C# Coding Standard

Version 2.0

The Salient development team has ratified a C# coding standard that is applicable to all C# software development henceforth.

- The foundation for the standard is contained here.
- This is an industry-proven coding standard that has been in use for years.

There are several extensions / clarifications to the basic rules contained therein that are outlined here:

No.	Coding Standard Rule
1	There shall be a maximum of 100 characters per line of source text (special allowances for string literals for violations per the programmer's discretion)
2	Representations of physical quantities (time for example) shall be named in such a manner as to make obvious the units of the physical quantity **Example:** a variable named “timeout” would instead be named “timeoutMs” to indicate the units as milliseconds
3	Properties shall use the upper camel case naming convention. **Example:** VideoStartTime
4	Method argument names and local variable names shall use the lower camel case naming convention. **Example:** videoStartTime
5	The naming convention for fields is similar except that a leading underscore shall be used **Example:** _videoGammaLevel
6	Method names shall use upper camel case.
7	When acronyms occur in a variable name, only the first letter in the acronym shall be capitalized. **Example:** localHttpConnection
8	Methods and member variables shall be organized such that there is a very obvious separation based on access control - public methods appear first in the source text.
9	Class hierachies shall contain no more than 7 levels of depth.
10	All public methods, properties and members shall be documented utilizing the built in C# documentation facitilies (XML). All comments shall be of high quality, without typos, and of a descriptive nature such that an unfamiliar developer can quickly comprehend the related code.
11	All source code files shall use tabs instead of spaces, with a tab being 4 spaces wide.
12	There shall be at most 1 line of white space between methods, code blocks, etc.
13	Braces shall always be used for all control structures. See Example 1.
14	All braces shall be on their own lines.
15	Comments shall include a space between the comment characters and the actual comment content, and shall begin with an uppercase letter. See Example 2.
16	All operators shall have spaces on either side. See Example 3.
17	All control statements shall have a space between the keyword and the parenthesis. See Example 4.
18	Classes shall reside in the same directory structure as the namespace. For example, if a class was in the namespace GenIIIWebClient.Views.UserControls, it should also be in the directory \GenIIIWebClient\Views\UserControls

No.	Coding Standard Rule
19	Event names shall be a past tense word or phrase. For example, we would use “Connected” over “DidConnect”
20	Cases inside a switch statement shall always be indented.
21	If a case inside a switch statement does not have a break statement, a comment shall be written above that case describing the intention for not including the break.
22	All statements shall be on their own lines.
23	All interface class names shall begin with the capital letter I. **Example:** ICaptureServer
24	Commented out code shall not be committed to master or maintenance branches. Commented code may be committed to personal topic or feature branches provided that it is removed upon opening a Pull Request or otherwise merging to the master or maintenance branches.
25	Regions shall be used and named in a consistent manner. The order of regions shall also be consistent between files. See Example 5.
26	There shall only be one class per file. The file name shall also match the class name.
27	Modern language features shall be used where possible, especially when Visual Studio lightbulbs are present. See Example 6.
28	Xaml elements shall use the self-closing tags where possible. See Example 7.
29	There shall only be one Xaml element per line.
30	There shall only be one Xaml attribute per line. See Example 8.
31	A namespace declaration shall match the directory structure and file path of which it is declared in. See Example 9.
32	A project name shall be used as the base of the namespace for all files contained within that project. See Example 10.

Proposed Rules

Here we have a list of rules that are up for discussion and have not yet been ratified into the actual standard by the Development team.

No.	Coding Standard Rule

Useful Utilities

If at all possible and budgets allow, all developers who regularly work on C# code should use Jetbrains ReSharper. If you are using ReSharper, apply our coding standard rules automatically by using our ReSharper settings file, located here: \\valhalla\Shared\Engineering\Dev Resources\ReSharper C# Settings.DotSettings

Rule Examples

Example 1

```
if (!someCondition)
    DoSomethingElse();
```

Should be:

```
if (!someCondition)
{
    DoSomethingElse();
}
```

Example 2

```
//this is a comment
```

Should be:

```
// This is a comment
```

Example 3

```
int i = j+10;
```

Should be:

```
int i = j + 10;
```

Example 4

```
if(result == 10)
{
    // Do something
}
```

Should be:

```
if (result == 10)
{
    // Do something
}
```

Example 5

Code shall be roughly organized as such, in the following order. If a certain region type is not needed (i.e. Dependency Properties), it may be left out of the code file.

- Constants and Enums
- Events / Delegates
- Dependency Properties

- Properties
- Constructors
- Public methods
- Private methods
- Fields

Example 6

Modern language features to be used:

- String interpolation
- Null conditional operator
- nameof()
- Auto property intializers
- async / await

Example 7

```
<TextBlock
   x:Name="TextBlockExample"
   Text="Test">
</TextBlock>
```

Should be:

```
<TextBlock
   x:Name="TextBlockExample"
   Text="Test" />
```

Example 8

```
<ComboBox x:Name="ComboBoxTest" Height="40" Width="250"
    Grid.Row="0" Grid.Column="1" />
```

Should be:

```
<ComboBox
    x:Name="ComboBoxTest"
    Height="40"
    Width="250"
    Grid.Row="0"
    Grid.Column="1" />
```

Example 9

If you have a class in the file path: **{base}\CompleteView\Desktop\Infrastructure\ViewModels**
The corresponding namespace should be:

```
namespace CompleteView.Desktop.Infrastructure.ViewModels
{
}
```

Example 10

If you have a project named: **CompleteView.Infrastructure**
The corresponding namespace for all files within should be:

```
namespace CompleteView.Infrastructure.{remaining structure}
{
}
```

Best Practices

The best practices listed herein should be typically followed when writing modern C # applications to ensure reliability and extensiblity.

No.	Best Practice
1	Code duplication should be avoided wherever possible.
2	Code files should be organized either in projects or folders based on their role and how they can be shared with other projects. If a certain list of files becomes hard to navigate, consider breaking it up into logical groupings in sub folders.
3	Unused or dead code should be avoided wherever possible. Experimental code can be kept in topic branches.
4	When using async / await, the async void signature should only be used on asynchronous event handlers.
5	When following the MVVM pattern, code behind files should be kept quite lean to improve test-ability.
6	ViewModel classes should not reference UI related classes to maintain separation of concerns.
7	All UI elements should have a unique name to ensure accessibility to Coded UI testing.
8	The SOLID principles should be followed to maintain a high level of test-ability. For example, dependency injection can be used to isolate code under test, the use of interfaces allows for the mocking of behavior and the single responsibility principle can be used to help keep testing more focused.