docs.microsoft.com

Convenciones de código de C# (Guía de programación de C#)

dotnet-bot

17-22 minutos

- 19/07/2015
- Tiempo de lectura: 13 minutos
- Colaboradores

En este artículo

Las convenciones de codificación tienen los objetivos siguientes:Coding conventions serve the following purposes:

- Crean una apariencia coherente en el código, para que los lectores puedan centrarse en el contenido, no en el diseño.They create a consistent look to the code, so that readers can focus on content, not layout.
- Permiten a los lectores comprender el código más rápidamente al hacer suposiciones basadas en la experiencia anterior.They enable readers to understand the code more quickly by making

assumptions based on previous experience.

- Facilitan la copia, el cambio y el mantenimiento del código.They facilitate copying, changing, and maintaining the code.
- Muestran los procedimientos recomendados de C#.They demonstrate C# best practices.

  Microsoft usa las instrucciones de este tema para desarrollar ejemplos y documentación.The guidelines in this topic are used by Microsoft to develop samples and documentation.

Convenciones de nomenclaturaNaming Conventions

- En ejemplos breves que no incluyen directivas using, use calificaciones de espacio de nombres.In short examples that do not include using directives, use namespace qualifications. Si sabe que un espacio de nombres se importa en un proyecto de forma predeterminada, no es necesario completar los nombres de ese espacio de nombres.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. Los nombres completos pueden partirse después de un punto (.) si son demasiado largos para una sola línea, como se muestra en el ejemplo siguiente.Qualified names can be broken after a dot (.) if they are too long for a single line, as shown in the following example.

  ```
  var currentPerformanceCounterCategory = new
  System.Diagnostics.
      PerformanceCounterCategory();
  ```

- No es necesario cambiar los nombres de objetos que se crearon con las herramientas del diseñador de Visual Studio para que se ajusten a otras directrices.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.

Convenciones de diseñoLayout Conventions

Un buen diseño utiliza un formato que destaque la estructura del código y haga que el código sea más fácil de leer.Good layout uses formatting to emphasize the structure of your code and to make the code easier to read. Las muestras y ejemplos de Microsoft cumplen las convenciones siguientes:Microsoft examples and samples conform to the following conventions:

- Utilice la configuración del Editor de código predeterminada (sangría automática, sangrías de 4 caracteres, tabulaciones guardadas como espacios).Use the default Code Editor settings (smart indenting, four-character indents, tabs saved as spaces). Para obtener más información, vea Opciones, editor de texto, C#, formato.For more information, see Options, Text Editor, C#, Formatting.
- Escriba solo una instrucción por línea.Write only one statement per line.
- Escriba solo una declaración por línea.Write only one declaration per line.
- Si a las líneas de continuación no se les aplica sangría automáticamente, hágalo con una tabulación (cuatro espacios).If continuation lines are not indented automatically, indent them one tab stop (four spaces).
- Agregue al menos una línea en blanco entre las definiciones de método y las de propiedad.Add at least one blank line between method definitions and property definitions.
- Utilice paréntesis para que las cláusulas de una expresión sean evidentes, como se muestra en el código siguiente.Use parentheses to make clauses in an expression apparent, as shown in the following code.

```
if ((val1 > val2) && (val1 > val3))
{
```

```
        // Take appropriate action.
    }
```

- Coloque el comentario en una línea independiente, no al final de una línea de código.Place the comment on a separate line, not at the end of a line of code.
- Comience el texto del comentario con una letra mayúscula.Begin comment text with an uppercase letter.
- Finalice el texto del comentario con un punto.End comment text with a period.
- Inserte un espacio entre el delimitador de comentario (///) y el texto del comentario, como se muestra en el ejemplo siguiente.Insert one space between the comment delimiter (///) and the comment text, as shown in the following example.

```
// The following declaration creates a query. It
does not run
// the query.
```

- No cree bloques con formato de asteriscos alrededor de comentarios.Do not create formatted blocks of asterisks around comments.

Convenciones de lenguajeLanguage Guidelines

En las secciones siguientes se describen las prácticas que sigue el equipo C# para preparar las muestras y ejemplos de código.The following sections describe practices that the C# team follows to prepare code examples and samples.

String (Tipo de datos)String Data Type

- Use interpolación de cadenas para concatenar cadenas cortas, como se muestra en el código siguiente.Use string interpolation to concatenate short strings, as shown in the following code.

```
string displayName = $"{nameList[n].LastName},
```

```
{nameList[n].FirstName}";
```

- Para anexar cadenas en bucles, especialmente cuando se trabaja con grandes cantidades de texto, utilice un objeto StringBuilder.To append strings in loops, especially when you are working with large amounts of text, use a StringBuilder object.

```
var phrase =
"lalalalalalalalalalalalalalalalalalalalalalalalalalalalalala
var manyPhrases = new StringBuilder();
for (var i = 0; i < 10000; i++)
{
    manyPhrases.Append(phrase);
}
//Console.WriteLine("tra" + manyPhrases);
```

Variables locales con asignación implícita de tiposImplicitly Typed Local Variables

- Use tipos implícitos para las variables locales cuando el tipo de la variable sea obvio desde el lado derecho de la asignación, o cuando el tipo exacto no sea importante.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.

```
// When the type of a variable is clear from the
context, use var
// in the declaration.
var var1 = "This is clearly a string.";
var var2 = 27;
var var3 = Convert.ToInt32(Console.ReadLine());
```

- No use var cuando el tipo no sea evidente desde el lado derecho de la asignación.Do not use var when the type is not apparent from the right side of the assignment.

```
// When the type of a variable is not clear from
the context, use an
```

```
// explicit type.
int var4 = ExampleClass.ResultSoFar();
```

- No confíe en el nombre de variable para especificar el tipo de la variable.Do not rely on the variable name to specify the type of the variable. Puede no ser correcto.It might not be correct.

  ```
  // Naming the following variable inputInt is
  misleading.
  // It is a string.
  var inputInt = Console.ReadLine();
  Console.WriteLine(inputInt);
  ```

- Evite el uso de `var` en lugar de dynamic.Avoid the use of `var` in place of dynamic.

- Use tipos implícitos para determinar el tipo de la variable de bucle en bucles for y foreach.Use implicit typing to determine the type of the loop variable in for and foreach loops.

  En el ejemplo siguiente se usan tipos implícitos en una instrucción `for`.The following example uses implicit typing in a `for` statement.

  ```
  var syllable = "ha";
  var laugh = "";
  for (var i = 0; i < 10; i++)
  {
      laugh += syllable;
      Console.WriteLine(laugh);
  }
  ```

  En el ejemplo siguiente se usan tipos implícitos en una instrucción `foreach`.The following example uses implicit typing in a `foreach` statement.

  ```
  foreach (var ch in laugh)
  {
      if (ch == 'h')
          Console.Write("H");
  ```

```
    else
        Console.Write(ch);
}
Console.WriteLine();
```

Tipo de datos sin signoUnsigned Data Type

- En general, utilice `int` en lugar de tipos sin signo.In general, use `int` rather than unsigned types. El uso de `int` es común en todo C#, y es más fácil interactuar con otras bibliotecas cuando se usa `int`.The use of `int` is common throughout C#, and it is easier to interact with other libraries when you use `int`.

MatricesArrays

- Utilice sintaxis concisa para inicializar las matrices en la línea de declaración.Use the concise syntax when you initialize arrays on the declaration line.

```
// Preferred syntax. Note that you cannot use var
here instead of string[].
string[] vowels1 = { "a", "e", "i", "o", "u" };


// If you use explicit instantiation, you can use
var.
var vowels2 = new string[] { "a", "e", "i", "o",
"u" };

// If you specify an array size, you must
initialize the elements one at a time.
var vowels3 = new string[5];
vowels3[0] = "a";
vowels3[1] = "e";
// And so on.
```

DelegadosDelegates

- Utilice sintaxis concisa para crear instancias de un tipo de delegado.Use the concise syntax to create instances of a delegate type.

```
// First, in class Program, define the delegate
type and a method that
// has a matching signature.

// Define the type.
public delegate void Del(string message);

// Define a method that has a matching signature.
public static void DelMethod(string str)
{
    Console.WriteLine("DelMethod argument: {0}",
str);
}
// In the Main method, create an instance of Del.

// Preferred: Create an instance of Del by using
condensed syntax.
Del exampleDel2 = DelMethod;

// The following declaration uses the full syntax.
Del exampleDel1 = new Del(DelMethod);
```

Instrucciones try-catch y using en el control de excepcionestry-catch and using Statements in Exception Handling

- Use una instrucción try-catch en la mayoría de casos de control de excepciones.Use a try-catch statement for most exception handling.

```
static string GetValueFromArray(string[] array,
int index)
{
    try
    {
        return array[index];
    }
    catch (System.IndexOutOfRangeException ex)
    {
        Console.WriteLine("Index is out of range:
{0}", index);
        throw;
    }
}
```

- Simplifique el código mediante la instrucción using de C#.Simplify your code by using the C# using statement. Si tiene una instrucción try-finally en la que el único código del bloque `finally` es una llamada al método Dispose, use en su lugar una instrucción `using`.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.

```
// This try-finally statement only calls Dispose
in the finally block.
Font font1 = new Font("Arial", 10.0f);
try
{
    byte charset = font1.GdiCharSet;
}
finally
{
    if (font1 != null)
    {
        ((IDisposable)font1).Dispose();
```

```
    }
}
```

```
// You can do the same thing with a using
statement.
using (Font font2 = new Font("Arial", 10.0f))
{
    byte charset = font2.GdiCharSet;
}
```

Operadores && y ||&& and || Operators

- Para evitar excepciones y aumentar el rendimiento omitiendo las comparaciones innecesarias, use && en lugar de & y || en lugar de | cuando realice comparaciones, como se muestra en el ejemplo siguiente.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.

```
Console.Write("Enter a dividend: ");
var dividend =
Convert.ToInt32(Console.ReadLine());

Console.Write("Enter a divisor: ");
var divisor = Convert.ToInt32(Console.ReadLine());

// If the divisor is 0, the second clause in the
following condition
// causes a run-time error. The && operator short
circuits when the
// first expression is false. That is, it does not
evaluate the
// second expression. The & operator evaluates
```

```
both, and causes
// a run-time error when divisor is 0.
if ((divisor != 0) && (dividend / divisor > 0))
{
    Console.WriteLine("Quotient: {0}", dividend /
divisor);
}
else
{
    Console.WriteLine("Attempted division by 0
ends up here.");
}
```

New (Operador)New Operator

- Utilice la forma concisa de la creación de instancias de objeto con tipos implícitos, como se muestra en la siguiente declaración.Use the concise form of object instantiation, with implicit typing, as shown in the following declaration.

  ```
  var instance1 = new ExampleClass();
  ```

  La línea anterior es equivalente a la siguiente declaración.The previous line is equivalent to the following declaration.

  ```
  ExampleClass instance2 = new ExampleClass();
  ```

- Utilice inicializadores de objeto para simplificar la creación de objetos.Use object initializers to simplify object creation.

  ```
  // Object initializer.
  var instance3 = new ExampleClass { Name =
  "Desktop", ID = 37414,
      Location = "Redmond", Age = 2.3 };

  // Default constructor and assignment statements.
  var instance4 = new ExampleClass();
  instance4.Name = "Desktop";
  ```

```
instance4.ID = 37414;
instance4.Location = "Redmond";
instance4.Age = 2.3;
```

Control de eventosEvent Handling

- Si va a definir un controlador de eventos que no es necesario quitar más tarde, utilice una expresión lambda.If you are defining an event handler that you do not need to remove later, use a lambda expression.

```
public Form2()
{
    // You can use a lambda expression to define
an event handler.
    this.Click += (s, e) =>
        {
            MessageBox.Show(

((MouseEventArgs)e).Location.ToString());
        };
}

// Using a lambda expression shortens the
following traditional definition.
public Form1()
{
    this.Click += new EventHandler(Form1_Click);
}

void Form1_Click(object sender, EventArgs e)
{

MessageBox.Show(((MouseEventArgs)e).Location.ToString());
}
```

Miembros estáticosStatic Members

- Llame a miembros estáticos con el nombre de clase *ClassName.StaticMember*.Call static members by using the class name: *ClassName.StaticMember*. Esta práctica hace que el código sea más legible al clarificar el acceso estático.This practice makes code more readable by making static access clear. No califique un miembro estático definido en una clase base con el nombre de una clase derivada.Do not qualify a static member defined in a base class with the name of a derived class. Mientras el código se compila, su legibilidad se presta a confusión, y puede interrumpirse en el futuro si se agrega a un miembro estático con el mismo nombre a la clase derivada.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.

Consultas LINQLINQ Queries

- Utilice nombres descriptivos para las variables de consulta.Use meaningful names for query variables. En el ejemplo siguiente, se utiliza `seattleCustomers` para los clientes que se encuentran en Seattle.The following example uses `seattleCustomers` for customers who are located in Seattle.

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

- Utilice alias para asegurarse de que los nombres de propiedad de tipos anónimos se escriben correctamente con mayúscula o minúscula, usando para ello la grafía Pascal.Use aliases to make sure that property names of anonymous types are correctly capitalized, using Pascal casing.

```
var localDistributors =
    from customer in customers
```

```
    join distributor in distributors on
customer.City equals distributor.City
    select new { Customer = customer, Distributor
= distributor };
```

- Cambie el nombre de las propiedades cuando puedan ser ambiguos en el resultado.Rename properties when the property names in the result would be ambiguous. Por ejemplo, si la consulta devuelve un nombre de cliente y un identificador de distribuidor, en lugar de dejarlos como `Name` e `ID` en el resultado, cambie su nombre para aclarar que `Name` es el nombre de un cliente e `ID` es el identificador de un distribuidor.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.

```
var localDistributors2 =
    from cust in customers
    join dist in distributors on cust.City equals
dist.City
    select new { CustomerName = cust.Name,
DistributorID = dist.ID };
```

- Utilice tipos implícitos en la declaración de variables de consulta y variables de intervalo.Use implicit typing in the declaration of query variables and range variables.

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

- Alinee las cláusulas de consulta bajo la cláusula from, como se muestra en los ejemplos anteriores.Align query clauses under the from clause, as shown in the previous examples.
- Use cláusulas where antes de otras cláusulas de consulta para

asegurarse de que las cláusulas de consulta posteriores operan en un conjunto de datos reducido y filtrado.Use where clauses before other query clauses to ensure that later query clauses operate on the reduced, filtered set of data.

```
var seattleCustomers2 = from cust in customers
                        where cust.City ==
"Seattle"
                        orderby cust.Name
                        select cust;
```

- Use varias cláusulas `from` en lugar de una cláusula join para obtener acceso a colecciones internas.Use multiple `from` clauses instead of a join clause to access inner collections. Por ejemplo, una colección de objetos `Student` podría contener cada uno un conjunto de resultados de exámenes.For example, a collection of `Student` objects might each contain a collection of test scores. Cuando se ejecuta la siguiente consulta, devuelve cada resultado superior a 90, además del apellido del alumno que recibió la puntuación.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.

```
// Use a compound from to access the inner
sequence within each element.
var scoreQuery = from student in students
                 from score in student.Scores
                 where score > 90
                 select new { Last =
student.LastName, score };
```

SeguridadSecurity

Siga las instrucciones de Instrucciones de codificación segura.Follow the guidelines in Secure Coding Guidelines.

Vea tambiénSee Also

- Convenciones de código de Visual BasicVisual Basic Coding Conventions
- Instrucciones de codificación seguraSecure Coding Guidelines

Comentarios

Nos gustaría conocer su opinión. Elija el tipo que desea proporcionar:

Nuestro nuevo sistema de comentarios está basado en los problemas de GitHub. Lea sobre este cambio en nuestra entrada de blog.

Ahora mismo no hay comentarios de este documento. Los comentarios enviados aparecerán aquí.