

Fluent API

Goal

The goal of a fluent API is to write more readable code

Write less code

Make it much more obvious what is happening

Eliminating temporary variables

Make code read like free-flowing language

Sample Scenario

Imagine ...

- (1) we have a number, and
 - (2) we wish to add another number to it,
 - (3) and we wish to multiple the result of (2) by another number,
 - (4) and we wish to output the result of (3)
- In English, we see we have four steps and the result flows through it
 - Why can't code be like that?

Traditional API

Definition

- `public class Calculation {`
- `public Int32 Result= {get; set;}`
- `public Calculation(Int32 x) { Result=x;}`
- `public void Add(Int32 y) { Result=Result+y; }`
- `public void Multiply(Int32 y) { Result=Result*y;}`
- `public void Output() { Console.WriteLine(Result.ToString());}`
- `}`

Usage

- `Calculation calc = new Calculation(4);`
- `calc.Add(6);`
- `calc.Multiply(28);`
- `calc.Output();`

Fluent API

Definition

- `public class Calculation {`
- `public Int32 Result= {get; set;}`
- `public Calculation Init(Int32 x) { Result = x; return this;}`
- `public Calculation Add(Int32 y) { Result += y; return this;}`
- `public Calculation Multiply(Int32 y){ Result *= y; return this;}`
- `public void Output() { Console.WriteLine(Result.ToString());}`
- `}`

Usage

- `Calculation calc = new Calculation().Init(4).Add(6).Multiply(28).Output();`

Naming

Term coined by:

- Martin Fowler (<http://martinfowler.com>) and
- Eric Evans - author of excellent Domain-Driven Design book (<http://domaindrivendesign.org/>)

Called Fluent API, Fluent Application Programming Interface, or just Fluent Interface

- In .NET world, often not delivered via C# interfaces, so we use Fluent API

Three ways of delivering it

- Method Chaining
- Nested Functions
- Function Sequences
- <http://martinfowler.com/dslwip/InternalOverview.html#FluentAndCommand-queryApis>

Method Chaining (Return Type)

No void return type

Instead, return type gives us the object for the next method call

- Hence concept of chaining of method calls
- What remains on top of stack for next call

Nested Functions

Passing method calls as parameter of method calls

- Nesting of method calls
- Look at Functional Composition within LINQ To XML
- This is not a unique a C# feature, rather a way of creating an easy to read API

Function Sequence

A list of function / method calls

- Simply call one method after another

Some concept of current instance

Methods work on current instance

- A method can also update current instance

Notes

Often see Fluent API used for configuration setting

Extensive use in LINQ scenarios

Need a little practice in designing Fluent APIs

- But with experience it becomes painless

Alternatives

Constructors

- Could pass initial property values in constructor (but not subsequent ones)

Temporary variables

- Store temporary variable after each API call

Fluent API

- Better

Setting Values

Three options:

- Fluent API
- Data Annotations
- Conventions
 - Use all three (see Code-First Entity Framework for good example)
 - Decide on precedence
 - Usually, convention is lowest, then data annotations, then fluent API is highest