



.NET Development (General) Technical Articles

## Overloading Methods in Visual Basic .NET

Â

Paul D. Sheriff  
PDSA, Inc.

November 2001

**Summary:** This article discusses how Microsoft Visual Basic .NET figures out which method to call during compile based on the parameter types that you pass. This technique is called *overloading* a method. (8 printed pages)

### Objectives

- Learn why to overload a method
- Create overloaded methods
- Create your own constructors for a class

### Assumptions

The following should be true for you to get the most out of this document:

- You understand OOP techniques
- You know how to build classes in Microsoft .NET

### Contents

[Overloading a Method](#)

[What's New Since Visual Basic 6.0?](#)

[Summary](#)

## Overloading a Method

Have you ever wanted to create a group of functions that each did essentially the same thing, with only their parameters changed? You can accomplish this by using optional parameters, or by changing the name of the function. Both techniques work, but they are not very elegant. In an OOP language such as Microsoft® Visual Basic® .NET, you are allowed to create methods in a class that have the same name but different argument lists. Visual Basic .NET can figure out which method to call during compile based on the parameter types that you pass. This technique is called *overloading* a method. The benefit of using the same name is that the user interface is kept consistent; only the inputs are different. The functionality within the method changes for each new overloaded method.

The Line class you created in the document "Creating Classes" uses a **GetWord** method to return the first word in its line of text. In this document, you will create two more versions of this **GetWord** method. In the first version, you will pass in a number representing the position of the word you wish to retrieve from the line. In the second version, you will pass in a search string and the method will return the first word in the line of text that contains that string.

The *signature* of a method consists of its name, parameters, and return type—in other words, the arguments that differentiate one "flavor" of the method from another. The signatures for these three methods look like the following lines of code:

 [Copy Code](#)

```
Function GetWord() As String  
Function GetWord(ByVal Position As Integer) As String  
Function GetWord(ByVal Search As String) As String
```

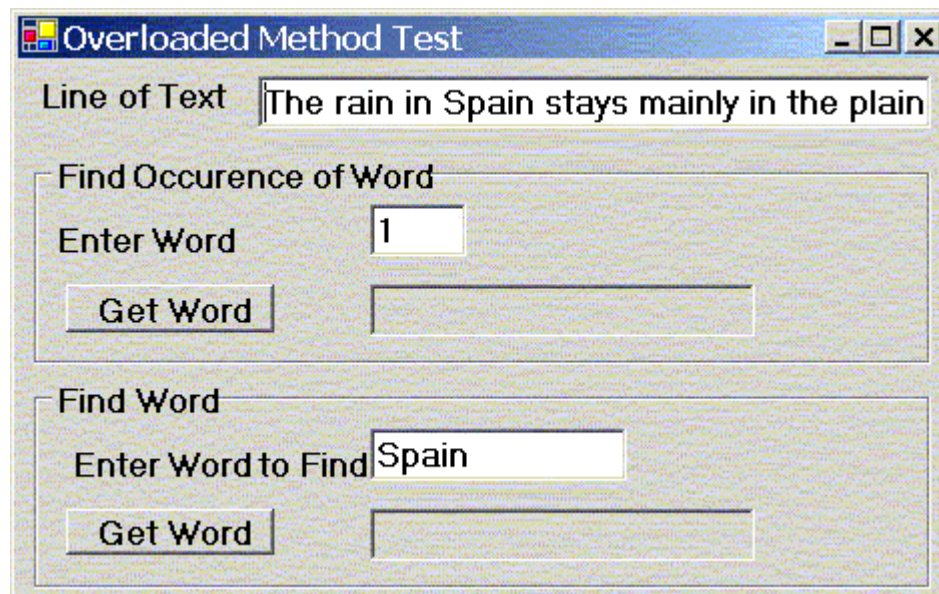
The name of the method (**GetWord**) stays the same in each of the above methods, but the argument list is different for each one. The compiler can resolve each of these names based on these different "signatures." Notice that the return value is the same for each method. Keeping the return value the same is not a requirement, but you normally will not change the return value on overloaded methods. Changing only the return type on a method does not overload a method. If you try to do this, Visual Basic .NET gives you an error message telling you that you cannot overload a method by changing the return type. You must change at least one argument for a method to have a different signature and be overloaded.

### Reasons to Overload a Method

Overloading a method allows you to keep your interface consistent, and allows you to logically call the same method regardless of the types of data you are passing in. You will find that using the same name will help you remember what a procedure does, as opposed to having to come up with new names or a naming convention, to help you keep things straight.

### Build Two Overloaded Methods

Let's create two new **GetWord** methods that overload the original **GetWord** method in the Line class. The first new **GetWord** method will accept an integer parameter and return the word at the position that you pass to the method. The second **GetWord** method will accept a string parameter, and will search for that string within the original line of text. If the search string is found, the word at that position will be returned from this method.



**Figure 1. Screen used to test the overloading of methods**

1. Create a new Windows Application project using Visual Studio .NET. Or, if you read the document called "Creating Classes," you can use that same project, and follow the steps below to change the Windows Form to work with this project.
2. Create the form shown in Figure 1 by adding the appropriate controls and setting the properties of those controls, as outlined in Table 1.

**Table 1. Controls used to build the form to test the GetWord overload method in the Line class**

Control Type	Property	Value
<b>Label</b>	Name	Label1
Â	Text	Line of Text
<b>TextBox</b>	Name	txtLine
Â	Text	The rain in Spain stays mainly in the plain
<b>CommandButton</b>	Name	btnDisplay
Â	Text	Display Length
<b>GroupBox</b>	Name	fraWord
Â	Text	Find Occurrence of Word
<b>Label</b>	Name	Label2
Â	Text	Enter Word
<b>TextBox</b>	Name	txtWordNum
Â	Text	1
<b>CommandButton</b>	Name	btnGetWord
Â	Text	Get Word
<b>TextBox</b>	Name	txtWord
Â	Text	Â
Â	ReadOnly	True
<b>GroupBox</b>	Name	fraWord2
Â	Text	Find Word
<b>Label</b>	Name	Label3
Â	Text	Enter Word to Find
<b>TextBox</b>	Name	txtSearch
Â	Text	Spain
<b>CommandButton</b>	Name	btnGetWord2
Â	Text	Get Word
<b>TextBox</b>	Name	txtWord2
Â	Text	Â
Â	ReadOnly	True

- From the Visual Studio .NET menu, add a new class by clicking **Project** and then clicking **Add Class**.
- When prompted, set the Name of this class to Line.vb and click **OK**.
- Add the following code to make this Line class:

 [Copy Code](#)

---

```
Public Class Line
    Private mstrLine As String

    Property Line() As String
        Get
            Return mstrLine
        End Get
        Set
            mstrLine = Value
        End Set
    End Property

    ReadOnly Property Length() As Integer
        Get
            Return mstrLine.Length
        End Get
    End Property

    Public Function GetWord() As String
        Dim astrWords() As String

        astrWords = mstrLine.Split(" ".ToCharArray())

        Return astrWords(0)
    End Function
End Class
```

6. Change the **GetWord** method and add the keyword **Overloads**, as shown in the code below.

 [Copy Code](#)

---

```
Public Overloads Function GetWord() As String
```

The **Overloads** keyword is required whenever you create several versions of the same method. You will be creating additional versions of the **GetWord** method next.

7. Create a new **GetWord** method that will accept an **Integer** value and still return a string. The declaration is in the code listing below.

 [Copy Code](#)

---

```
Public Overloads Function GetWord(_
    ByVal Position As Integer) As String
    Dim astrWords() As String

    astrWords = mstrLine.Split(" ".ToCharArray())

    If Position > astrWords.Length Then
        Return ""
    Else
        Return astrWords(Position - 1)
    End If
End Function
```

The **GetWord** method converts the string to an array of individual words using the **Split** method of the

**String** class. Because the `mstrLine` variable is defined as a string object, it has a **Split** method. This **Split** method takes a Char array of delimiters. Instead of declaring a Char array variable and populating it with a single space, you can take advantage of a technique known as *boxing*.

Boxing is the process of converting a literal, like a single space, to the implied object (in this case, a string object) for that literal. Once this boxing has occurred, you can apply the **ToCharArray** method of this newly created string object to return an array of one character. This single element array is passed to the **Split** method to use as the delimiter that breaks apart the words in the string.

Now that you have an array of words, you can return the word that corresponds to the number you passed into this method. Of course, you should always check to make sure that the number you passed is not greater than the number of words in the array.

### Try It Out

1. Open the `frmLineTest.vb` form.
2. Double-click the first **Get Word** button on the form and add the following code to the Click event.

 [Copy Code](#)

---

```
Protected Sub btnGetWord_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles btnGetWord.Click
    Dim oLine As Line = New Line()

    oLine.Line = txtLine.Text
    txtWord.Text = _
        oLine.GetWord(txtWordNum.Text.ToInt32())
End Sub
```

Because this version of the **GetWord** method expects an Integer data type, you need to use the **ToInt32** method to convert the value in the **Text** property to an **Integer**.

1. To run the project, place a number in the first text box, and click the **Get Word** button. Depending on the number you typed in, a word should now appear in the read-only text box next to this first button.
2. Add another new version of the **GetWord** method that will accept a search string as an argument. This version of **GetWord** needs to search for that string within the line of text, and return the word where that string is found.

 [Copy Code](#)

---

```
Public Overloads Function GetWord(_
    ByVal Search As String) As String
    Dim astrWords() As String
    Dim intLoop As Integer

    astrWords = mstrLine.Split(" ".ToCharArray())

    For intLoop = 0 To astrWords.Length - 1
        If astrWords(intLoop).IndexOf(Search) > -1 Then
            Exit For
        End If
    Next
    If intLoop < astrWords.Length Then
        Return astrWords(intLoop)
    End If
End Function
```

In this method, after you create the array of words, you loop through the array and check to see if the search string you passed is contained in any word in the array. To accomplish this, you use the **IndexOf** method of the string to see whether the string you are searching for is part of the string. If the **IndexOf** method finds the search string, it will return a number greater than 0. Once you have found the string, the **For** loop exits, and the word at that position is returned from this method.

### Try It Out

1. Open the frmLineTest.vb form.
2. Double-click the second **Get Word** button and add the following code to the Click event procedure for this button.

 [Copy Code](#)

```
Protected Sub btnGetWord2_Click(ByVal sender As Object, _  
    ByVal e As System.EventArgs) Handles btnGetWord2.Click  
    Dim oLine As Line = New Line()  
  
    oLine.Line = txtLine.Text  
    txtWord2.Text = _  
        oLine.GetWord(txtSearch.Text)  
End Sub
```

In the btnGetWord2\_Click event procedure, you pass the search string to the **GetWord** method. The **Text** property returns a **String** data type, so the **GetWord** method that accepts the **String** data type is the method that is called.

3. To run the project, place a few characters into the second text box on this form, and then click the second **Get Word** button. Depending on the characters you typed in, you should see a word appear in the text box next to the button, or a blank string.

### Overloaded Constructors

When you instantiate a new object, Visual Basic .NET calls an implicit **New** method. If you don't write the **New** method yourself, Visual Basic .NET builds one for you. If you wish to initialize any variables within an object at the time you create the new object, you can write an overloaded **New** method. You can then pass one or more values to this **New** method. You will probably find that overloading the **New** method is your most common use of overloading.

1. Open the Line.vb class module and add the code shown below.

 [Copy Code](#)

```
Public Sub New()  
    ' Do Nothing  
End Sub  
  
Public Sub New(ByVal Line As String)  
    mstrLine = Line  
End Sub
```

The first **New** method you created handles the creation of a new **Line** object when you do not pass any parameters to it. You need to use the **Overloads** keyword because you will also be creating another **New** method with a parameter.

The other **New** method accepts a string argument. This string argument will be passed when the object is

created and assigned to the private variable that holds the line of text.

### Try It Out

1. Open the frmLineTest.vb form.
2. Double-click on either of the GetWord command buttons.
3. Change the code to match the code below.

 [Copy Code](#)

```
Protected Sub btnGetWord_Click(ByVal sender As Object, _  
    ByVal e As System.EventArgs) Handles btnGetWord.Click  
    Dim oLine As Line = New Line(txtLine.Text)  
    txtWord.Text = _  
        oLine.GetWord(txtWordNum.Text.ToInt32())  
End Sub
```

4. Press **F5** to try out this sample.

Notice that the declaration of the **oLine** object now passes the value from the **txtLine** text box. This is different from Visual Basic 6.0 where a **Dim** statement was not executable code. The **Dim** statement is now executable.

### What's New Since Visual Basic 6.0?

The ability to overload methods is completely new to Visual Basic .NET and has no equivalent in previous versions of Visual Basic.

### Summary

Overloading methods is a great way to keep the intent of a method clear while allowing different parameters to affect the value you receive back from the method. Remember that you must change the parameters, and not just the return type to overload a method. When deciding which methods to overload, the intent of the methods should stay consistent. Just because you can overload a method does not necessarily mean you should. Always think about the intent of the method and whether the various methods you create do essentially the same thing.

### About the Author

[Paul D. Sheriff](#) is the owner of PDSA, Inc., a custom software development and consulting company in Southern California. Paul is the MSDN Regional Director for Southern California, is the author of a book on Visual Basic 6 called *Paul Sheriff Teaches Visual Basic*, and has produced over 72 videos on Visual Basic, SQL Server, .NET and Web Development for Keystone Learning Systems. Paul has co-authored a book entitled [ASP.NET Jumpstart](#) [ <http://service.bfast.com/bfast/click?bfmid=43945&sourceid=0039384188&bfpid=0672323575&bfmtype=Book> ]. Visit the PDSA, Inc. Web site ([www.pdsa.com](http://www.pdsa.com)) for more information.

### About Informant Communications Group

Informant Communications Group, Inc. ([www.informant.com](http://www.informant.com)) is a diversified media company focused on the information technology sector. Specializing in software development publications, conferences, catalog publishing and Web sites, ICG was founded in 1990. With offices in the United States and the United Kingdom, ICG has served as a respected media and marketing content integrator, satisfying the burgeoning appetite of IT professionals for quality technical information.

Copyright © 2001 Informant Communications Group and Microsoft Corporation

Technical Editing: PDSA, Inc.