Visual Basic Language Reference

Do...Loop Statement (Visual Basic)

Repeats a block of statements while a **Boolean** condition is **True** or until the condition becomes **True**.

Do { While | Until } condition
    [ statements ]
    [ Exit Do ]
    [ statements ]
Loop

-or-

Do
    [ statements ]
    [ Exit Do ]
    [ statements ]
Loop { While | Until } condition

Parts

**While**
   Required unless **Until** is used. Repeat the loop until condition is **False**.

**Until**
   Required unless **While** is used. Repeat the loop until condition is **True**.

**condition**
   Optional. **Boolean** expression. If condition is **Nothing**, Visual Basic treats it as **False**.

**statements**
   Optional. One or more statements that are repeated while, or until, condition is **True**.

**Exit Do**
   Optional. Transfers control out of the **Do** loop.

**Loop**
   Required. Terminates the definition of the **Do** loop.

Remarks

Use a **Do...Loop** structure when you want to repeat a set of statements an indefinite number of times, until a condition is satisfied. If you want to repeat the statements a set number of times, the **For...Next Statement** is usually a better choice.

The **Do...Loop** structure gives you more flexibility than the **While...End While Statement (Visual Basic)** because it allows you to choose whether to end the loop when condition stops being **True** or when it first becomes **True**. It also allows you to test condition at either the beginning or the end of the loop.

Rules

- **Nature of Condition.** The condition usually results from a comparison of two values, but it can be any expression that evaluates to a **Boolean Data Type (Visual Basic)** value (**True** or **False**). This includes values of other data types, such as numeric types, that have been converted to **Boolean**.
- **Testing the Condition.** You can test condition only once, at either the beginning or the end of the loop. You can use either **While** or **Until** to specify condition, but not both.
- **Number of Iterations.** If you test condition at the beginning of the loop (in the **Do** statement), the loop might never run even once. If you test at the end of the loop (in the **Loop** statement), the loop always runs at least once.
Nesting Loops. You can nest **Do** loops by placing one loop within another. You can also nest different kinds of control structures within one another. For more information, see [Nested Control Structures](#).

Transferring Out of the Loop. The [Exit Statement](#) transfers control immediately to the statement following the **Loop** statement. You might want to exit a loop if you detect a condition that makes it unnecessary or impossible to continue iterating, such as an erroneous value or a termination request. You can place any number of **Exit Do** statements anywhere in the **Do** loop. **Exit Do** is often used after evaluating some condition, for example in an **If...Then...Else** structure.

Endless Loops

One use of **Exit Do** is to test for a condition that could cause an *endless loop*, which is a loop that could run an extremely large or even infinite number of times. If you detect such a condition, you can use **Exit Do** to escape the loop. Otherwise, the loop continues running.

In the following example, **number** is assigned a value that could cause the loop to run more than $2^{31}$ times. The **If** statement checks for this condition and exits if it exists, preventing endless looping.

Visual Basic  Copy Code

```vbnet
Sub exitDoExample()
    Dim counter As Integer = 0
    Dim number As Integer = 8
    Do Until number = 10
        If number <= 0 Then Exit Do
        number -= 1
        counter += 1
    Loop
    MsgBox("The loop ran " & counter & " times.")
End Sub
```

**Note:**

to stop an endless loop, press ESC or CTRL+BREAK.

Example

The following example illustrates nested **Do**..**Loop** structures, as well as the use of **While** and **Until**, and testing at the beginning (**Do** statement) and end (**Loop** statement) of the loop.

Visual Basic  Copy Code

```vbnet
Sub DoExample()
    Dim check As Boolean = True
    Dim counter As Integer = 0
    Do
        Do While counter < 20
            counter += 1
            If counter = 10 Then check = False
        Exit Do
        End If
    Loop
    Loop Until check = False
End Sub
```
In the preceding example, the inner `Do...Loop` structure loops 10 times, sets the value of the flag to `False`, and exits prematurely using the `Exit Do` statement. The outer loop exits immediately upon checking the value of the flag.

See Also

**Concepts**
- Loop Structures
- Nested Control Structures

**Reference**
- `For...Next Statement (Visual Basic)`
- `Boolean Data Type (Visual Basic)`
- `Exit Statement (Visual Basic)`
- `While...End While Statement (Visual Basic)`