


My Standard C# Style Guide

👤 Created by	 Royce Gill
🕒 Created time	@January 8, 2025 10:27 AM
🏷️ Tags	Guides
# Version	1.01

Document Versioning

Version	Effective Date	Maintainer	Comments
1.00	12/01/2024	Royce Gill	The Initial version of my C# Style Guide
1.01	13/01/2024	Royce Gill	Moved Document Versioning to top of document, export formatting.

Table of Contents

[Document Versioning](#)

[Table of Contents](#)

[Formatting](#)

[Naming Conventions](#)

[Formatting](#)

[Good Naming](#)

[Using Lambda](#)

[LINQ](#)

[Expression-bodied members](#)

[Null-coalescing and Null-conditional Operators](#)

[Pattern Matching](#)

[Documentation](#)

[Inline Code Comments](#)

[Do's](#)

[Don'ts](#)

[XML Comments](#)

[Testing](#)

Formatting

Consistent Indentation

- 4 space Indentation.
- Add a level of indentation
 - Within a set of curly braces
 - When Wrapping LINQ Expressions

Line Length

- 100 characters
- Keep Readability font of mind when exceeding this char limit.

White Spaces before

- Closing Parentheses
- Closing Braces

White Spaces after

- Commas
- Opening Parentheses
- Opening Braces

White Spaces before & after

- Operators

```
public Goat( string name, int age ) => ( Name, Age ) = ( name
```

Brace Styles

- Allman Style
- Start a new line for opening of curly braces and another new line for a closing bracer.

```
public void DoAThing()  
{  
    // Do the thing.  
}
```

Blank line

- Use a blank line to show a break in code sections.

Naming Conventions

The purpose of naming something is to communicate with clarity and conciseness what something is. Avoid names that people could be misunderstood and avoid names that have more information than is necessary.

Formatting

PascalCase:

- Methods/Functions
- Classes
- Interfaces
- Enums
- Properties

camelCase:

- Variables
- Fields

UPPER_SNAKE_CASE:

- Constants

Special Cases:

- private fields begin with "_"

- When using an acronym (HTML) for a variable or Object each letter should be the same case if the case would normally be camel the the who acronym is lower case if the Case would normally be Pascal Case the the whole acronym should be in Upper Case.

Variables:

- Temporary variables: should be "_"
- loop Indices: a single letter abbreviations (i, j, k, l)

Files:

- Package names: all lowercase, avoid language keywords

Good Naming

It's Important is to use descriptive names for everything you create, avoid abbreviating unnecessarily and being overly Verbose can hinder Readability

Files:

- Folder names should be descriptive and should be to group related functionality, features and modules together.

Bools:

- Any bool value or method with a bool return should have a name that is Positive. IsValid > IsValid.
- Boolean Variables: Prefixed with is or has.

Interfaces:

- In general Prefixing an I at the beginning Shows that the Object is an interface.
- Use Descriptive nouns, noun phrases or Adjective when naming your interface.

Classes

- Use Nouns for names.
- If your class is the only class that inherits from an interface then that class should be named Exactly like the interface minus the I prefix.

Methods

- Use a Verb or Verb Phrase.

- Be consistent with your naming.
 - Open/Load/Read Pick one and use it in all similar situations or layers
- If you are having difficulty Naming a Method, Reevaluate the Method and ensure it is only doing one thing.
- If your Method contains the word "and" or "or" then you should split the method into two different methods.

Collections

- Use the proper noun and Collection as a suffix in variable and Properties of collections.
- It is acceptable to use a plural noun if the singular and the plural are different.

Events

- Use past and present tense to differentiate when the event is occurring in the call stack.
 - OnXChanged or OnXChanging

Attributes

- Prefix with the Word Attribute.

Generic Type Parameters

- Nouns should be used.
- Prefix with "T".

Enums

- Use Singular for Normal Enums.
- For Bit use Enums use a Plural nouns.
- Values should use PascalCasing.

Using Lambda

LINQ

Utilize Language Integrated Query (LINQ) for querying and manipulating collections in a declarative manner.

```
var adultGoats = goats.Where(goat => goat.Age >= 2).ToList();
```

Wrap LINQ Queries to break down the Call chain in a readable way.

```
var adultGoatsNames = goats.Where(goat => goat.Age >= 2)
                              .Select(goat => g
                              .ToList();
```

Expression-bodied members

For simple Methods, Properties, Class Constructors use Expression-bodied members.

```
public class Goat {
    public string Name { get; }
    public int Age { get; }

    public Goat(string name, int age) => (Name, Age) = (name,
    public void MakeSound() => Console.WriteLine($"{Name} say
}
```

Null-coalescing and Null-conditional Operators

Simplify null checks with the null-coalescing operator and the null Conditional Operators.

```
//Do this
public string GetGoatName(Goat goat) => goat?.Name ?? "Unknow

//Not this
public string GetGoatName(Goat goat)
{
```

```

        if (goat != null)
        {
            return goat.Name;
        }
        else
        {
            return "Unknown";
        }
    }
}

```

Pattern Matching

Enhance Readability by using Pattern Matching for type Checks and Conditional Logic.

Perform a type check and assign object a variable name in the same operation.

```

public void FeedAnimal(object animal) {
    if (animal is Goat goat) {
        Console.WriteLine($"Feeding {goat.Name}");
    }
}

```

Using the new switch statement syntax, assign variables from Specific Properties from within a conditional statement.

```

object shape = new Circle(5);

string description = shape switch
{
    Circle { Radius: var r } => $"Circle with radius {r}",
    Rectangle { Width: var w, Height: var h } => $"Rectangle {w} x {h}",
    _ => "Unknown shape"
};

Console.WriteLine(description);

```

Documentation

Inline Code Comments

Do's

Make comments on functionality and outcomes.

```
public class Goal
{
    public DateTime DueDate { get; set; }

    public Goal(DateTime date)
    {
        DueDate = date
    }

    // DateTime is non nullable value type so default bri
    public void RemoveDueDate()
    {
        DueDate = default;
    }
}
```

Make Comments that explain what situations you may use this item.

```
public class Goal
{
    public DateTime DueDate { get; set; }

    // For high level organistion of Task Managment.
    public Goal(DateTime date)
    {
        DueDate = date
    }

    public void RemoveDueDate()
    {
```



```

        DueDate = default;
    }
}

```

Dispel Complexity.

```

class Program
{
    static void Main()
    {
        string input = "racecar";
        bool isPalindrome = IsPalindrome(input);
        Console.WriteLine($"{input} is palindrome: {isPalindr
    }

    static bool IsPalindrome(string s)
    {
        return s == ReverseString(s, 0);
    }

    static string ReverseString(string s, int index)
    {
        // if the input string is the last char in the st
        if (index == s.Length - 1) return s[index].ToString()
        // loops though the string from start to finish using
        return s[s.Length - 1 - index] + ReverseString(s, ind
    }
}

```

Don'ts

Don't make redundant comments.

```

// This is a Goal Class
public class Goal
{

}

```

Don't make comments that require Frequent updating.

```
// This is a Goal Class
public class Objective
{

}
```

Don't comment for comments sake.

```
// This is a Goal Class
public class Goal
{
    // This a Constructor
    public Goal()
    {

    }
}
```

Don't add Comments about changes to the code base especially when using Source Control, Add the Comment to Commit the change occurred.

```
/* This is an Objective Class
 * This Class's Name was change From Goal to Objective
 */
public class Objective
{
    public Goal()
    {

    }
}
```

Don't Comment out Blocks of code and leave it in the Code base.

```

public class Goal
{
    public DateTime DueDate { get; set; }
    public Goal(DateTime dueDate)
    {
        DueDate = dueDate
    }

    //public void RemoveDueDate()
    //{
    //    DueDate = default;
    //}
}

```

Don't add a Comments where correct Naming can explain.

```

public class Goal
{
    public DateTime DueDate { get; set; }

    // date param is the date the Goal is Due.
    public Goal(DateTime date)
    {
        DueDate = date
    }

    public void RemoveDueDate()
    {
        DueDate = default;
    }
}

```

XML Comments

XML Comments add additional functionality to your IDE and other applications such as Swagger that map API Endpoints using XML Comment Content.

Use XML Comments on the Public Interfaces of a class and have the class inherit the comments from the Interface for Inteli sense

```
public interface IImageUploadRepository
{
    /// <summary>
    /// Creates a record of the uploaded image in the Database
    /// </summary>
    /// <param name="dto">Uploaded File Details</param>
    /// <returns>Database Entity Task for Async Operations</returns>
    Task<ImageUpload> AddImageUploadResult(UploadResultDto dto)
}
```

Testing

Naming Tests

- MethodName_Scenario_Expectations

Assertions

- Use Fluent Assertions library for added readability.
- Don't use Double negatives.

Refactoring

Things to Consider during the Refactoring Stages

Methods

1. Documentation
2. Strong Naming
3. Does the Method Reach across Abstraction Levels
4. Code Composition
 - a. Does the Method do one thing?

- b. Can the Method be split up into different Methods
 - i. Does the new Method belong to a new Class?
 - ii. Do you have an existing Class that has similar Functionality?
 - iii. If you were to test this new method would it be easier if the method was in a different class?
 - c. New Classes Created for functionality need to be tested.
5. Is this code as optimized as it could be?

Reference Material

Website: [Foundational Code Standards - GoatStyles](#) - Code examples.

Website: [C# - GoatStyles](#) - Code Examples

Website: [Identifier names - rules and conventions - C# | Microsoft Learn](#)