# Code Format Enforcer

Use the provided custom .editorconfig file to enforce many of the code format and style guidelines from the IDesign Coding Standard.

Visual Studio (or Rider) follows the code formatting instruction in the .editorconfig file. Simply copy the .editorconfig file from the .zip file to the root of the solution, and restart the IDE.

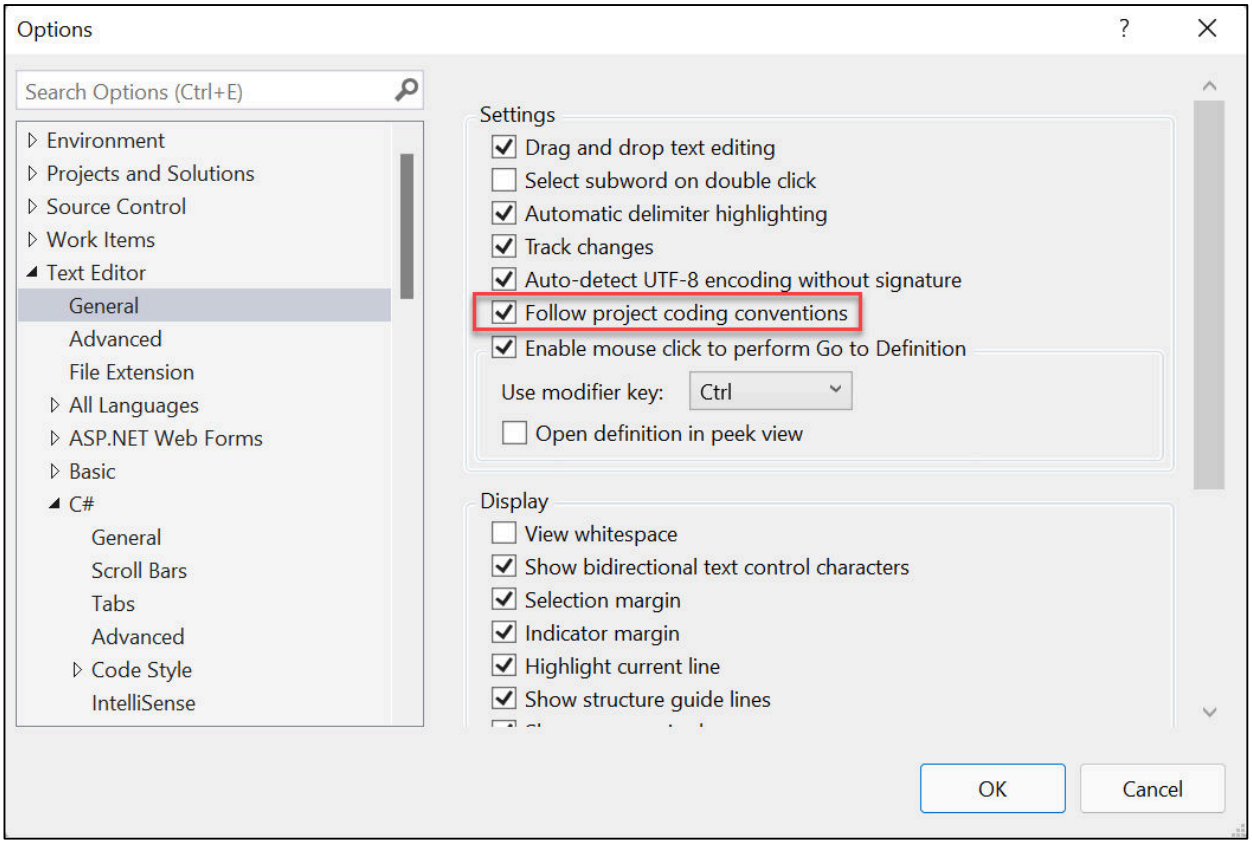
The settings in the .editorconfig file take precedence over global Visual Studio text editor settings. You can still set your own personal editor preferences in the Visual Studio Options dialog box. Those settings apply when working in a codebase without an .editorconfig file, or when the .editorconfig file doesn't override a particular setting.

With an editorconfig file in the project, new lines of code are formatted based on the editorconfig settings. The formatting of existing code is not changed unless you run one of the following commands:

- Code Cleanup (Ctrl+K, Ctrl+E), which applies any white-space settings, such as indent style, and selected code style settings.

- Edit > Advanced > Format Document (or Ctrl+K, Ctrl+D in the default profile), which only applies white-space settings.

Make sure to enable following the instructions in .editorconfig file by checking the "Follow project coding conventions" under Tools > Options > Text Editor > General:



The .editorconfig file enforces naming rules, symbol specifications, and naming styles.
Specifically, it enforces from the standard the following elements:

| Element | Comment |
|---|---|
| 1.1.1.1 | |
| 1.1.1.2 | |
| 1.1.1.3 | |
| 1.1.1.4 | |
| 1.1.1.5 | Only with public access modifier. |
| 1.1.1.6 | Only with public access modifier. |
| 1.1.1.12 | Only suppress suggestion to prefix with T. (CA1715, https://learn.microsoft.com/en-us/dotnet/fundamentals/code-analysis/quality-rules/ca1715) |
| 1.2.2 | |
| 1.2.3 | |
| 1.3.5 | |
| 1.3.8 | |
| 1.3.9 | |
| 1.3.10 | |
| 1.3.11 | |
| 1.3.15 | Only formatted correctly when using braces. |
| | |
| 1.4.1 | IDE provides confusing message but highlights the error correctly and only compiles if fixed. |
| 1.4.2 | IDE provides confusing message but highlights the error correctly and only compiles if fixed. |
| 2.1.8 | Set to en-us. |
| 2.2.1 | |
| 2.5.2 | |
| 2.5.3 | |
| | |
| 2.6.1 | No, turned off CA1810 to not discourage use of static constructor. |
| 2.7.3 | No, turn off on project level. |
| 2.7.16 | |
| 2.7.37 | |
| 2.7.39 | |

For the 4.* items, install the Threading Analyzer:
https://www.nuget.org/packages/microsoft.visualstudio.threading.analyzers

| | |
|---|---|
| 4.1 | |
| 4.5 | |
| 4.6 | |
| 4.8 | |
| 4.30 | |
| | |
| 6.1 | |

# IDesign Coding Standard Analyzers

Enforcing a few of the coding standard guidelines goes beyond the ability of the .editorconfig file. To enforce those guidelines, IDesign wrote custom analyzers.

The `IDesign.Analyzers.nupkg` NuGet package installs the following diagnostic analyzers. After installing, please restart the IDE and ensure the analyzers are loaded correctly.

| ID | Title | Severity |
|---|---|---|
| **IDESIGN100** | Group all framework namespaces together and put custom or third-party namespaces underneath, followed by solution namespaces. Order them alphabetically in a group. | Warning |
| **IDESIGN101** | Avoid `IEnumerable<T>` in service interfaces. Use arrays instead. | Warning |
| **IDESIGN102** | Avoid inline member initialization. | Warning |
| **IDESIGN103** | Avoid compound statements. | Warning |
| **IDESIGN104** | With automatic properties, place the get and the set under the property name. | Warning |

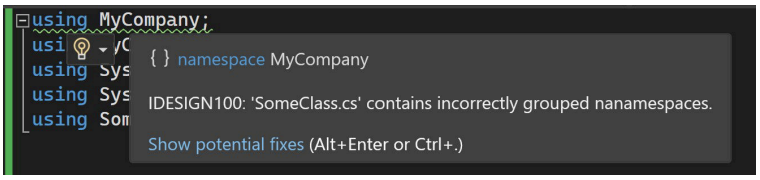
# IDESIGN100 Group namespaces

Group all framework namespaces together and put custom or third-party namespaces underneath, followed by project namespaces.

This analyzer enforces item **1.2.5** from the IDesign Coding Standard.

## Example the analyzer flags

```
using MyCompany;
using MyControls;
using System;
using System.Collections.Generic;
using SomeFramework;
```




## Solution

Sort usings to group all framework namespaces together and put custom or third-party namespaces underneath, followed by solution namespaces.

```
using System;
using System.Collections.Generic;
using SomeFramework;
using MyCompany;
using MyControls;
```

## Configuration

You can configure this analyzer. The analyzer looks for a file in the project root called `UsingAnalyzer.txt`. The file name is case sensitive.

Add the file to your project following the instructions below.

- In Visual Studio, right click project in Solution Explorer, and choose "Add -> New Items", then select "Text File" in "Add new item" dialog.
- Or, create the file at the location of your choice, then add the following text to your project/target file (replace file path with its actual location):

```
<ItemGroup>
  <AdditionalFiles Include="UsingAnalyzer.txt" />
</ItemGroup>
```

Each line of the config file must be a regular expression that can match the namespace to order. The analyzer orders the namespaces based on the line order in the configuration file, then alphabetically. If a namespace matches multiple lines, the analyzer places it as a member of the last group that matches. The analyzer puts any namespaces that do not match any of the regular expressions underneath the last matched namespace.

## Code fix

Analyzer IDESIGN100 comes with a code fix. The following example shows how to configure and use the code fix in Visual Studio.

Given the following `UsingAnalyzer.txt` and the following using block.

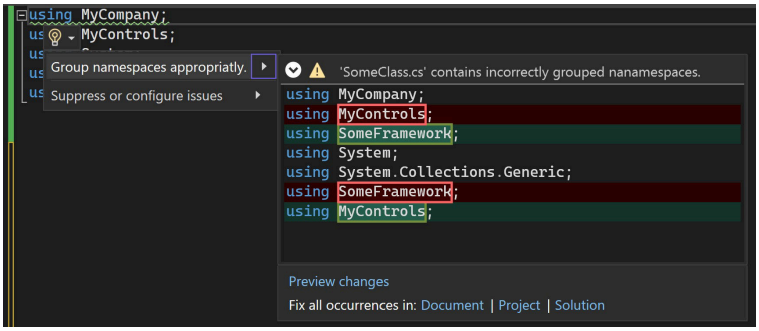
`UsingAnalyzer.txt`

```
MyCompany.*
SomeFramework.*
System.*
```

**Using block**

```
using MyCompany;
using SomeFramework;
using System;
using System.Collections.Generic;
using MyControls;
```

Visual Studio (or Rider) now suggests a fix for the violation. Before applying the fix, Visual Studio shows a preview of the changes. You can apply the fix for the current document, the entire project or the entire solution.



# IDESIGN101 Avoid `IEnumerable<T>` in interfaces. Use arrays instead.

Avoid the use of `IEnumerable<T>` in interfaces to prevent passing unmaterialized data.

This analyzer enforces item **2.7.18** from the IDesign Coding Standard.

## Example the analyzer flags


## Solution

Use an `Array`.

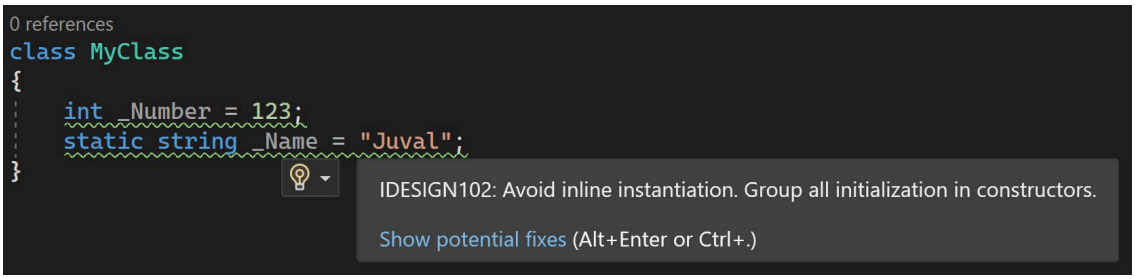
```
public interface IMyService
{
    string[] Load();
}
```

# IDESIGN102 Avoid inline member initialization

Avoid inline member initialization. Group all initialization in constructors.

This analyzer enforces item **2.6.2** from the IDesign Coding Standard.

## Example the analyzer flags




## Solution

Use constructor instead of inline instantiation.

```
class MyClass
{
    int _Number;
    static string _Name;

    public MyClass()
    {
        _Number = 123;
    }

    static MyClass()
    {
        _Name = "Juval";
    }
}
```

# IDESIGN103 Avoid compound statements

Avoid compound statements. This analyzer skips the use of `nameof` and skips Lambda statements.

This analyzer enforces item **2.7.32** from the IDesign Coding Standard.

## Example the analyzer flags


## Solution

Define variables to store temporary values.

```
//Avoid:
Bar(Foo());

//Correct:
int number = Foo();
Bar(number);

//Avoid:
return Foo();

//Correct:
int number = Foo();
return number;

//Avoid:
MyClass myClass = new MyClass(Foo());

//Correct:
int number = Foo();
MyClass myClass = new MyClass(number);
```

# IDESIGN104 With automatic properties, place the get and the set under the property name

---

With automatic properties, place the get and the set under the property name.

## Example the analyzer flags


## Solution

Format properties. Place the get and the set under the property name.

```
    public string Prop5
    { get; set; }
```

## Code fix

This analyzer has an associated Code Fix available. The code fix formats automatic properties, placing the accessors that are present and the curly braces on a new line and respects the whitespace. The following example shows how to use the code fix in Visual Studio.

**Before code fix**

```
    public string Prop1 { get; set; }

    public string Prop2 { get; private set; }
```

**After code fix**

```
    public string Prop1
    { get; set; }

    public string Prop2
    { get; private set; }
```

Visual Studio (or Rider) suggests a fix for the violation. Before applying the fix, Visual Studio shows a preview of the changes. You can apply the fix for the current document, the entire project or the entire solution.

# Project Settings Enforcer

You can enforce common project settings. Copy the .props and json files to the root of the repository. These files are:

- **Directory.Build.props** – enforces common project settings.
- **Directory.Package.props** – enforces central package management.
- **global.json** – enforces the same dotnet version across developer machines and build server.