Special Characters in Code

(Visual Basic)

Sometimes you have to use special characters in your code, that is, characters that are not alphabetical or numeric. The punctuation and special characters in the Visual Basic character set have various uses, from organizing program text to defining the tasks that the compiler or the compiled program performs. They do not specify an operation to be performed.

Parentheses

Use parentheses when you define a procedure, such as a Sub or Function. You must enclose all procedure argument lists in parentheses. You also use parentheses for putting variables or arguments into logical groups, especially to override the default order of operator precedence in a complex expression. The following example illustrates this.

```
VB
Dim a, b, c, d, e As Double
a = 3.2
b = 7.6
c = 2
d = b + c / a
e = (b + c) / a
```

Following execution of the previous code, the value of d is 8.225 and the value of e is 3. The calculation for d uses the default precedence of / over + and is equivalent to d = b + (c / a). The parentheses in the calculation for e override the default precedence.

Separators

Separators do what their name suggests: they separate sections of code. In Visual Basic, the separator character is the colon (:). Use separators when you want to include multiple statements on a single line instead of separate lines. This saves space and improves the readability of your code. The following example shows three statements separated by colons.

```
VB
a = 3.2 : b = 7.6 : c = 2
```

For more information, see How to: Break and Combine Statements in Code (Visual Basic).

The colon (:) character is also used to identify a statement label. For more information, see How to: Label Statements (Visual Basic).

Concatenation
Use the & operator for concatenation, or linking strings together. Do not confuse it with the + operator, which adds together numeric values. If you use the + operator to concatenate when you operate on numeric values, you can obtain incorrect results. The following example demonstrates this.

VB
```
var1 = "10.01"
var2 = 11
resultA = var1 + var2
resultB = var1 & var2
```
Following execution of the previous code, the value of resultA is 21.01 and the value of resultB is "10.0111".
Member Access Operators
To access a member of a type, you use the dot (.) or exclamation point (!) operator between the type name and the member name.
Dot (.) Operator
Use the . operator on a class, structure, interface, or enumeration as a member access operator. The member can be a field, property, event, or method. The following example illustrates this.

VB
```
Dim nextForm As New System.Windows.Forms.Form
' Access Text member (property) of Form class (on nextForm object).
nextForm.Text = "This is the next form"
' Access Close member (method) on nextForm.
nextForm.Close()
```
Exclamation Point (!) Operator
Use the ! operator only on a class or interface as a dictionary access operator. The class or interface must have a default property that accepts a single String argument. The identifier immediately following the ! operator becomes the argument value passed to the default property as a string. The following example demonstrates this.

VB
```
Public Class hasDefault
  Default Public ReadOnly Property index(ByVal s As String) As Integer
    Get
      Return 32768 + AscW(s)
    End Get
  End Property
```

```
End Class
Public Class testHasDefault
  Public Sub compareAccess()
    Dim hD As hasDefault = New hasDefault()
    MsgBox("Traditional access returns " & hD.index("X") & vbCrLf &
      "Default property access returns " & hD("X") & vbCrLf &
      "Dictionary access returns " & hD!X)
  End Sub
End Class
```

The three output lines of MsgBox all display the value 32856. The first line uses the traditional access to property index, the second makes use of the fact that index is the default property of class hasDefault, and the third uses dictionary access to the class.

Note that the second operand of the ! operator must be a valid Visual Basic identifier not enclosed in double quotation marks (" "). In other words, you cannot use a string literal or string variable. The following change to the last line of the MsgBox call generates an error because "X" is an enclosed string literal.

```
"Dictionary access returns " & hD!"X")
```