

Coding Conventions

Clean Code Document



March 8, 2020

Mothol

Table of Contents

[Naming conventions 2](#_Toc34573254)

[Names of Methods 2](#_Toc34573255)

[Names of Properties 3](#_Toc34573256)

[Names of Events 4](#_Toc34573257)

[Names of Fields 4](#_Toc34573258)

[Naming Parameters 5](#_Toc34573259)

[Naming Operator Overload Parameters 5](#_Toc34573260)

[Layout Conventions 5](#_Toc34573261)

[Commenting Conventions 6](#_Toc34573262)

[Structure Example 7](#_Toc34573263)

**Coding Conventions**

Coding conventions serve the following purposes:

* They create a consistent look to the code, so that readers can focus on content, not layout.
* They enable readers to understand the code more quickly by making assumptions based on previous experience.
* They facilitate copying, changing, and maintaining the code.
* They demonstrate best practices.

The guidelines in this article are used by Microsoft to develop samples and documentation.

# Naming conventions

In addition to the rules, there are a number of identifier [naming conventions](https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines) used. By convention, programs use PascalCase for type names, namespaces, and all public members. In addition, the following conventions are common:

* Interface names start with a capital I.
* Attribute types end with the word Attribute.
* Enum types use a singular noun for non-flags, and a plural noun for flags.
* Identifiers should not contain two consecutive \_ characters. Those names are reserved for compiler generated identifiers.

## Names of Methods

Because methods are the means of taking action, the design guidelines require that method names be verbs or verb phrases. Following this guideline also serves to distinguish method names from property and type names, which are noun or adjective phrases.

**✔ DO** give methods names that are verbs or verb phrases.

public class String {

public int CompareTo(...);

public string[] Split(...);

public string Trim();

}

## Names of Properties

Unlike other members, properties should be given noun phrase or adjective names. That is because a property refers to data, and the name of the property reflects that. PascalCasing is always used for property names.

**✔ DO** name properties using a noun, noun phrase, or adjective.

❌ DO NOT have properties that match the name of "Get" methods as in the following example:

public string TextWriter { get {...} set {...} } public string GetTextWriter(int value) { ... }

This pattern typically indicates that the property should really be a method.

**✔ DO** name collection properties with a plural phrase describing the items in the collection instead of using a singular phrase followed by "List" or "Collection".

**✔ DO** name Boolean properties with an affirmative phrase (CanSeek instead of CantSeek). Optionally, you can also prefix Boolean properties with "Is", "Can", or "Has", but only where it adds value.

✔ CONSIDER giving a property the same name as its type.

For example, the following property correctly gets and sets an enum value named Color, so the property is named Color:

public enum Color {...}

public class Control {

public Color Color { get {...} set {...} }

}

## Names of Events

Events always refer to some action, either one that is happening or one that has occurred. Therefore, as with methods, events are named with verbs, and verb tense is used to indicate the time when the event is raised.

**✔ DO** name events with a verb or a verb phrase.

Examples include Clicked, Painting, DroppedDown, and so on.

**✔ DO** give events names with a concept of before and after, using the present and past tenses.

For example, a close event that is raised before a window is closed would be called Closing, and one that is raised after the window is closed would be called Closed.

❌ DO NOT use "Before" or "After" prefixes or postfixes to indicate pre- and post-events. Use present and past tenses as just described.

**✔ DO** name event handlers (delegates used as types of events) with the "EventHandler" suffix, as shown in the following example:

public delegate void ClickedEventHandler(object sender, ClickedEventArgs e);

**✔ DO** use two parameters named sender and e in event handlers.

The sender parameter represents the object that raised the event. The sender parameter is typically of type object, even if it is possible to employ a more specific type.

**✔ DO** name event argument classes with the "EventArgs" suffix.

## Names of Fields

The field-naming guidelines apply to static public and protected fields. Internal and private fields are not covered by guidelines, and public or protected instance fields are not allowed by the [member design guidelines](https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/member).

**✔ DO** use PascalCasing in field names.

**✔ DO** name fields using a noun, noun phrase, or adjective.

❌ DO NOT use a prefix for field names.

For example, do not use "g\_" or "s\_" to indicate static fields.

## Naming Parameters

Beyond the obvious reason of readability, it is important to follow the guidelines for parameter names because parameters are displayed in documentation and in the designer when visual design tools provide Intellisense and class browsing functionality.

**✔ DO** use camelCasing in parameter names.

**✔ DO** use descriptive parameter names.

✔ CONSIDER using names based on a parameter’s meaning rather than the parameter’s type.

## Naming Operator Overload Parameters

**✔ DO** use left and right for binary operator overload parameter names if there is no meaning to the parameters.

**✔ DO** use value for unary operator overload parameter names if there is no meaning to the parameters.

✔ CONSIDER meaningful names for operator overload parameters if doing so adds significant value.

❌ DO NOT use abbreviations or numeric indices for operator overload parameter names.

# Layout Conventions

Good layout uses formatting to emphasize the structure of your code and to make the code easier to read. Microsoft examples and samples conform to the following conventions:

* Use the default Code Editor Settings (smart indenting, four-character indents, tabs saved as spaces). For more information.
* Write only one statement per line.
* Write only one declaration per line.
* If continuation lines are not indented automatically, indent them one tab stop (four spaces).
* Add at least one blank line between method definitions and property definitions.
* Use parentheses to make clauses in an expression apparent, as shown in the following code.

if ((val1 > val2) && (val1 > val3))

{

// Take appropriate action.

}

# 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 the comment text, as shown in the following example.

// The following declaration creates a query. It does not run

// the query.

* Do not create formatted blocks of asterisks around comments.

# Structure Example

// The Following Code is a basic Example of Code Convention

using System;

namespace YourNamespace

{

class YourClass

{

}

struct YourStruct

{

}

interface IYourInterface

{

}

delegate int YourDelegate();

enum YourEnum

{

}

namespace YourNestedNamespace

{

struct YourStruct

{

}

}

class YourMainClass

{

static void Main(string[] args)

{

//Your program starts here...

}

}

}