Net Deep Copy/Cloning

Below are some references for different methods of accomplishing deep copies of object types.

NOTE: You will need to evaluate which technique is better for your use case, as there is no single solution for all. This is because of several factors:

- Time-cost
  Faster methods generally require adding cloning methods to each class type.
- Reflection usage (or not)
  Reflection can be very slow, if not building an expression tree to reuse
- If the type is serializable
  Not all types can be serialized, without losing state.
  Specifically, an object type must have enough serializable properties to recreate its state.
  If part of its state is stored some private field, then serialization will not produce a faithful copy.
- If the type is derived from another
  Some serialization methods have trouble handling properties of a type's base classes.
  And, there are cases where serializing a list of objects that have a shared base type, ends up with a list of base type copies, losing all derived type data.
  This is especially true when cloning, via generic T serialization, from a list of derived objects of a common base.

Quick and Dirty

Obviously, any deep cloning method that uses serialization, will ensure a complete copy.

Good reference: https://medium.com/@dayanandthombare/object-cloning-in-c-a-comprehensive-guide---d3b79ed6ebcd

This article benchmarks the different methods: https://code-maze.com/csharp-deep-copy-of-object/

https://github.com/havard/copyable

https://github.com/force-net/DeepCloner

Reflection Deep Copy

Here’s a reflective method that performs a deep copy.

It inspects the actual type of the given instance, in case the given instance was passed in as a base type.
Doing so, allows it to work with collections of derived types that share a common base.

NOTE: This method uses reflection, so it’s slow.
NOTE: This method also ignore any private fields of the type, which may lose state during the copy.

```
static public T DeepCopyReflection<T>(T input)
{
    // Get the actual type of the instance, in case it's a derived type,
    //  but was passed in as a base.
    var type = input.GetType();
    // Get the properties of the fully-derived type...
    var properties = type.GetProperties();
    // Create an instance of the fully derived type, and cast it back down
    //  to the base type given...
    T clonedObj = (T)Activator.CreateInstance(type);

    // Iterate properties of the type...
    foreach (var property in properties)
    {
        if (property.CanWrite)
        {
            object value = property.GetValue(input);
            if (value != null && value.GetType().IsClass && !value.GetType().FullName.StartsWith("System."))
            {
                property.SetValue(clonedObj, DeepCopyReflection(value));
            }
            else
            {
                property.SetValue(clonedObj, value);
            }
        }
    }
    return clonedObj;
}
```

Revision #2
Created 8 January 2025 04:33:47 by glwhite
Updated 19 March 2025 15:40:08 by glwhite