Control Flow in Visual Basic

Visual Studio 2015

Left unregulated, a program proceeds through its statements from beginning to end. Some very simple programs can be written with only this unidirectional flow. However, much of the power and utility of any programming language comes from the ability to change execution order with control statements and loops.

Control structures allow you to regulate the flow of your program's execution. Using control structures, you can write Visual Basic code that makes decisions or that repeats actions. Other control structures let you guarantee disposal of a resource or run a series of statements on the same object reference.

In This Section

Decision Structures (Visual Basic)
: Describes control structures used for branching.

Loop Structures (Visual Basic)
: Discusses control structures used to repeat processes.

Other Control Structures (Visual Basic)
: Describes control structures used for resource disposal and object access.

Nested Control Structures (Visual Basic)
: Covers control structures inside other control structures.

Related Sections

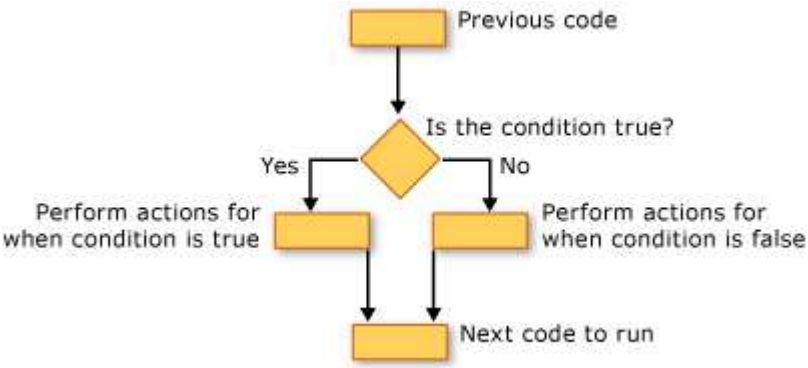
Control Flow Summary (Visual Basic)
: Provides links to language reference pages on this subject.

Decision Structures (Visual Basic)

Visual Studio 2015

Visual Basic lets you test conditions and perform different operations depending on the results of that test. You can test for a condition being true or false, for various values of an expression, or for various exceptions generated when you execute a series of statements.

The following illustration shows a decision structure that tests for a condition being true and takes different actions depending on whether it is true or false.



Taking different actions when a condition is true and when it is false

If...Then...Else Construction

If...Then...Else constructions let you test for one or more conditions and run one or more statements depending on each condition. You can test conditions and take actions in the following ways:

- Run one or more statements if a condition is **True**
- Run one or more statements if a condition is **False**
- Run some statements if a condition is **True** and others if it is **False**
- Test an additional condition if a prior condition is **False**

The control structure that offers all these possibilities is the If...Then...Else Statement (Visual Basic). You can use a single-line version if you have just one test and one statement to run. If you have a more complex set of conditions and actions, you can use the multiple-line version.

Select...Case Construction

The **Select...Case** construction lets you evaluate an expression one time and run different sets of statements based on different possible values. For more information, see Select...Case Statement (Visual Basic).

Try...Catch...Finally Construction

Try...Catch...Finally constructions let you run a set of statements under an environment that retains control if any one of your statements causes an exception. You can take different actions for different exceptions. You can optionally specify a block of code that runs before you exit the whole **Try...Catch...Finally** construction, regardless of what occurs. For more information, see Try...Catch...Finally Statement (Visual Basic).

Note
For many control structures, when you click a keyword, all of the keywords in the structure are highlighted. For instance, when you click **If** in an **If...Then...Else** construction, all instances of **If**, **Then**, **ElseIf**, **Else**, and **End If** in the construction are highlighted. To move to the next or previous highlighted keyword, press CTRL+SHIFT+DOWN ARROW or CTRL+SHIFT+UP ARROW.

See Also

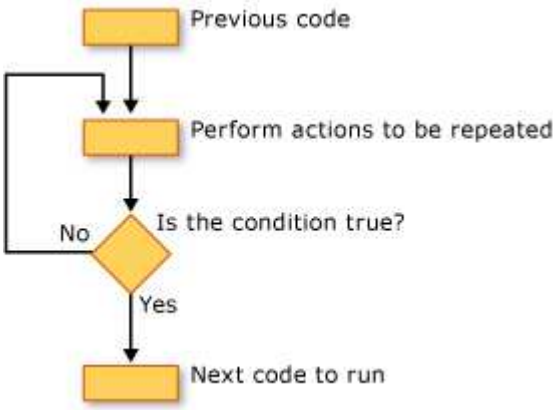
Control Flow in Visual Basic
Loop Structures (Visual Basic)
Other Control Structures (Visual Basic)
Nested Control Structures (Visual Basic)
If Operator (Visual Basic)

Loop Structures (Visual Basic)

Visual Studio 2015

Visual Basic loop structures allow you to run one or more lines of code repetitively. You can repeat the statements in a loop structure until a condition is **True**, until a condition is **False**, a specified number of times, or once for each element in a collection.

The following illustration shows a loop structure that runs a set of statements until a condition becomes true.



Running a set of statements until a condition becomes true

While Loops

The **While**...**End While** construction runs a set of statements as long as the condition specified in the **While** statement is **True**. For more information, see While...End While Statement (Visual Basic).

Do Loops

The **Do**...**Loop** construction allows you to test a condition at either the beginning or the end of a loop structure. You can also specify whether to repeat the loop while the condition remains **True** or until it becomes **True**. For more information, see Do...Loop Statement (Visual Basic).

For Loops

The **For**...**Next** construction performs the loop a set number of times. It uses a loop control variable, also called a *counter*, to keep track of the repetitions. You specify the starting and ending values for this counter, and you can optionally specify the amount by which it increases from one repetition to the next. For more information, see For...Next Statement (Visual Basic).

For Each Loops

The **For Each**...**Next** construction runs a set of statements once for each element in a collection. You specify the loop control variable, but you do not have to determine starting or ending values for it. For more information, see For Each...Next Statement (Visual Basic).

See Also

Control Flow in Visual Basic
Decision Structures (Visual Basic)
Other Control Structures (Visual Basic)
Nested Control Structures (Visual Basic)

Other Control Structures (Visual Basic)

Visual Studio 2015

Visual Basic provides control structures that help you dispose of a resource or reduce the number of times you have to repeat an object reference.

Using...End Using Construction

The **Using...End Using** construction establishes a statement block within which you make use of a resource such as a SQL connection. You can optionally acquire the resource with the **Using** statement. When you exit the **Using** block, Visual Basic automatically disposes of the resource so that it is available for other code to use. The resource must be local and disposable. For more information, see Using Statement (Visual Basic).

With...End With Construction

The **With...End With** construction lets you specify an object reference once and then run a series of statements that access its members. This can simplify your code and improve performance because Visual Basic does not have to re-establish the reference for each statement that accesses it. For more information, see With...End With Statement (Visual Basic).

See Also

Control Flow in Visual Basic
Decision Structures (Visual Basic)
Loop Structures (Visual Basic)
Nested Control Structures (Visual Basic)
Using Statement (Visual Basic)
With...End With Statement (Visual Basic)

Nested Control Structures (Visual Basic)

Visual Studio 2015

You can place control statements inside other control statements, for example an **If...Then...Else** block within a **For...Next** loop. A control statement placed inside another control statement is said to be *nested*.

Nesting Levels

Control structures in Visual Basic can be nested to as many levels as you want. It is common practice to make nested structures more readable by indenting the body of each one. The integrated development environment (IDE) editor automatically does this.

In the following example, the procedure `sumRows` adds together the positive elements of each row of the matrix.

```
Public Sub sumRows(ByVal a(,) As Double, ByRef r() As Double)
    Dim i, j As Integer
    For i = 0 To UBound(a, 1)
        r(i) = 0
        For j = 0 To UBound(a, 2)
            If a(i, j) > 0 Then
                r(i) = r(i) + a(i, j)
            End If
        Next j
    Next i
End Sub
```

In the preceding example, the first **Next** statement closes the inner **For** loop and the last **Next** statement closes the outer **For** loop.

Likewise, in nested **If** statements, the **End If** statements automatically apply to the nearest prior **If** statement. Nested **Do** loops work in a similar fashion, with the innermost **Loop** statement matching the innermost **Do** statement.

Note

For many control structures, when you click a keyword, all of the keywords in the structure are highlighted. For instance, when you click **If** in an **If...Then...Else** construction, all instances of **If**, **Then**, **ElseIf**, **Else**, and **End If** in the construction are highlighted. To move to the next or previous highlighted keyword, press CTRL+SHIFT+DOWN ARROW or CTRL+SHIFT+UP ARROW.

Nesting Different Kinds of Control Structures

You can nest one kind of control structure within another kind. The following example uses a **With** block inside a **For Each** loop and nested **If** blocks inside the **With** block.

```
For Each ctl As System.Windows.Forms.Control In Me.Controls
    With ctl
        .BackColor = System.Drawing.Color.Yellow
        .ForeColor = System.Drawing.Color.Black
        If .CanFocus Then
            .Text = "Colors changed"
            If Not .Focus() Then
                ' Insert code to process failed focus.
            End If
        End If
    End With
Next ctl
```

Overlapping Control Structures

You cannot overlap control structures. This means that any nested structure must be completely contained within the next innermost structure. For example, the following arrangement is invalid because the **For** loop terminates before the inner **With** block terminates.

```
For i As Integer = 1 to maxValue

    With sendButton

Next i

    End With
```

Invalid nesting of For and With structures

The Visual Basic compiler detects such overlapping control structures and signals a compile-time error.

See Also

Control Flow in Visual Basic
Decision Structures (Visual Basic)
Loop Structures (Visual Basic)
Other Control Structures (Visual Basic)