תקנים לכתיבת קוד

Coding Standards

- Naming Conventions
- Statements Structure
- Delegates and Events
- Coding Style

Table Of Contents – תוכן עניינים

1 Coding Standards and Style
3 Part A – Standards
3 Variables and Class Members
4 Statements
4 Properties, Methods and Indexers
5 Classes and Structs
5 Enums
6 Delegates and Events
7 Part B - Style
7 Conditions
7 Layout
8 Spaces
8 Method Implementations

Part A – Standards

Variables and Class Members

1. Declare variables and class members with **meaningful names** (even if they come out very long)
2. Indices with no special meaning may be named "i", "j", etc. (even advisable)
3. **Local Variables**
   a. First letter should be lower-cased.
   b. First letter of each new word should be upper-case (Camel-Casing with first-letter-lower-case).
   c. Example:

```
int numOfItems; //GOOD
int numofitems; //BAD
int NumOfItems; //BAD
int NUMOFITEMS; //BAD
```

4. **Local** `const bool`:
   a. Starts with a 'v_' and the Camel Case.
   b. Use instead of `true`/`false` literals for parameters to methods.
   c. Always `true`. Use '!' to send negative value.

```
//GOOD:
const bool v_IncludeSubFolders = true;
string line = SearchEngine.FindLine(@"C:\SomeFolder", v_IncludeSubFolders);
string line = SearchEngine.FindLine(@"C:\SomeFolder", !v_IncludeSubFolders);

//BAD:
string line = SearchEngine.FindLine(@"C:\SomeFolder", true);
string line = SearchEngine.FindLine(@"C:\SomeFolder", false);
```

5. **Class/Struct Members**
   a. **Regular** data member names will be in the following format m_SomeName:

```
private int m_NumOfItems; //GOOD
private int m_numOfitems; //BAD
private int myNumOfItems; //BAD
private int numOfItems;   //BAD
```

   b. **Const** member names **and local consts** will always start with a lower-cased **'k'** and an underscore:

```
protected const int k_NumOfItems = 20; //GOOD
```

   c. **Static** member names will always start with a lower-cased **'s'** and an underscore:

```
protected static int s_NumOfItems = 20; //GOOD
```

   d. **Readonly** member names will always start with a lower-cased **'r'** and an underscore:

```
protected readonly int r_NumOfItems = 20; //GOOD
```

   e. **Static Readonly** member names will always start with a lower-cased **'sr'** and an underscore:

```
protected static readonly int sr_NumOfItems = 20; //GOOD
```

Statements

All statement bodies should be wrapped with block wrappers (curly braces)

if:

```
//GOOD
if(!falseBoolParam)
{
     code...
}
else
{
     code...
}

//BAD
if(int i = 0; i < 15; ++i)
     one line of code…
else(int i = 0; i < 15; ++i)
     one line of code…
```

Loops:

```
//GOOD
for (int i = 0; i < 15; ++i)
{
     code...
}

//BAD
for (int i = 0; i < 15; ++i)
     one line of code…
```

Properties, Methods and Indexers

1. **Naming**
   a. Declare with meaningful names (even if they come out very long).
   b. Do not add "_" (underscore) to the name.
   c. Use Camel-Casing (Each new word starts with a capital).
   d. `private` methods should start **lower cased**

```
public    int GetNumOfItems();          //GOOD
protected int GetNumOfItems();          //GOOD
private   int getNumOfItems();          //GOOD
public    int getnumofitems();          //BAD
public    int get_num_of_items();       //BAD
public    int Getnumofitmes();          //BAD
public    int GETNUMOFITEMS();          //BAD
```

2. **Declaration** (Method's signature)
   a. Use SPACE between parameter declarations (and not TAB).
   b. Method declaration should be in a single line.
   c. In case of many parameters, split parameters declaration to several lines.

3. **Parameters**
   a. Declare parameters with meaningful names (even if they come out very long).
   b. Parameter name will pre-fixed with:
      - ***i_*** – The value is relevant inside the method only
      - ***o_*** – The value is relevant outside the method only
      - ***io_*** – The value is relevant both inside and outside the method

GOOD

```
public int        GetNumOfItems()
{...}

protected void    AddCustomer(string i_Name, int i_Age)
{...}

private string    addProducer(string i_Name, string i_ContactName,
                      ref string io_PhoneNumber, out string o_Age);
```

BAD

```
public int
GetNumOfItems(){...} // (should be in one line)

protected void AddCustomer(string i_Name,int i_Age) // (no space)
{...}

private string addProducer(string name, string contactName,
                           ref string phoneNumber,
                           out string age) // (naming params)
{...}
```

Classes and Structs

1. Declare with meaningful names (even if they come out very long).
2. Use Camel-Casing (Each new word starts with a capital).

Enums

1. Declare with meaningful names (even if they come out very long).
2. Use Camel-Casing (Each new word starts with a capital) with the prefix 'e'.
3. Use plural ('s') for flagged enums.
4. Use Camel-Casing for the list of enums also and do not prefix them like in C++.

```
public enum eFuelType  //GOOD
{
     Octan96,
     Octan95,
     Octan98,
}

[Flags]
public enum eFuelTypes //GOOD
{
     Octan96,
     Octan95,
     Octan98,
}

public enum Fuel_Type //BAD
{
     eftOctan96, // bad - no need for prefix
     octan95,    // bad - not camel-cased
     octan_98,   // bad - no need for underscore
}
```

Delegates and Events

1. Declare with meaningful names (even if they come out very long).
2. Delegate name convention is the same as class name convention (delegate are classes…) with an extension of the word 'Delegate'.

```
public delegate void ErrorNotificationDelegate(string i_ErrorMsg);
```

3. The extension for a delegate which is defined to be an event handler will be 'EventHandler' (and not 'Delegate'):

```
public delegate void EventHandler(object sender, EventArgs e);
public delegate void CancelEventHandler(object sender CancelEventArgs e);
public delegate void MouseEventHandler(object sender MouseEventArgs e);
```

4. Event naming is Camel-Cased and will described the event in the best way (usually will be an action description).

```
public event EventHandler Closed; // this means I am notifing after I was closed
public event CancelEventHandler Closing; // this means I am in the closing process
public event EventHandler AfterEdit; // being raised after edit process is completed
public event EventHandler BeforeEdit; // being raised before edit process was started
```

5. The name of the event handler method which is declared in the subscriber will be constructed from the name of the instance that raises the event, underscore and the name of the event:

```
private void buttonOK_Click(object sender, EventArgs e){...}
```

6. The name of the internal method (in the class that raises the event) which is responsible for raising the event will be constructed with the format of OnXXX (while XXX is the event's name):

```
protected virtual void OnClick(EventArgs e)
{
      // check if someone is listening
      if(Click != null){
            // raise the event:
            Click.Invoke(this, e)
      }
}
```

Part B - Style

Conditions

1. **Boolean assignments**
   Use Boolean assignments when relevant:

```
// BAD - START
if (age > 50)
{
     isOldMan = true;
}
else
{
     isOldMan = false;
}
// BAD - END

//GOOD
isOldMan = age > 50;
```

2. **Duplicated code**
   Be aware of cases in which duplicated code resides in 'if' and 'else' statements:

```
// BAD - START
if (age > 50)
{
     CheckEyes();
     CheckHealth();
}
else
{
     CheckYearsOfLiscense();
     CheckHealth();
}
// BAD - END

//GOOD - START
if (age > 50)
{
     CheckEyes();
}
else
{
     CheckYearsOfLiscense();
}
CheckHealth();
//GOOD - END
```

Layout

1. Use TABs (and not SPACEs) for indentations.
2. Each block will be tabbed inside its parent block:

```
public static void Main()
{
     if (boolVar)
     {
          for (int j = 0 ; j < 44 ; ++j)
          {
               someLinesOfCode...
          }
     }
}
```

Spaces

1. **No double space lines.**
2. **Do not** leave redundant space lines.
3. **Use** spaces in the following cases:
   a. Between operators and operands:

```
x = t + 5;      //GOOD
x=t+5;          //BAD
if(x == 5)      //GOOD
if(x==5)        //BAD
```

   b. Between method parameter declatrations:

```
Console.WriteLine("Hello {0} and {1}!", name1, name2); //GOOD
Console.WriteLine("Hello {0} and {1}!",name1,name2);    //BAD
```

4. **Do not use** spaces in the following cases:
   a. Before method braces:

```
Console.WriteLine("hello world");          //GOOD
Console.WriteLine ("hello world");         //BAD
DoSomeThing(5, 4);                         //GOOD
DoSomeThing (5, 4);                        //BAD
```

5. **Use space lines** in the following cases:
   a. After local variable declarations section

```
public int PrintNameAndAge()
{
   int         age = 18;
   string      name = "moshe"

   Console.WriteLine("{0} is {1} years old", name, age);
   ...
```

   b. After a closing curly bracket (end of block), unless followed by another closing curly, or an 'else' statement:

```
        Console.WriteLine("Last line in this block");
    }

    Console.WriteLine("First line after the block");
  }
}
```

   c. Before return statement **(which should be single in a method!)**

```
   ...
   Console.WriteLine("Have a nice day");

   return retVal;
}
```

Method Implementations

1. Use only **one** `return` **statement** in a method.
2. **Methods should implement a flow of calling other methods**