Coding Standards – Dot Net

Change History

Date	Ver	Change Description	Prepared By	Reviewed By	Approved By
16-07-2012	1.0	First Release	Senthil	Deepa	Mukesh V
26-08-2016	1.1	Standards have been added	Ranga	Vinod P	Harihara G
07-02-2017	1.2	Updated the	Deepa	Deepa	Harihara.G
01-02-19	1.3	Updated the document ID	Document Controller	MR	CISO

Coding Guidelines Summary

Communication – Simplicity –, Flexibility

Meaningful Names

- Use Intention-Revealing Names
- Use Solution Domain Names or Problem Domain Names
- Avoiding generic names (tmp, retVal)
- Provide more information, by using a suffix or prefix
- Don't use Hungarian notation
- Use capitalization, underscores, and so on in a meaningful way

Functions

- Small functions; Do One Thing
- One Level of Abstraction per Function
- Document parameters, error conditions and exceptions

Organization

- If multiple blocks of code are doing similar things, try to give them the same format/style.
- Aligning parts of the code into "columns" can make code easy to skim through.
- Use empty lines to break apart large blocks into logical "paragraphs."
- Separate the generic code from the project specific code.
- Do only one task at a time.

Comments

- Keep Comments Compact
- Describe Function Behavior Precisely
- State the high-level intent of your code, rather than the obvious details.
- Avoid pronouns like "it" and "this" when they can refer to multiple things.
- Describe a function's behavior with as much precision as is practical.
- Illustrate your comments with carefully chosen input/output examples.
- Use inline comments (e.g., Function(/* arg = */ ...)) to explain mysterious function arguments.

Loops & Logic

- Make your code's control flow easier to read.
- You can also reorder the blocks of an if/else statement. Generally, try to handle the positive/easier/interesting case first.
- Certain programming constructs, like the ternary operator (: ?), the do/while loop, and goto often result in unreadable code. Don't use them.
- Nested code blocks require more concentration to follow along.
- Returning early can remove nesting and clean up code in general.

Expressions & Variables

- Giant expressions are hard to think about. One simple technique is to introduce "explaining variables".
- Eliminate temporary variables that just get in the way, handle the result immediately.
- Reduce the scope of each variable to be as small as possible.
- Prefer write-once variables. Variables that are set only once (or const, final, or otherwise immutable) make code easier to understand.

Test code

- The top level of each test should be as concise as possible; ideally, each test input/output can be described in one line of code.
- If your test fails, it should emit an error message that makes the bug easy to track down and fix.
- Use the simplest test inputs that completely exercise your code.
- Give your test functions a fully descriptive name so it's clear what each is testing. Instead of Test1(), use a name like Test_<FunctionName>_<Situation>.

Class/Object related:

- Class—Use a class to say, “This data goes together and this logic goes with it.”
- Simple Superclass Name—Name the roots of class hierarchies with simple names drawn from the same metaphor.
- Qualified Subclass Name—Name subclasses to communicate the similarities and differences with a superclass.
- Abstract Interface—Separate the interface from the implementation.
- Interface—Specify an abstract interface which doesn’t change often with a C# / Java interface.
- Abstract Class—Specify an abstract interface which will likely change with an abstract class.
- Value Object—Write an object that acts like a mathematical value.
- Subclass—Express one-dimensional variation with a subclass.
- Inner Class—Bundle locally useful code in a private class.
- Delegation—Vary logic by delegating to one of several types of objects.
- Anonymous Inner Class—Vary logic by overriding one or two methods right in the method that is creating a new object.
- Library Class—Represent a bundle of functionality that doesn’t fit into any object as a set of static methods.

Method-related:

- Intention-Revealing Name—Name methods after what they are intended todo.
- Method Visibility—Make methods as private as possible.
- Method Object—Turn complex methods into their own objects.
- Overridden Method—Override methods to express specialization.
- Overloaded Method—Provide alternative interfaces to the same computation.
- Method Return Type—Declare the most general possible return type.
- Method Comment—Comment methods to communicate information noteasily read from the code.
- Helper Method—Create small, private methods to express the main computationmore succinctly.
- Debug Print Method—Use toString() to print useful debugging information.
- Conversion—Express the conversion of one type of object to anothercleanly.
- Conversion Constructor—provide a method on the converted object’s class that takes the source object as a parameter.
- Creation—Express object creation clearly.
- Complete Constructor—Write constructors that return fully formed objects.
- Collection Accessor Method—Provide methods that allow limited access to collections.
- Boolean Setting Method—If possible, provide two methods to set boolean values, one for each state.
- Equality Method—Define equals() and hashCode() together.

Common Collections

- Array—Arrays are the simplest and least flexible collection: fixed size, simple accessing syntax, and fast.
- Iterable—The basic collection interface, allowing a collection to be used for iteration but nothing else.
- Collection—Offers adding, removing, and testing for elements.
- List—A collection whose elements are ordered and can be accessed by their location in the collection (i.e., “give me the third element”).
- Set—A collection with no duplicates.
- SortedSet—An ordered collection with no duplicates.
- Map—A collection whose elements are stored and retrieved by key.