Coding Guidelines

Prepared By:
Triveni Global Software Services LLP

Last Updated:

March 6, 2013

Table of Contents

General Guidelines

1. Avoid putting multiple classes in a single file.
2. A single file should contribute types to only a single namespace. Avoid having multiple namespaces in the same file.
3. Avoid files with more than 500 lines (excluding machine-generated code).
4. Avoid methods with more than 200 lines.
5. Avoid methods with more than 5 arguments. Use structures for passing multiple arguments.
6. Lines should not exceed 120 characters.
7. Do not manually edit any machine-generated code.
    - If modifying machine generated code, modify the format and style to match this coding standard.
8. Avoid comments that explain the obvious. Code should be self-explanatory. Good code with readable variable and method names should not require comments. However, do comment uncommon/complicated code.
9. With the exception of zero and one, never hard-code a numeric value; always declare a constant instead.
10. Use the const directive only on natural constants such as the number of days of the week.
11. Catch only exceptions for which you have explicit handling.
12. In a catch statement that throws an exception, always throw the original exception (or another exception constructed from the original exception) to maintain the stack location of the original error:

```
catch(Exception exception)
{
   MessageBox.Show(exception.Message);
     throw;
}
```

13. Avoid error codes as method return values.
14. Avoid function calls in Boolean conditional statements. Assign into local variables and check on them.

```
bool IsEverythingOK()
{...}

//Avoid:
if(IsEverythingOK())
{...}
```

```
//Correct:
bool ok = IsEverythingOK();
if(ok)
{...}
```

15. Always use zero-based arrays.
16. With indexed collection, use zero-based indexes
17. Do not provide public or protected member variables. Use properties instead.
18. Never hardcode strings that will be presented to end users. Use resources instead.
19. Never hardcode strings that might change based on deployment such as connection strings.

1. Use String.Empty instead of "":
2.
3. //Avoid
4. string `name = "";`
5.
6. //Correct
7. string `name = String.Empty;`
8.
9. When building a long string, use StringBuilder, not string.
10. Always have a default case in a switch statement that asserts.

Naming Conventions and Style

1. Use Pascal casing for type and method names and constants:

```
public class SomeClass
{
     public void SomeMethod()
   {}
}
```

2. Use Pascal casing for local variable names and method arguments. Use lowercase prefix only for data types specified at the bottom.

```
void MyMethod(int iSomeNumber)
{
     int iNumber;
     string strSomeString;
   //used to be slccCustomClass but the abbreviations were getting
   out of hand
     SomeLongCustomClass CustomClass;
}
```

3. Prefix interface names with I

```
interface IMyInterface
{...}
```

4. Prefix private member variables with _. Use Pascal casing for the rest of a member variable name following the _.

```
public class SomeClass
{
     private int _Number;
}
```

5. Suffix custom attribute classes with Attribute.
6. Suffix custom exception classes with Exception.
7. Name methods using verb-object pair, such as ShowDialog().
8. Methods with return values should have a name describing the value returned, such as GetObjectState().
9. Use descriptive variable names.

    1. Avoid single character variable names, with the exceptions of loop indexes (eg. i). Even then, it is recommended to use meaningful name (eg. Index, ArrayIndex, etc)
    2. Do not abbreviate words (such as num instead of number).

10. Always use C# predefined types rather than the aliases in the System namespace.

For example:

```
object NOT Object
string NOT String
int NOT Int32
```

11. With generics, use capital letters for types. Reserve suffixing Type when dealing with the .NET type Type.

```
//Correct:
public class LinkedList<K,T>
{...}

//Avoid:
public class LinkedList<KeyType,DataType>
{...}
```

12. Use meaningful namespaces such as the product name or the company name.
13. Avoid fully qualified type names. Use the using statement instead.
14. Avoid putting a using statement inside a namespace.
15. Group all framework namespaces together and put custom or third-party namespaces underneath.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using MyCompany;
using MyControls;
```

16. Use delegate inference instead of explicit delegate instantiation.

```
delegate void SomeDelegate();
public void SomeMethod()
{...}

SomeDelegate someDelegate = SomeMethod;
```

17. Indent comments at the same level of indentation as the code you are documenting.
18. All comments should pass spell checking. Misspelled comments indicate sloppy development.
19. All member variables should be declared at the top, with one line separating them from the properties or methods.

```
public class MyClass
{
    int _Number;
    string _Name;

    public void SomeMethod1()
   {}

    public void SomeMethod2()
   {}
}
```

20. Declare a local variable as close as possible to its first use.
21. A file name should reflect the class it contains.
22. When using partial types and allocating a part per file, name each file after the logical part that part plays. For example:

```
//In MyClass.cs
public partial class MyClass
{...}

//In MyClass.Designer.cs
public partial class MyClass
{...}
```

23. Always place an open curly brace ({) in a new line.
24. With anonymous methods, mimic the code layout of a regular method, aligned with the delegate declaration. (complies with placing an open curly brace in a new line):

```
delegate void SomeDelegate(string someString);

//Correct:
void InvokeMethod()
{
   SomeDelegate someDelegate = delegate(string name)
   {
        MessageBox.Show(name);
    };

    someDelegate("Juval");
}

//Avoid
void InvokeMethod()
{
    SomeDelegate someDelegate = delegate(string
    name){MessageBox.Show(name);};

     someDelegate("Juval");
}
```

25. Use empty parentheses on parameter-less anonymous methods. Omit the parentheses only if the anonymous method could have been used on any delegate:

```
delegate void SomeDelegate();

//Correct
SomeDelegate someDelegate1 = delegate()
{
    MessageBox.Show("Hello");
};

//Avoid
SomeDelegate someDelegate1 = delegate
{
    MessageBox.Show("Hello");
};
```

26. With Lambda expressions, mimic the code layout of a regular method, aligned with the delegate declaration. Omit the variable type and rely on type inference,yet use parentheses:

```
delegate void SomeDelegate(string someString);

SomeDelegate someDelegate = (name)=>
{
    Trace.WriteLine(name);
    MessageBox.Show(name);
};
```

Only use in-line Lambda expressions when they contain a single simple statement. Avoid multiple statements that require a curly brace or a return statement with inline expressions. Omit parentheses:

```
delegate void SomeDelegate(string someString);

void MyMethod(SomeDelegate someDelegate)
{...}

//Correct:
MyMethod(name=>MessageBox.Show(name));

//Avoid
MyMethod((name)=>{Trace.WriteLine(name);MessageBox.Show(name);});
```

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.
- Always 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
- Recursively indent all code blocks contained within braces.
- Use white space (CR/LF, Tabs, etc) liberally to separate and organize code.
- If in doubt, always err on the side of clarity and consistency.

Type Notation

Button	btn
CheckBox	cb
CheckBoxList	cbl
ComboBox	cmb
Container	ctr
DataGrid	dg
DetailView	dv
DropDownList	ddl
FormView	fv
GridView	gv
ImageList	iml
Label	lb
ListBox	lbx
ListView	lv
Panel	pnl
RadioButton	rb
r.a.d.ajax	ra
r.a.d.calendar	rcal
r.a.d.chart	rch
r.a.d.combobox	rcmb
r.a.d.dock	rd
r.a.d.editor	re
r.a.d.grid	rg
r.a.d.input	ri
r.a.d.menu	rm
r.a.d.panelbar	rpb
r.a.d.rotator	rr
r.a.d.spell	rs
r.a.d.tabstrip	rts
r.a.d.toolbar	rtb
r.a.d.treeview	rtv
r.a.d.upload	ru
r.a.d.window	rw
TextBox	tb
UserControl	uc