robbiblubber.org

C# Coding Guidelines

Version 1.0

Introduction

robbiblubber.org coding guidelines typically follow the coding style recommendations and conventions accepted by the community for a specific programming language while trying to maintain common ideas and traditions, especially when denoting scope and visibility of code elements.

Generally, class, member, parameter names should be identical in all language ports of a given piece of code, except for naming style (meaning a method may be called “CopyElement”, “copyElement”, or “copy_element” in different languages, but should never be named “copy”).

All language constructs should always be commented in a way that supports automatic documentation generation.

In this document, rules for protected members also apply to private protected and internal protected. Static and non-static elements follow the same rules.

Each type should be defined in a file with the type name. An assembly should not contain more than one root namespace.

1 Visibility

Visibility rules apply to all types and members.

Public types or members are unmarked.

Private, protected, and internal types or members start with a leading underscore.

Types or members that should only be used under specific circumstances start with two leading underscores.

2 Namespaces

Namespaces are Pascal case, starting with "Robbiblubber.".

3 Types

This chapter defines naming conventions for types.

3.1 Classes

Classes and Structs are always Pascal case. The class name should be a noun or a noun with descriptive attributes.

Classes derived from *Exception* end with "Exception". Classes derived from *Attribute* end with "Attribute". Also, classes derived from *EventArgs* end with "EventArgs"

3.2 Interfaces

Interfaces are always Pascal case, starting with “I”. The interface name should be an adjective if applicable.

3.3 Enumerations

Enumerations are always Pascal case. The name should be singular and should never end with “Enum”.

3.4 Delegates

Delegates are always Pascal case. The name should reassemble a method name, therefore most likely being a verb, and should never end with “Delegate”.

Event handler delegates end with "EventHandler".

4 Members

This chapter defines naming conventions for type members.

4.1 Fields

Fields are Pascal case. The field name is typically a noun or adjective.

4.2 Properties

Properties are Pascal case. The property name is typically a noun or adjective.

4.3 Methods

Methods are Pascal case. Method names should be verbs.

4.4 Events

Events are Pascal case. Event names frequently are participles.

4.5 Constants and Enumeration Values

Constants and enumeration values are upper case. Especially static read-only properties may be considered constants if this makes sense in the given context.

5 Variables and Parameters

This chapter defines naming conventions for type variables and parameters.

5.1 Variables

Local variables are camel case.

5.2 Parameters

Parameters are camel case.

6 Comments

This chapter describes the usage of comments.

6.1 Documentation Comments

Each type or member should have an XML documentation comment ("///").

6.2 Member Grouping

Fields, constructors, properties, methods, events, overrides should be grouped by a box of 80 slashes regarding visibility. Nested types and interface implementations should also be introduced by such a box, named [class|enum|…|override|interface] and type name.

6.3 Code Comments

Inline comments typically start at position 81 and are single line comments ("//"). Longer, descriptive comments may be multi-line ("/*", "*/") if useful.

7 Example

```
namespace Robbiblubber.Naming.Example
{
    /// <summary>This is an example class for coding guidelines.</summary>
    public class Example: __SampleBase, IUsable
    {
        ////////////////////////////////////////////////////////////////////////////////
        // private static members                                                     //
        ////////////////////////////////////////////////////////////////////////////////

        /// <summary>Private static field.</summary>
        private static int _SomeNumber = 42;



        ////////////////////////////////////////////////////////////////////////////////
        // constructors                                                               //
        ////////////////////////////////////////////////////////////////////////////////

        /// <summary>Creates a new instance of this class.</summary>
        /// <param name="arg">Argument.</param>
        public Example(int arg)
        {}



        ////////////////////////////////////////////////////////////////////////////////
        // public properties                                                          //
        ////////////////////////////////////////////////////////////////////////////////

        /// <summary>Gets or sets the name.</summary>
        public string Name { get; set; }



        ////////////////////////////////////////////////////////////////////////////////
        // public methods                                                             //
        ////////////////////////////////////////////////////////////////////////////////

        /// <summary>Generates a number.</summary>
        /// <returns>Returns a number.</returns>
        public int GenerateNumber()
        {
            return _SomeNumber;                                 // a comment
        }



        ////////////////////////////////////////////////////////////////////////////////
        // [override] _SomeBase                                                       //
        ////////////////////////////////////////////////////////////////////////////////

        /// <summary>Initializes the instance.</summary>
        protected override void _Init()
        {}



        ////////////////////////////////////////////////////////////////////////////////
        // [interface] IUsable                                                        //
        ////////////////////////////////////////////////////////////////////////////////

        /// <summary>Uses the item.</summary>
        public void Use()
        {}
    }
}
```

Sample: C# code

Version History

Version 1.0	2017-04-22	released