Hidden Features of VB.NET?

Asked 16 years, 3 months ago Modified 13 years ago Viewed 30k times

121

votes

Locked. This question and its answers are locked because the question is off-topic but has historical significance. It is not currently accepting new answers or interactions.

I have learned quite a bit browsing through Hidden Features of C# and was surprised when I couldn't find something similar for VB.NET.

So what are some of its hidden or lesser known features?

vb.net hidden-features

Share

edited May 23, 2017 at 12:10

community wiki
5 revs, 2 users 40%
Sean Gough

Comments disabled on deleted / locked posts / reviews

128 votes

The `Exception When` clause is largely unknown.

Consider this:

```
Public Sub Login(host as string, user as String, passwo
                                Optional bRetry as Boolean
Try
   ssh.Connect(host, user, password)
Catch ex as TimeoutException When Not bRetry
   ''//Try again, but only once.
   Login(host, user, password, True)
Catch ex as TimeoutException
   ''//Log exception
End Try
End Sub
```

Share

edited Dec 20, 2011 at 18:05

SteveC

16.7k ● 25 ● 112 ● 179

answered Sep 19, 2008 at 14:16

torial

13.1k ● 9 ● 65 ● 89

9 useful if you wish to catch a specific SQLException, say -2 which if i remember correctly is network timeout: Catch ex as sqlException where ex.code = -2 – Pondidum Dec 27, 2008 at 18:06

Wow! I just read this and put it to use immediately to simplify a try/catch block I just wrote last week. I never new this existed. – John M Gant Jun 2, 2009 at 16:14

1 +1 And here's where the NET CLR team blog explains why exception filters are useful blogs.msdn.com/clrteam/archive/2009/02/05/… – MarkJ Jun 9, 2009 at 14:32

5 Not only is this hidden, but it is also not available in C#. – Cheeso Aug 25, 2009 at 20:43

82 votes

Custom `Enum`s

One of the real *hidden* features of VB is the `completionlist` XML documentation tag that can be used to create own `Enum`-like types with extended functionality. This feature doesn't work in C#, though.

One example from a recent code of mine:

```
'
''' <completionlist cref="RuleTemplates"/>
Public Class Rule
    Private ReadOnly m_Expression As String
    Private ReadOnly m_Options As RegexOptions

    Public Sub New(ByVal expression As String)
        Me.New(expression, RegexOptions.None)
    End Sub

    Public Sub New(ByVal expression As String, ByVal opt
        m_Expression = expression
        m_options = options
    End Sub
```

```
    Public ReadOnly Property Expression() As String
        Get
            Return m_Expression
        End Get
    End Property

    Public ReadOnly Property Options() As RegexOptions
        Get
            Return m_Options
        End Get
    End Property
End Class

Public NotInheritable Class RuleTemplates
    Public Shared ReadOnly Whitespace As New Rule("\s+")
    Public Shared ReadOnly Identifier As New Rule("\w+")
    Public Shared ReadOnly [String] As New Rule("""([^"'
End Class
```

Now, when assigning a value to a variable declared as `Rule`, the IDE offers an IntelliSense list of possible values from `RuleTemplates`.

/EDIT:

Since this is a feature that relies on the IDE, it's hard to show how this looks when you use it but I'll just use a screenshot:

[Completion list in action http://page.mi.fu-berlin.de/krudolph/stuff/completionlist.png](http://page.mi.fu-berlin.de/krudolph/stuff/completionlist.png)

In fact, the IntelliSense is 100% identical to what you get when using an `Enum`.

Share

edited Jun 2, 2009 at 16:07

Joel Coehoorn

415k ● 114 ● 577 ● 813

answered Sep 19, 2008 at 14:22

Konrad Rudolph

545k ● 138 ● 956 ● 1.2k

Interesting - could you show an example of how this looks when you consume it? – Brian MacKay Sep 19, 2008 at 15:15

It looks like someone could still create their own rule by calling the Rule constructor directly? If so, and if you wanted to stop this, could you declare the constructor as "Friend" in your library? – Joel Coehoorn Oct 7, 2008 at 14:40

Joel, in my example code I intentially didn't forbid this. Rather, I offer the user some common rules and allow the creation of own, specialized rules. You can of course prevent this by marking the constructor as `friend` or by using the same class as the enum: `Rule` instead of `RuleTemplate`.
– Konrad Rudolph Oct 8, 2008 at 9:06

is there somewhere with a list of hidden useful attribute uses? this one at face seems incredible, but I'm not sure yet where I would use it or if it could solve a primary gripe in some cases with having no ability to have a generic restricted to enums.
– Maslow May 22, 2009 at 13:23

@Maslow: there's not attribute involved here. That is an xml comment. – Joel Coehoorn Jun 2, 2009 at 15:45

49

votes

Have you noticed the Like comparison operator?

```
    Dim b As Boolean = "file.txt" Like "*.txt"
```

More from MSDN

```
Dim testCheck As Boolean

' The following statement returns True (does "F" satisfy
testCheck = "F" Like "F"

' The following statement returns False for Option Compa
'    and True for Option Compare Text (does "F" satisfy
testCheck = "F" Like "f"

' The following statement returns False (does "F" satisf
testCheck = "F" Like "FFF"

' The following statement returns True (does "aBBBa" hav
'    beginning, an "a" at the end, and any number of cha
'    between?)'
testCheck = "aBBBa" Like "a*a"

' The following statement returns True (does "F" occur i
'    characters from "A" through "Z"?)'
testCheck = "F" Like "[A-Z]"

' The following statement returns False (does "F" NOT oc
'    set of characters from "A" through "Z"?)'
testCheck = "F" Like "[!A-Z]"

' The following statement returns True (does "a2a" begir
'    an "a" and have any single-digit number in between?
testCheck = "a2a" Like "a#a"

' The following statement returns True (does "aM5b" begi
'    followed by any character from the set "L" through
'    by any single-digit number, and end with any charac
'    the character set "c" through "e"?)'
testCheck = "aM5b" Like "a[L-P]#[!c-e]"

' The following statement returns True (does "BAT123khg"
'    "B", followed by any single character, followed by
```

```
'    with zero or more characters of any type?)'
testCheck = "BAT123khg" Like "B?T*"

' The following statement returns False (does "CAT123khg
'    a "B", followed by any single character, followed b
'    end with zero or more characters of any type?)'
testCheck = "CAT123khg" Like "B?T*"
```

Share

edited Oct 19, 2011 at 8:59

community wiki
5 revs, 5 users 66%
Eduardo Molteni

3 wait, what? That's new to me! Hmm, that's a helluva lot better than the alternative with VB.NET string manipulation :D – STW May 5, 2009 at 14:37

..!! I didn't know that although I worked on several vb.net projects! Interesting feature... – Meta-Knight May 13, 2009 at 3:44

wooooooooooooooooooooooow! thanks! that's awsome! does it work in ELinq, DLinq too? what about XLinq?
– Shimmy Weitzhandler Jul 14, 2009 at 9:49

@dotjoe Hmm? There's nothing 'lazy' about these globs.
– Josh Lee Nov 27, 2009 at 7:16

How does this perform, whats happening under the hood? Is it s synonym for reg ex libraries? – brumScouse Aug 25, 2010 at 22:00

48

votes

Typedefs

VB knows a primitive kind of `typedef` via `Import` aliases:

```
Imports S = System.String

Dim x As S = "Hello"
```

This is more useful when used in conjunction with generic types:

```
Imports StringPair = System.Collections.Generic.KeyValue
```

Share

edited Jul 13, 2009 at 17:37

answered Sep 19, 2008 at 14:40

Konrad Rudolph

545k ● 138 ● 956 ● 1.2k

2 please show an example the word Import is unrecognized in my IDE. – Shimmy Weitzhandler Jul 13, 2009 at 15:55

5 `Imports` it should be. ;-) Somehow, this error has gone undetected (and garnered 28 upvotes) for nearly a whole year. – Konrad Rudolph Jul 13, 2009 at 17:37

I always wondered how i could make a new "type" with defaults for a generic! Cool! – eidylon Sep 2, 2009 at 23:42

3 Import, Imports, Importz, whatever! Sheesh, you think we *read* these posts before we upvote them! – MarkJ Sep 4, 2009 at

16:27

Superb for using xunit in visual studio tests i.e. `Imports Assert = xUnit.Assert` – wheelibin Jul 19, 2010 at 10:43

45

votes

Oh! and don't forget XML Literals.

```
Dim contact2 = _
        <contact>
          <name>Patrick Hines</name>
          <%= From p In phoneNumbers2 _
            Select <phone type=<%= p.Type %>><%= p.Numbe
          %>
        </contact>
```

Share

answered Sep 19, 2008 at 14:34

Nescio

28.4k ● 10 ● 55 ● 76

That's what I was going to say. In fact, I've used XML literals for a purpose in which it wasn't intended. I used it to produce Javascript... swortham.blogspot.com/2009/03/vb-2008.html – Steve Wortham Jul 13, 2009 at 21:26

9 Among other things, you can use XML Literals to get around ugly string escaping, e.g. when you're using strings that themselves contain double quotes. Just put the string inside an XML Literal and call Value, like this: `<string>This string contains "quotes" and it's OK.</string>.Value` (I found this especially handy when writing tests on parsing CSV files where every field was in quotes. It would *not* have been fun to escape all those quotes by hand in my test lines.) – Ryan Lundy Dec 17, 2009 at 22:26

2 @Kyralessa: +1, great comment. In fact, it's also a great way to specify multi-line strings (hugs SQL statements, etc.). – Heinzi Dec 27, 2009 at 19:41

Inline XML is one of the worst VB.NET features ever to be conceived. Unsubstantiated fact. – Grant Thomas Feb 1, 2012 at 15:01

39

votes

Object initialization is in there too!

```
Dim x as New MyClass With {.Prop1 = foo, .Prop2 = bar}
```

Share

answered Sep 19, 2008 at 14:17

Nescio

28.4k ● 10 ● 55 ● 76

1 I can't believe they went with curly braces. We already have a With statement.. they could have just reused that syntax. – Sam Axe May 19, 2009 at 19:26

2 I know, it's made just to show you that there is no cop on the road... lol JK – Shimmy Weitzhandler Jul 13, 2009 at 15:56

4 Thi sis what happens when all of the compiler writers are themselves C/C++ programmers at heart. They keep slipping C syntax into other languages because they cannot conceive of anything being better. – RBarryYoung Sep 19, 2009 at 6:28

38

votes

`DirectCast`

`DirectCast` is a marvel. On the surface, it works similar to the `CType` operator in that it converts an object from one type into another. However, it works by a much stricter set of rules. `CType`'s actual behaviour is therefore often opaque and it's not at all evident which kind of conversion is executed.

`DirectCast` only supports two distinct operations:

- Unboxing of a value type, and
- upcasting in the class hierarchy.

Any other cast will not work (e.g. trying to unbox an `Integer` to a `Double`) and will result in a compile time/runtime error (depending on the situation and what can be detected by static type checking). I therefore use `DirectCast` whenever possible, as this captures my intent best: depending on the situation, I either want to unbox a value of known type or perform an upcast. End of story.

Using `CType`, on the other hand, leaves the reader of the code wondering what the programmer really intended because it resolves to all kinds of different operations, including calling user-defined code.

Why is this a hidden feature? The VB team has published a guideline[1] that discourages the use of `DirectCast` (even though it's actually faster!) in order to make the code more uniform. I argue that this is a bad guideline that should be reversed: **Whenever possible, favour `DirectCast` over the more general `CType` operator.** It makes the code much clearer. `CType`, on the other hand, should only be

called if this is indeed intended, i.e. when a narrowing `CType` operator (cf. operator overloading) should be called.

[1)] I'm unable to come up with a link to the guideline but I've found Paul Vick's take on it (chief developer of the VB team):

> In the real world, you're hardly ever going to notice the difference, so you might as well go with the more flexible conversion operators like CType, CInt, etc.

(EDIT by Zack: Learn more here: How should I cast in VB.NET?)

Share

edited May 23, 2017 at 10:30

Community Bot

1 ● 1

answered Sep 19, 2008 at 16:06

Konrad Rudolph

545k ● 138 ● 956 ● 1.2k

I actually like TryCast() a bit better. It won't throw an exception, just return Nothing if the cast fails. If you expect the cast to fail often, it's faster than If TypeOf x Is T...DirectCast(x,T) and certainly faster than catching the exception if DirectCast fails. – Bob King Sep 22, 2008 at 22:48

---

6 DirectCast() and TryCast() are invaluable when used correctly as a pair. DirectCast() should be used if the object being cast is

always expected to be the target type (if it isn't you'll get an error, a good thing since it's an unexpected situation). TryCast() should be used if the object being cast *could* be of the target type, or of several target types. Using One or the other exclusively will either lead to extra overhead (if typeof x is y then directcast(x, y) is inefficient) or to avoiding valid errors (using TryCast() for cases where the object should *always* be the target type) – STW May 5, 2009 at 14:28

Yoooder: 100% correct. It's a shame I didn't mention `TryCast` back then since I mainly had a bone to pick with the pervasive use of `CType`. – Konrad Rudolph May 5, 2009 at 17:35

@Maslow: of course it doesn't, since Enums are value types and `TryCast` only works on reference types, as per the documentation. – Konrad Rudolph May 22, 2009 at 15:03

+1: Heh. I've got to admit, I just read this and thought "Oh yeah, DirectCast, how did I forget about that?" And then I used it on my very next line of code (as I really dislike CType).
– RBarryYoung Sep 19, 2009 at 6:53

37

votes

`If` conditional and coalesce operator

> I don't know how hidden you'd call it, but the Iif([expression],[value if true],[value if false]) As Object function could count.

It's not so much hidden as **deprecated**! VB 9 has the `If` operator which is much better and works exactly as C#'s

conditional and coalesce operator (depending on what you want):

```
Dim x = If(a = b, c, d)

Dim hello As String = Nothing
Dim y = If(hello, "World")
```

Edited to show another example:

This will work with `If()`, but cause an exception with `IIf()`

```
Dim x = If(b<>0,a/b,0)
```

Share

edited Oct 1, 2008 at 16:40

Joel Coehoorn

415k ● 114 ● 577 ● 813

answered Sep 19, 2008 at 14:14

Konrad Rudolph

545k ● 138 ● 956 ● 1.2k

Nice, I didn't know this! I just used IIF yesterday so I'm going to revisit that code block. – Sean Gough Sep 19, 2008 at 14:16

4 Tell VS 2005 that. Not all of us get to work with the latest and greatest. – Sam Erwin Sep 19, 2008 at 18:31

3 @Slough, nonsense. This method is 100% type safe and it returns an object of the same type as its (second and third) argument. Additionally, there must be a widening conversion between the arguments, else there will be a compile error

because the types don't match. – Konrad Rudolph Nov 27, 2008 at 17:53

1 Yes, its IIf() that is not type safe – Pondidum Jul 8, 2009 at 12:53

2 @Br.Bill In fact, it's *completely equivalent* to C and Perl's `:?` operator, it's not just a simplified version. – Konrad Rudolph Mar 12, 2011 at 8:55

---

32 votes

This is a nice one. The Select Case statement within VB.Net is very powerful.

Sure there is the standard

```
Select Case Role
  Case "Admin"
         ''//Do X
  Case "Tester"
         ''//Do Y
  Case "Developer"
         ''//Do Z
  Case Else
       ''//Exception case
End Select
```

But there is more...

You can do ranges:

```
Select Case Amount
 Case Is < 0
    ''//What!!
 Case 0 To 15
   Shipping = 2.0
```

```
 Case 16 To 59
    Shipping = 5.87
 Case Is > 59
    Shipping = 12.50
 Case Else
    Shipping = 9.99
 End Select
```

And even more...

You can (although may not be a good idea) do boolean checks on multiple variables:

```
Select Case True
 Case a = b
    ''//Do X
 Case a = c
    ''//Do Y
 Case b = c
    ''//Do Z
 Case Else
   ''//Exception case
 End Select
```

Share

edited Sep 27, 2011 at 14:10

John

30.5k ● 18 ● 92 ● 133

answered Dec 26, 2008 at 19:59

torial

13.1k ● 9 ● 65 ● 89

5 Actually you missed a couple: a) use of "Select Case True" to test more than one variable, b) use of "Case A, B, ..." form, and even c) applying the ":" to in-line the execution statement with the condition clause (though many do not like this).
– RBarryYoung Sep 19, 2009 at 6:24

6 Please don't use Select Case True. Just use an If statement.
– Ryan Lundy Dec 2, 2009 at 21:29

4 I find Select Case True much easier to read than a giant ifelse statement. – dwidel Dec 30, 2010 at 16:43

The trouble with `Select Case True` is that it *looks* as if it evaluates *each* of the `Case` statements and runs the code for each one which is true. But in fact it evaluates them one-by-one and only runs the code for the *first* one which is true. The syntax for `If` is much clearer in this regard (`If...Else If...Else If...Else`). – Ryan Lundy Dec 30, 2011 at 18:56

31 votes

One major time saver I use all the time is the **With** keyword:

```
With ReallyLongClassName
    .Property1 = Value1
    .Property2 = Value2
    ...
End With
```

I just don't like typing more than I have to!

Share

answered Sep 19, 2008 at 14:10

Jasha87
327 ● 2 ● 3

But it also create some hidden bugs, especially when you have With within With – Varun Mahajan Sep 19, 2008 at 14:24

2 I didn't even know you could put a new With within an existing With. That's just sloppy! – Bob King Sep 22, 2008 at 22:44

2Bob: it's not sloppy, I'd suggest.. it's just a language construct to be used with care. For setting lots of properties in successive lines, it's great, but finding a random .Foo in a complex piece of code and then having to hunt the With statement x lines above is not a good use of the feature. – ChrisA Dec 19, 2008 at 15:50

2 Wish C# has this? Or have I been asleep and is that in the C# hidden-features answers already...? ;-) – peSHIr Jan 15, 2009 at 14:41

1 @Boo: You're right but it's still an annoyance that you can't add it to the Watch list. – Meta-Knight Jun 22, 2009 at 17:50

31 votes

The best and easy CSV parser:

```
Microsoft.VisualBasic.FileIO.TextFieldParser
```

By adding a reference to Microsoft.VisualBasic, this can be used in any other .Net language, e.g. C#

Share

answered Jun 2, 2009 at 16:13

community wiki
cjk

5 +1 It's weird how the C# folks run to FileHelpers without ever considering this. I'm sure FileHelpers is excellent, but it is an external dependency. – MarkJ Jun 9, 2009 at 14:45

@MarkJ I presume it's because of ignorance – Nathan Koop Jun 9, 2009 at 19:07

I googled the hell out of trying to find this class and never did, thanks for flaging it up! – pingoo May 24, 2011 at 8:32

26

votes

- AndAlso/OrElse logical operators

(EDIT: Learn more here: Should I always use the AndAlso and OrElse operators?)

Share

edited May 23, 2017 at 12:02

Community Bot

1 ● 1

answered Sep 19, 2008 at 14:10

Joel Coehoorn

415k ● 114 ● 577 ● 813

I wouldn't call it hidden at all, If Contact IsNot Nothing AndAlso Contact.ContactId > 0 Then Do it, If you will use And you will have an exception thrown. – Shimmy Weitzhandler Jul 14, 2009 at 9:55

25

votes

Static members in methods.

For example:


```
Function CleanString(byval input As String) As String
    Static pattern As New RegEx("...")

    return pattern.Replace(input, "")
End Function
```

In the above function, the pattern regular expression will only ever be created once no matter how many times the function is called.

Another use is to keep an instance of "random" around:

```
Function GetNextRandom() As Integer
    Static r As New Random(getSeed())

    Return r.Next()
End Function
```

Also, this isn't the same as simply declaring it as a Shared member of the class; items declared this way are guaranteed to be thread-safe as well. It doesn't matter in this scenario since the expression will never change, but there are others where it might.

Share

edited Jun 2, 2009 at 15:13

answered Oct 3, 2008 at 16:49

Joel Coehoorn
415k ● 114 ● 577 ● 813

1 One use of this is to keep a counter that will increment each time the method is called. If the variable is marked Static, it won't be reinitialized on each method call; it'll only be initialized on the first call, and thereafter will retain its value. – Ryan Lundy Jan 6, 2009 at 20:56

This is the reason why static class members are named "shared" in VB.NET – Enrico Campidoglio Apr 9, 2009 at 20:15

Static is bad form in VB.NET even more than it was in classic VB. Static variables should be avoided whenever and wherever possible. – Sam Axe May 19, 2009 at 19:39

6 @Boo - that's pretty sweeping. What's your justification? I think static variables are useful. – MarkJ Jun 9, 2009 at 14:36

4 Static, used as in the examples above, allows a unique form of encapsulation: class-level variables that have method-level scope. Without it, you'd have to create a class-level variable that would be accessible to any class member, even if you're only using it in one method. – Ryan Lundy Dec 2, 2009 at 21:41

25

votes

In vb there is a different between these operators:

`/` is `Double`

`\` is `Integer` ignoring the remainder

```
Sub Main()
    Dim x = 9 / 5
    Dim y = 9 \ 5
    Console.WriteLine("item x of '{0}' equals to {1}", x
    Console.WriteLine("item y of '{0}' equals to {1}", y

    'Results:
```

```
    'item x of 'System.Double' equals to 1.8
    'item y of 'System.Int32' equals to 1
End Sub
```

Share

edited Apr 3, 2010 at 5:29

community wiki
2 revs, 2 users 94%
Shimmy

1 i learned this the hard way when trying to find a needle in a million lines of code. regular versus integer division. good tip! – Jason Irwin Sep 18, 2009 at 18:47

23 votes

I really like the **"My" Namespace** which was introduced in Visual Basic 2005. *My* is a shortcut to several groups of information and functionality. It provides quick and intuitive access to the following types of information:

- **My.Computer**: Access to information related to the computer such as file system, network, devices, system information, etc. It provides access to a number of very important resources including My.Computer.Network, My.Computer.FileSystem, and My.Computer.Printers.
- **My.Application**: Access to information related to the particular application such as name, version, current directory, etc.

- **My.User**: Access to information related to the current authenticated user.
- **My.Resources**: Access to resources used by the application residing in resource files in a strongly typed manner.
- **My.Settings**: Access to configuration settings of the application in a strongly typed manner.

Share

answered Oct 10, 2008 at 11:13

splattne
104k ● 53 ● 205 ● 250

This is just great and every vb.net guy out there should know stuff in My namespace, it's so useful. – dr. evil Dec 19, 2008 at 15:50

My.* FTW :) . – dr. evil Apr 29, 2009 at 13:56

3 It's sort of useful, but I hate the dumbed down name. Reminds me of this secretgeek.net/refactvb.asp – MarkJ Jun 9, 2009 at 14:37

23 votes

Custom Events

Though seldom useful, event handling can be heavily customized:

```
Public Class ApplePie
    Private ReadOnly m_BakedEvent As New List(Of EventHa

    Custom Event Baked As EventHandler
        AddHandler(ByVal value As EventHandler)
```

```
            Console.WriteLine("Adding a new subscriber:
            m_BakedEvent.Add(value)
        End AddHandler

        RemoveHandler(ByVal value As EventHandler)
            Console.WriteLine("Removing subscriber: {0}'
            m_BakedEvent.Remove(value)
        End RemoveHandler

        RaiseEvent(ByVal sender As Object, ByVal e As Ev
            Console.WriteLine("{0} is raising an event.'
            For Each ev In m_BakedEvent
                ev.Invoke(sender, e)
            Next
        End RaiseEvent
    End Event

    Public Sub Bake()
        ''// 1. Add ingredients
        ''// 2. Stir
        ''// 3. Put into oven (heated, not pre-heated!)
        ''// 4. Bake
        RaiseEvent Baked(Me, EventArgs.Empty)
        ''// 5. Digest
    End Sub
End Class
```

This can then be tested in the following fashion:

```
Module Module1
    Public Sub Foo(ByVal sender As Object, ByVal e As Ev
        Console.WriteLine("Hmm, freshly baked apple pie.
    End Sub

    Sub Main()
        Dim pie As New ApplePie()
        AddHandler pie.Baked, AddressOf Foo
        pie.Bake()
        RemoveHandler pie.Baked, AddressOf Foo
    End Sub
End Module
```

Share

edited Jun 2, 2009 at 15:11

Joel Coehoorn

415k ● 114 ● 577 ● 813

answered Sep 19, 2008 at 17:20

Konrad Rudolph

545k ● 138 ● 956 ● 1.2k

seems cool, but if I ever need this I think I will have done something wrong ;-). It just seems against the KISS principle. – chrissie1 Sep 19, 2008 at 18:38

4 It is really, really useful when you want to make sure every sink gets the event even if one or more throw an exception. – Jonathan Allen Oct 4, 2008 at 5:40

Another value (I just read it on the MSDN site) is if your class throws lots of events, you can force it to use a hash table/dictionary as a container, rather than declaring a field for each event. msdn.microsoft.com/en-us/library/8627sbea(VS.71).aspx – torial Dec 13, 2008 at 0:46

Cool. I thought this one of the features that set apart C# and VB.NET (as in one can, but the other has no syntax for it). Nice to at least know I was wrong in this respect. – peSHIr Jan 15, 2009 at 14:40

21

votes

I just found an article talking about the "!" operator, also know as the "dictionary lookup operator". Here's an excerpt from the article at:

http://panopticoncentral.net/articles/902.aspx

> The technical name for the ! operator is the "dictionary lookup operator." A dictionary is any collection type that is indexed by a key rather than a number, just like the way that the entries in an English dictionary are indexed by the word you want the definition of. The most common example of a dictionary type is the System.Collections.Hashtable, which allows you to add (key, value) pairs into the hashtable and then retrieve values using the keys. For example, the following code adds three entries to a hashtable, and looks one of them up using the key "Pork".

```
Dim Table As Hashtable = New Hashtable
Table("Orange") = "A fruit"
Table("Broccoli") = "A vegetable"
Table("Pork") = "A meat"
Console.WriteLine(Table("Pork"))
```

> The ! operator can be used to look up values from any dictionary type that indexes its values using strings. The identifier after the ! is used as the key in the lookup operation. So the above code could instead have been written:

```
Dim Table As Hashtable = New Hashtable
Table!Orange = "A fruit"
Table!Broccoli = "A vegetable"
Table!Pork = "A meat"
Console.WriteLine(Table!Pork)
```

The second example is completely equivalent to the first, but just looks a lot nicer, at least to my eyes. I find that there are a lot of places where ! can be used, especially when it comes to XML and the web, where there are just tons of collections that are indexed by string. One unfortunate limitation is that the thing following the ! still has to be a valid identifier, so if the string you want to use as a key has some invalid identifier character in it, you can't use the ! operator. (You can't, for example, say "Table!AB$CD = 5" because $ isn't legal in identifiers.) In VB6 and before, you could use brackets to escape invalid identifiers (i.e. "Table![AB$CD]"), but when we started using brackets to escape keywords, we lost the ability to do that. In most cases, however, this isn't too much of a limitation.

To get really technical, x!y works if x has a default property that takes a String or Object as a parameter. In that case, x!y is changed into x.DefaultProperty("y"). An interesting side note is that there is a special rule in the lexical grammar of the language to make this all work. The ! character is also used as a type character in the language, and type characters are eaten before operators. So without a special rule, x!y would be scanned as "x! y" instead of "x ! y". Fortunately, since there is no place in the language where two identifiers in a row are valid, we just introduced the rule that if the

> next character after the ! is the start of an identifier, we consider the ! to be an operator and not a type character.

Share

answered Mar 14, 2009 at 16:22

community wiki
torial

11 That's one of those features I used then intentionally forgot. It saves a few keystrokes but messes with my code highlighting and readability. *forgetting again right.... NOW* – STW May 5, 2009 at 14:39

3 Interesting, but not really useful. Is this the kind of stuff the VB team works on instead of adding missing features like the yield keyword? :P – Meta-Knight May 13, 2009 at 3:53

5 This feature is carried for backward compatibility from VB3 (AFAIK) – Eduardo Molteni May 13, 2009 at 13:43

1 do those classes that implement a keyed index have a common interface they inherit from? like IKeyed the same way integer indexed containers implement IENumberable? – Maslow May 22, 2009 at 12:39

2 This feature also works with DataRows (i.e. dr!ID) which is VERY handy in LINQ to DataSets. – Paul Aug 24, 2009 at 20:31

19 This is built-in, and a definite advantage over C#. The ability to implement an interface Method without having to

votes

use the same name.

Such as:

```
Public Sub GetISCSIAdmInfo(ByRef xDoc As System.Xml.XmlD
IUnix.GetISCSIInfo

End Sub
```

Share

answered Sep 19, 2008 at 14:28

torial

13.1k ● 9 ● 65 ● 89

Im pritty sure that you can do something like this in C#. In VB.NET its just forced, and in C# its optional?
– Jesper Blad Jensen Feb 1, 2009 at 11:43

5 You can also make the sub private, which is a great way to hide stuff like the calls to non-generic deprecated versions of interfaces. – Craig Gidney May 26, 2009 at 17:31

3 It can be a good idea. The classic example would be if you want a Public Close method that also acts as your Dispose implementation for IDisposable. – MarkJ Jun 9, 2009 at 14:41

1 That's also pretty useful if you happen to implement two interfaces that share a method name. – Eric Nicholson Jan 5, 2010 at 19:25

I've seen this and always wish I hadn't. Should not be allowed.
– FastAl Sep 7, 2010 at 19:03

17

votes

Forcing ByVal

In VB, if you wrap your arguments in an extra set of parentheses you can override the ByRef declaration of the method and turn it into a ByVal. For instance, the following code produces 4, 5, 5 instead of 4,5,6

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e A
Handles Me.Load
    Dim R = 4
    Trace.WriteLine(R)
    Test(R)
    Trace.WriteLine(R)
    Test((R))
    Trace.WriteLine(R)
End Sub
Private Sub Test(ByRef i As Integer)
    i += 1
End Sub
```

See Argument Not Being Modified by Procedure Call - Underlying Variable

Share

answered May 14, 2010 at 20:45

community wiki
Chris Haas

6 Oh my goodness... that's a remarkable feature, though I don't think I'd know what it was doing if I read it in someone else's code. If you have to comment it just to know what it's doing, you might as well have made the throw away variable to pass in instead. – mattmc3 Jul 22, 2010 at 0:10

7 This is actually a side effect of the use of parenthesis - Parenthesis create a temp value of what's inside, even if just one item. This effect KILLED ME in vb6 - Sub calls didn't take parens, but me coming from C instinctively put the parens in. 6.0 blew up on multiple parameters, but for one parameter subs, it happily passed a temp value and REFUSED to honor my 'byref'. Happened about every 3 years, about the time it took me to forget the last incident. – FastAl Sep 7, 2010 at 19:11

16 votes

Passing parameters by name and, so reordering them

```
Sub MyFunc(Optional msg as String= "", Optional displayC

    'Do stuff

End function
```

Usage:

```
Module Module1

    Sub Main()

        MyFunc() 'No params specified

    End Sub

End Module
```

Can also be called using the ":=" parameter specification in any order:

```
MyFunc(displayOrder:=10, msg:="mystring")
```

Share

edited Jul 14, 2009 at 10:03

community wiki
3 revs, 3 users 49%
rich

Wow, this is super cool! I seen this when using Web Services, etc but I didn't know you could do this any any ol' normal method. – RichC Dec 22, 2008 at 16:26

2 Definately a *very* handy tool when you encounter a method that takes too many arguments. I try to name each parameter and put the name:=value on its own line. It's a lot more intuitive and clean for methods that take > 5 (my rule of thumb) parameters. – STW May 5, 2009 at 14:33

Especially useful for Office automation, where you have to cope with methods with dozens of optional arguments. – MarkJ Jun 9, 2009 at 14:38

1 What's also cool is that you can *mix* the two: start by specifying the required parameters in order, then switch to named parameters for the optional arguments! – RBarryYoung Sep 19, 2009 at 7:10

15 votes

The Using statement is new as of VB 8, C# had it from the start. It calls dispose automagically for you.

E.g.

```
Using lockThis as New MyLocker(objToLock)

End Using
```

Share

answered Sep 19, 2008 at 14:24

torial

13.1k ● 9 ● 65 ● 89

23 It's worth noting (only because I've forgotten at least twice) that you can have one Using statement wrap several Disposable objects. The syntax is "Using objA as new object, objB as new object...." It's a lot cleaner than nesting multiple Using statements. – STW May 5, 2009 at 14:35

Definately one of my favorites as well. – Sam Axe May 19, 2009 at 19:31

14

votes

Import aliases are also largely unknown:

```
Import winf = System.Windows.Forms

''Later
Dim x as winf.Form
```

Share

answered Sep 19, 2008 at 14:22

torial

13.1k ● 9 ● 65 ● 89

1 I think we had the same idea. – chrissie1 Sep 19, 2008 at 14:27

3 @Boo -- here's a simple example where import aliases are not evil. stackoverflow.com/questions/92869/… – torial Jun 4, 2009 at 0:04

Okay - there's an obvious avenue for abuse, but regardless of @torial's rant this is a great feature. Everytime I include System.Text.RegularExpressions I risk having name collisions with the "Group" class since that's a common class name. The alias lets you avoid having to fully qualify ambiguous classes which is a huge time saver and actually **enhances** readability when used properly. The feature is great, but your specific example does sort of invite some ridicule - sorry to say.
– mattmc3 Jul 22, 2010 at 0:01

---

14 votes

Consider the following event declaration

```
Public Event SomethingHappened As EventHandler
```

In C#, you can check for event subscribers by using the following syntax:

```
if(SomethingHappened != null)
{
  ...
}
```

However, the VB.NET compiler does not support this. It actually creates a hidden private member field which is not visible in IntelliSense:

```
If Not SomethingHappenedEvent Is Nothing OrElse
SomethingHappenedEvent.GetInvocationList.Length = 0 Ther
```

```
...
End If
```

More Information:

http://jelle.druyts.net/2003/05/09/BehindTheScenesOfEventsInVBNET.aspx
http://blogs.msdn.com/vbteam/archive/2009/09/25/testing-events-for-nothing-null-doug-rothaus.aspx

Share

edited Sep 25, 2009 at 21:48

answered Sep 19, 2008 at 16:44

Technobabble
1,046 ● 11 ● 25

Why would you ever want to do that? I can't imagine a use case where you need to know the number of subscribers to an event in VB. – Konrad Rudolph Sep 19, 2008 at 16:58

In certain circumstances, C# throws an exception if you raise the event and there are no handlers. VB.Net will not. Hence the need to check. – Joel Coehoorn Sep 19, 2008 at 20:23

2 I used this for a business object event which raised validation error messages to the subscribers. I wanted to check to see if the event was being handled so that I knew the validation errors were being received. Otherwise, I had the business object throw an exception. – Technobabble Nov 18, 2008 at 17:12

2 Another handy use for this private member is to get the Event's invocation list. I've used it in several cases to fire the event in an async manner to all callers (prevents Listener A from

modifying the event before Listener B receives it; also it prevents Listener A from delaying the delivery to Listener B). I've used this a lot in custom data sync scenarios, and also in APIs. – STW May 5, 2009 at 14:31

14 votes

If you need a variable name to match that of a keyword, enclose it with brackets. Not nec. the best practice though - but it can be used wisely.

e.g.

```
Class CodeException
Public [Error] as String
''...
End Class

''later
Dim e as new CodeException
e.Error = "Invalid Syntax"
```

e.g. Example from comments(@Pondidum):

```
Class Timer
Public Sub Start()
''...
End Sub

Public Sub [Stop]()
''...
End Sub
```

Share

edited Nov 20, 2009 at 23:26

answered Sep 19, 2008 at 14:48

torial

13.1k ● 9 ● 65 ● 89

I think this example would be better if you didn't use "If" as the example keyword. – Sean Gough Sep 19, 2008 at 18:53

4 timer.Start and timer.Stop spring to mind as examples of good use of this – Pondidum Dec 26, 2008 at 20:13

5 +1 for pointing it out with a disclaimer. There are several framework classes that require this to resolve correctly, such as [Assembly] – STW May 5, 2009 at 14:36

5 [Enum] is another good example of a case where you need the brackets in order to use the class instead of the keyword. – Ryan Lundy Dec 17, 2009 at 22:30

13 votes

There are a couple of answers about XML Literals, but not about this specific case:

You can use XML Literals to enclose string literals that would otherwise need to be escaped. String literals that contain double-quotes, for instance.

Instead of this:

```
Dim myString = _
    "This string contains ""quotes"" and they're ugly."
```

You can do this:

```
Dim myString = _
    <string>This string contains "quotes" and they're ni
```

This is especially useful if you're testing a literal for CSV parsing:

```
Dim csvTestYuck = _
    """Smith"", ""Bob"", ""123 Anywhere St"", ""Los Ange

Dim csvTestMuchBetter = _
    <string>"Smith", "Bob", "123 Anywhere St", "Los Ange
</string>.Value
```

(You don't have to use the `<string>` tag, of course; you can use any tag you like.)

Share

answered Dec 17, 2009 at 22:50

community wiki
Ryan Lundy

3 `<q>` would be a good tag, similar to usage in Perl/Ruby. Anyway, that's quite a nice idiom. LIKE! – Konrad Rudolph May 12, 2010 at 15:46

12 votes

DateTime can be initialized by surrounding your date with #

```
Dim independanceDay As DateTime = #7/4/1776#
```

You can also use type inference along with this syntax

```
Dim independanceDay = #7/4/1776#
```

That's a lot nicer than using the constructor

```
Dim independanceDay as DateTime = New DateTime(1776, 7,
```

Share

answered Jun 29, 2009 at 14:51

community wiki
danlash

6 Not if you have Option Strict On – danlash Sep 21, 2009 at 20:35

12

votes

You can have 2 lines of code in just one line. hence:

```
Dim x As New Something : x.CallAMethod
```

Share

edited Nov 27, 2009 at 7:58

community wiki
2 revs, 2 users 77%
Parsa

whoa... i thought this was only possible w/class and inheritance – Jason Jun 2, 2009 at 16:25

Don't forget `Call (New Something).CallAMethod()` – Jonathan Allen Jul 18, 2010 at 13:18

This is a holdever from MS-Basic on the Apple][! In my shop I would be just as ridiculed for using Gotos :-/ – FastAl Sep 7, 2010 at 19:07

Most of the time this is to be avoided but where I like to use it is in case statements where the lines are really short e.g. Case 4:x=22 – dwidel Dec 30, 2010 at 16:49

11

votes

Optional Parameters

Optionals are so much easier than creating a new overloads, such as :

```
Function CloseTheSystem(Optional ByVal msg AS String = '
system...")
   Console.Writeline(msg)
   ''//do stuff
End Function
```

Share

edited Jun 2, 2009 at 16:06

Joel Coehoorn

415k ● 114 ● 577 ● 813

answered Nov 27, 2008 at 16:24

dr. evil

27.2k ● 37 ● 134 ● 202

1 I wasn't aware that C# was going to get them. They are nice, but you have to be careful to use them *only* where you're sure the code won't be consumed by C# since it doesn't support them. FxCop/Code Analysis will tell you to overload the method instead. – STW May 5, 2009 at 14:34

...I just found a great use for Optional parameters, while still keeping them out of production code. I wrote-up a short post about it on my site: yoooder.com/wordpress/?p=62 – STW Aug 21, 2009 at 20:30

2 Ah, I despise this so much... But useful for Office automation – dance2die Oct 20, 2009 at 2:39

9

votes

Title Case in VB.Net can be achieved by an old VB6 fxn:

```
StrConv(stringToTitleCase, VbStrConv.ProperCase,0) ''0 i
```

Share

answered Sep 19, 2008 at 14:51

torial

13.1k ● 9 ● 65 ● 89

1 its also in the textinfo class. not sure what namespace that is in. probably system.text – Shawn Oct 13, 2008 at 4:23

You are better of to be .net language neutral and use the Globalization TextInfo class to convert ToTitleCase. If you code ever needs to be converted to C# you'll have lots of little nasties that require Microsoft.VisualBasic – Jeremy Jun 29, 2011 at 4:03

9

votes

Properties with parameters

I have been doing some C# programming, and discovered a feature that was missing that VB.Net had, but was not mentioned here.

An example of how to do this (as well as the c# limitation) can be seen at: [Using the typical get set properties in C#... with parameters]

I have excerpted the code from that answer:

```
Private Shared m_Dictionary As IDictionary(Of String, Ob
             New Dictionary(Of String, Object)

Public Shared Property DictionaryElement(ByVal Key As St
    Get
        If m_Dictionary.ContainsKey(Key) Then
            Return m_Dictionary(Key)
        Else
            Return [String].Empty
        End If
    End Get
    Set(ByVal value As Object)
        If m_Dictionary.ContainsKey(Key) Then
            m_Dictionary(Key) = value
        Else
            m_Dictionary.Add(Key, value)
        End If

    End Set
End Property
```

Share

edited May 23, 2017 at 10:30

community wiki
3 revs, 2 users 89%
torial

This is interesting, but I'm not entirely sure where this would be useful. myObj.Something("abc") looks more like you're accessing a Function than a property. Not sure what this buys you. – mattmc3 Jul 22, 2010 at 0:06

I hate ambiguity. What should it be. A method, or a property. Some refactoring tools suggest creating both in certain situations too, looks like they dont even know... – brumScouse Aug 25, 2010 at 21:57