For...Next Statement (Visual Basic)

Repeats a group of statements a specified number of times.

```
For counter [ As datatype ] = start To end [ Step step ]
    [ statements ]
    [ Exit For ]
    [ statements ]
Next [ counter ]
```

**Parts**

- **counter**: Required in the `For` statement. Numeric variable. The control variable for the loop.
- **datatype**: Required if `counter` is not already declared. Data type of `counter`.
- **start**: Required. Numeric expression. The initial value of `counter`.
- **end**: Required. Numeric expression. The final value of `counter`.
- **step**: Optional. Numeric expression. The amount by which `counter` is incremented each time through the loop.
- **statements**: Optional. One or more statements between `For` and `Next` that run the specified number of times.
- **Exit For**: Optional. Transfers control out of the `For` loop.
- **Next**: Required. Terminates the definition of the `For` loop.

**Remarks**

Use a `For...Next` structure when you want to repeat a set of statements a set number of times.

A `While...End While Statement (Visual Basic)` or `Do...Loop Statement (Visual Basic)` works well when you do not know in advance how many times you need to run the statements in the loop. However, when you expect to run the loop a specific number of times, a `For...Next` loop is a better choice. You determine the number of iterations when you first enter the loop.

The value of `step` can be either positive or negative. It determines loop processing as follows:

<table>
<thead>
<tr>
<th>Step value</th>
<th>Loop executes if</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive or zero</td>
<td><code>counter &lt;= end</code></td>
</tr>
<tr>
<td>Negative</td>
<td><code>counter &gt;= end</code></td>
</tr>
</tbody>
</table>

If not specified, `step` defaults to 1.

**Rules**

- **Data Types.** The data type of `counter` is usually `Integer` but can be any type that supports the greater than or equal to (`>=`), less than or equal to (`<=`), addition (`+`), and subtraction (`-`) operators. It can even be a user-defined type provided it supports all these operators.

  The `start`, `end`, and `step` expressions usually evaluate to type `Integer` but can evaluate to any data type that widens to the type of `counter`. If you use a user-defined type for `counter`, this means you might have to define the `CType` conversion operator to convert the types of `start`, `end`, or `step` to the type of `counter`. 
• Declaration. If *counter* has not been declared outside this loop, you must declare it within the **For** statement. In this case, the scope of *counter* is the body of the loop. However, you cannot declare *counter* both outside and inside the loop.

• Number of Iterations. Visual Basic evaluates the iteration values *start*, *end*, and *step* only once, before the loop begins. If your statement block changes *end* or *step*, these changes do not affect the iteration of the loop.

• Nesting Loops. You can nest **For** loops by placing one loop within another. However, each loop must have a unique *counter* variable. The following construction is valid.

```visual-basic
For i As Integer = 1 To 10
    For j As Integer = 1 To 10
        For k As Integer = 1 To 10
            ' Insert statements to operate with current values of i, j, and k.
        Next k
    Next j
Next i
```

You can also nest different kinds control structures within one another. For more information, see 'Nested Control Structures'.

**Note:**

If a **Next** statement of an outer nesting level is encountered before the **Next** of an inner level, the compiler signals an error. However, the compiler can detect this overlapping error only if you specify *counter* in every **Next** statement.

• Identifying the Control Variable. You can optionally specify *counter* in the **Next** statement. This improves the readability of your program, especially if you have nested **For** loops. You must specify the same variable as the one that appears in the corresponding **For** statement.

• Transferring Out of the Loop. The **Exit Statement (Visual Basic)** transfers control immediately to the statement following the **Next** 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. Also, if you catch an exception in a **Try...Catch...Finally**, you can use **Exit For** at the end of the **Finally** block.

You can place any number of **Exit For** statements anywhere in the **For** loop. **Exit For** is often used after evaluating some condition, for example in an **If...Then...Else** structure.

• Endless Loops. One use of **Exit For** 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 For** to escape the loop. For more information, see **Do...Loop Statement (Visual Basic)**.

**Behavior**

• Entry into the Loop. When execution of the **For...Next** loop begins, Visual Basic evaluates *start*, *end*, and *step* for the only time. It then assigns *start* to *counter*. Before it runs the statement block, it compares *counter* to *end*. If *counter* is already past the end value, the **For** loop terminates and control passes to the statement following the **Next** statement. Otherwise the statement block runs.

• Iterations of the Loop. Each time Visual Basic encounters the **Next** statement, it increments *counter* by *step* and returns to the **For** statement. Again it compares *counter* to *end*, and again it either runs the block or terminates the loop depending on the result. This process continues until *counter* passes *end* or an **Exit For** statement is encountered.

• Termination of the Loop. The loop does not terminate until *counter* has passed *end*. If *counter* is equal to *end*, the loop continues. The comparison that determines whether to run the block is *counter* <= *end* if *step* is positive and *counter* >= *end* if *step* is negative.

• Changing Iteration Values. Changing the value of *counter* while inside a loop can make it more difficult to read and debug your code. Changing the value of *start*, *end*, or *step* does not affect the iteration values determined when the loop was first entered.
Example

The following example demonstrates nested `For...Next` structures with different step values.

Visual Basic  

```vbnet
dim words, digit as integer
Dim thisString as string = ""
For words = 10 to 1 step -1
    For digit = 0 to 9
        thisString &= CStr(digit)
    Next digit
    thisString &= " 
Next words
```

The preceding example creates a string that contains 10 instances of the numbers 0 through 9, each string separated from the other by a single space. The outer loop decrements a loop counter variable each time through the loop.

See Also

Concepts
- Loop Structures
- Nested Control Structures

Reference
- While...End While Statement (Visual Basic)
- Do...Loop Statement (Visual Basic)
- Exit Statement (Visual Basic)
- For Each...Next Statement (Visual Basic)