Guía de estilos C#

Visual Studio 2012

Nomenclatura

Las convenciones generales de nomenclatura explican la elección de los nombres más adecuados para los elementos de sus bibliotecas. Estas instrucciones se aplican a todos los identificadores. Las secciones posteriores tratan la nomenclatura de elementos concretos, como espacios de nombres o propiedades.

Elección de palabra

Elija los nombres fácilmente legibles para los identificadores. Por ejemplo, una propiedad denominada HorizontalAlignment es más legible en inglés que AlignmentHorizontal.

Es preferible la legibilidad a la brevedad. El nombre de propiedad CanScrollHorizontally es mejor que ScrollableX (una referencia oculta al eje X).

No utilice guiones de subrayado, guiones ni ningún otro carácter no alfanumérico.

No utilice la notación húngara.

La notación húngara consiste en incluir un prefijo en los identificadores para codificar ciertos metadatos sobre el parámetro, como puede ser el tipo de datos del identificador.

Evite utilizar identificadores que están en conflicto con palabras clave de lenguajes de programación ampliamente utilizados.

Aunque los lenguajes conformes a CLS deben proporcionar una manera de utilizar palabras clave como palabras normales, los procedimientos recomendados indican que no debería obligar a los desarrolladores a saber cómo hacerlo. Para la mayoría de los lenguajes de programación, la documentación de la referencia del lenguaje contiene una lista de las palabras clave utilizada por los lenguajes. La tabla siguiente proporciona vínculos a la documentación de referencia de algunos lenguajes de programación ampliamente utilizados.

Abreviaturas y acrónimos

En general, no debería utilizar abreviaturas ni acrónimos. Estos elementos hacen que los nombres sean menos legibles. De igual forma, es difícil saber cuándo es seguro suponer que un acrónimo es ampliamente reconocido.

Para conocer las reglas de uso de mayúsculas para las abreviaturas, vea Normas referentes al uso de minúsculas y mayúsculas.

No utilice abreviaturas ni contracciones como parte de nombres de identificadores.

Por ejemplo, use OnButtonClick en lugar de OnBtnClick.

No utilice cualquier ningún acrónimo que no esté ampliamente aceptado y, además, sólo cuando necesario.

Estilos de grafía

Las condiciones siguientes describen distintas maneras de usar mayúsculas y minúsculas de los identificadores.

Grafía Pascal

La primera letra del identificador y la primera letra de las siguientes palabras concatenadas están en mayúsculas. El estilo de mayúsculas y minúsculas Pascal se puede utilizar en identificadores de tres o más caracteres. Por ejemplo:

BackColor

Grafía Camel
La primera letra del identificador está en minúscula y la primera letra de las siguientes palabras concatenadas en mayúscula. Por ejemplo:

backColor

Mayúsculas
Todas las letras del identificador van en mayúsculas. Por ejemplo:

IO

Reglas de uso de mayúsculas y minúsculas para los identificadores
Cuando un identificador está compuesto de varias palabras, no utilice separadores, como guiones de subrayado ("_") ni guiones ("-"), entre las palabras. En su lugar, utilice la grafía para señalar el principio de cada palabra.

Las instrucciones siguientes proporcionan reglas generales para los identificadores.

Utilice la grafía Pascal para todos los nombres de miembros públicos, tipos y espacios de nombres que constan de varias palabras.

Tenga en cuenta que esta regla no se aplica a las instancias de campos. Por los motivos que se detallan en Instrucciones de diseño de miembros, no debería utilizar campos de instancia públicos.

Utilice el uso combinado de mayúsculas y minúsculas tipo Camel para los nombres de parámetros.

En la tabla siguiente se resumen las reglas de uso de mayúsculas y minúsculas para los identificadores y se proporcionan ejemplos de los diferentes tipos de identificadores.

identificador	Case	Ejemplo
Clase	Pascal	**AppDomain**
Tipo de enumeración	Pascal	**ErrorLevel**
Valores de enumeración	Pascal	**FatalError**
Event	Pascal	**ValueChanged**
Clase de excepción	Pascal	**WebException**
Campo estático de sólo lectura	Pascal	**RedValue**
Interface	Pascal	**IDisposable**
Método	Pascal	**ToString**
namespace	Pascal	**System.Drawing**
Parámetro	Camel	**typeName**
Propiedad	Pascal	**BackColor**

Reglas de uso de mayúsculas para los acrónimos

Un acrónimo es una palabra formada a partir de las letras de las palabras de un término o frase. Por ejemplo, HTML es un acrónimo de Hypertext Markup Language (lenguaje de marcado de hipertexto). Sólo debería incluir acrónimos en identificadores cuando son muy conocidos y están bien entendidos. Los acrónimos se diferencian de las abreviaturas en que una abreviatura acorta una sola palabra. Por ejemplo, ID es una abreviatura de identifier. En general, los nombres de biblioteca no deberían utilizar las abreviaturas.

Nota
Las dos abreviaturas que se pueden utilizar en los identificadores son ID y OK. En identificadores con grafía Pascal, deberían aparecer como Id y Ok. Si se utilizan como la primera palabra de un identificador con grafía tipo Camel, deberían aparecer como id y ok, respectivamente.

La grafía de los acrónimos depende de la longitud del acrónimo. Todos los acrónimos tienen al menos dos caracteres. A efectos de estas instrucciones, si un acrónimo tiene exactamente dos caracteres, se considera un acrónimo corto. Un acrónimo de tres o más caracteres es un acrónimo largo.

Las instrucciones siguientes especifican la grafía apropiada para los acrónimos cortos y los largos. Las reglas de grafía de identificadores tienen prioridad sobre las reglas de grafía de acrónimos.

Ponga en mayúsculas ambos caracteres de los acrónimos de dos caracteres, excepto cuando se trata de la primera palabra de un identificador con grafía tipo Camel.

Una propiedad denominada DBRate es un ejemplo de un acrónimo corto (DB) utilizado como la primera palabra de un identificador con grafía Pascal. Un parámetro denominado ioChannel es un ejemplo de un acrónimo corto (IO) utilizado como la primera palabra de un identificador con grafía Camel.

Ponga en mayúsculas sólo el primer carácter de los acrónimos con tres o más caracteres, excepto si es la primera palabra de un identificador con grafía Camel.

Una clase denominada XmlWriter es un ejemplo de un acrónimo largo usado como primera palabra de un identificador con grafía Pascal. Un parámetro denominado htmlReader es un ejemplo de un acrónimo largo utilizado como la primera palabra de un identificador con grafía Camel.

No ponga en mayúsculas ninguno de los caracteres de los acrónimos, independientemente de su longitud, al principio de un identificador con grafía Camel.

Un parámetro denominado xmlStream es un ejemplo de un acrónimo largo (xml) utilizado como la primera palabra de un identificador con grafía Camel. Un parámetro denominado dbServerName es un ejemplo de un acrónimo corto (db) usado como primera palabra de un identificador con grafía Camel.

Reglas de uso de mayúsculas para palabras compuestas y términos comunes

No ponga en mayúsculas todas las palabras en las denominadas palabras compuestas con formato cerrado. Éstas son palabras compuestas escritas como una sola palabra, como por ejemplo "endpoint".

Por ejemplo, hashtable es una palabra compuesta con formato cerrado que se debería tratar como una sola palabra y se debería utilizar la grafía correspondiente. En la grafía Pascal, es Hashtable; mientras que en la grafía Camel, es hashtable. Para determinar si una palabra es una palabra compuesta con formato cerrado, compruebe un diccionario actualizado.

La lista siguiente identifica algunos términos comunes que no son palabras compuestas con formato cerrado. La palabra se muestra en la grafía Pascal seguida por el formato de grafía Camel entre paréntesis.

- BitFlag (bitFlag)

- FileName (fileName)
- LogOff (logOff)
- LogOn (logOn)
- SignIn (signIn)
- SignOut (signOut)
- UserName (userName)
- WhiteSpace (whiteSpace)

Distinguir mayúsculas de minúsculas

Las instrucciones del uso de mayúsculas solamente se utilizan para facilitar la lectura y el reconocimiento de los identificadores. La grafía no se puede utilizar como medio para evitar que se produzcan colisiones entre los elementos de biblioteca.

No dé por supuesto que todos los lenguajes de programación distinguen entre mayúsculas y minúsculas. No es así. Los nombres no se pueden diferenciar exclusivamente por su grafía.

Convenciones de Código

En la Especificación del lenguaje C# no se define un estándar de codificación. Sin embargo, Microsoft usa las directrices que se describen en este tema para desarrollar los ejemplos y la documentación.

Las convenciones de codificación tienen la finalidad siguiente:

- Dan un aspecto coherente al código, de modo que los lectores pueden centrar su atención en el contenido y no en el diseño.
- Permiten a los lectores entender el código con más rapidez, ya que pueden hacer suposiciones en función de su experiencia anterior.
- Facilitan la copia, modificación y mantenimiento del código.
- Muestran los procedimientos recomendados de C#.

Convenciones de nomenclatura

- En los ejemplos cortos que no incluyan directivas using, use las calificaciones de espacios de nombres. Si sabe que, de manera predeterminada, se importa un espacio de nombres en un proyecto, no es necesario completar los nombres de ese espacio de nombres. Los nombres calificados se pueden interrumpir después de un punto (.) si son demasiado largos para una sola línea, como se muestra en el ejemplo siguiente.

  C#

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

- No tiene que cambiar los nombres de los objetos creados mediante las herramientas de diseño de Visual Studio para que

quepan otras instrucciones.

Convenciones de diseño

En un buen diseño, el formato se usa para hacer hincapié en la estructura del código y facilitar su lectura. Los ejemplos y muestras de Microsoft se ajustan a las convenciones siguientes:

- Se usa la configuración predeterminada del editor de código (sangría automática, sangrías de cuatro caracteres, tabuladores guardados como espacios). Para obtener más información, vea Opciones, editor de texto, C#, formato.
- Se escribe únicamente una instrucción por línea.
- Se escribe solamente una declaración por línea.
- Si la sangría no se aplica automáticamente a las líneas de continuación, se aplica una tabulación (cuatro espacios).
- Se agrega al menos una línea en blanco entre las definiciones de método y las definiciones de propiedad.
- Se usan paréntesis para dotar a las cláusulas de un formato de expresión, tal y como se muestra en el código siguiente.

C#

```
if ((val1 > val2) && (val1 > val3))
{
    // Take appropriate action.
}
```

Convenciones de los comentarios

- El comentario se sitúa en una línea independiente, no al final de una línea de código.
- El texto del comentario comienza con una letra mayúscula.
- El texto del comentario finaliza con un punto.
- Entre el delimitador del comentario (/) y el texto del comentario se inserta un espacio, tal y como se muestra en el ejemplo siguiente.

C#

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

- No se crean bloques de asteriscos con formato alrededor de los comentarios.

Comentario de documentación de Métodos

Escriba /// seguido de cualquier cadena de texto o etiqueta XML. Si especifica /// en la línea antes de la definición, el editor crea

una plantilla de un comentario de documentación y rellena los parámetros y otra información.

```
///<summary>
///summary description
///</summary>
///<remarks>
///This is a test.
///</remarks>
```

Convenciones de lenguaje

En las siguientes secciones se describen los procedimientos que sigue el equipo de C# para preparar ejemplos y las muestras de código.

String (Tipo de datos)

- El operador + se usa para concatenar cadenas cortas, tal y como se muestra en el código siguiente.

C#

```
string displayName = nameList[n].LastName + ", " + nameList[n].FirstName;
```

- Para anexar cadenas en bucles, sobre todo cuando se trabaja con grandes cantidades de texto, se usa un objeto StringBuilder.

C#

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

Variables locales tipificadas implícitamente

- La tipificación implícita se usa en las variables locales cuando el tipo de la variable resulta obvio desde lado derecho de la asignación o cuando el tipo exacto no es importante.

C#

```
// 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 se usa var cuando el tipo no resulta evidente desde el lado derecho de la asignación.

C#

```
// When the type of a variable is not clear from the context, use an
// explicit type.
int var4 = ExampleClass.ResultSoFar();
```

- No se toma como base el nombre de la variable para especificar el tipo de variable. Puede no ser correcto.

C#

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

- Conviene evitar el uso de **var** en lugar de dynamic.
- Se usan tipos implícitos para determinar el tipo de la variable de bucle en los bucles for y foreach.

  En el ejemplo siguiente se usa la tipificación implícita en una instrucción **for**.

C#

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

  En el ejemplo siguiente se usa la tipificación implícita en una instrucción **foreach**.

C#

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

Tipo de datos sin signo

- En general, conviene usar **int** en lugar de tipos sin signo. El uso de **int** es común en C#, y es más fácil interactuar con otras bibliotecas cuando se usa **int**.

Matrices

- Conviene usar la sintaxis concisa al inicializar matrices en la línea de declaración.

C#

```
// 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.
```

Delegados

- Conviene usar la sintaxis concisa para crear instancias de un tipo de delegado.

C#

```
// 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);
}
```

C#

```
// 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 excepciones

- Se usa una instrucción try-catch para controlar la mayor parte de las excepciones.

C#

```
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;
    }
}
```

- Puede simplificar el código usando la instrucción using de C#. Si tiene una instrucción try-finally en la que solo el código del bloque **finally** es una llamada al método Dispose, use en su lugar una instrucción **using**.

C#

```
// 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 ||

- Para evitar excepciones y mejorar el rendimiento omitiendo comparaciones innecesarias, use && en lugar de & y || en lugar de |

cuando realice comparaciones, tal y como se muestra en el ejemplo siguiente.

C#

```
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)

- Utilice el formato conciso de la instanciación de objetos, con tipificación implícita, tal y como se muestra en la siguiente declaración.

C#

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

La línea anterior es equivalente a la siguiente declaración.

C#

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

- Use inicializadores de objetos para simplificar la creación de objetos.

C#

```
// 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 eventos

- Si va a definir un controlador de eventos que no tiene que quitar más adelante, use una expresión lambda.

C#

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

C#

```
// 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 estáticos

- Llame a miembros static usando el nombre de clase: *NombreClase.MiembroEstático*. No acceda a un miembro estático definido en una clase base de una clase derivada.

Consultas LINQ

- Use nombres descriptivos para las variables de consulta. En el ejemplo siguiente se usa `seattleCustomers` para los clientes que se encuentran en Seattle.

C#

```
var seattleCustomers = from cust in customers
```

```
                        where cust.City == "Seattle"
                        select cust.Name;
```

- Use alias para asegurarse de que los nombres de propiedades de tipos anónimos se escriben con las mayúsculas correctas mediante la grafía Pascal:

C#

```
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 los nombres de propiedad en el resultado puedan ser ambiguos. Por ejemplo, si la consulta devuelve un nombre de cliente y un identificador del distribuidor, en lugar de dejarlos como `Name` y `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.

C#

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

- Use la tipificación implícita en la declaración de variables de consulta y variables de rango:

C#

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

- Alinee las cláusulas de consulta bajo la cláusula from, tal y como se muestra en los ejemplos anteriores.

- Use las cláusulas where antes que otras cláusulas de consulta para asegurarse de que las cláusulas de consulta posteriores operan en el conjunto de datos reducido y filtrado:

C#

```
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 las colecciones internas. Por ejemplo, cada objeto

`Student` de una colección podría contener una colección de puntuaciones de exámenes. Cuando se ejecuta la consulta siguiente, devuelve cada una de las puntuaciones que es superior a 90 junto con el apellido del estudiante que recibió la puntuación.

C#

```
// 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 };
```