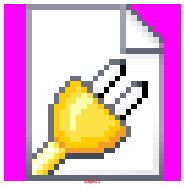
- Equivalencias VB / C#

Equivalencias entre Visual Basic .NET y C#

Publicado el 14/Abr/2003
Actualizado el 06/Ago/2006

En esta página (y posiblemente otras más) te explicaré cómo usar el mismo código en Visual Basic .NET y C# para que te sea más fácil usar cualquiera de estos dos lenguajes de punto NET.



Las equivalencias publicadas hasta ahora:

1. Consideraciones generales
2. Declarar variables con o sin asignación del valor por defecto
3. Ámbitos (niveles de accesibilidad)
4. Ámbitos predeterminados (si no se indica)
5. Bucles For y For Each (foreach)
6. Bucles While, Do... Loop
7. Abandonar un bucle o procedimiento
8. Comparaciones: IF, Else...
9. Operadores lógicos y aritméticos
10. Procedimientos/Métodos (funciones, propiedades)

- **22/Jul/2006:** La segunda parte de las equivalencias.
  - Declarar variables por referencia y asignar un valor
  - Arrays: declararlas, con y sin tamaño, cambiar el tamaño...
  - Propiedades por defecto o indizadores
  - Constructores y destructores
  - Llamar a otro constructor desde un constructor (incluso al de la clase base)

- **06/Ago/2006:** La tercera parte de las equivalencias.
  - Select Case / switch
  - Conversiones entre datos
  - Sobre los redondeos de Visual Basic
  - Parámetros opcionales (y sobrecargas)
  - Array de parámetros opcionales
  - Parámetros por valor y por referencia

1- Consideraciones generales

C#:
Una cosa que debes tener muy presente cuando quieras escribir código en C#, es que todas las líneas deben acabar en un punto y coma (;). Debido a esta peculiaridad, puedes "alargar" cada línea de código C# en varias líneas, ya que el fin de una "sentencia" viene indicada por el punto y coma.
En C#, se pueden agrupar líneas de código en bloques, los bloques **siempre** estarán dentro de un par de llaves: { y }
En C# todas las variables deben estar declaradas con un tipo de datos específico, lo mismo ocurre cuando asignamos datos de distintos tipos, siempre debe indicarse el tipo al que se quiere convertir.

Visual Basic .NET:
En Visual Basic .NET cada línea física es una sentencia, si necesitas que ocupe más de una línea, tendrás que usar el guión bajo al final de la línea y continuar en la siguiente. Esto no es aplicable a los comentarios.
En VB.NET no se obliga a que se especifique el tipo de datos, pero si quieres hacer las cosas bien, deberías tener siempre conectado **Option Strict On**, esto te obligará a declarar las variables con el tipo de datos adecuado y así cuando hagas una asignación entre tipos diferentes de datos, tendrás que especificar el tipo, para ello hay que hacer una conversión explícita de datos (casting). Esto último es algo que muchos programadores de VB no suelen hacer, pero te lo recomiendo encarecidamente que lo hagas.

2- Declarar una variable, con y sin asignación del valor por defecto

En Visual Basic .NET se utiliza de la siguiente forma:
<ámbito> <variable> As <tipo>
<ámbito> <variable> As <tipo> = <valor>

En C# se declara de la siguiente forma:
<ámbito> <tipo> <variable>;
<ámbito> <tipo> <variable> = <valor>;

Visual Basic .NET	C#

`Dim i As Integer`
`Dim i1 As Integer = 10`
`Dim d As Double`
`Dim d1 As Double = 3.5#`
`Dim f As Single`
`Dim s As String`
	`int i;`
	`int i1 = 10;`
	`double d;`
	`double d1 = 3.5;`
	`float f;`
	`string s;`

```vb
Dim c As Char
Dim l As Long
Dim m As Decimal

Dim o As MiClase
Dim o1 As New MiClase()
Dim o2 As MiClase = New MiClase()

Public Sp As String
Private s1 As String
```

```csharp
char c;
long l;
decimal m;

MiClase o;
MiClase o1 = new MiClase();
MiClase o2 = new MiClase();

public string Sp;
private string s1;
```

En Visual Basic .NET cuando se declaran variables dentro de un procedimiento (método o propiedad) sólo se puede indicar Dim ya que esas variables serán privadas al procedimiento.
En estos casos, en C# no se indicará el ámbito, simplemente el tipo de la variable.

3- Ámbito (niveles de accesibilidad)

Cualquiera de los miembros de una clase pueden tener un ámbito o nivel de accesibilidad que dependerá de cómo y desde dónde se podrá acceder a dicho miembro.
Los ámbitos pueden ser: privados, públicos, a nivel de ensamblado, etc. Para más información consulta en la documentación de Visual Studio .NET, los siguientes links te muestran la explicación de los mismos tanto para C# como para Visual Basic .NET:

C#: ms-help://MS.VSCC/MS.MSDNVS.3082/csref/html/vclrfDeclaredAccessibilityPG.htm
VB: ms-help://MS.VSCC/MS.MSDNVS.3082/vbcn7/html/vbconaccessibility.htm
VB: ms-help://MS.VSCC/MS.MSDNVS.3082/vbcn7/html/vbconscopelevels.htm

Veamos las equivalencias de los modificadores de ámbito entre Visual Basic y C#, así como una pequeña descripción de esos mismos ámbitos.

Visual Basic .NET	C#	Descripción del ámbito
Private	**private**	Accesible dentro del mismo módulo, clase o estructura.
Friend	**internal**	Accesible desde dentro del mismo proyecto, pero no desde fuera de él.
Protected	**protected**	Accesible desde dentro de la misma clase o desde una clase derivada de ella.
Protected Friend	**protected internal**	Accesible desde clases derivadas o desde dentro del mismo proyecto, o ambos.
Public	**public**	Accesible desde cualquier parte del mismo proyecto, desde otros proyectos que hagan referencia al proyecto, y desde un ensamblado generado a partir del proyecto.

4- Ámbitos predeterminados (si no se indica)

Cuando declaramos una variable, método, etc. y no indicamos el ámbito que tienen, el compilador le asigna uno de forma predeterminada, según el lenguaje que estemos usando o el tipo de elemento en el que estemos haciendo esa declaración, tendrá un ámbito o nivel de accesibilidad diferente.

En la siguiente tabla, podemos ver qué ámbito tendría, dependiendo de dónde se declare.
Nota: Esta información está en la ayuda de Visual Studio .NET, para más información usa links indicados.

C#: ms-help://MS.VSCC/MS.MSDNVS.3082/csref/html/vclrfDeclaredAccessibilityPG.htm

Miembros de	Accesibilidad predeterminada	Accesibilidades declaradas permitidas
enum	**public**	**Ninguna**
class	**private**	**public** **protected** **internal** **private** **protected internal**
interface	**public**	**Ninguna**
struct	**private**	**public** **internal** **private**

VB: No he encontrado una lista con esta información... así que en algunos casos, simplemente lo he comprobado.

Miembros de	Accesibilidad predeterminada	Accesibilidades declaradas permitidas
Enum	**Public**	**Ninguna**
Class **Module**	**Public**	**Public** **Protected** **Friend** **Private** **Protected Friend**
Interface	**Public**	**Ninguna**
Structure	**Siempre hay que indicar el ámbito**	**Public** **Friend** **Private**

.

5- Bucles For y For Each (foreach)

Nota: En las versiones de Visual Basic .NET incluidas en Visual Studio .NET 2003 (.NET Framework 1.1) se pueden declarar variables para usar en el propio bucle

(**For** o **For Each**).
En C#, cuando se usa un bucle **foreach**, siempre hay que declarar la variable a usar con el bucle.

Visual Basic .NET	C#

`Dim i As Integer`
`For i = 1 To 10`
`    ' ...`
`Next`
	`int i;`
	`for(i = 1; i<= 10; i++)`
	`{`
	`    // ...`
	`}`

`' Sólo en Visual Studio .NET 2003` `For i As Integer = 1 To 10`	`for(int i = 1; i<=10; i++)`
`Dim objeto As <Tipo>` `For Each objeto In colección`	`foreach(<Tipo> objeto in colección)`
`' Sólo en Visual Studio .NET 2003` `For Each objeto As <Tipo> In colección`	`foreach(<Tipo> objeto in colección)`

.

6- Bucles While, Do... Loop

Visual Basic .NET	C#

`Do`
`    '...`
`Loop While <expresión>`
	`do`
	`{`
	`    //...`
	`}while(<expresión>);`

`While <expresión>`
`    '...`
`End While`
	`while(<expresión>)`
	`{`
	`    //...`
	`}`

`Do While <expresión>`
`    '...`
`Loop`
	`while(<expresión>)`
	`{`
	`    //...`
	`}`

`Do`
`    '...`
`Loop`
	`while(true)`
	`{`
	`    //...`
	`}`

`Do Until <expresión>`
`    '...`
`Loop`
	`while(!<expresión>)`
	`{`
	`    //...`

	}

Do
    '...
Loop Until <expresión>
	do
	{
	    //...
	}while(!<expresión>);

7- Abandonar un bucle o procedimiento

Visual Basic .NET:
Para abandonar un bucle Do... Loop, se utiliza Exit Do.
Para abandonar un bucle While... End While, se utiliza Exit While.
Para abandonar un bucle For o For Each, se utiliza Exit For
Para abandonar un procedimiento Function (función), se utilizará Exit Function.
Para abandonar un procedimiento Sub, se utilizará Exit Sub.
Para abandonar un procedimiento Property, se utilizará Exit Property.

C#:
Para abandonar cualquier tipo de bucle, se utilizará break.
Para abandonar cualquier tipo de procedimiento, se utilizará return.

8- Comparaciones, If, Else...

Visual Basic .NET	C#

If x = 10 Then
    '...
End If
	if(x == 10)
	{
	    //...
	}

If x = 10 Then
    '...
Else
    '...
End If
	if(x == 10)
	{
	    //...
	}
	else
	{
	    //...
	}

If x = 10 Then
    '...
ElseIf x > 50 Then
    '...
End If
	if(x == 10)
	{
	    //...
	}
	else if(x > 50)
	{
	    //...
	}

En C# no es necesario usar llaves para indicar la instrucción a ejecutar cuando se cumpla la condición del IF, pero si hay más de una instrucción, es necesario usar las llaves.
Estos ejemplos lo aclararán mejor:

```
// con una instrucción, no es necesario usar llaves
if(x == 10)
    Console.WriteLine(x);

// con más de una instrucción, hay que usar las llaves
if(x == 10)
{
    Console.WriteLine(x);
    x += 10;
}
```

En Visual Basic no es necesario usar End If si el código a ejecutar se incluye en una línea, pero esa instrucción (o instrucciones separadas por dos puntos (:) deberán estar en la misma línea.

```
' con una instrucción
If x = 10 Then Console.WriteLine(x)

' con más de una instrucción, pero en una sola línea
' pero separando cada instrucción con dos puntos
If x = 10 Then Console.WriteLine(x): x += 10
```

.

9- Operadores lógicos y aritméticos

Visual Basic .NET	C#
And	**&**
AndAlso	**&&**
Or	**|**
OrElse	**||**
XOr	**^**
Not	**!**
=	**==**
<>	**!=**
& (concatenación de cadenas)	**+**
\ (división de números enteros)	**/**
\=	**/=**
Mod	**%**
Is Nothing	**== null**

En C# sólo se utiliza el símbolo / para división tanto de números enteros como decimales

En VB la división de números enteros se realiza con \, la división de números decimales se hace con /.

.

10- Procedimientos / Métodos (funciones, propiedades)

En Visual Basic existen tres tipos de procedimientos: Sub, Function y Property
En C# los procedimientos pueden ser funciones o propiedades. Las funciones pueden o no devolver algún valor, en caso de que no devuelvan un valor se comportan como los Subs de Visual Basic.

Declarar un procedimiento de tipo Sub en Visual Basic y el equivalente en C#

Visual Basic .NET	C#

<ámbito> Sub <nombre>()
End Sub
	<ámbito> void <nombre>()
	{
	}

Declarar un procedimiento de tipo Function en Visual Basic y el equivalente en C#

Visual Basic .NET	C#

<ámbito> Function <nombre>() As <tipo>
End Function
	<ámbito> <tipo> <nombre>()
	{
	}

Declarar un procedimiento de tipo Property en Visual Basic y el equivalente en C#

Visual Basic .NET	C#

<ámbito> Property <nombre>() As <tipo>
Get
'...
End Get
Set
'...
End Set
End Property
	<ámbito> <tipo> <nombre>
	{
	get{
	//...
	}
	set{
	//..
	}
	}

<ámbito> ReadOnly Property <nombre> As <tipo>
Get
'...
End Get
End Property
	<ámbito> <tipo> <nombre>
	{
	get{
	//...
	}
	}

<ámbito> WriteOnly Property <nombre> As <tipo>
Set
'...
End Set
	<ámbito> <tipo> <nombre>
	{
	set{
	//...

```
End Property
```

```
    }
}
```

.