Using the PropertyChangeLogger library

Project location: https://github.com/ScottLilly/PropertyChangeLogger

Contact e-mail: github@scottlilly.com

Contents

What is it?

PropertyChangeLogger is a library that lets you easily see if any of the (non-list) properties of your objects were changed. If a property was changed, you can see its original value and current value.

Why would I want to use it?

If you need to create an audit log entry, whenever someone changes a value, this is an easy way for you to do that. You don't need to manually write unique code in each class, maintain that class' original values, and have a unique routine for each class to look for changes.

How can I use it?

1. Download the PropertyChangeLogger project (or DLL) from here and include it in your solution.
2. Create your business class, using PropertyChangeLogger as its base class.
3. Apply any extra attributes to your properties (if needed).
4. If you have a constructor for your class that accepts parameters, and assigns those values to your properties, call the SetInitialPropertyValues() method after you've set the values.
5. To see if your object has changed properties, check the Boolean property "HasChangedProperties"on your class.
6. To get a list of the changed properties, check the "ChangedProperies" property. It returns a list of PropertyChange object which contain the PropertyName, OriginalValue, and CurrentValue.

Here's how a sample class would look:

Customer.cs

```
using PropertyChangeLogger;

namespace TestPropertyChangeLogger
{
    public class Customer : PropertyChangeLoggingClass
    {
        public string Name { get; set; }
        [DoNotLogValuesWhenThisPropertyChanges]
        public string Password { get; set; }
        [DoNotLogChangesToThisProperty]
        public double ComputedValue { get; set; }

        public Customer()
        {
        }

        public Customer(string name, string password, double
computedValue)
        {
            Name = name;
            Password = password;
            ComputedValue = computedValue;

            SetInitialPropertyValues();
        }
    }
}
```

In this class, if the "Name" property is changed, HasChangedProperties will return true. If the "Password" property is changed, HasChangedProperties will return true; however, the OriginalValue and ChangedValue will be null when you check the ChangedProperties list. If the "ComputedValue" property is changed, it will not cause HasChangedProperties to be true, and it will not add a PropertyChange item to the ChangedProperties list.

Here is a unit test (from the solution) that shows you how to detect if a property was changed.

```
[TestMethod]
public void
EmptyConstructor_OneLoggedOneLoggedWithoutValuesPropertyChanged()
{
    Customer customer = new Customer();

    customer.Name = CUSTOMER_ORIGINAL_NAME;
    customer.Password = CUSTOMER_ORIGINAL_PASSWORD;

    Assert.IsTrue(customer.HasChangedProperties);
    Assert.AreEqual(2, customer.ChangedProperties.Count);
    Assert.AreEqual(null, customer.ChangedProperties.First(x =>
x.PropertyName == "Name").OriginalValue);
```

```
    Assert.AreEqual(CUSTOMER_ORIGINAL_NAME,
customer.ChangedProperties.First(x => x.PropertyName ==
"Name").CurrentValue);
    Assert.AreEqual(null, customer.ChangedProperties.First(x =>
x.PropertyName == "Password").OriginalValue);
            Assert.AreEqual(null, customer.ChangedProperties.First(x
=> x.PropertyName == "Password").CurrentValue);
}
```

If your class has a property that is a type which has PropertyChangeLogger as its base class, you will also see any changes made to the child (and all descendant children) properties that use PropertyChangeLogger as their base.

For example:

ExpandedCustomer.cs

```
using System.Collections.Generic;

using PropertyChangeLogger;

namespace TestPropertyChangeLogger
{
    public class ExpandedCustomer : PropertyChangeLoggingClass
    {
        public string Name { get; set; }
        [DoNotLogValuesWhenThisPropertyChanges]
        public string Password { get; set; }
        [DoNotLogChangesToThisProperty]
        public double ComputedValue { get; set; }
        public Address BillingAddress { get; set; }
        public List<string> EmployeeName { get; set; } // List
properties are currently not checked for changes

        public ExpandedCustomer(string name, string password, double
computedValue, Address billingAddress)
        {
            Name = name;
            Password = password;
            ComputedValue = computedValue;
            BillingAddress = billingAddress;

            SetInitialPropertyValues();
        }
    }
}
```

Address.cs

```
using PropertyChangeLogger;

namespace TestPropertyChangeLogger
{
    public class Address : PropertyChangeLoggingClass
    {
        public string StreetAddress { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public PostalCode PostCode { get; set; }

        public Address()
        {
        }

        public Address(string streetAddress, string city, string
state, PostalCode postCode)
        {
            StreetAddress = streetAddress;
            City = city;
            State = state;
            PostCode = postCode;

            SetInitialPropertyValues();
        }
    }
}
```

PostalCode.cs

```
using System;

using PropertyChangeLogger;

namespace TestPropertyChangeLogger
{
    public class PostalCode : PropertyChangeLoggingClass
    {
        [DoNotLogChangesToThisProperty]
        public string Prefix { get; set; }
        [DoNotLogChangesToThisProperty]
        public string Suffix { get; set; }

        public string Value
        {
            get
            {
                return String.IsNullOrWhiteSpace(Suffix) ?
                               Prefix :
                               string.Format("{0}-{1}", Prefix, Suffix);
            }
        }

        public PostalCode(string prefix, string suffix)
        {
            Prefix = prefix;
            Suffix = suffix;

            SetInitialPropertyValues();
        }
    }
}
```

Here's a unit test that detects a change to the Value property, which is a "grand-child" from the ExpandedCustomer class.

Note that the PropertyName of the PropertyChange object has the names of the parent, child, and grandchild properties, separated by a ".".

```
[TestMethod]
public void PopulatedConstructor_OneChildPropertyChanged()
{
    Address address = new Address("1234 Main Street", "Houston",
"Texas", new PostalCode("77777", "6666"));
    ExpandedCustomer customer = new ExpandedCustomer("CustomerName",
"CustomerPassword", 123, address);

    customer.BillingAddress.PostCode.Prefix = "11111";

    Assert.IsTrue(customer.HasChangedProperties);
    Assert.AreEqual(1, customer.ChangedProperties.Count);
    Assert.AreEqual("BillingAddress.PostCode.Value",
customer.ChangedProperties[0].PropertyName);
```

```
    Assert.AreEqual("77777-6666",
customer.ChangedProperties[0].OriginalValue);
    Assert.AreEqual("11111-6666",
customer.ChangedProperties[0].CurrentValue);
}
```

What are the available property attributes?

DoNotLogChangesToThisProperty

If you add this attribute before a property, it will not be checked for changes to the original value.

DoNotLogValuesWithThisPropertyChange

If you add this attribute before a property, it will be checked for changes. However, the PropertyChange object for this property will have null values for the OriginalValue and CurrentValue.

This is useful if you are going to record property changes to an audit log, but don't want to have the property values shown - for security reasons. For instance, properties for passwords, government ID numbers, salaries, etc.