Coding Conventions and Guidelines

Xavier Morera

Helping .NET developers create amazing applications

@xmorera / www.xaviermorera.com / www.bigdatainc.org

It is no secret that programmers like it when things “feel familiar”

In fact, familiarity leads to comfort

Coding conventions are a set of guidelines for a specific programming language

Recommend the programming style, practices, and methods for each aspect of a program written in that particular language

Recommended Coding Conventions and Guidelines

Coding Conventions and Guidelines

Naming

Layout

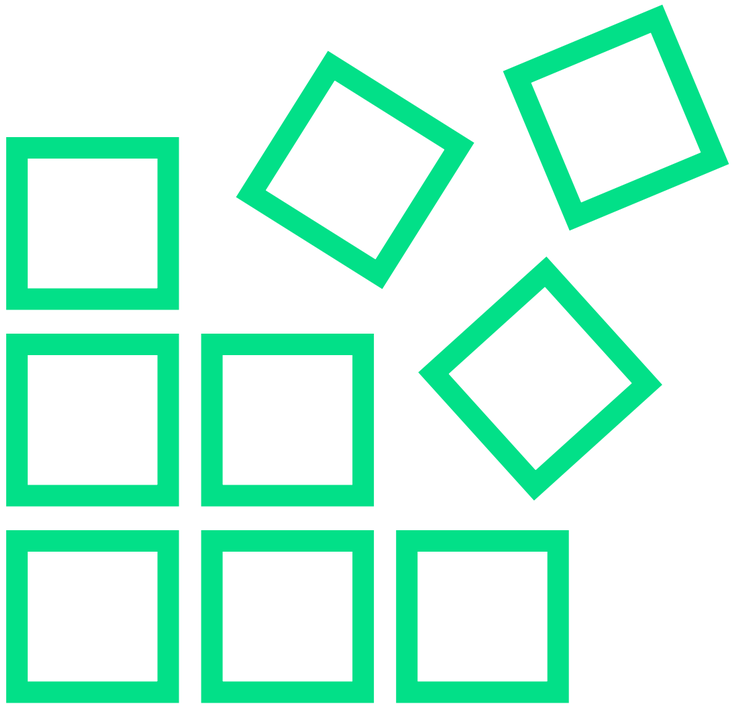
Comments

Cleaner Code

1. String interpolation
2. Implicitly typed local variables
3. Unsigned types
4. Static
5. Exception handling and event handlers

What’s the difference between a coding convention and a guideline?

Coding Convention



Set of rules used for coding in a specific programming language that recommends

- Programming style
- Practices
- Methods for each aspect of a program written in that language

Guidelines

Give some general suggestions regarding the coding style to improve the understandability and readability of the code

Benefits

Create a consistent look of your code

Enable those reading your code to understand the code

Facilitate copying, changing, and maintaining the code

Demonstrate C# Best Practices

C# Coding Conventions

.NET Runtime
&
C# Coding Style

Naming Conventions

Naming Convention

Way in which you write the name of a variable, type, function or any other entity

Naming Conventions

Use a representative name for variables

- `address` is better than just a

Multiple words convey more information

- `trailaddress`

Casing

Lower case

Camel case

Pascal case

PascalCasing

Class, Record, or Struct

```
public class Product
{
    // ...
}

public struct Coords
{
    public double Latitude;
    public double Longitude;
    public override string ToString() => $"Coords({Latitude}, {Longitude})";
}
```

Multiple Words

```
public record TrailAddress(string City, string State, string ZipCode);
```

Interfaces

```
public interface IProduct
{
    string GetDetails();
}
```

Public Members

```
public class ClimbingShoes
{
    // public properties
    public string? Name { get; set; }
    public bool InStock;

   // An event
    public event Action EventCheckInventory;

   // Method
    public void StartCheckInventory()
    {
        // ...
    }
}
```

Positional Records

```
public record TrailAddress(string City, string State, string ZipCode);
```

Demo

PascalCase

camelCase

Private or Internal Fields

```
public class ClimbingShoes
{
    private readonly int _uniqueIdentifier;
    // ...
}
```

Static Private or Internal

```
public class Product
{
    private static int s_reviewsQueue;

    [ThreadStatic]
    private static TimeSpan t_timeSpan;
    // ...
}
```

Method Parameters

```
public class Trail
{
    public int TrailNumber { get; set; }

    public void SaveTrail(int trailNumber, bool isRegistered)
    {
        // ...
    }
}
```

Demo

camelCase

Additional Naming Conventions

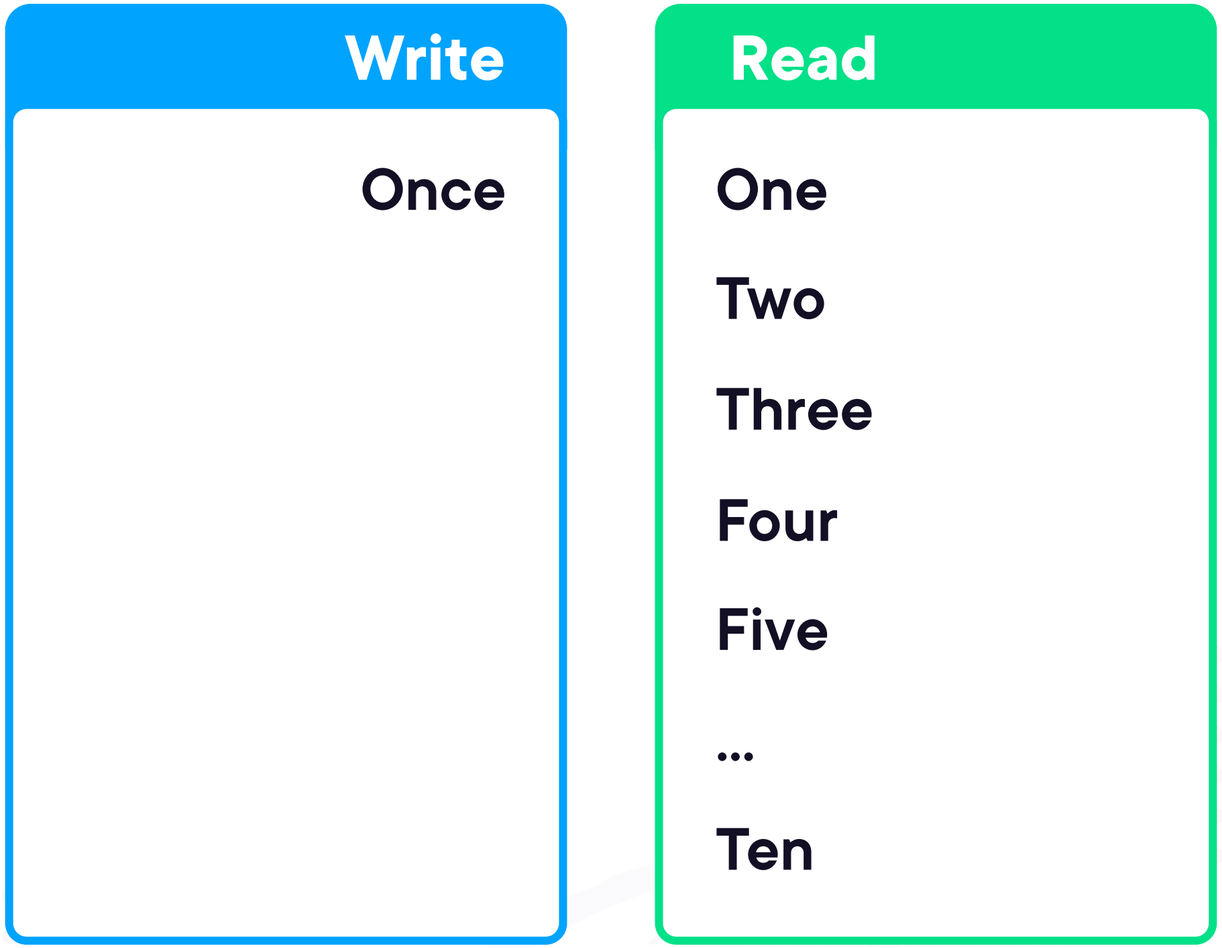
```
var reallyLongVariableName = new CarvedRock.Common.
                                 GroupProductsBySport();
```

Code Layout Conventions

It’s Harder to Read Code than to Write it

By a prolific programmer

Estimated Ratio



Code Layout Conventions

Layout Conventions

Use the default code editor settings

- Smart indenting
- Four-character indents
- Tabs saved as spaces

Write only one…

- Statement per line
- Declaration per line

Layout Conventions

If continuation lines are not indented automatically

- Indent them one tab stop (four spaces)

Add at least one blank line between property definitions and methods

Use parentheses to make clauses in an expression apparent

Demo

Code Layout Conventions

Commenting Conventions

```
// Comments

/* Comments */
```

Commenting Conventions

Place the comment on a separate line

- Not at the end of a line of code

Begin comment text with an uppercase letter

End comment text with a period

Insert one space between the comment delimiter and comment text

XML Comments

Use XML documentation comments

- Purpose
- Parameters
- Output

Structured comments

- Produce API documentation

Can be used to generate human-readable docs

- PDF or HTML

Demo

Commenting Conventions

Language Guidelines

Language Guidelines

Do or Don’t

Be careful

String Interpolation

Use string interpolation to concatenate short strings

```
string displayName = $"{firstName} {lastName}";
```

String Interpolation

Method of concatenating, formatting, and manipulating strings
Use the $ special character to identify an interpolated string
    Literal that can contain interpolated expressions, for example {firstName}
    Replaced by string representations of the expression results

StringBuilder

Use a StringBuilder object to append strings in loops

StringBuilder

Strings are immutable

Concatenating strings creates new objects

Performance decreases as the number of new objects are created

Only use when necessary

Version
.NET 6
Search
EncodingExtensions
EncodingInfo
EncodingProvider
NormalizationForm
Rune
SpanLineEnumerator
SpanRuneEnumerator
StringBuilder
StringBuilder
Constructors
Properties
Methods
Explicit Interface Implementations
StringBuilder.AppendInterpolatedStringHandler
StringBuilder.ChunkEnumerator
StringRuneEnumerator
UnicodeEncoding
UTF32Encoding
UTF7Encoding
UTF8Encoding

StringBuilder Class

Reference

Definition

Namespace: System.Text
Assembly: System.Runtime.dll

Represents a mutable string of characters. This class cannot be inherited.

C# Copy

```
public sealed class StringBuilder : System.Runtime.Serialization.ISerializable
```

Inheritance Object → StringBuilder

Implements ISerializable

Examples

The following example shows how to call many of the methods defined by the StringBuilder class.

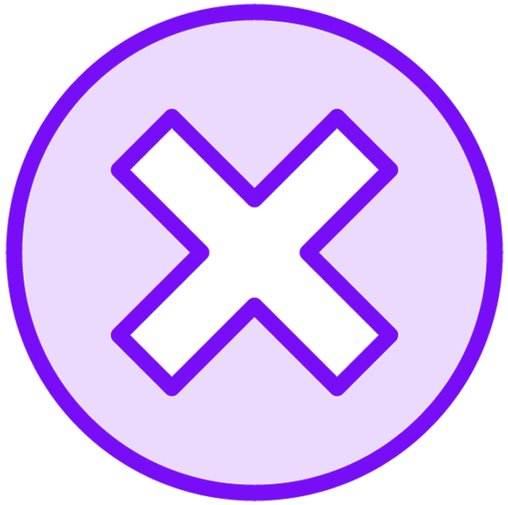
C# Copy Run

```
using System;
```

In this article
Definition
Examples
Remarks
Notes to Callers
Show more

Implicitly Typed Local Variables

Declare implicitly typed variables via the use of the var keyword



Unsigned

Don’t unsigned types

Use int instead

Arrays

Use the concise syntax when initializing in a single line

Use var for explicit instantiation

If you specify size

- Initialize elements one at a time

Func and Action

Use Func and Action instead of delegate

Delegates

Type that represents references to methods

- Includes parameter list and return type

Used to pass methods as arguments to other methods

Event handlers are methods invoked through delegates

Defining Delegates

Define a custom delegate

- Should match the signature of the method

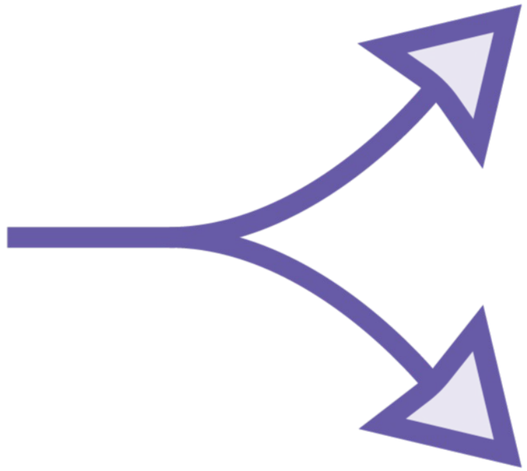
Create an instance of the delegate

- Point it to the method

Invoke the method

Better Way

Delegate



Func<>

Action<>

Delegate vs. Action

```
static void DoSomething(int i)
{
    // Perform some important function
}
```

```
public delegate void importantDelegate(int
val);

static void Main(string[] args)
{
    importantDelegate del = DoSomething;
    del(10);
}
```

```
static void Main(string[] args)
{
    Action<int> importantAction = DoSomething;
    importantAction(10);
}
```

Advantages of Action and Func

`Func<>`

`Action<>`

Easier to define

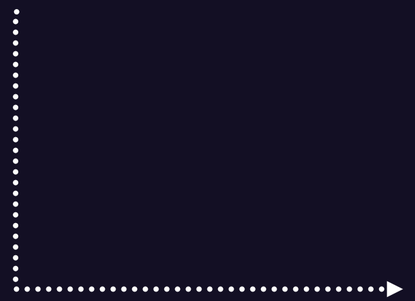
Shorter code

Compatible type throughout your code

Event Handler

```
public ExampleForm()
{
    this.Click += new EventHandler(Form1_Click);
}

void ExampleForm_Click(object? sender, EventArgs e)
{
    MessageBox.Show(((MouseEventArgs)e).Name);
}
```



```
public ExampleForm()
{
    this.Click += (s, e) =>
        {
            MessageBox.Show(((MouseEventArgs)e).Name);
        };
}
```

new

Use the new syntax that does not require braces

New new Syntax

```
var backpack = new Backpack();
```

```
Backpack backpack = new();
```

&& and ||

Use the short-circuit operators

When the first expression is false, the second expression is not evaluated

Avoids exceptions and increases performance

&& and ||

using

Simplify your code with the `using` **statement**

Can use in try-finally **where only** Dispose **is called in the** finally **block**

Use the syntax without braces

Object Initializers

Simplify object creation and make your code more readable by using object initializers

Static

Be careful with static

Call static members using the class name

LINQ

LINQ

Language INtegrated Query

Allows querying capabilities directly in your C# code

LINQ Do's

Use meaningful names for query variables

Use aliases

- Ensure property names of anonymous types are correctly capitalized
  - Pascal casing

Rename properties

- When the property names in the result would be ambiguous

LINQ Do's

Use implicit typing in the declaration of query variables and range variables

Align query clauses under the `from` clause

Use `where` clauses before other query clauses

- Ensures later query clauses operate on the reduced, filtered set of data

Demo

LINQ

Takeaway

Naming conventions are a set of rules on how to name entities

- Make code more readable and easier to understand

Use representative names for entities

- Multiple words can convey more information

Different naming conventions

- Lowercase, pascal case, camel case

Takeaway

Pascal case

- Class, record, struct, interfaces, public members, positional records

Camel case

- Private and internal fields
- Method parameters

Don't rename auto-generated names

Takeaway

Code layout conventions

- More time reading than writing code

Use the IDE functionality

- Default code editor settings
- Format document

Only one statement and declaration per line

Add one blank line between property definitions and methods

Use parenthesis to make clauses in expressions apparent

Takeaway

Commenting conventions

- Use comments as they help with readability
- Place comments in separate lines
  - Can also use multiline comments
- Begin with an uppercase letter and end with a period
- Insert space after the comment
- Use XML comments

Takeaway

Language guidelines

- String interpolation
- StringBuilder
- Implicitly typed variables
- Arrays
- Func and Action
- new
- Short-circuit operators
- using
- Object initializers
- Static

Takeaway

LINQ

- Meaningful names
- Use aliases
- Rename properties that might be ambiguous
- Use implicit typing
- Align query clauses under the `from` clause
- Use `where` clauses before other query clauses