Project Conventions

Coding Standards and Practices

C#

Contents

1.Naming Conventions

1.1. 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”.

1.2. Name Usage & Syntax

Class or Struct	• Pascal Case. • Use a noun or noun phrase for class name. • Add an appropriate class-suffix when sub-classing another type when possible. **Examples:** private class MyClass {…} internal class SpecializedAttribute : Attribute {…} public class CustomerCollection : CollectionBase {…} public class CustomEventArgs : EventArgs {…} private struct ApplicationSettings {…}
Interface	• Pascal Case. • Always prefix interface name with capital “I”. **Example**: interface ICustomer {…}
Method	• Pascal Case. • Try to use a Verb or Verb-Object pair. **Example:** public void Execute() {…} private string GetAssemblyVersion(Assembly target) {…}
Property	• Pascal Case. • Never prefix property names with “Get” or “Set”.

	Example:
	public string Name
	{
	get{…}
	set{…}
	}

Field **(Public, Protected,** **or Internal**	• Pascal Case. • Avoid using non-private Fields! Use Properties instead. **Example:** public string Name; protected IList InnerList;
Field (Private)	• Camel Case and prefix with a single underscore (_) character. **Example:** private string _name;
Variable	• Camel Case. • Avoid using single characters like "x" or "y" except in FOR loops. • Avoid enumerating variable names like text1, text2, text3 etc.
Parameter	• Camel Case. **Example:** public void Execute(string commandText, int iterations) {…}

Treat constants, static fields, enums, delegates and events as fields.

2. Coding Style

2.1. Formatting

- Never declare more than 1 namespace per file.
- Avoid putting multiple classes in a single file.
- Always place curly braces ({ and }) on a new line.
- 4Always use curly braces ({ and }) in conditional statements.
- Always use a Tab & Indention size of 4.
- Declare each variable independently – not in the same statement.

- Place namespace “using” statements together at the top of file. Group .NET namespaces above custom
- namespaces.
- Group internal class implementation by type in the following order:
    - Member variables.
    - Constructors & Finalizers.
    - Nested Enums, Structs, and Classes.
    - Properties
    - Methods
- Sequence declarations within type groups based upon access modifier and visibility:
    - Public
    - Protected
    - Internal
    - Private
- Segregate interface Implementation by using #region statements.
- Recursively indent all code blocks contained within braces.
- Use white space (CR/LF, Tabs, etc) liberally to separate and organize code.

2.2. Code Commenting

- Use // or ///
- Include comments using Task-List keyword flags to allow comment-filtering.
    - // TODO: Work to be done
    - // UNDONE: Code removed
    - // HACK: Temporary fix
- Always apply C# comment-blocks for documenting the API.
    - Always include <summary> comments. Include <param>, <return>, and <exception> comment sections where applicable.

3. Language Usage

3.1. General

- Do not omit access modifiers. Explicitly declare all identifiers with the appropriate access modifier instead of allowing the default

3.2. Variables and Types

- Try to initialize variables where you declare them.
- Always choose the simplest data type, list, or object required.
- Always use the built-in C# data type aliases, not the .NET common type system (CTS).
    - Example:
        - short NOT System.Int16
        - int NOT System.Int32
        - long NOT System.Int64
        - string NOT System.String
- Only declare member variables as private. Use properties to provide access to them with public, protected, or internal access modifiers.
- Try to use int for any non-fractional numeric values that will fit the int datatype - even variables for nonnegative numbers.
- Only use long for variables potentially containing values too large for an int.
- Try to use double for fractional numbers to ensure decimal precision in calculations.
- Only use float for fractional numbers that will not fit double or decimal.

3.3. Flow Control

- Avoid invoking methods within a conditional expression.
- Use the ternary conditional operator only for trivial conditions. Avoid complex or compound ternary operations.
    - **Example**:
        - **Good:**
            - int result = isValid ? 9 : 4;
        - **Bad:**

- int resul = (object.IsValid && otherObject.CanBeUsed(object.Value)) || z == null ? 9 : 4

- Avoid evaluating Boolean conditions against true or false.
- Avoid assignment within conditional statements.
    - **Example:** if((i=2)==2)
- Avoid compound conditional expressions – use Boolean variables to split parts into multiple manageable expression.
    - **Example:**
        - **Bad**
            - if (((value > _highScore) && (value != _highScore)) && (value < _maxScore))
        - **Good**
            - isHighScore = (value >= _highScore);
            - isTiedHigh = (value == _highScore);
            - isValid = (value < _maxValue);
            - if ((isHighScore && ! isTiedHigh) && isValid)
- Prefer polymorphism over switch/case to encapsulate and delegate complex operations.

3.4. Exceptions

- Do not use try/catch blocks for flow-control.
- 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
- Use validation to avoid exceptions
- Always set the innerException property on thrown exceptions so the exception chain & call stack are maintained

4.Object model and API Design

- Always prefer aggregation over inheritance.

- Avoid “Premature Generalization”. Create abstractions only when the intent is understood.
- Always separate presentation layer from business logic.
- Always prefer interfaces over abstract classes.
- Try to include the design-pattern names such as “Bridge”, “Adapter”, or “Factory” as a suffix to class names where appropriate.

5. Sources

http://se.inf.ethz.ch/old/teaching/ss2007/251-0290-00/project/CSharpCodingStandards.pdf