Some C# style preferences

Michael A. Covington **2012 May 4**

Unlike syntax rules, **style guidelines are made to be broken.** Their purpose is to make programs easier to read and understand. Never hesitate to break them when doing something else will foster readability.

The following are a few preferences that I am adopting experimentally, mainly to settle points on which I had no firm habit. For the most part, I follow Microsoft's coding guidelines. This document is to be viewed as a supplement. **It is not a description of what I have always done or what anyone else must do.** Some other useful guidelines (which do not all agree with each other) are at http://www.amazedsaint.com/2010/11/top-6-coding-standards-guideline.html. I particularly like the Encodo C# Handbook.

Layout

I use Visual Studio's default setting (Tab = 4 spaces) and prefer to convert all tabs to spaces upon saving (though if the program is only edited with Visual Studio, that is a moot point).

I avoid aligning with anything other than the first nonblank character in a preceding line. That is:

```
Abcdef(def, ghi,
       jkl, mno, pqr);   // not this – attempting exact alignment

Abcdef(def, ghi,
    jkl, mno, pqr);      // but this – just indent one unit
```

The reason is that I am experimenting with viewing C# programs in proportional-pitch type. This often makes them much more readable, but it eliminates alignments that depend on matching the width of characters. (It also precludes aligning end-of-line comments, which is why I am not entirely comfortable with it.) On average, proportional type is much narrower than fixed-width type.

Braces

I use K&R brace style in C, not C#, to help me distinguish the two languages. In C#, every **{** and **}** is on a line by itself:

```
if (x < y)
{
    DoSomething();
}
else
{
    DoSomethingElse();
}
```

The only exception is when the material between the braces is very short and fits on one line:

```
if (x < y) { DoSomething(); }

set { _abcdef = value; }
get { return _abcdef; }
```

Even in these cases I retain the braces. (This is not something I've done consistently; it is a new practice I'm adopting.)

Expressions

Use parentheses generously; do not try to remember a lot of operator precedences. If you can remember all the operator precedences, you know too few programming languages!

Break expressions after an operator, not before:

```
x = a + b + c + d + e + f +
   g + h + i + j + k;

if ((x1 > 35 || x > 25)  &&
  (y1 > 35 || y > 25)  &&  z > 25)
{
    DoSomething();
}
```

Extra spaces around **&&** and **||** can contribute to readability.

Parentheses

When the material between parentheses is long, the parentheses can and should be put on lines by themselves just like braces. This applies to complicated "if" expressions (common in AI work) and also, in a slightly different way, to parameter declarations:

```
if
(
    (x1 > 35 || x > 25)  &&
    (y1 > 35 || y > 25)  &&
    z > 25
)
{
    DoSomething();
}
```

However, never separate the '**(**' that introduces an argument list from the name of the method; this helps make it obvious that it is a method name (and is a syntactic requirement in some languages though not in C#):

```
string MyMethod(                    // preferred layout for a long arg list
    int abc,
    char def,
    string ghi,
    SomeOtherType jkl
)
{
   …
}
```

Conciseness

Use **`var`** to avoid repeating a type declaration, especially a long one:

```
StringBuilder sb = new StringBuilder();    // not this
var sb = new StringBuilder();              // but this
```

Naming

I use **`PascalCase`** (first letter of each word capitalized) for all names except as follows.

Microsoft recommends **`camelCase`** (first letter lowercase, subsequent word-initial letters capitalized) for parameters and for local variables. I prefer to use **`lowercase`** since it is more visibly distinct from PascalCase. I also prefer to make this a moot point by choosing short, single-word names for parameters and local variables.

I prepend an underscore mark (`_`) to names of private fields that correspond to properties. This avoids having two names that differ only in capitalization. I also begin names with an underscore in the case of some temporary names associated with testing and debugging.

Taboos

`goto` statements are permitted whenever they make the algorithm easier to understand. (Knuth published some well-known examples in the 1970s.) The C# language specification recommends **`goto`** as the way to exit from more than one loop at a time, since there is no labeled **`break`** in C#.

Public fields are permitted. Properties are only advantageous if (1) some form of data validation is performed, or (2) the field may be changed to a property later and your code is going into a DLL that may be changed without re-compiling the calling program. Like almost all other names, names of public fields are in PascalCase.