Some sort of creational pattern needed in C#

Asked 16 years, 3 months ago Modified 16 years, 3 months ago Viewed 495 times

4

I have the following type :

```
// incomplete class definition
public class Person
{
    private string name;

    public string Name
    {
        get { return this.name; }
    }
}
```

I want this type to be **created** and **updated** with some sort of dedicated controller/builder, but I want it to remain **read-only for other types**.

This object also needs to fire an event every time it is updated by its controller/builder.

To summary, according to the previous type definition skeleton :

- The `Person` could only be instantiated by a specific controller
- This controller could **update** the state of the `Person` (`name` field) at any time
- The `Person` need to send a **notification** to the rest of the world when it occurs
- All other types should only be able to **read** `Person` attributes

How should I implement this ? I'm talking about a controller/builder here, but all others solutions are welcome.

Note : *I would be able to rely on the `internal` modifier, but ideally all my stuff should be in the same assembly.*

c# .net design-patterns

Share Improve this question Follow

asked Sep 17, 2008 at 19:32

Romain Verdier

13k ● 7 ● 59 ● 77

6 Answers

Sorted by: Highest score (default)

5

Create an interface IReadOnlyPerson which exposes only get accessors. Have Person implement IReadOnlyPerson. Store the reference to Person in your controller. Give other clients only the read only version.

This will protect against mistakes, but not fraud, as with most OO features. Clients can runtime cast to Person if they happen to know (or suspect) IReadOnlyPerson is implemented by Person.

Update, per the comment:

The Read Only interface may also expose an event delegate, just like any other object. The idiom generally used in C# doesn't prevent clients from messing with the list of listeners, but convention is only to add listeners, so that should be adequate. Inside any set accessor or function with state-changing side effects, just call the event delegate with a guard for the null (no listeners) case.

Share
Improve this answer
Follow

edited Sep 18, 2008 at 16:58

answered Sep 17, 2008 at 19:39

JasonTrue
19.6k ● 5 ● 37 ● 61

This answer could be the good one, but it doesn't cover the second part of the question : The Person need to send a notification to the rest of the world when it occurs – Seb Sep 18, 2008 at 7:51

2 Check out the "System.ComponentModel.INotifyPropertyChanged" interface for the standard method of notifying interested parties that a property of the object has changed. – Adrian Clark Sep 18, 2008 at 17:07

1

I like to have a read-only interface. Then the builder/controller/whatever can reference the object directly, but when you expose this object to the outside you show only the interface.

Share Improve this answer Follow

answered Sep 17, 2008 at 19:34

Jason Cohen
83k ● 26 ● 110 ● 114

1

Use an interface `IPerson` and a nested class:

```
public class Creator
{
```

```
    private class Person : IPerson
    {
        public string Name { get; set; }
    }

    public IPerson Create(...) ...


    public void Modify(IPerson person, ...)
    {
        Person dude = person as Person;
        if (dude == null)
            // wasn't created by this class.
        else
            // update the data.
    }
}
```

Share Improve this answer Follow

answered Sep 17, 2008 at 19:38

fryguybob

4,410 ● 2 ● 30 ● 36

1

I think `internal` is the least complex and best approach (this of course involves multiple assemblies). Short of doing some overhead intensive stack walking to determine the caller in the property setter you could try:

```
interface IPerson
{
    Name { get; set; }
}
```

and implement this interface explicitly:

```
class Person : IPerson
{
    Name { get; private set; }
    string IPerson.Name { get { return Name; } set { Name = value; } }
}
```

then perform explicit interface casts in your builder for setting properties. This still doesn't protect your implementation and isn't a good solution though it does go some way to emphasize your intention.

In your property setters you'll have to implement an event notification. Approaching this problem myself I would not create separate events and event handlers for each property but instead create a single PropertyChanged event and fire it in each property when a change occurs (where the event arguments would include the property name, old value, and new value).

answered Sep 17, 2008 at 19:48

cfeduke
23.2k ● 10 ● 63 ● 65

1

Seems odd that, though I cannot change the name of the Person object, I can simply grab its controller and change it there. That's not a good way to secure your object's data.

But, notwithstanding, here's a way to do it:

```
    /// <summary>
    /// A controlled person.  Not production worthy code.
    /// </summary>
    public class Person
    {
        private string _name;
        public string Name
        {
            get { return _name; }
            private set
            {
                _name = value;
                OnNameChanged();
            }
        }
        /// <summary>
        /// This person's controller
        /// </summary>
        public PersonController Controller
        {
            get { return _controller ?? (_controller = new
PersonController(this)); }
        }
        private PersonController _controller;

        /// <summary>
        /// Fires when <seealso cref="Name"/> changes.  Go get the new name
yourself.
        /// </summary>
        public event EventHandler NameChanged;

        private void OnNameChanged()
        {
            if (NameChanged != null)
                NameChanged(this, EventArgs.Empty);
        }

        /// <summary>
        /// A Person controller.
        /// </summary>
        public class PersonController
        {
            Person _slave;
            public PersonController(Person slave)
            {
```

```
                _slave = slave;
            }
            /// <summary>
            /// Sets the name on the controlled person.
            /// </summary>
            /// <param name="name">The name to set.</param>
            public void SetName(string name) { _slave.Name = name; }
        }
    }
```

Share Improve this answer Follow

answered Sep 17, 2008 at 20:00

user1228

0

Maybe something like that ?

```
public class Person
{
    public class Editor
    {
        private readonly Person person;

        public Editor(Person p)
        {
            person = p;
        }

        public void SetName(string name)
        {
            person.name = name;
        }

        public static Person Create(string name)
        {
            return new Person(name);
        }
    }

    protected string name;

    public string Name
    {
        get { return this.name; }
    }

    protected Person(string name)
    {
        this.name = name;
    }
}

Person p = Person.Editor.Create("John");
Person.Editor e = new Person.Editor(p);
e.SetName("Jane");
```

Not pretty, but I think it works. Alternatively you can use properties instead of SetX methods on the editor.

Share

Improve this answer

Follow

edited Sep 17, 2008 at 19:47

answered Sep 17, 2008 at 19:41

David Thibault

8,736 ● 3 ● 39 ● 51