OmniAccessor.NET Documentation

Release 1.0.0

Brandon Sara

November 25, 2016

Contents

CHAPTER 1

Table of Contents

1.1 Introduction to OmniAccessor.NET

1.1.1 Why?

Name

Why the name "OmniAccessor"? Because this library gives you access to everything and the definition of the English prefix "omni" is as follows:

> Omni is a Latin prefix meaning "all" or "every".

Purpose

This library was created to help with two things:

1. Simplify the use of the reflectoin APIs provided natively in .NET
2. Unit Testing

Simplified Reflection API

A simplified reflection API is needed in native .NET, and this library provides that. Do you have a situation where you need to use reflection to access **public** fields? No problem! For invoking a method, just call `OmniAccessor.InvokeMethod(myObj, "myMethod")`, For invoking a property getter, call `OmniAccessor.GetPropertyValue(myObj, "MyProp")` (by default, only `public` fields are allowed, if you want to access fields that are not `public`, you would add a second parameter of type `BindingFlags`, which is native to .NET. Example: `OmniAccessor.GetPropertyValue(myObj, 'myProp', BindingFlags.NonPublic))`. On top of that, all of the methods provided by the library are extension methods, allowing you to easily find each available method via your code completion in Visual Studio or Mono Develop. It also allows for a much more sensible syntax: `OmniAcessor.InvokeMethod("MyMethod")`.

You also don't have to know which class in the inheritance tree implements the thing that you want to access. If you want to use reflection to access something that is inherited from a parent class but is not overriden by the child class, then you can't use the native .NET reflection API to access the field unless you try to do so on the parent class. This library eliminates the need to do this, you simply make the call to access the field in some way, and it will happen, regardless of whether or not the object that owns the field actually implements that field itself.

Unit Tests

When writing unit tests, you may want to change a value here or there to ensure that everthing fails the way you want it to. Granted, generally, the best way to accomplish this is through creating interfaces for objects being tested and substituting in custom test objects which implement those inerfaces, allowing one to control everything that is returned by the object's property getters and methods. Or, one can use stubs and shims (which essentially uses the same concept of creating interfaces for test objects). However, it is still sometimes useful to change data via direct maniuplation. One obvious case is if you need to alter data coming from a third-party library that you don't have any ability to alter. If the third-party library does not provide interfaces for the objects that you're testing, you can just change it **with one simple method call**.

1.1.2 Basic Usage

Note: The example below does not contain an example for every single method made available by the library. See the reference documentation for a full list of available methods and how to use them.

```
using OmniAccessor;


namespace MyNamespace
{
  public class MyClass
  {
    public static class MyMethod(MyObj obj)
    {
      // Invoke property getter via static method
      OmniAccessor.GetPropertyValue(obj, "MyProperty");


      // Invoke property getter/setter via extension method
      var propVal = obj.GetPropertyValue("PropName");
      obj.SetPropertyValue("Propname", 42);


      // Invoke public instance method
      var instanceMethodReturnVal = obj.InvokeMethod("MyInstanceMethod");


      // Invoke public static method
      var staticMethodReturnVal = MyObj.InvokeStaticMethod("MyStaticMethod");


      // Invoke private/protected method
      var nonPublicMethodReturnVal = obj.InvokeMethod("MyPrivateMethod", System.Reflection.BindingFla


      // Get/Set value of public instance variable
      var instanceFieldVal = obj.GetFieldValue("MyPublicInstanceVar");
      obj.SetFieldValue("MyPublicInstanceVar", "Fish fingers and custard!");


      // Get/Set value of public static variable
      var staticFieldVal = MyObj.GetStaticFieldValue("MyPublicStaticVar");
      MyObj.SetStaticFieldValue("MyPublicStaticVar", "Did you hear about Pluto?");
```

```
      // Get method custom attributes (you can also get the custom attributes of
      // any other type of field: properties, variables, etc.)
      Object[] instanceMethodAttrs = obj.GetMethodCustomAttributes("MyInstanceMethod");
      Object[] staticMethodAttrs   = MyObj.GetMethodCustomAttributes("MyStaticMethod");


      // Get method info (you can also get the info object of any other type of
      // field: properties, variables, etc.)
      System.Reflection.MethodInfo methodInfo = obj.GetMethodInfo("MyInstanceMethod");
    }
  }
}
```

1.2 Reference

1.2.1 MissingPropertyException Class

1.2.2 OmniAccessor Class

1.3 Change Log

1.4 License