VB.NET Language Pocket Reference

Whether you’re using Visual Basic .NET to create ASP.NET pages, develop Windows Forms applications, or write middle-tier components, this book will be your constant companion.

Based on the bestselling *Visual Basic .NET Language in a Nutshell*, this book details every VB.NET language element—every compiler directive, statement, function, and object—recognized by the compiler or implemented in the Microsoft.VisualBasic namespace. Entries are arranged alphabetically by topic, so that you can, for instance, easily find details about that string-handling function that you can’t quite remember.

Regardless of how much experience you have programming with VB.NET, this is the book you’ll pick up time and time again as your standard quick reference guide to the Visual Basic .NET language. It is an indispensable reference for anyone writing code with VB.NET.

Visit O’Reilly on the Web at www.oreilly.com
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Font Conventions</td>
<td>2</td>
</tr>
<tr>
<td>Visual Basic Conventions</td>
<td>2</td>
</tr>
<tr>
<td>Data Types</td>
<td>3</td>
</tr>
<tr>
<td>Variables</td>
<td>5</td>
</tr>
<tr>
<td>Operators and Precedence</td>
<td>6</td>
</tr>
<tr>
<td>Constants</td>
<td>7</td>
</tr>
<tr>
<td>Call Type Constants</td>
<td>7</td>
</tr>
<tr>
<td>Comparison Constants</td>
<td>7</td>
</tr>
<tr>
<td>Date and Time Constants</td>
<td>8</td>
</tr>
<tr>
<td>Date Format Constants</td>
<td>8</td>
</tr>
<tr>
<td>Error Constant</td>
<td>8</td>
</tr>
<tr>
<td>File Attributes</td>
<td>8</td>
</tr>
<tr>
<td>Logical and Tristate Constants</td>
<td>8</td>
</tr>
<tr>
<td>Message Box Constants</td>
<td>8</td>
</tr>
<tr>
<td>Special Character Constants</td>
<td>9</td>
</tr>
<tr>
<td>Variable Subtype Constants</td>
<td>9</td>
</tr>
<tr>
<td>Window Style Constants</td>
<td>10</td>
</tr>
</tbody>
</table>
Enumerations

AppWinStyle Enumeration 10
CallType Enumeration 10
CompareMethod Enumeration 11
ControlChars Class 11
DateFormat Enumeration 11
DateInterval Enumeration 11
DueDate Enumeration 12
FileAttribute Enumeration 12
FirstDayOfWeek Enumeration 12
FirstWeekOfYear Enumeration 12
MsgBoxResult Enumeration 13
MsgBoxStyle Enumeration 13
OpenAccess Enumeration 13
OpenMode Enumeration 13
OpenShare Enumeration 14
TriState Enumeration 14
VariantType Enumeration 14
VbStrConv Enumeration 15

Object Programming

.NET Framework Class Library 15
Custom Types and Classes 16
Fields, Properties, and Methods 17

Program Structure 19

Array Handling 20

Collection Objects 22
Conditional Compilation 24

Other Conversion 34
Introduction

Visual Basic .NET is a radically new version of Microsoft Visual Basic, the world's most widely used rapid application development (RAD) package. Visual Basic .NET is designed to work directly with Microsoft's .NET platform, which includes a Framework Class Library to facilitate application development, a Common Language Runtime to provide a managed execution environment, and a Common Type System to insure interoperability among all languages that support the .NET platform.

The Visual Basic .NET Language Pocket Reference is a quick reference guide to Visual Basic 7.0, the first version of Visual Basic .NET. It contains a concise description of all language elements by category. These include language elements implemented by the Visual Basic compiler, as well as all procedures and functions implemented in the Microsoft.VisualBasic namespace. It does not attempt to document the core classes of the .NET Framework Class Library.

The purpose of this quick reference is to aid readers who need to look up some basic detail of Visual Basic syntax or usage. It is not intended to be a tutorial or user guide; at least a basic familiarity with Visual Basic is assumed.
Font Conventions

Constant Width
Used to indicate code examples, types and type members, statements, constants, and keywords.

Constant Width Italic
Used to indicate replaceable parameters.

Italic
Used to indicate new terms, filenames, URLs, and email addresses.

Visual Basic Conventions

The “rules” for Visual Basic (VB) code are very simple:

• VB is a case-insensitive programming language; that is, the compiler ignores case when reading VB code. So myVar, MyVar, Myvar, and MYVAR all refer to the same variable. Note that Visual Studio imposes a uniform casing on all language elements, although this is not a requirement of the compiler.

• White space (except for line breaks) is ignored when reading VB code.

• Line breaks mark the end of a complete statement; complete VB statements must occupy a single line.

• If you want to break a single statement over several lines, you can use the line continuation character, an underscore (_), which must be preceded by a space and must be the last character on the line that is to be continued.

• If you want to combine multiple statements on a single line, you can use the colon (:). Among other uses, it is commonly used to imitate C++, Java, and C# syntax for inheritance. For example, the code fragment:

```vbnet
Public Class MainForm
    Inherits Form
```
can be shortened as follows:

```vbnet
Public Class MainForm : Inherits Form
```

- Two comment symbols are used: the apostrophe (’) and the `Rem` keyword. They may appear at any place within a line.

## Data Types

Whether VB is a weakly or a semi-strongly typed language depends on the `Option Explicit` setting. (The statement must appear at the top of a code module.) If `Off`, VB is a weakly typed language; variables need not be declared in advance, and all undeclared variables will be cast as type `Object` until they are initialized. If `On` (its default setting), each variable must be declared in advance, but its data type need not be specified. If no type is explicitly declared, variables are cast as type `Object` until their first use.

Although VB recognizes a number of “intrinsic” data types, each is really a wrapper around a data type found in the .NET Common Type System (CTS). VB recognizes the following intrinsic types:

- **Boolean**
  A logical (True or False) value. Corresponds to `System.Boolean`.

- **Byte**

- **Char**

- **Date**
  A date or time value. Corresponds to `System.DateTime`.
Decimal
A decimal or currency value. Corresponds to System.Decimal.

Double
A double-precision floating point value. Corresponds to System.Double.

Integer

Long
A signed 64-bit integral data type. Corresponds to System.Int64.

Object
A reference to an object. Object is VB’s “universal” data type and corresponds to System.Object.

Short

Single

String
A reference type pointing to a fixed-length character string. Corresponds to System.String.

A number of other data types are available from the .NET CTS but are not wrapped by a corresponding VB intrinsic data type. These include:

System.SByte
A signed 8-bit integral data type.

System.UInt16
An unsigned 16-bit integral data type.
System.UInt32
   An unsigned 32-bit integral data type.
System.UInt64
   An unsigned 64-bit integral data type.

VB also allows you to create user-defined reference types by using the Class...End Class construct and user-defined value types by using the Structure...End Structure construct.

Any of these data types can be used as a member of an array. Arrays can have a single dimension, or they can be multidimensional (up to 60 dimensions). The first element in an array is always at position 0.

Variables

VB does not require that variables be declared before they are used unless Option Explicit is On (its default value). In that case, you can declare variables using the Dim, Public, Private, Protected, Friend, or Protected Friend statements.

A variable name in VB must satisfy the following requirements:

• It must be 16,383 or fewer characters in length.
• It must begin with an alphabetic character or an underscore.
• It cannot include embedded spaces.
• It cannot contain any special (i.e., non-alphabetic, non-numeric) character other than an underscore.
• It must be unique within its scope.
Operators and Precedence

VB supports the following operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition, string concatenation</td>
</tr>
<tr>
<td>+=</td>
<td>Increment and assign</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction, unary operator</td>
</tr>
<tr>
<td>-=</td>
<td>Subtract and assign</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>/=</td>
<td>Divide and assign</td>
</tr>
<tr>
<td>\</td>
<td>Integer division (no remainder)</td>
</tr>
<tr>
<td>\</td>
<td>Integer division and assignment</td>
</tr>
<tr>
<td>Mod</td>
<td>Modulo arithmetic</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>*=</td>
<td>Multiply and assign</td>
</tr>
<tr>
<td>^</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>^=</td>
<td>Exponentiation and assignment</td>
</tr>
<tr>
<td>&amp;</td>
<td>String concatenation</td>
</tr>
<tr>
<td>&amp;=</td>
<td>String concatenation and assignment</td>
</tr>
<tr>
<td>=</td>
<td>Equality, assignment</td>
</tr>
<tr>
<td>Is</td>
<td>Equality (for object references)</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=, &lt;&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=, =&gt;</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;&gt;, &lt;&lt;</td>
<td>Not equal to</td>
</tr>
<tr>
<td>And</td>
<td>Logical or bitwise conjunction</td>
</tr>
<tr>
<td>AndAlso</td>
<td>Logical conjunction with short-circuiting</td>
</tr>
<tr>
<td>Or</td>
<td>Logical or bitwise disjunction</td>
</tr>
<tr>
<td>OrElse</td>
<td>Logical disjunction with short-circuiting</td>
</tr>
</tbody>
</table>
Expressions are evaluated in the following order:

1. Arithmetic operators
   a. Exponentiation
   b. Division and multiplication
   c. Integer division
   d. Modulo arithmetic
   e. Addition and subtraction
2. Concatenation operators
3. Logical operators
   a. Not
   b. And, AndAlso
   c. Or, OrElse
   d. X

If two or more operators in an expression have the same order of precedence, they are evaluated from left to right.

**Constants**

VB recognizes the following intrinsic constants:

**Call Type Constants**

- `vbGet`
- `vbLet`
- `vbMethod`
- `vbSet`

**Comparison Constants**

- `vbBinaryCompare`
- `vbTextCompare`
**Trim Function**

**Description**: Performs special conversions on a string.

**Return Value**: A String converted according to conversion.

---

**StrDup Function**

```vba
StrDup(number, character)
```

**number** required; Integer

- The number of times to duplicate the first character in string

**character** required; String, Char, or Object containing a String or Char

- The String or Char whose first character is to be duplicated

**Return Value**: A String containing the character duplicated the specified number of times.

**Description**: Returns a string that consists of the first character of character duplicated number times.

---

**StrReverse Function**

```vba
StrReverse(expression)
```

**expression** required; String

- The string whose characters are to be reversed

**Return Value**: String

**Description**: Returns a string that is the reverse of the string passed to it. For example, if the string and is passed to it as an argument, StrReverse returns the string dna.

---

**Trim Function**

```vba
Trim(str)
```

**str** required; String

- Any string expression

**Return Value**: String

**Description**: Removes both leading and trailing spaces from a given string.
UCase Function

Microsoft.VisualBasic.Strings

UCase(value)

value

A valid string expression

Return Value: String

Description: Converts a string to uppercase.

Val Function

See “Val Function” entry under “Data Type Conversion” section.

VBFixedString Attribute

Microsoft.VisualBasic.VBFixedStringAttribute

Applies to: Field

Description: Defines a fixed-length string. It is the rough equivalent of the VB 6 declaration:

Dim s As String * length

It can be used to define fixed-length strings within structures, particularly structures that are to be passed to Win32 API functions, as well as to define fixed-length strings to be written to and read from random access files.

Constructor: New(length)

length

Integer

The length of the string

Properties: Length

Integer

Read-only. The length of the string. Its value is set by the length parameter in the class constructor.