# 4. Coding convention

**C# Coding Standards and Naming Conventions**

| Object Name | Notation | Length | Plural | Prefix | Suffix | Abbreviation | Char Mask | Underscores |
|---|---|---|---|---|---|---|---|---|
| Namespace name | PascalCase | 128 | Yes | Yes | No | No | [A-z][0-9] | No |
| Class name | PascalCase | 128 | No | No | Yes | No | [A-z][0-9] | No |
| Constructor name | PascalCase | 128 | No | No | Yes | No | [A-z][0-9] | No |
| Method name | PascalCase | 128 | Yes | No | No | No | [A-z][0-9] | No |
| Method arguments | camelCase | 128 | Yes | No | No | Yes | [A-z][0-9] | No |
| Local variables | camelCase | 50 | Yes | No | No | Yes | [A-z][0-9] | No |
| Constants name | PascalCase | 50 | No | No | No | No | [A-z][0-9] | No |
| Field name | camelCase | 50 | Yes | No | No | Yes | [A-z][0-9] | Yes |
| Properties name | PascalCase | 50 | Yes | No | No | Yes | [A-z][0-9] | No |
| Delegate name | PascalCase | 128 | No | No | Yes | Yes | [A-z] | No |
| Enum type name | PascalCase | 128 | Yes | No | No | No | [A-z] | No |

1. Do use PascalCasing for class names and method names

```
-============================
public class ClientActivity
{
  public void ClearStatistics()
  {
    //...
  }
  public void CalculateStatistics()
  {
    //...
  }
}
-============================
```

2. Do use camelCasing for method arguments and local variables:

```
-=======================
public class UserLog
{
  public void Add(LogEvent logEvent)
  {
    int itemCount = logEvent.Items.Count;
    // ...
  }
}
-=======================
```

3. Do not use Hungarian notation or any other type identification in identifiers

```
-=======================
// Correct
int counter;
string name;
// Avoid
int iCounter;
string strName;
-=======================
```

4. Do not use Screaming Caps for constants or readonly variables:

```
-===============
// Correct
public const string ShippingType = "DropShip";
// Avoid
public const string SHIPPINGTYPE = "DropShip";
-===============
```

5. Use meaningful names for variables

```
-================
var seattleCustomers = from customer in customers
  where customer.City == "Seattle"
  select customer.Name;
-================
```

6. Avoid using Abbreviations. Exceptions: abbreviations commonly used as names, such as Id, Xml, Ftp, Uri

```
-===============
// Correct
UserGroup userGroup;
Assignment employeeAssignment;
// Avoid
UserGroup usrGrp;
Assignment empAssignment;
// Exceptions
CustomerId customerId;
XmlDocument xmlDocument;
FtpHelper ftpHelper;
UriPart uriPart;
-===============
```

7. Do not use Underscores in identifiers. Exception: you can prefix private fields with an underscore:

```
-===============
// Correct
public DateTime clientAppointment;
public TimeSpan timeLeft;
// Avoid
public DateTime client_Appointment;
public TimeSpan time_Left;
// Exception (Class field)
private DateTime _registrationDate;
-===============
```

8. Do use predefined type names (C# aliases) like int, float, string for local, parameter and member declarations. Do use .NET Framework names like I**nt32, Single, String** when accessing the type's static members like **Int32.TryParse** or **String.Join.**

```
-===============
// Correct
string firstName;
int lastIndex;
bool isSaved;
string commaSeparatedNames = String.Join(", ", names);
int index = Int32.Parse(input);
// Avoid
String firstName;
Int32 lastIndex;
Boolean isSaved;
string commaSeparatedNames = string.Join(", ", names);
int index = int.Parse(input);
-===============
```

9. Do use implicit type var for local variable declarations. Exception: primitive types (int, string, double, etc) use predefined names

```
-===============
var stream = File.Create(path);
var customers = new Dictionary();
// Exceptions
int index = 100;
```

```
string timeSheet;
bool isCompleted;
-===============
```

10. Do use noun or noun phrases to name a class.

```
-===============
public class Employee
{
}
public class BusinessLocation
{
}
public class DocumentCollection
{
}
-===============
```

11. Do prefix interfaces with the letter I. Interface names are noun (phrases) or adjectives

```
-============
public interface IShape
{
}
public interface IShapeCollection
{
}
public interface IGroupable
{
}
-============
```

12. Do organize namespaces with a clearly defined structure

```
-============
// Examples
namespace Company.Technology.Feature.Subnamespace
{
}
namespace Company.Product.Module.SubModule
{
}
namespace Product.Module.Component
{
}
namespace Product.Layer.Module.Group
{
}


-============
```

13. Do vertically align curly brackets:

```
-============
// Correct
class Program
{
  static void Main(string[] args)
  {
    //...
  }
}
-============
```

14. Do declare all member variables at the top of a class, with static variables at the very top.

```
-============
// Correct
public class Account
{
  public static string BankName;
  public static decimal Reserves;
```

```
  public string Number { get; set; }
  public DateTime DateOpened { get; set; }
  public DateTime DateClosed { get; set; }
  public decimal Balance { get; set; }
  // Constructor
  public Account()
  {
    // ...
  }
}
-============
```

15. Do use singular names for enums. Exception: bit field enums.

```
-============
// Correct
public enum Color
{
  Red,
  Green,
  Blue,
  Yellow,
  Magenta,
  Cyan
}
// Exception
[Flags]
public enum Dockings
{
  None = 0,
  Top = 1,
  Right = 2,
  Bottom = 4,
  Left = 8
}
-============
```

16. Do not explicitly specify a type of an enum or values of enums (except bit fields):

```
-============
// Don't
public enum Direction : long
{
  North = 1,
  East = 2,
  South = 3,
  West = 4
}
// Correct
public enum Direction
{
  North,
  East,
  South,
  West
}
-============
```

17. Do not use an "Enum" suffix in enum type names:

```
-============
// Don't
public enum CoinEnum
{
  Penny,
  Nickel,
  Dime,
  Quarter,
  Dollar
```

```
}
// Correct
public enum Coin
{
  Penny,
  Nickel,
  Dime,
  Quarter,
  Dollar
}
-============
```

18. Do not use "Flag" or "Flags" suffixes in enum type names:

```
-============
// Don't
[Flags]
public enum DockingsFlags
{
  None = 0,
  Top = 1,
  Right = 2,
  Bottom = 4,
  Left = 8
}
// Correct
[Flags]
public enum Dockings
{
  None = 0,
  Top = 1,
  Right = 2,
  Bottom = 4,
  Left = 8
}
-============
```

19. Do use suffix EventArgs at creation of the new classes comprising the information on event:

```
-============
// Correct
public class BarcodeReadEventArgs : System.EventArgs
{
}
-============
```

20. Do name event handlers (delegates used as types of events) with the "EventHandler" suffix, as shown in the following example:

```
-============
public delegate void ReadBarcodeEventHandler(object sender, ReadBarcodeEventArgs e);
-============
```

21. Do not create names of parameters in methods (or constructors) which differ only by the register:

```
-============
// Avoid
private void MyFunction(string name, string Name)
{
  //...
}
-============
```

22. Do use suffix Exception at creation of the new classes comprising the information on exception:

```
-============
// Correct
public class BarcodeReadException : System.Exception
{
}
-============
```

23. Do use prefix Any, Is, Have or similar keywords for boolean identifier:

```
-============
// Correct
public static bool IsNullOrEmpty(string value) {
    return (value == null || value.Length == 0);
}
-============
```

24. Use Named Arguments in method calls:

```
-============
// Method
public void DoSomething(string foo, int bar)
{
...
}

// Avoid
DoSomething("someString", 1);
// Correct
DoSomething(foo: "someString", bar: 1);
-============
```