C# Coding Standards

Capitalization

Use PascalCase for naming:

- Classes/Types
- Pubic Properties
- Methods
- Events
- Namespaces
- Enumerated Types and Values
- Interfaces

Use camelCase for naming:

- Private fields
- Parameters
- Local Variables
- Fields
    - Fields should be private and exposed via properties, hence not PascalCase

Prefixes

Interfaces should be prefixed with a capital “I”

```
Interface ISomeInterface
```

Booleans should be prefixed with an affirmative assertion with the appropriate capitalization. Examples:

- IsPending
- HasExecuted
- NeedsService

Exceptions

Custom Exception types should be used sparingly, but end Exception type names with the word “Exception”.

```
public class BadDataException : Exception
```

Favor throwing exceptions over returning error codes to ensure they are handled. Handle any exceptions that may be thrown by called code. Exceptions thrown in private methods can bubble up and be handled by the public caller if appropriate.

White Space and Formatting

Use the following white space guidelines:

- All tabs should be formatted as spaces
- Tabs should be 4 spaces long
- Always use a space after “for”, “if”, and other similar statements before the parenthesized condition(s)

- Always use spaces between parameters and arguments
- Always separate operands and operators with spaces.

Use the following guidelines for formatting and general coding conventions

- Only one statement or declaration should be made per line
- All curly braces should go on their own lines.

Example snippet:

```
class SomeClass(int firstParam, int secondParam)
{
    public bool IsPositive(int firstNum, int secondNum)
    {
        int sum = firstNum + secondNum;
        if (sum > 0)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

Try to keep lines 80 characters long or fewer. Exceptions will occur, but

in most cases, lines of code exceeding 80 characters can be further modularized to be more readable and concise.

Documentation

Headers

All Classes Should be documented with a header explaining the class's conceptual purpose or any special, non-obvious considerations. The specifics of a class's behavior should be documented in its methods and properties

All methods and properties should be documented with XML headers.

For Methods, this should include at a minimum the following tags. Additional tags should be used as appropriate.

- <summary>
- <exception> for every exception the method can throw
- <param> for every parameter
- <returns> if there is a return value

For classes, the summary tag should be included. Various supplemental tags can be included as appropriate.

For properties, a header is not necessary if the property's purpose is obvious from the name. If not, the <value> tag should be included at a minimum.

Formatting tags (<c>, <code>, etc.) are recommended when appropriate.

Comments

In general, comments should be placed on their own lines rather than appended at the end of a line of code. There will be exceptions where the latter is more appropriate, but this should be the general guideline. Comments should be separated from the comment indicator ("//") with a space, should begin with capitalization, and should end with punctuation

Example from above, (overly, for the example) documented:

```
/// <summary>This class does some things with numbers.</su
mmary>
class SomeClass(int firstParam, int secondParam)
{
    /// <summary> Check if the sum of two numbers is posit
ive </summary>
    /// <param name="firstNum">The first number to sum.</p
aram>
    /// <param name="secondNum"> The second number to sum.
</param>
    /// <returns> True if the sum of the inputs is positiv
e. False otherwise.</returns>
    public bool IsPositive(int firstNum, int secondNum)
```

```
    {
        int sum = firstNum + secondNum;

        // Check if the sum of the inputs is positive.
        if (sum > 0)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```