# C# Coding Guidelines and Best Practices

## Capitalization Summary

| Identifier | Rules for Naming | Notes/Examples |
|---|---|---|
| Class | Pascal Case | |
| Attribute Class | Pascal Case | Has a suffix of Attribute |
| Exception Class | Pascal Case | Has a suffix of Exception |
| Constant | Pascal Case | |
| Enum type | Pascal Case | |
| Enum values | Pascal Case | |
| Event | Pascal Case | |
| Interface | Pascal Case | Has a prefix of I |
| Local variable | Camel Case | |
| Method | Pascal Case | |
| Namespace | Pascal Case | |
| Property | Pascal Case | |
| Public Instance Field | Pascal Case | Rarely used (use a property instead) |
| Protected Instance Field | Camel Case | Rarely used (use a property instead) |
| Parameter | Camel Case | |

## Name Usage & Syntax

| | |
|---|---|
| **Class or Struct** | • Pascal Case.<br>• Use a noun or noun phrase for class name.<br>• Add an appropriate class-suffix when sub-classing another type when possible.<br><br>**Examples:**<br>private class MyClass<br>{…}<br>internal class SpecializedAttribute : Attribute<br>{…}<br>public class CustomerCollection : CollectionBase<br>{…}<br>public class CustomEventArgs : EventArgs<br>{…}<br>private struct ApplicationSettings<br>{…} |
| **Interface** | • Pascal Case.<br>• Always prefix interface name with capital "I".<br><br>**Example:**<br>interface ICustomer<br>{…} |

| | |
|---|---|
| **Generic Class**<br>&<br>**Generic Parameter Type** | • Always use a single capital letter, such as T or K.<br><br>**Example:**<br>public class FifoStack<T><br>{<br>public void Push(<T> obj)<br>{…}<br>public <T> Pop()<br>{…}<br>} |
| **Method** | • Pascal Case.<br>• Try to use a Verb or Verb-Object pair.<br><br>**Example:**<br>public void Execute() {…}<br>private string GetAssemblyVersion(Assembly target)<br>{…} |
| **Property** | • Pascal Case.<br>• Never prefix property names with “Get” or “Set”.<br>**Example:**<br>public string Name<br>{<br>get{…}<br>set{…}<br>} |
| **Field**<br>**(Public, Protected, or Internal** | • Pascal Case.<br>• Avoid using non-private Fields! Use Properties instead.<br><br>**Example:**<br>public string Name;<br>protected IList InnerList; |
| **Field (Private)** | • Camel Case and prefix with a single underscore (_) character.<br><br>**Example:**<br>private string _name; |
| **Variable** | • Camel Case.<br>• Avoid using single characters like “x” or “y” except in FOR loops.<br>• Avoid enumerating variable names like text1, text2, text3 etc. |

| | |
|---|---|
| **Parameter** | • Camel Case.<br><br>**Example:**<br>public void Execute(string commandText, int iterations)<br>{…} |
| **Enum** | • Pascal Case (both the Type and the Options).<br>• Add the FlagsAttribute to bit-mask multiple options.<br><br>**Example:**<br>public enum CustomerTypes<br>{<br>Consumer,<br>Commercial<br>} |

## General Guidelines

- Always use Camel Case or Pascal Case names.
- Avoid numeric characters.
- Avoid using abbreviations unless the full name is excessive.
- Do not include the parent class name within a property name.
- Try to prefix Boolean variables and properties with “Can”, “Is” or “Has”.

## Exception

- Do not use try/catch blocks for flow-control.
- Only catch exceptions that you can handle.
- Never declare an empty catch block.
- Avoid nesting a try/catch within a catch block.
- Always catch the most derived exception via exception filters.
- Order exception filters from most to least derived exception type.
- Avoid re-throwing an exception. Allow it to bubble-up instead.( https://www.pluralsight.com/guides/throw-re-throw-expectations)
- If re-throwing an exception, preserve the original call stack by omitting the exception argument from the throw statement.

**Example**:-

```
// Bad!
catch(Exception ex)
{
        Log(ex);
        throw ex;
}

// Good!
catch(Exception)
{
        Log(ex);
        throw;
```

```
}
```

- Only use the finally block to release resources from a try statement.

## Code Commenting

- All comments should be written in the propper language, be grammatically correct, and contain appropriate punctuation.
- Use // or /// but never "flowerbox" comment blocks(/* … */).
  Example:

```
// ***************************************
        // Comment block
// ***************************************
```

- Use inline-comments to explain assumptions, known issues, and algorithm insights.
- Do not use inline-comments to explain obvious code. Well written code is self-documenting.
- Always apply C# comment-blocks (///) to public, protected, and internal declarations.
- Always include <summary> comments. Include <param>, <return>, and <exception> comment sections where applicable.

## Flow Control

- Use below structure for class creation in this order with each section wrapped in a #region:

```
Class
Private members
Public properties
Constructors
Public methods
Private methods
```

Example:

```
public class myClass
{
        #region Private Members

        #endregion
        #region Public Properties

        #endregion
        #region Constructors

        #endregion
        #region Public Methods

        #endregion
        #region Private Methods

        #endregion
}
```