

C# Coding Guidelines and Best Practices

1. 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 | Enm prefix , ex enmEmployeeType |
| 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 | |

2. Name Usage & Syntax

| | |
|-----------------|--|
| Class or Struct | <ul style="list-style-type: none">• Pascal Case.• Use a noun or noun phrase for class name.• Add an appropriate class-suffix when sub-classing another type when possible. <p>Examples:</p> <pre>private class MyClass {...} internal class SpecializedAttribute : Attribute {...} public class CustomerCollection : CollectionBase {...} public class CustomEventArgs : EventArgs {...} private struct ApplicationSettings {...}</pre> |
|-----------------|--|

| | |
|---|--|
| Interface | <ul style="list-style-type: none"> • Pascal Case. • Always prefix interface name with capital “I”. <p>Example: interface ICustomer {...}</p> |
| Generic Class & Generic Parameter Type | <ul style="list-style-type: none"> • Always use a single capital letter, such as T or K. <p>Example: public class FifoStack<T> { public void Push(<T> obj) {...} public <T> Pop() {...} }</p> |
| Method | <ul style="list-style-type: none"> • Pascal Case. • Try to use a Verb or Verb-Object pair. <p>Example: public void Execute() {...} private string GetAssemblyVersion(Assembly target) {...}</p> |
| Property | <ul style="list-style-type: none"> • Pascal Case. • Never prefix property names with “Get” or “Set”. <p>Example: public string Name { get{...} set{...} }</p> |
| Field (Public, Protected, or Internal | <ul style="list-style-type: none"> • Pascal Case. • Avoid using non-private Fields! Use Properties instead. <p>Example: public string Name;</p> |

| | |
|------------------------|---|
| | <code>protected IList InnerList;</code> |
| Field (Private) | <ul style="list-style-type: none"> • Camel Case and prefix with a single underscore (_) character. <p>Example: <code>private string _name;</code></p> |
| Variable | <ul style="list-style-type: none"> • Camel Case. • Avoid using single characters like “x” or “y” except in FOR loops. • Avoid enumerating variable names like text1, text2, text3 etc. |
| Parameter | <ul style="list-style-type: none"> • Camel Case. <p>Example: <code>public void Execute(string commandText, int iterations)</code> <code>{...}</code></p> |
| Enum | <ul style="list-style-type: none"> • Pascal Case (both the Type and the Options). • Add the FlagsAttribute to bit-mask multiple options. <p>Example: <code>public enum CustomerTypes</code> <code>{</code> <code>Consumer,</code> <code>Commercial</code> <code>}</code></p> |

3. 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”.

4. 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.
- 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.

5. Code Commenting

- All comments should be written in the same language, be grammatically correct, and contain appropriate punctuation.
- Use `//` or `///` but never `/* ... */`
- Do not “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.
- **Method Comment**
 - Purpose of Method
 - Calling From GUI (When call from GUI)
 - Parameter Description

6. 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
}
```

7. Proper use of External Resource

- Take care of proper release external resource after use.
- Database connection always use with 'using'
- Reading large file using stream and after finish operation then dispose this stream

8. Nested Database Call

- Don't use database connection inside loop to retrieved data from database.
- Use only when necessary condition
- Don't use transaction table for nested call