Tips for writing fluent interfaces in C# 3

Asked 16 years, 2 months ago Modified 6 years, 7 months ago

Viewed 17k times

43

I'm after some good tips for fluent interfaces in C#. I'm just learning about it myself but keen to hear what others think outside of the articles I am reading. In particular I'm after:

1. when is fluent too much?
2. are there any fluent patterns?
3. what is in C# that makes fluent interfaces more fluent (e.g. extension methods)
4. is a complex fluent interface still a fluent one?
5. refactoring to arrive at a fluent interface or refactoring an existing fluent interface
6. any good examples out there that you have worked with or could recommend?

If you could post one tip or thought, or whatever per post. I want to see how they get voted on, too.

Thank you in advance.

c# design-patterns fluent-interface

Share

Improve this question

Follow

edited Jul 12, 2013 at 18:58

dreftymac

32.3k ● 26 ● 124 ● 187

asked Oct 22, 2008 at 7:13

Scott McKenzie

16.2k ● 9 ● 48 ● 74

here's a tip: discoverability and simplicity are much more important than the little added readability a fluent API gives in most cases. Fluent the way LinQ (the methods synatax) does it is great, building a fluent DSL with C# just creates an API that is impossible to understand. – AK_ Jan 30, 2014 at 9:07

8 Answers

Sorted by: Highest score (default)

27

The single biggest challenge I have experienced as a consumer of fluent interfaces is that most of them aren't really fluent intefaces -- instead they are really instances of what I tend to refer to as 'legible interfaces'.

A fluent interface implies that its primary goal is to make it easy to SPEAK it whereas a legible interface implies that its primary goal is to be easy to READ it. Most fluent interfaces only tend to be ridiculously difficult to code with but conversely incredibly easy to READ later by others.

```
Assert().That().This(actual).Is().Equal().To(expected)
    Except().If(x => x.GreaterThan(10));
```

...is alot easier to read later than it is to actually compose in code!

Share Improve this answer

Follow

edited Feb 16, 2009 at 17:55

Jeff Atwood

63.9k ● 48 ● 151 ● 153

answered Feb 1, 2009 at 13:27

sbohlen

2,019 ● 1 ● 16 ● 19

13 Is it really easier to read, than just plain `actual==expected || actual>10` ? – Yaroslav Yakovlev Mar 14, 2014 at 13:17

18

On your 4th point;

Yes I think that a complex fluent interface can still be fluent.

I think fluent interfaces are somewhat of a compromise. (although a good one!) There has been much research into using natural language for programming and generally natural language isn't precise enough to express programs.

Fluent interfaces are constructed so that they write like a programming language, only a small subset of what you can express in a natural language is allowed, but they read like a natural language.

If you look at rhino mocks for example the writing part has been complicated compared to a normal library. I took me longer to learn mostly due to the fluent interface but it makes code a lot easier to read. Because programs are usually written once and read a lot more than once this is a good tradeoff.

So to qualify my point a bit. A fluent interface that's complex to write but easy to read can still be fluent.

Share Improve this answer

Follow

edited Oct 22, 2008 at 7:51

Ishmaeel

14.4k ● 9 ● 73 ● 84

answered Oct 22, 2008 at 7:42

Mendelt

37.5k ● 6 ● 75 ● 97

Rhino mocks is a good example. Also, like your view on what's good. I think that fluent interfaces, whether exposed to the public or not, is about providing easy to use interfaces for users, other coders, and so on. So the complexity pain is not all bad. – Scott McKenzie Oct 22, 2008 at 9:00

11

You'll hit a brick when using inheritance along with fluent interfaces because using polymorphic methods break your call chains and you definitely don't want to make your interfaces non-fluent by using ugly casting and paranthesis where they are not needed. I've written an article about a pattern that provides you with a workaround using generic builders and generic extension

methods with generic constraints: http://liviutrifoi.wordpress.com/2009/02/16/fluent-interfaces-constraints-at-compile-time/

Share Improve this answer

Follow

edited May 8, 2013 at 22:25

Csaba Toth

10.6k ● 6 ● 83 ● 140

answered Feb 16, 2009 at 17:52

jhonny

1 Nice article. I'll give it a try. – David Robbins Jan 18, 2010 at 0:56

1 Looks like the article link has changed. Seem to now be at: liviutrifoi.wordpress.com/2009/02/16/… – dharmatech Jan 26, 2012 at 21:48

8

Moq hides unreleated methods such as equals, `ToString` and so on to make their fluent interface even easier to use.

Hiding System Object is an article explaining the benefit of doing this.

Share Improve this answer Follow

edited Dec 4, 2012 at 8:42

answered Aug 25, 2009 at 10:44

Finglas

15.7k ● 11 ● 59 ● 90

1 Here is the updated link – Pascal Berger Sep 21, 2016 at 15:07

7

And on your 2nd and 3rd question;

Three fluent patterns i've noticed

The first uses the using statement (C# 2.0) to run code in a certain context for example:

```
using(var transaction = new Transaction())
{
  // ..
  // ..
}
```

This uses the constructor and disposer of Transaction to set up a transaction and then runs the code in this context.

The second does almost the same but with lambda's, this is used a lot in Rhino Mocks for example.

```
(new Transaction()).Run( () => mycode(); );
```

The best known fluent interface is to use return types to chain method calls. Mostly methods return this so you can chain calls on the same object. But you can also return different objects to change the context depending on the method called. If you've got an object that can only run in a transaction (sorry can't think of a different example) you can give it a StartTransaction method that returns an initialized transaction where you can run call run and stoptransaction, in pseudocode:

```
class Runner
{
  Transaction StartTransaction()
  {
    return new Transaction(this);
  }
}

class Transaction
{
  Transaction Run()
  Transaction StopTransaction()
}
```

where the call looks like

```
var runner = new Runner();
runner
  .StartTransaction()
  .Run()
  .StopTransaction();
```

Of course you need to add all kinds of error handling etc.

Share Improve this answer answered Oct 22, 2008 at 7:57

Follow

Mendelt
37.5k • 6 • 75 • 97

7

I too am just jumping on learning how to write a fluent interface for a small app at work. I've asked around and researched a little and found that a good approach for writing a fluent interface is using the "Builder pattern", read more about it here.

In essence, this is how I started mine:

```
public class Coffee
{
    private bool _cream;
    private int _ounces;

    public Coffee Make { get new Coffee(); }

    public Coffee WithCream()
    {
        _cream = true;
        return this;
    }

    public Coffee WithOuncesToServe(int ounces)
    {
        _ounces = ounces;
        return this;
    }
}
```

Here's a cross post to a similar question I have for implementing a closure in a fluent interface.

Share Improve this answer edited May 23, 2018 at 10:18

Follow

Thomas Flinkow
5,065 ● 6 ● 32 ● 70

answered Nov 25, 2009 at 5:54

sam
763 ● 1 ● 7 ● 15

4

One thing is that you have to account for the morphology of English syntax and ensure that you have not introduced undocumented sequential coupling underneath.

```
// Snarky employees get a raise.
employees.WhereSnarky().GiveRaise();
```

vs.

```
// Depending on implementation, everyone may get a rai
employees.GiveRaise().WhereSnarky();
```

Share Improve this answer

Follow

answered Aug 20, 2013 at 14:51

An00bus
111 ● 1 ● 2

2

Sometime ago I had the same doubts you are having now. I've done some research and now I'm writing some posts to help in those topics.

Check it at my blog:

Guidelines to Fluent Interface design in C# part 1

And in the following posts I will cover every one of the points you mentioned.

Best regards André Vianna

Share Improve this answer

Follow

edited Feb 24, 2017 at 22:43

Tony

17.6k ● 16 ● 83 ● 144

answered Aug 5, 2010 at 1:24

Andre Vianna

1,732 ● 2 ● 18 ● 31

Link is dead; pretty good example of why link-only answers are not recommended. I assume this is you?: codeproject.com/Articles/99542/… – Tieson T. Jul 26, 2014 at 20:47