# Preface

## Rule of Thumb

1. Readability first (your code should be your documentation most of the time)
2. Follow IDE's auto formatted style unless you have really good reasons not to do so. (Ctrl + K + D in Visual Studio)
3. Learn from existing code

## References

This coding standards is inspired by these coding standards

* [Unreal engine 4 coding standard](https://docs.unrealengine.com/latest/INT/Programming/Development/CodingStandard/)
* [Doom 3 Code Style Conventions](ftp://ftp.idsoftware.com/idstuff/doom3/source/codestyleconventions.doc)
* [IDesign C# Coding Standard](http://www.idesign.net/downloads/getdownload/1985)

# 1. Naming Conventions and Style

1. Use Pascal casing for class and structs  
   class SomeClass;  
   struct SomeStruct;
2. Use camel casing for local variable names and function parameters  
   void SomeMethod(int someParameter)  
   {  
    int someNumber;  
   }
3. Use pascal casing for all method names  
     
   public uint GetAge()  
   {  
    // function implementation...  
   }
4. Use verb-object pairs for method names  
     
   public uint GetAge()  
   {  
    // function implementation...  
   }
5. Post fix any non-public method with "Internal"  
     
   Private uint GetAgeInternal()  
   {  
    // function implementation...  
   }
6. Use ALL\_CAPS\_SEPARATED\_BY\_UNDERSCORE for constants and  
   const int SOME\_CONSTANT = 1;
7. Use pascal casing for namespaces  
    namespace System.Graphics
8. prefix boolean variables with b and prefix boolean properties with Is   
   bool bFired; // for local variable  
   Private bool mbFired; // for private class member variable  
   Public bool IsFired { get; private set; }
9. prefix interfaces with I  
   interface ISomeInterface;
10. prefix enums with E  
    public enum EDirection  
    {  
     North,  
     South  
    }
11. prefix private member variables with m. Use Pascal casing for the rest of a member variable  
    Public class Employee  
    {  
     public int DepartmentID { get; set; }  
     private int mAge;  
    }
12. Methods with return values must have a name describing the value returned  
    public uint GetAge();
13. Use descriptive variable names. e.g index or employee instead of i or e unless it is a trivial index variable used for loops.
14. Capitalize every characters in acronyms if they have only 2 characters.  
    public int ID { get; private set; }
15. Capitalize only first character in acronyms if they have more than 2 characters.  
    public string HttpAddress { get; private set; }
16. Prefer properties over getter setter functions  
    Use:  
    public class Employee  
    {  
     public string Name {get; set;}  
    }  
    Instead of:  
    public class Employee  
    {  
     private string mName;  
     public string GetName();  
     public string SetName(string name);  
    }
17. Use real tabs that are equal to 4 spaces
18. Declare local variables as close as possible to the first line where it is being used.
19. Always place an opening curly brace ({) in a new line
20. Add curly braces even if there's only one line in the scope  
    if (bSomething)  
    {  
     return;  
    }
21. Use precision specification for floating point values unless there is an explicit need for a double  
    float f = 0.5F;
22. Always have a default case for a switch statement.  
    switch (number)  
    {  
     case 0:  
     ...   
     break;  
     default:  
     break;
23. Add //fallthrough comment for switch case fall through unless there is no code in the case statement.   
    switch (number)  
    {  
     case 0:  
     DoSomething();  
     //fallthrough  
     case 1:  
     DoFallthrough();  
     break;  
     case 2:  
     case 3:  
     DoNotFallthrough();  
     default:  
     break;  
    }
24. If default case must not happen in a switch case, always add Assert(false). In our assert implementation, this will add optimization hint for release build.  
    switch (type)  
    {  
     case 1:  
     ...   
     break;  
     Default:  
     Debug.Fail("unknown type");  
     break;  
    }
25. Names of recursive functions end with "Recursive"  
    void FibonacciRecursive();
26. Order of class variables and methods must be as follows:
    1. public variables
    2. internal variables
    3. protected variables
    4. private variables
    5. public methods
    6. Internal methods
    7. protected methods
    8. private methods
27. Function overloading must be avoided in most cases  
    Use:  
    Anim GetAnimByIndex(int index);  
    Anim GetAnimByName(string name);  
      
    Instead of:  
    Anim GetAnim(int index);  
    Anim GetAnim(string name);
28. Each class must be in a separate source file unless it makes sense to group several smaller classes.
29. The filename must be the same as the name of the class including upper and lower cases  
    class Anim {}  
      
    Anim.cs
30. Use assert for any assertion you have. Assert is not recoverable. (e.g, most function will have Debug.Assert( not null parameters ) )
31. The name of a bitflag enum must be suffixed by Flags  
    public enum EVisibilityFlags  
    {  
    }
32. Prefer overloading over default parameters
33. When default parameters are used, restrict them to natural immutable constants such as null, false or 0.
34. Shadowed variables are not allowed.  
    public class SomeClass  
    {  
     public int Count {get;set;}  
     public void Func(int Count)  
     {  
     for (int count = 0; count != 10; ++count)  
     {  
     // Use Count  
     }  
     }  
    }
35. Always use containers from System.Collections.Generic over ones from System.Collections. Using pure array is fine as well.
36. Prefer to use real type over var. Acceptable var usage includes Enumerator
37. Use static class, not singleton pattern
38. Use async Task instead of async void. The only place async void is allowed is for event handler.
39. Validate any external data at the boundary and return before passing the data into our functions. This means that we assume all data is valid after this point.
40. Therefore, do not throw any exception from inside our function. This should be handled at the boundary only.
41. Prefer not to allow null parameter in your function, especially from a public one.
42. If null parameter is used, and postfix the parameter name with OrNull
43. Prefer not to return null from any function, especially from a public one. However, you sometimes need to do this to avoid throwing exceptions.
44. If null is returned from any function. Postfix the function name with OrNull.  
      
    public string GetNameOrNull();
45. Try not to use object initializer. Use explicit constructor with named parameters instead. Two exceptions.
    1. When the object is created at one place only. (e.g, one-time DTO)
    2. When the object is created inside a static method of the owning class. (e.g, factory pattern)

# 2. Framework Specific Guidelines

## 2.1. XAML Controls

1. Do not name (i.e, x:name) a control unless you absolutely need it
2. Use pascal casing with prefixed 'x' character for the name.  
   xLabelName
3. Prefix the name with full control type  
   xLabelName  
   xButtonAccept

## 2.2. ASP .NET Core

1. When using DTO(Data Transfer Object)s for a request body for a RESTful API, make each value-type property as nullable so that model validation is automatic  
     
   [Required]  
   public Guid? ID { get; set; }

1. Validate all the requests as the first thing in any controller method. Once validation passes, all inputs are assumed to be correct. So no [required] nullable properties will be null.
2. Unlike above, [RouteParam] will not have ?  
     
   public bool GetAsync([RouteParam]Guid userID)