# Coding Standards and Naming Convention

greenwich.edu.vn

UNIVERSITY of GREENWICH

Alliance with FPT Education

## Goal: Self-Documenting Code

- Self-documenting explains itself without need for external documentation, like flowcharts, UML diagrams, process-flow diagrams, etc.
  - Doesn’t imply we don’t like/use those documents!
- Coding conventions target:
  - How you write statements in the language, organize them into “modules,” format them in the source files
  - How you create names
  - How you write comments

## Coding Conventions Apply To...

- Comments, 3 types:
  - – File headers
  - – Function headers
  - – Explanations of variables and statements
- Names (chosen by programmer)
- Statements
  - – Organization: files, “modules,” nesting
  - – Format: spacing and alignment

## Naming Convention

| Object Name | Notation | Length | Plural | Prefix | Suffix | Abbreviation | Char Mask | Underscores |
|---|---|---|---|---|---|---|---|---|
| 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 |

## Naming convention

- Do use PascalCasing for class names and method names:

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

## Naming convention

- Do use camelCasing for method arguments and local variables:

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

## Naming convention

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

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

## Naming convention

- Do not use Screaming Caps for constants or readonly variables:

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

## Naming convention

- Use meaningful names for variables. The following example uses seattleCustomers for customers who are located in Seattle:

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

## Naming convention

- 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;
```

## Naming convention

- Do use PascalCasing or camelCasing (Depending on the identifier type) for abbreviations 3 characters or more (2 chars are both uppercase when PascalCasing is appropriate or inside the identifier).:

```
HtmlHelper htmlHelper;
FtpTransfer ftpTransfer, fastFtpTransfer;
UIControl uiControl, nextUIControl;
```

## Naming convention

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

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

## Naming convention

- Do organize namespaces with a clearly defined structure:

```
// Examples
namespace Company.Product.Module.SubModule
{
}
namespace Product.Module.Component
{
}
namespace Product.Layer.Module.Group
{
}
```

## Naming convention

- Do vertically align curly brackets:

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

## Naming convention

- 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()
  {
    // ...
  }
}
```

## Naming convention

- Do use singular names for enums.

```
// Correct
public enum Color
{
  Red,
  Green,
  Blue,
  Yellow,
  Magenta,
  Cyan
}
```

## Naming convention

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

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

## Naming convention

- 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);
}
```

## Naming convention

- Use Named Arguments in method calls

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

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

## Naming convention

- 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;
```

## Naming convention

- Do use predefined type names (C# aliases) like int, float, string for local, parameter and member declarations. Do use .NET Framework names like Int32, 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);
```

## Naming convention

- 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;
```

## Naming convention

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

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

## Naming convention

- 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
}
```

## Naming convention

- 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
}
```