# 

# C SHARP CODING CONVENTION FOR

# RIDE SHARING WITH GPS

# Version 1.0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Revision History** | | | | |
| **Date** | **Version** | **Description** | **Author** | **Approved** |
| Apr-8-2015 | 1.0 | First creation | Terry |  |
|  |  |  |  |  |
| **www.enclaveIT.com** | | | | |

|  |  |  |
| --- | --- | --- |
| Approver | Reviewer | Creator |
|  |  |  |
| Lieu (Lyn) T.B. TRAN | Hoang (Hanley) T. NGUYEN | Thanh (Terry) D. TRAN |

**Table of Contents**

[1. Introduction 4](#_Toc416304998)

[1.1 Purpose 4](#_Toc416304999)

[1.2 Application scope 4](#_Toc416305006)

[1.3 Related documents 4](#_Toc416305007)

[1.4 Definition 4](#_Toc416305008)

[2. General Rules 5](#_Toc416305009)

[2.1 Naming conventions 5](#_Toc416305010)

[2.2 Layout conventions 5](#_Toc416305011)

[2.3 Commenting conventions 5](#_Toc416305012)

[2.4 Languages guidelines 6](#_Toc416305013)

[2.4.1 String data type 6](#_Toc416305014)

[2.4.2 Implicitly Typed Local Variables 6](#_Toc416305015)

[2.4.3 Unsigned Data Type 8](#_Toc416305016)

[2.4.4 Arrays 8](#_Toc416305017)

[2.4.5 Delegates 9](#_Toc416305018)

[2.4.6 try-catch and using Statements in Exception Handling 9](#_Toc416305019)

[2.4.7 && and || Operators 10](#_Toc416305020)

[2.4.8 New Operator 11](#_Toc416305021)

[2.4.9 Event Handling 12](#_Toc416305022)

[2.4.10 Static Members 13](#_Toc416305023)

[2.4.11 LINQ Queries 13](#_Toc416305024)

# Introduction

## Purpose

## The C# Language Specification does not define a coding standard. However, the guidelines in this topic are used by Microsoft to develop samples and documentation.

## Coding conventions serve the following purposes:

## They create a consistent look to the code, so that readers can focus on content, not layout.

## They enable readers to understand the code more quickly by making assumptions based on previous experience.

## They facilitate copying, changing, and maintaining the code.

## They demonstrate C# best practices.

## Application scope

## Related documents

|  |  |  |
| --- | --- | --- |
| No. | Code | Name of documents |
| 1 | C# Coding Conventions | C# Programming Guide – MSDN.  <http://msdn.microsoft.com> |

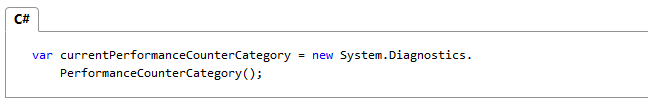
## Definition

|  |  |
| --- | --- |
| Terminology | Explanation |
|  |  |
|  |  |
|  |  |

# General Rules

## Naming conventions

In short examples that do not include using directives, use namespace qualifications. If you know that a namespace is imported by default in a project, you do not have to fully qualify the names from that namespace. Qualified names can be broken after a dot (.) if they are too long for a single line, as shown in the following example.



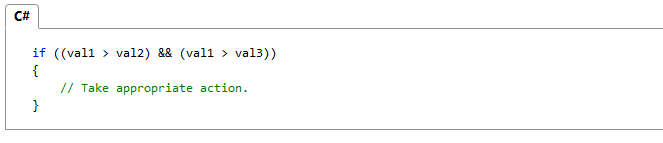
You do not have to change the names of objects that were created by using the Visual Studio designer tools to make them fit other guidelines.

## Layout conventions

Good layout uses formatting to emphasize the structure of your code and to make the code easier to read. Microsoft examples and samples conform to the following conventions:

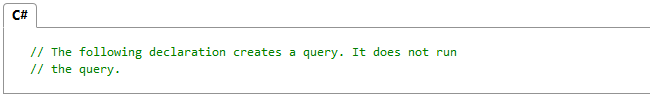
Use the default Code Editor settings (smart indenting, four-character indents, tabs saved as spaces). For more information, see Options, Text Editor, C#, Formatting.

* Write only one statement per line.
* Write only one declaration per line.
* If continuation lines are not indented automatically, indent them one tab stop (four spaces).
* Add at least one blank line between method definitions and property definitions.
* Use parentheses to make clauses in an expression apparent, as shown in the following code.



## Commenting conventions

* Place the comment on a separate line, not at the end of a line of code.
* Begin comment text with an uppercase letter.
* End comment text with a period.
* Insert one space between the comment delimiter (//) and the comment text, as shown in the following example.

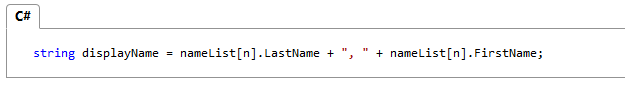


## Languages guidelines

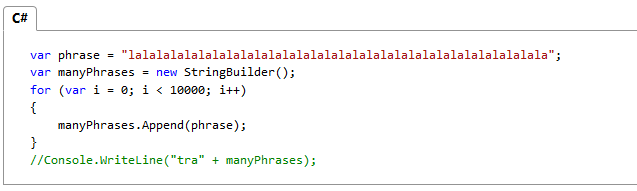
The following sections describe practices that the C# team follows to prepare code examples and samples.

### String data type

* Use the + operator to concatenate short strings, as shown in the following code.

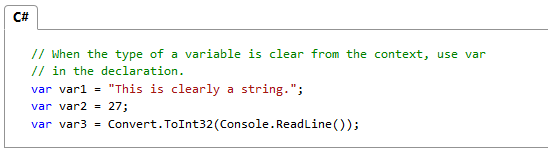


* To append strings in loops, especially when you are working with large amounts of text, use a **StringBuilder** object.

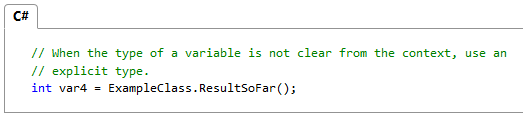


### Implicitly Typed Local Variables

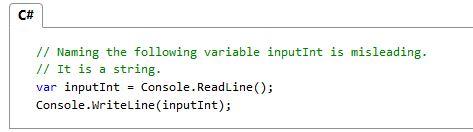
* Use implicit typing for local variables when the type of the variable is obvious from the right side of the assignment, or when the precise type is not important.



* Do not use **var** when the type is not apparent from the right side of the assignment.

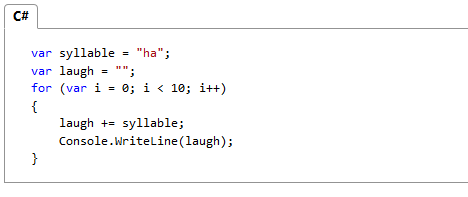


* Do not rely on the variable name to specify the type of the variable. It might not be correct.

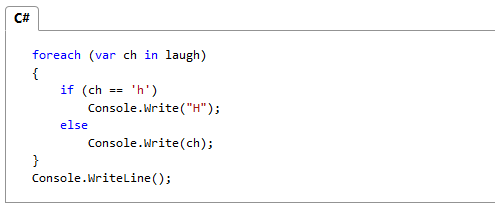


* Avoid the use of **var** in place of dynamic.
* Use implicit typing to determine the type of the loop variable in for and **foreach** loops.

The following example uses implicit typing in a **for** statement.



The following example uses implicit typing in a **foreach** statement.

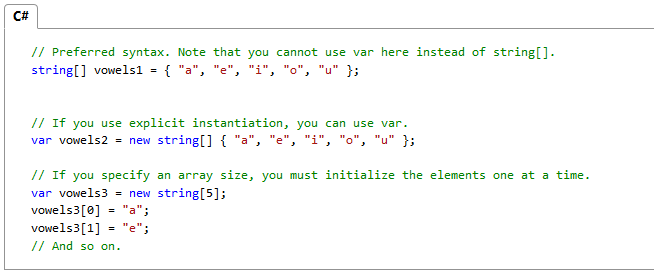


### Unsigned Data Type

In general, use **int** rather than unsigned types. The use of **int** is common throughout C#, and it is easier to interact with other libraries when you use **int**.

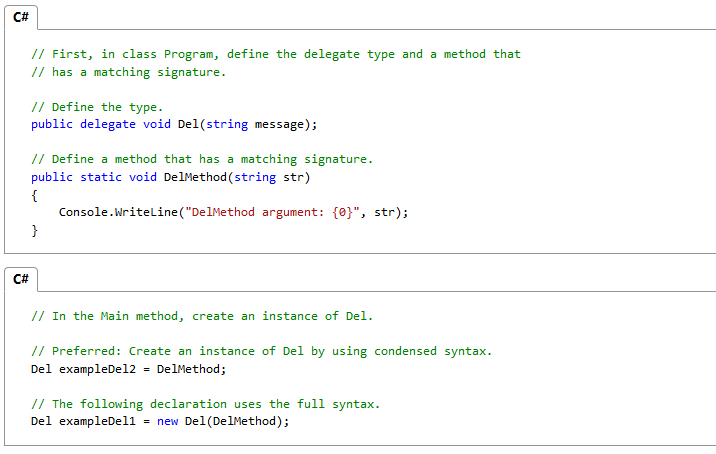
### Arrays

Use the concise syntax when you initialize arrays on the declaration line.



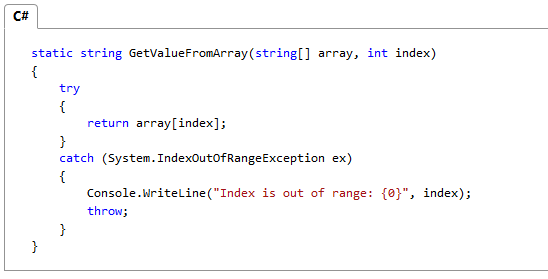
### Delegates

Use the concise syntax to create instances of a delegate type.

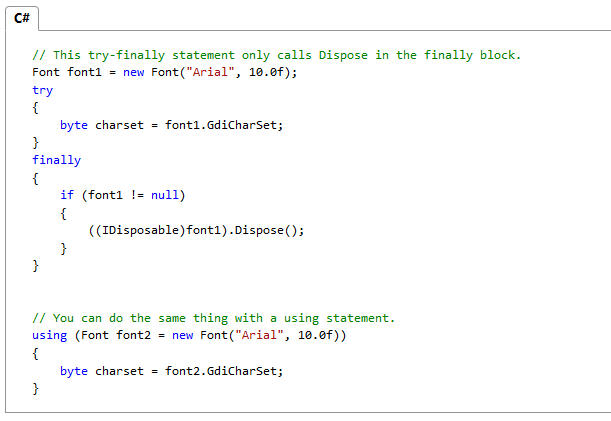


### try-catch and using Statements in Exception Handling

* Use a **try-catch** statement for most exception handling.

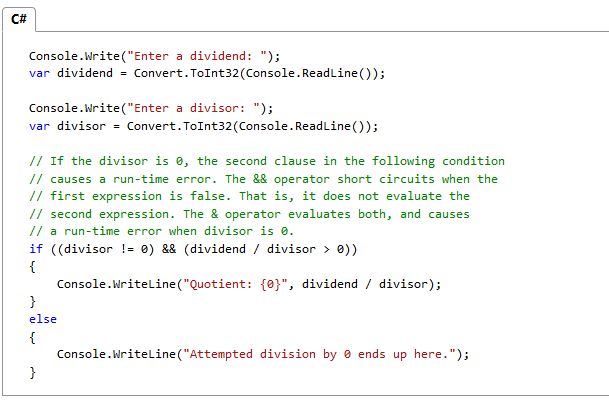


* Simplify your code by using the C# using statement. If you have a **try-finally** statement in which the only code in the **finally** block is a call to the **Dispose** method, use a **using** statement instead.



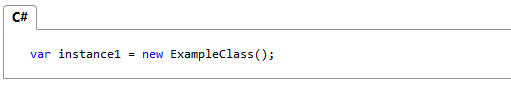
### && and || Operators

To avoid exceptions and increase performance by skipping unnecessary comparisons, use && instead of & and || instead of | when you perform comparisons, as shown in the following example.

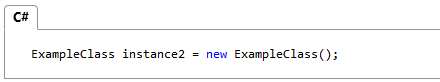


### New Operator

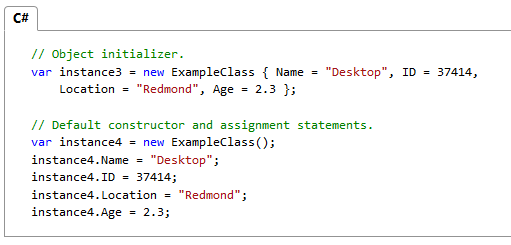
* Use the concise form of object instantiation, with implicit typing, as shown in the following declaration.



The previous line is equivalent to the following declaration.

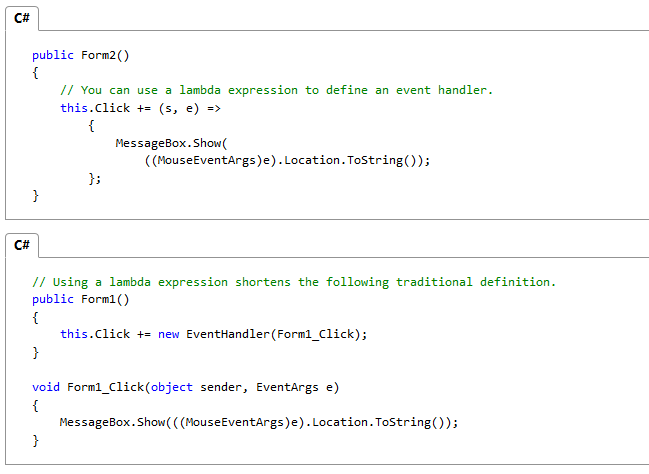


* Use object initializers to simplify object creation.



### Event Handling

If you are defining an event handler that you do not need to remove later, use a lambda expression.

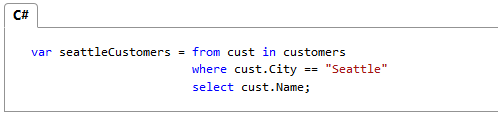


### Static Members

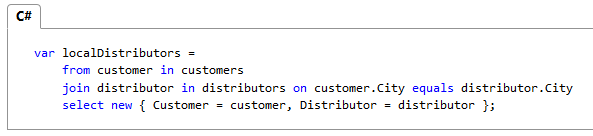
Call **static** members by using the class name: **ClassName.StaticMember**. This practice makes code more readable by making static access clear. Do not qualify a static member defined in a base class with the name of a derived class. While that code compiles, the code readability is misleading, and the code may break in the future if you add a static member with the same name to the derived class.

### LINQ Queries

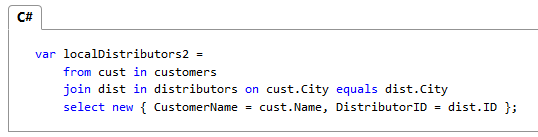
* Use meaningful names for query variables. The following example uses **seattleCustomers** for customers who are located in Seattle.



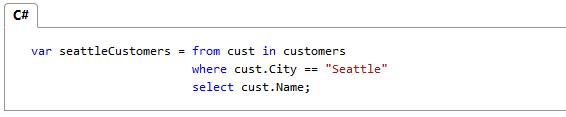
* Use aliases to make sure that property names of anonymous types are correctly capitalized, using Pascal casing.



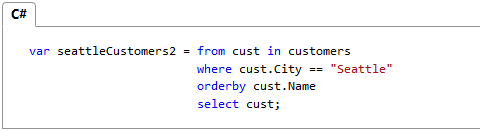
* Rename properties when the property names in the result would be ambiguous. For example, if your query returns a customer name and a **distributor ID**, instead of leaving them as **Name** and **ID** in the result, rename them to clarify that **Name** is the name of a customer, and **ID** is the **ID** of a distributor.



* Use implicit typing in the declaration of query variables and range variables.



* Align query clauses under the **from** clause, as shown in the previous examples.
* Use **where** clauses before other query clauses to ensure that later query clauses operate on the reduced, filtered set of data.



* Use multiple **from** clauses instead of a **join** clause to access inner collections. For example, a collection of Student objects might each contain a collection of test scores. When the following query is executed, it returns each score that is over 90, along with the last name of the student who received the score.

