a speed comparison.
    public class MyDeepCopy
    {
        public static T DeepCopy<T>(T obj)
        {
            object result = null;
            using (var ms = new MemoryStream())
            {
                var formatter = new BinaryFormatter();
                formatter.Serialize(ms, obj);
                ms.Position = 0;
                result = (T)formatter.Deserialize(ms);
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
    }
}
```

Then, call the demo from main:

```
    void MyMain(string[] args)
    {
        {
            Console.Write("Demo of shallow and deep copy, u
MemberwiseCopy:\n");
            var Bob = new Person(30, "Lamborghini");
            Console.Write("  Create Bob\n");
            Console.Write("    Bob.Age={0}, Bob.Purchase.D
Bob.Purchase.Description);
            Console.Write("  Clone Bob >> BobsSon\n");
            var BobsSon = Bob.DeepCopy();
            Console.Write("  Adjust BobsSon details\n");
            BobsSon.Age = 2;
            BobsSon.Purchase.Description = "Toy car";
            Console.Write("    BobsSon.Age={0}, BobsSon.Pu
BobsSon.Age, BobsSon.Purchase.Description);
            Console.Write("  Proof of deep copy: If BobsSo
adjusting BobsSon details will not affect Bob:\n");
            Console.Write("    Bob.Age={0}, Bob.Purchase.D
Bob.Purchase.Description);
            Debug.Assert(Bob.Age == 30);
            Debug.Assert(Bob.Purchase.Description == "Lamb
            var sw = new Stopwatch();
            sw.Start();
            int total = 0;
            for (int i = 0; i < 100000; i++)
            {
                var n = Bob.DeepCopy();
                total += n.Age;
            }
            Console.Write("  Elapsed time: {0},{1}\n", sw.
        }
        {
            Console.Write("Demo of shallow and deep copy, u
            var Bob = new PersonStruct(30, "Lamborghini");
            Console.Write("  Create Bob\n");
            Console.Write("    Bob.Age={0}, Bob.Purchase.D
Bob.Purchase.Description);
            Console.Write("  Clone Bob >> BobsSon\n");
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
            BobsSon.Purchase.Description = "Toy car";
            Console.Write("    BobsSon.Age={0}, BobsSon.Pu
BobsSon.Age, BobsSon.Purchase.Description);
            Console.Write("  Proof of deep copy: If BobsSo
adjusting BobsSon details will not affect Bob:\n");
            Console.Write("    Bob.Age={0}, Bob.Purchase.D
Bob.Purchase.Description);
            Debug.Assert(Bob.Age == 30);
            Debug.Assert(Bob.Purchase.Description == "Lamb
            var sw = new Stopwatch();
            sw.Start();
            int total = 0;
            for (int i = 0; i < 100000; i++)
            {
                var n = Bob.DeepCopy();
                total += n.Age;
            }
            Console.Write("  Elapsed time: {0},{1}\n", sw.
        }
        {
            Console.Write("Demo of deep copy, using class
serialize/deserialize:\n");
            int total = 0;
            var sw = new Stopwatch();
            sw.Start();
            var Bob = new Person(30, "Lamborghini");
            for (int i = 0; i < 100000; i++)
            {
                var BobsSon = MyDeepCopy.DeepCopy<Person>(
                total += BobsSon.Age;
            }
            Console.Write("  Elapsed time: {0},{1}\n", sw.
        }
        Console.ReadKey();
    }
```

Again, note that **if** you use **Nested MemberwiseClone for a deep copy**, you have to manually implement a ShallowCopy for each nested level in the class, and a DeepCopy which calls all said ShallowCopy methods to create a complete clone. This is simple: only a few lines in total, see the demo code above.

Note that when it comes to cloning an object, there is is a

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

- If you have a "struct", it's a value type so you can just copy it, and the contents will be cloned.
- If you have a "class", it's a reference type, so if you copy it, all you are doing is copying the pointer to it. To create a true clone, you have to be more creative, and use a method which creates another copy of the original object in memory.
- Cloning objects incorrectly can lead to very difficult-to-pin-down bugs. In production code, I tend to implement a checksum to double check that the object has been cloned properly, and hasn't been corrupted by another reference to it. This checksum can be switched off in Release mode.
- I find this method quite useful: often, you only want to clone parts of the object, not the entire thing. It's also essential for any use case where you are modifying objects, then feeding the modified copies into a queue.

Update

It's probably possible to use reflection to recursively walk through the object graph to do a deep copy. WCF uses this technique to serialize an object, including all of its children. The trick is to annotate all of the child objects with an attribute that makes it discoverable. You might lose some performance benefits, however.

Update

Quote on independent speed test (see comments below):

> I've run my own speed test using Neil's serialize/deserialize extension method, Contango's Nested MemberwiseClone, Alex Burtsev's reflection-based extension method and AutoMapper, 1 million

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

> seconds. Much faster was the reflection-based method which took 2.4 seconds. By far the fastest was Nested MemberwiseClone, taking 0.1 seconds. Comes down to performance versus hassle of adding code to each class to clone it. If performance isn't an issue go with Alex Burtsev's method. – Simon Tewsi

edited Jun 8 '16 at 16:32

answered Dec 30 '11 at 19:05

Contango

42k 47 190 239

1 Good post. Any idea why serialization is so much slower? Also, how would your checksum work? Why not just have an equality checker? – user420667 Apr 1 '12 at 15:33

3 I can confirm that this is much faster than the serialization method. The cost is: writing more code; the maintenance risk of adding a field without adding it to the clone method; need to write helper classes for any 3rd party classes (such as Dictionary<>) – Neil May 23 '12 at 14:37

1 You can create an extension method that works on any object, see my answer stackoverflow.com/a/11308879/235715 – Alex Burtsev Jul 12 '12 at 4:16

3 It's too bad neither Java nor .NET distinguishes among references that encapsulate identity, mutable state, both, or neither. Conceptually, there should only be one type of "clone": a new object where each reference encapsulates the same thing as in the corresponding reference in the original. If a reference encapsulates identity, the clone's reference must refer to the *same* object. If it encapsulates mutable state but *not* identity, the clone must receive a reference to a different object with the same state [otherwise both references would erroneously... – supercat Sep 23 '13 at 19:53

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

that object--a feat which is often difficult or impossible. While references to some types of object will usually be used to encapsulate identity, and references to others will usually encapsulate mutable state, knowing the type of an object is not sufficient to the purpose for which a reference is held. – supercat Sep 23 '13 at 20:01

15

I believe that the BinaryFormatter approach is relatively slow (which came as a surprise to me!). You might be able to use ProtoBuf .NET for some objects if they meet the requirements of ProtoBuf. From the ProtoBuf Getting Started page (http://code.google.com/p/protobuf-net/wiki/GettingStarted):

Notes on types supported:

Custom classes that:

- Are marked as data-contract
- Have a parameterless constructor
- For Silverlight: are public
- Many common primitives, etc.
- *Single* dimension arrays: T[]
- List<T> / IList<T>
- Dictionary<TKey, TValue> / IDictionary<TKey, TValue>
- any type which implements IEnumerable<T> and has an Add(T) method

The code assumes that types will be mutable around the elected members. Accordingly, custom structs are not supported, since they should be immutable

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
public static void deepCopy<T>(ref T object2Copy, ref T ob
{
    using (var stream = new MemoryStream())
    {
        Serializer.Serialize(stream, object2Copy);
        stream.Position = 0;
        objectCopy = Serializer.Deserialize<T>(stream);
    }
}
```

Which is VERY fast indeed...

Edit:

Here is working code for a modification of this (tested on .NET 4.6). It uses System.Xml.Serialization and System.IO. No need to mark classes as serializable.

```
public void DeepCopy<T>(ref T object2Copy, ref T objectCop
{
    using (var stream = new MemoryStream())
    {
        var serializer = new XS.XmlSerializer(typeof(T));

        serializer.Serialize(stream, object2Copy);
        stream.Position = 0;
        objectCopy = (T)serializer.Deserialize(stream);
    }
}
```

edited Nov 20 '17 at 12:03

HockeyJ
8,769 8 58 86

answered Jan 3 '12 at 5:21

Kurt Richardson
151 1 4

Wonder how fast it is compared to the Nested

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

7

You can try this

```
public static object DeepCopy(object obj)
{
    if (obj == null)
        return null;
    Type type = obj.GetType();

    if (type.IsValueType || type == typeof(string))
    {
        return obj;
    }
    else if (type.IsArray)
    {
        Type elementType = Type.GetType(
             type.FullName.Replace("[]", string.Empty)
        var array = obj as Array;
        Array copied = Array.CreateInstance(elementTyp
        for (int i = 0; i < array.Length; i++)
        {
            copied.SetValue(DeepCopy(array.GetValue(i)
        }
        return Convert.ChangeType(copied, obj.GetType(
    }
    else if (type.IsClass)
    {

        object toret = Activator.CreateInstance(obj.Ge
        FieldInfo[] fields = type.GetFields(BindingFla
                    BindingFlags.NonPublic | BindingFl
        foreach (FieldInfo field in fields)
        {
            object fieldValue = field.GetValue(obj);
            if (fieldValue == null)
                continue;
            field.SetValue(toret, DeepCopy(fieldValue)
        }
        return toret;
    }
    else
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

Thanks to DetoX83 article on code project.

answered Apr 1 '12 at 3:59

Suresh Kumar Veluswamy

2,625 2 14 24

1 It works only if your obj has default constructor! – Cyrus Aug 23 '14 at 12:50

5

Maybe you only need a shallow copy, in that case use `Object.MemberWiseClone()` .

There are good recommendations in the documentation for `MemberWiseClone()` for strategies to deep copy: -

http://msdn.microsoft.com/en-us/library/system.object.memberwiseclone.aspx

edited Jul 4 '11 at 19:33

axel22

28.2k 9 110 129

answered Dec 1 '10 at 3:00

David Thornley

876 1 10 14

3 Good try, but he specifically asked for a deep clone. – Levitikon Sep 26 '11 at 17:38

You can do a deep clone with MemberwiseClone, all you do is add nesting. See answer from @Gravitas above. – Contango Dec 30 '11 at 19:22

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

3

The best way is:

```
public interface IDeepClonable<T> where T : class
{
    T DeepClone();
}

public class MyObj : IDeepClonable<MyObj>
{
    public MyObj Clone()
    {
        var myObj = new MyObj();
        myObj._field1 = _field1;//value type
        myObj._field2 = _field2;//value type
        myObj._field3 = _field3;//value type

        if (_child != null)
        {
            myObj._child = _child.DeepClone(); //refer
does the same
        }

        int len = _array.Length;
        myObj._array = new MyObj[len]; // array / coll
        for (int i = 0; i < len; i++)
        {
            myObj._array[i] = _array[i];
        }

        return myObj;
    }

    private bool _field1;
    public bool Field1
    {
        get { return _field1; }
        set { _field1 = value; }
    }

    private int _field2;
    public int Property2
    {
        get { return _field2; }
        set { _field2 = value; }
    }
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
        {
            get { return _field3; }
            set { _field3 = value; }
        }

        private MyObj _child;
        private MyObj Child
        {
            get { return _child; }
            set { _child = value; }
        }

        private MyObj[] _array = new MyObj[4];
    }
```

edited Feb 2 '17 at 21:47

Peter Mortensen
14k 19 87 114

answered Apr 22 '12 at 11:23

alex
39 2

Using CGbR Clone Generator you get the same result without manually writing the code. – Toxantron Jun 9 '16 at 20:50

0

The MSDN documentation seems to hint that Clone should perform a deep copy, but it is never explicitly stated:

The ICloneable interface contains one member, Clone, which is intended to support cloning beyond that supplied by MemberWiseClone… The MemberwiseClone method creates a shallow copy…

You can find my post helpful.

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

answered Feb 13 '13 at 13:50

Eugene

1

The problem with ICloneable is that the Clone method does not explicitly specify whether it is performing a shallow or deep copy, so callers can never be sure. Hence, there is some [discussion|blogs.msdn.com/brada/archive/2004/05/03/125427.aspx] about making ICloneable obsolete in the .NET Framework. – Mahmoud Samy Aug 8 '16 at 18:54

0

```
    public static object CopyObject(object input)
    {
        if (input != null)
        {
            object result = Activator.CreateInstance(input
            foreach (FieldInfo field in
input.GetType().GetFields(Consts.AppConsts.FullBindingList
            {
                if (field.FieldType.GetInterface("IList", 
                {
                    field.SetValue(result, field.GetValue(
                }
                else
                {
                    IList listObject = (IList)field.GetVal
                    if (listObject != null)
                    {
                        foreach (object item in ((IList)fi
                        {
                            listObject.Add(CopyObject(item
                        }
                    }
                }
            }
            return result;
        }
        else
        {
            return null;
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

This way is a few times faster than `BinarySerialization`
AND this does not require the `[Serializable]` attribute.

edited Jul 17 '17 at 21:11

Peter Mortensen
14k 19 87 114

answered Jul 4 '11 at 13:57

Basil
25 1

1 You're not continuing the deep copy down your non-IList branch and I think you would have issues with ICollection/IEnumerable. – Rob McCready Jul 5 '11 at 7:04

Using the "Nested MemberwiseClone" technique is an order of magnitude faster again (see my post under @Gravitas). – Contango Jan 1 '12 at 23:29

4 what are Consts ? – user5447154 Nov 13 '15 at 15:47

2 What is `Consts.AppConsts.FullBindingList` ? – Developer Apr 24 '18 at 7:10

I have a simpler idea. Use LINQ with a new selection.

-6

```
public class Fruit
{
  public string Name {get; set;}
  public int SeedCount {get; set;}
}

void SomeMethod()
{
  List<Fruit> originalFruits = new List<Fruit>();
  originalFruits.Add(new Fruit {Name="Apple", SeedCount=10
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
    List<Fruit> deepCopiedFruits = from f in originalFruits
                select new Fruit {Name=f.Name, SeedCount=f.S
}
```

edited Feb 2 '17 at 21:49

Peter Mortensen
14k 19 87 114

answered Jan 25 '13 at 11:42

Jordan Morris
940 12 30

4 It's not simpler when you have mutable reference types, many properties, lists with sublists, etc. This can work in a few simple scenarios, but is still error-prone when your `Fruit` class adds another property and you forget to change your method. – Nelson Rothermel Feb 4 '13 at 19:32

Reference assignment 'Name=f.Name' is shallow copying. – Tarec Feb 5 '14 at 10:35

1 if we have 100 properties then our whole day will be consumed just writing such code for one object and so on... – Umar Abbas Apr 10 '14 at 5:49

@Tarec, if Name is a string, and strings are immutable, it shouldn't matter. – Arturo Torres Sánchez May 20 '15 at 17:58

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.