

# Differences with Csharp

This page notes some differences between C# and boo.

## C# to Boo Converter

The easiest way to see how code in boo differs from C# is to use the C# to boo converter included in the [Boo AddIn For SharpDevelop](#).

The converter is also accessible online via a webform here: <http://developer.sharpdevelop.net/codeconvert.net/>

While viewing a C# file in SharpDevelop, you can select from the menu: Tools -> Convert C# to boo. Or you can convert an entire C# project at once by right-clicking on the project icon and selecting: Convert -> C# to boo. Note, however, the converter is still new and in development.

The converter will convert C# to legal boo code, but it will not show you the simplest way to write that code. For example, instead of "System.Console.WriteLine  ", in boo you can simply say "print x". And many times the type will be declared "x as int" when it is unnecessary since the type can be inferred by the boo compiler: "x = 3" or "m = MyClass()".

## C# vs. Boo Syntax Examples

Here is a table noting some differences between C# and boo, based on this [table](#) from the [Nemerle](#) programming language site. See also the [C# language specification](#).

### Expressions

C#	Boo	Remarks
<pre>int x = 3; string y = "foo"; FooBarQux fbq = make_fbq ();</pre>	<pre>x = 3 y = "foo" fbq = make_fbq()</pre>	Types are inferred, no semicolons needed
<pre>expr_1 = expr_2 = expr_3;</pre>	<pre>expr_1 = expr_2 = expr_3</pre>	
<pre>Class c = new Class (params);</pre>	<pre>c = Class(params)</pre>	No "new" keyword required

<pre>Class[] c = new Class [size]</pre>	<pre>c = array(Class, size)</pre>	See <a href="#">Lists and Arrays</a>
<pre>GenericClass&lt;int&gt; c = new GenericClass&lt;int&gt; ();</pre>	<pre>c = GenericClass[of int]()</pre>	See <a href="#">Generics</a>
<pre>Type[] l = new Type[] { expr_1, expr_1, ..., expr_n }</pre>	<pre>l = (expr_1, expr_1, ..., expr_n)</pre>	Types are inferred when you use arrays
<pre>if (cond) return foo; do_something (); return bar;</pre>	<pre>if cond:     return foo //You can also say: return foo if cond do_something() return bar</pre>	
<pre>if (cond) answer = 42;</pre>	<pre>if cond:     answer = 42 //or answer = 42 if cond</pre>	
<pre>if (!cond) answer = 42;</pre>	<pre>answer = 42 if not cond</pre>	

```
try ...
catch
(FooException e)
{ ... }
catch
(BarException e)
{ ... }
```

```
try:
...
except e as
FooException:
...
except e as
BarException:
...
```

```
try { foo (); bar
(); }
catch (Exception
e) { baz (); }
finally { qux ();
}
```

```
try:
foo()
bar()
except e:
baz()
ensure:
qux()
```

```
throw new
System.ArgumentEx
ception ("foo");
```

```
raise
System.ArgumentEx
ception("foo")
```

Use "raise" to generate an exception.

```
type t = ((type)
expr)
```

```
t = expr cast
type
//or
//null if cast
fails:
t = expr as type
```

See [Casting Types](#)

```
using
System.Windows.Fo
rms;
Button button =
control as
Button;
if (button !=
null) ...
else ...
```

```
import
System.Windows.Fo
rms
button = control
as Button
if button !=
null:
...
else:
...
```

<pre>using System; using SWF = System.Windows.Fo rms; using System.Xml; ... Console.WriteLine ("foo"); SWF.Form x = new SWF.Form (); XmlDocument doc = new XmlDocument ();</pre>	<pre>import System import System.Windows.Fo rms as SWF import System.Xml  print "foo" x = SWF.Form() doc = XmlDocument()</pre>	<p>You can create aliases for namespaces. Also use <code>print</code> instead of <code>console.WriteLine</code>.</p>
<pre>x++; ++x;</pre>	<pre>x++ ++x</pre>	
<pre>readonly int X = 2; const int Y = 3;</pre>	<pre>final X = 2 static final Y = 3</pre>	<p>Read-only and constant fields.</p>

## Type definitions

C#	Boo	Remarks
<pre>static int foo (int x, string y) { ... }</pre>	<pre>static def foo(x as int, y as string) as int: ... </pre>	<p>Use "def" for methods and functions. Types are declared with the "as" keyword like VB.</p>
<pre>class Foo : Bar { public Foo (int x) : base (x) { ... } }</pre>	<pre>class Foo (Bar): def constructor(x as int): super(x) ... </pre>	<p>Illustrates a class with a superclass and a constructor</p>

<pre>class Foo {   int x; }</pre>	<pre>class Foo:   x as int</pre>	<p>a class with a field</p>
<pre>class Foo {   readonly int x; }</pre>	<pre>class Foo:   final x as   int</pre>	<p>or else a "const" keyword may be added to boo</p>
<pre>class C : I1, I2 {   void I1.m () {   ... }   void I2.m () {   ... } }</pre>	<pre>class C (I1, I2):   def I1.m():   ...   def I2.m():   ...</pre>	<p>Implementing two interfaces that have the same named method.</p>
<pre>using System.Runtime.Co mpilerServices; class C { public object this [int i] { ... } [CSharp.IndexerNa me("MyItem")] public int this [string name] { ... } }</pre>	<pre>import System.Reflection  [DefaultMember("M yItem")] class A:   private _val = (1,2,3,4)    MyItem(index as long) as int:   get:   return _val[index]   set:   _val[index] = value  a = A() print a[2] a[2] = 10 print a[2]</pre>	<p>Use the DefaultMember attribute.</p>

<pre>char c = 'a';</pre>	<pre>c = char('a')</pre>	char() builtin
<pre>float v = 1.0f;</pre>	<pre>f = 1.0f //or f as single = 1.0</pre>	Uses "single" instead of "float"

## Miscellaneous Differences and Similarities

C#	Boo	Remarks
<pre>// A comment. /* A possibly multiline comment. */</pre>	<pre>// A comment. /* A possibly multiline comment. */ # Another comment</pre>	C++ style commenting plus Python # comments.
<pre>@"foo\bar"</pre>	<pre>"""foo\bar"""</pre>	Quoted string literals. Triple-quoted strings can span multiple lines, too.

## Other Notes on Differences Between C# and Boo

These are some early notes from looking at the [C# language specification](#).

### Assigning multiple variables at once

In C#, you can assign a value to two variables at once like so:

```
int a, b = 1
```

In boo, however, "a, b" refers to a sequence. So you would instead use:

```
a = b = 1
```

Note also declaring the type is unnecessary because of boo's [Type Inference](#).

Use "a, b" when unpacking multiple values. For example:

```
name = "First Last"  
firstname, lastname = @/ /.Split(name)  
print firstname, lastname
```

Also note in boo you can use the "print" statement instead of System.Console.WriteLine.

### **Buffer overflow checking**

You can turn off overflow checking in boo like so:

```

try:
    checked:
        i = 100000
        i += 1000000000
        i += 1000000000
        i += 1000000000
except:
    print "did overflow i"

unchecked:
    j = 100000
    j += 1000000000
    j += 1000000000
    j += 1000000000
    print "didn't overflow j: $j"

```

Variable number of parameters.

```

static void F(params int[] args) {
    Console.WriteLine("# of arguments:
{0}", args.Length);
    for (int i = 0; i < args.Length; i++)
        Console.WriteLine("\targs[{0}] = {1}",
i, args[i]);
}

```

Boo uses syntax similar to python:

```
def F(*args as (int)):  
    print "# of arguments: ${len(args)}"  
    for arg in args:  
        print arg
```

By reference and output parameters.

In C# you can pass types to functions by reference using "ref" or "out" keywords.

```
static void Swap(ref int a, ref int b) {  
    int t = a;  
    a = b;  
    b = t;  
}  
....  
Swap(ref x, ref y);
```

Boo supports the "ref" keyword, too, but not "out", which is unnecessary in boo.

```
def Swap(ref a as int, ref b as int):  
    t = a  
    a = b  
    b = t  
  
x = 1  
y = 2  
Swap(x,y)
```

Note though that in this particular sample, you can swap two values more easily like so:

```
x = 1
y = 2

x, y = y, x
print x, y //-> 2 1
```

As a more general alternative to by reference parameters, in boo you can also return multiple values from a function instead:

```
def Swap(a as int, b as int):
    return b, a

a, b = Swap(a, b)
```

## Things in C# but not Boo

### do..while loop

Boo doesn't have do..while or do..until loops like C# or VB.NET.

You can emulate them with "while true" loops.

### unsafe code

```
static void WriteLocations(byte[] arr) {
    unsafe {
        fixed (byte* pArray = arr) {
            byte* pElem = pArray;
            for (int i = 0; i < arr.Length;
i++) {
                byte value = *pElem;
                Console.WriteLine("arr[{0}]
at 0x{1:X} is {2}",
                i, (uint)pElem, value);
                pElem++;
            }
        }
    }
}
```