Yet another style guide for C#

Why?

I'm sure you are wondering why do you need a style guide if you already know how to program. True that, but let me tell you something (and I don't mean to be rude): **Anyone can write code**. Anyone with some months of experience, maybe a lecture or even an internet video can write a functional app in little time. Writing code that works is easy, writing good code that works is way harder.

The Good Code

About The Good Code, everyone seems to have a distinct opinion, yet there are some points where those opinions come together, based on those I dare to say that The Good Code is:

- Efficient
- Maintainable
- Well organized
- Legible

Writing The Good Code is an art that you must learn, practice and encourage. To adopt an style (or create your own) could help you to achieve that.

About style guides

Contrary to what it might seem, this guide (and any other you may happen to find) isn't a rulebook that you must follow strictly, there may be things that simply you cant integrate into your organization due philosophical reasons, compatibility with legacy systems or complicated errors in The Matrix. So, take this kind of guides as suggestions that, if applied to your organization, might help to keep an standardized code, easily legible and maintainable for everyone in your dev team.

The Guide

My name is Asdflkj, James Asdflkj

Incredibly, choosing the right names for your variables, methods and classes could make the difference between wanting to use an old code or write the same code from scratch. You may not want to find out for what you were using that `c` variable inside the `buysomething` method and prefer to write the `muysomething2` method instead. Think about it.

Recommendations about naming

- Use comprehensible names for classes, methods, variables, etc.

- In case of class properties, unless another logic is needed, always use Auto-Implemented properties.
  ```
  int Property { get; set; }
  ```

- Use an _ before a member variable, then write the name using *camel casing*.

```
private int _clientId
private string _name;
```

- When using boolean variables, use prefix words like “can”, “is”, or “has”, depending on what they represent.

```
public bool IsCool { get; set; }
private bool _hasChildren;
```

- Avoid using variable names that look like C# (or any other language) reserved keywords.

- Use full names when possible, avoid abbreviations. If you are working with acronyms keep only the first character in uppercase.

```
string addr; // wrong
string address; // right
void PublishHtml() { ... // right
void PublishHypertextTrans... // wrong
```

- Use a single letter variable name only for cycles where it is used only to count iterations.

```
for(int i = 0; i < value; i++) {
```

- Use *pascal casing* to name classes, methods and scructs.

```
class SharpGuide { ...
void FixMatrix(int matrixId) { ...
```

- Use *camel casing* to name variables and method parameters.

```
string awesomeProgrammingLanguage = "C#";
void SetLanguage(string someLanguage) { ...
```

- For interfaces, prefix an `I` to the name in *camel casing*.
  `interface IStyleGuide { ...`

- For `namespaces` follow the next convention `<company name>.<product name>.<top module>.<bottom module>`

- Al files must be named after the class or interface they contain.

Comments? really?

Maybe you just finished writing a poetry with code (including variables and method names) and your code isn't as understandable as it should be due some crazy business logic or an efficiency trick you applied, that is when a good comment helps other developers. Or, maybe the company you're wants you to document every single method you created. Here are some recommendations for comments:

- Do not narrate your code through the comments or, in other words, don't write a comment for eery line of code.

- Write comments on empty lines and above the section the comment refers to. As an exception, write a comment on the same line of a variable declaration if needed.

- Use `//` or `///` to write comments, avoid `/* ... */` (and avoid writing comments of more than one line).

- Your comments are beautiful as they are, there is no need to put a fancy `********` around them, so avoid this as well.

- Check your comments spelling and make sure they make sense when read.

- There is this cool special comment that you must use when you leave unfinished code, and that is `// TODO: [description]` some IDEs would identify them an put them in a task list so you can easily come back and finish the work.

- In case a method or a class isn't as self documented as you want, use XML comments `///`, some IDEs will grab those comments and show them when a developer makes a reference to the documented method or class.

- Remember that C# allows to write XML documentation, make sure it is well formed.

File and class structure

There are some recommendation about files, it structure and content.

- Try to write a class (or interface) per file, there might be some cases when that couldn't be possible, generally it is.

- The file name must be the same as the class or interface it contains.

- Use `tab` of size 4 to indent the code instead of spaces.

- Always write curly brackets `{ }` on empty lines.

- Group lines of code for what they do, leave an empty line between groups of code.

- Leave an empty line between every method declaration.

- La estructura general de un archivo es:
    - *using* sentences
    - *namespace*
    - Class declaration
- Private variables
    - Private properties
    - Public properties
    - Constructors
    - Public methods
    - Private methods
- Use the `#region` and `#endregion` directives to group the mentioned categories and make it easy to identify them.

Other good practices

- Avoid writing a lot of lines per method, keep them around 30 lines maximum, if your method goes beyond that mark, stop and see if it could be broken down into two or more small pieces.

- Once again, **self document** your methods, if the name is clear enough there is no need to explain a lot using the comments.

- Do not mix task in the same method. One action for each method.

- Use the data types alias to declare variables and method parameters, choose `string awesome = "awesome!";` over `String awesome = "awesome!".`

- On the other hand, use the data types defined in `System` to make reference to constants or class methods, choose `Int32.ParseInt("0");` over `int.ParseInt("0");`.

- Not everything inside a class must be either private or public, use the `internal` access modifier for classes, methods or properties that must be accesible only within the same assembly.

- If a method receives more than 5 or 6 parameter, consider to renace them by an struct or a class that contains those parameters as properties.

- **NEVER, never** *hardcode*[1] numbers, strings or any other value. Use external data sources like files, databases or even the command line.

- Use `String.Empty` instead of an empty double quoted string `""`.

- To check if a string is empty use the class methods `String.IsNullOrEmpty` or `String.IsNullOrWhiteSpace`.

- Always try to override the `Equals` method for user defined classes.

Contributing

I'm just a human being who likes to code, specially using C#, so as I said: this is not a rulebook but a few things that I consider as good practices. If you want to make a comment, complain or suggestion about this guide you can do it at https://github.com/fferegrino/cool-sharp/issues, or antonio.feregrino@hotmail.es.

---

1. Awful development practice that consists of placing business logic data directly in the source code of an application.