Visual Basic Language Reference 1
Keywords 3
Attributes 8
Constants and Enumerations 9
Directives 13
Modifiers 14
Async 16
Narrowing 19
Shadows 20
Unicode 22

Visual Basic Language Reference

Visual Studio 2015

This section provides reference information for various aspects of the Visual Basic language.

In This Section

Typographic and Code Conventions (Visual Basic)
: Summarizes the way that keywords, placeholders, and other elements of the language are formatted in the Visual Basic documentation.

Visual Basic Runtime Library Members
: Lists the classes and modules of the Microsoft.VisualBasic namespace, with links to their member functions, methods, properties, constants, and enumerations.

Keywords (Visual Basic)
: Lists all Visual Basic keywords and provides links to more information.

Attributes (Visual Basic)
: Documents the attributes available in Visual Basic.

Constants and Enumerations (Visual Basic)
: Documents the constants and enumerations available in Visual Basic.

Data Type Summary (Visual Basic)
: Documents the data types available in Visual Basic.

Directives (Visual Basic)
: Documents the compiler directives available in Visual Basic.

Functions (Visual Basic)
: Documents the run-time functions available in Visual Basic.

Modifiers (Visual Basic)
: Lists the Visual Basic run-time modifiers and provides links to more information.

Modules (Visual Basic)
: Documents the modules available in Visual Basic and their members.

Nothing (Visual Basic)
: Describes the default value of any data type.

Objects (Visual Basic)
: Documents the objects available in Visual Basic and their members.

Operators (Visual Basic)
: Documents the operators available in Visual Basic.

Properties (Visual Basic)

Documents the properties available in Visual Basic.

Queries (Visual Basic)
: Provides reference information about using Language-Integrated Query (LINQ) expressions in your code.

Statements (Visual Basic)
: Documents the declaration and executable statements available in Visual Basic.

Recommended XML Tags for Documentation Comments (Visual Basic)
: Describes the documentation comments for which IntelliSense is provided in the Visual Basic Code Editor.

XML Axis Properties (Visual Basic)
: Provides links to information about using XML axis properties to access XML directly in your code.

XML Literals (Visual Basic)
: Provides links to information about using XML literals to incorporate XML directly in your code.

Error Messages (Visual Basic)
: Provides a listing of Visual Basic compiler and run-time error messages and help on how to handle them.

Related Sections

Visual Basic
: Provides comprehensive help on all areas of the Visual Basic language.

Visual Basic Command-Line Compiler
: Describes how to use the command-line compiler as an alternative to compiling programs from within the Visual Studio integrated development environment (IDE).

Keywords (Visual Basic)

Visual Studio 2015

The following tables list all Visual Basic language keywords.

Reserved Keywords

The following keywords are *reserved*, which means that you cannot use them as names for programming elements such as variables or procedures. You can bypass this restriction by enclosing the name in brackets (**[ ]**). For more information, see "Escaped Names" in Declared Element Names (Visual Basic).

> **Note**
>
> We do not recommend that you use escaped names, because it can make your code hard to read, and it can lead to subtle errors that can be difficult to find.

AddHandler	AddressOf	Alias	And
AndAlso	As	Boolean	ByRef
Byte	ByVal	Call	Case
Catch	CBool	CByte	CChar
CDate	CDbl	CDec	Char
CInt	Class Constraint	Class Statement	CLng
CObj	Const	Continue	CSByte
CShort	CSng	CStr	CType
CUInt	CULng	CUShort	Date
Decimal	Declare	Default	Delegate
Dim	DirectCast	Do	Double
Each	Else	ElseIf	End Statement
End <keyword>	**EndIf**	Enum	Erase

Error	Event	Exit	False
Finally	For (in For...Next)	For Each...Next	Friend
Function	Get	GetType	GetXMLNamespace
Global	**GoSub**	GoTo	Handles
If	If()	Implements	Implements Statement
Imports (.NET Namespace and Type)	Imports (XML Namespace)	In	In (Generic Modifier)
Inherits	Integer	Interface	Is
IsNot	Let	Lib	Like
Long	Loop	Me	Mod
Module	Module Statement	MustInherit	MustOverride
MyBase	MyClass	Namespace	Narrowing
New Constraint	New Operator	Next	Next (in Resume)
Not	Nothing	NotInheritable	NotOverridable
Object	Of	On	Operator
Option	Optional	Or	OrElse
Out (Generic Modifier)	Overloads	Overridable	Overrides
ParamArray	Partial	Private	Property
Protected	Public	RaiseEvent	ReadOnly
ReDim	REM	RemoveHandler	Resume
Return	SByte	Select	Set
Shadows	Shared	Short	Single
Static	Step	Stop	String
Structure Constraint	Structure Statement	Sub	SyncLock
Then	Throw	To	True

Try	TryCast	TypeOf...Is	UInteger
ULong	UShort	Using	**Variant**
Wend	When	While	Widening
With	WithEvents	WriteOnly	Xor
#Const	#Else	#ElseIf	#End
#If	=	&	&=
*	*=	/	/=
\	\=	^	^=
+	+=	-	-=
>> Operator (Visual Basic)	>>= Operator (Visual Basic)	<<	<<=

Note

EndIf, **GoSub**, **Variant**, and **Wend** are retained as reserved keywords, although they are no longer used in Visual Basic. The meaning of the **Let** keyword has changed. **Let** is now used in LINQ queries. For more information, see Let Clause (Visual Basic).

Unreserved Keywords

The following keywords are not reserved, which means you can use them as names for your programming elements. However, doing this is not recommended, because it can make your code hard to read and can lead to subtle errors that can be difficult to find.

Aggregate	Ansi	Assembly	Async
Auto	Await	Binary	Compare
Custom	Distinct	Equals	Explicit
From	Group By	Group Join	Into
IsFalse	IsTrue	Iterator	Join
Key (Visual Basic)	Mid	Off	Order By

Preserve	Skip	Skip While	Strict
Take	Take While	Text	Unicode
Until	Where	Yield	#ExternalSource
#Region			

Related Topics

Title	Description
Arrays Summary (Visual Basic)	Lists language elements that are used to create, define, and use arrays.
Collection Object Summary (Visual Basic)	Lists language elements that are used for collections.
Compiler Directive Summary (Visual Basic)	Lists directives that control compiler behavior.
Control Flow Summary (Visual Basic)	Lists statements that are used for looping and controlling procedure flow.
Conversion Summary (Visual Basic)	Lists functions that are used to convert numbers, dates, times, and strings.
Data Types Summary (Visual Basic)	Lists data types. Also lists functions that are used to convert between data types and verify data types.
Dates and Times Summary (Visual Basic)	Lists language elements that are used for dates and times.
Declarations and Constants Summary (Visual Basic)	Lists statements that are used to declare variables, constants, classes, modules, and other programming elements. Also lists language elements that are used to obtain object information, handle events, and implement inheritance.
Directories and Files Summary (Visual Basic)	Lists functions that are used to control the file system and to process files.
Errors Summary (Visual Basic)	Lists language elements that are used to catch and return run-time error values.
Financial Summary (Visual Basic)	Lists functions that are used to perform financial calculations.

Input and Output Summary (Visual Basic)	Lists functions that are used to read from and write to files, manage files, and print output.
Information and Interaction Summary (Visual Basic)	Lists functions that are used to run other programs, obtain command-line arguments, manipulate COM objects, retrieve color information, and use control dialog boxes.
Math Summary (Visual Basic)	Lists functions that are used to perform trigonometric and other mathematical calculations.
My Reference (Visual Basic)	Lists the objects contained in **My**, a feature that provides access to frequently used methods, properties, and events of the computer on which the application is running, the current application, the application's resources, the application's settings, and so on.
Operators Summary (Visual Basic)	Lists assignment and comparison expressions and other operators.
Registry Summary (Visual Basic)	Lists functions that are used to read, save, and delete program settings.
String Manipulation Summary (Visual Basic)	Lists functions that are used to manipulate strings.

See Also

Visual Basic Runtime Library Members

Attributes (Visual Basic)

Visual Studio 2015

Visual Basic provides several attributes that allow objects interoperate with unmanaged code, and one attribute that enables module members to be accessed without the module name. The following table lists the attributes used by Visual Basic.

ComClassAttribute	Instructs the compiler to add metadata that allows a class to be exposed as a COM object.
HideModuleNameAttribute	Allows the module members to be accessed using only the qualification needed for the module.
VBFixedArrayAttribute	Indicates that an array in a structure or non-local variable should be treated as a fixed-length array.
VBFixedStringAttribute	Indicates that a string should be treated as if it were fixed length.

See Also

Attributes (C# and Visual Basic)

Constants and Enumerations (Visual Basic)

Visual Studio 2015

Visual Basic supplies a number of predefined constants and enumerations for developers. Constants store values that remain constant throughout the execution of an application. Enumerations provide a convenient way to work with sets of related constants, and to associate constant values with names.

Constants

Conditional Compilation Constants

The following table lists the predefined constants available for conditional compilation.

Constant	Description
CONFIG	A string that corresponds to the current setting of the **Active Solution Configuration** box in the **Configuration Manager**.
DEBUG	A **Boolean** value that can be set in the **Project Properties** dialog box. By default, the Debug configuration for a project defines **DEBUG**. When **DEBUG** is defined, Debug class methods generate output to the **Output** window. When it is not defined, Debug class methods are not compiled and no Debug output is generated.
TARGET	A string representing the output type for the project or the setting of the command-line **/target** option. The possible values of **TARGET** are: • "winexe" for a Windows application. • "exe" for a console application. • "library" for a class library. • "module" for a module. • The **/target** option may be set in the Visual Studio integrated development environment. For more information, see /target (Visual Basic).
TRACE	A **Boolean** value that can be set in the **Project Properties** dialog box. By default, all configurations for a project define **TRACE**. When **TRACE** is defined, Trace class methods generate output to the **Output** window. When it is not defined, Trace class methods are not compiled and no **Trace** output is generated.
VBC_VER	A number representing the Visual Basic version, in *major.minor* format. The version number for Visual Basic 2005 is 8.0.

Print and Display Constants

When you call print and display functions, you can use the following constants in your code in place of the actual values.

Constant	Description
vbCrLf	Carriage return/linefeed character combination.
vbCr	Carriage return character.
vbLf	Linefeed character.
vbNewLine	Newline character.
vbNullChar	Null character.
vbNullString	Not the same as a zero-length string (""); used for calling external procedures.
vbObjectError	Error number. User-defined error numbers should be greater than this value. For example: `Err.Raise(Number) = vbObjectError + 1000`
vbTab	Tab character.
vbBack	Backspace character.
vbFormFeed	Not used in Microsoft Windows.
vbVerticalTab	Not useful in Microsoft Windows.

Enumerations

The following table lists and describes the enumerations provided by Visual Basic.

Enumeration	Description
AppWinStyle	Indicates the window style to use for the invoked program when calling the Shell function.
AudioPlayMode	Indicates how to play sounds when calling audio methods.
BuiltInRole	Indicates the type of role to check when calling the IsInRole method.

CallType	Indicates the type of procedure being invoked when calling the CallByName function.
CompareMethod	Indicates how to compare strings when calling comparison functions.
DateFormat	Indicates how to display dates when calling the FormatDateTime function.
DateInterval	Indicates how to determine and format date intervals when calling date-related functions.
DeleteDirectoryOption	Specifies what should be done when a directory that is to be deleted contains files or directories.
DueDate	Indicates when payments are due when calling financial methods.
FieldType	Indicates whether text fields are delimited or fixed-width.
FileAttribute	Indicates the file attributes to use when calling file-access functions.
FirstDayOfWeek	Indicates the first day of the week to use when calling date-related functions.
FirstWeekOfYear	Indicates the first week of the year to use when calling date-related functions.
MsgBoxResult	Indicates which button was pressed on a message box, returned by the MsgBox function.
MsgBoxStyle	Indicates which buttons to display when calling the MsgBox function.
OpenAccess	Indicates how to open a file when calling file-access functions.
OpenMode	Indicates how to open a file when calling file-access functions.
OpenShare	Indicates how to open a file when calling file-access functions.
RecycleOption	Specifies whether a file should be deleted permanently or placed in the Recycle Bin.
SearchOption	Specifies whether to search all or only top-level directories.
TriState	Indicates a **Boolean** value or whether the default should be used when calling number-formatting functions.
UICancelOption	Specifies what should be done if the user clicks **Cancel** during an operation.
UIOption	Specifies whether or not to show a progress dialog when copying, deleting, or moving files or directories.
VariantType	Indicates the type of a variant object, returned by the VarType function.
VbStrConv	Indicates which type of conversion to perform when calling the StrConv function.

See Also

- Visual Basic Language Reference
- Visual Basic
- Constants Overview (Visual Basic)
- Enumerations Overview (Visual Basic)

Directives (Visual Basic)

Visual Studio 2015

The topics in this section document the Visual Basic source code compiler directives.

In This Section

#Const Directive -- Define a compiler constant

#ExternalSource Directive -- Indicate a mapping between source lines and text external to the source

#If...Then...#Else Directives -- Compile selected blocks of code

#Region Directive -- Collapse and hide sections of code in the Visual Studio editor

#Disable, #Enable -- Disable and enable specific warnings for regions of code.

VB

```
#Disable Warning BC42356 ' suppress warning about no awaits in this method
    Async Function TestAsync() As Task
        Console.WriteLine("testing")
    End Function
#Enable Warning BC42356
```

You can disable and enable a comma-separated list of warning codes too.

Related Sections

Visual Basic Language Reference

Visual Basic

Modifiers (Visual Basic)

Visual Studio 2015

The topics in this section document Visual Basic run-time modifiers.

In This Section

Ansi (Visual Basic)

Assembly (Visual Basic)

Async (Visual Basic)

Auto (Visual Basic)

ByRef (Visual Basic)

ByVal (Visual Basic)

Default (Visual Basic)

Friend (Visual Basic)

In (Generic Modifier) (Visual Basic)

Iterator (Visual Basic)

Key (Visual Basic)

Module <keyword> (Visual Basic)

MustInherit (Visual Basic)

MustOverride (Visual Basic)

Narrowing (Visual Basic)

NotInheritable (Visual Basic)

NotOverridable (Visual Basic)

Optional (Visual Basic)

Out (Generic Modifier) (Visual Basic)

Overloads (Visual Basic)

Overridable (Visual Basic)

Overrides (Visual Basic)

ParamArray (Visual Basic)

Partial (Visual Basic)

Private (Visual Basic)

Protected (Visual Basic)

Public (Visual Basic)

ReadOnly (Visual Basic)

Shadows (Visual Basic)

Shared (Visual Basic)

Static (Visual Basic)

Unicode (Visual Basic)

Widening (Visual Basic)

WithEvents (Visual Basic)

WriteOnly (Visual Basic)

Related Sections

Visual Basic Language Reference

Visual Basic

Async (Visual Basic)

Visual Studio 2015

The **Async** modifier indicates that the method or lambda expression that it modifies is asynchronous. Such methods are referred to as *async methods*.

An async method provides a convenient way to do potentially long-running work without blocking the caller's thread. The caller of an async method can resume its work without waiting for the async method to finish.

Note

The **Async** and **Await** keywords were introduced in Visual Studio 2012. For an introduction to async programming, see Asynchronous Programming with Async and Await (C# and Visual Basic).

The following example shows the structure of an async method. By convention, async method names end in "Async."

VB

```
Public Async Function ExampleMethodAsync() As Task(Of Integer)
    ' . . .

    ' At the Await expression, execution in this method is suspended and,
    ' if AwaitedProcessAsync has not already finished, control returns
    ' to the caller of ExampleMethodAsync. When the awaited task is
    ' completed, this method resumes execution.
    Dim exampleInt As Integer = Await AwaitedProcessAsync()

    ' . . .

    ' The return statement completes the task. Any method that is
    ' awaiting ExampleMethodAsync can now get the integer result.
    Return exampleInt
End Function
```

Typically, a method modified by the **Async** keyword contains at least one Await expression or statement. The method runs synchronously until it reaches the first **Await**, at which point it suspends until the awaited task completes. In the meantime, control is returned to the caller of the method. If the method doesn't contain an **Await** expression or statement, the method isn't suspended and executes as a synchronous method does. A compiler warning alerts you to any async methods that don't contain **Await** because that situation might indicate an error. For more information, see the compiler error.

The **Async** keyword is an unreserved keyword. It is a keyword when it modifies a method or a lambda expression. In all other contexts, it is interpreted as an identifier.

Return Types

An async method is either a Sub procedure, or a Function procedure that has a return type of Task or Task(Of TResult). The method cannot declare any ByRef parameters.

You specify **Task(Of TResult)** for the return type of an async method if the Return statement of the method has an operand of type TResult. You use **Task** if no meaningful value is returned when the method is completed. That is, a call to the method returns a **Task**, but when the **Task** is completed, any **Await** statement that's awaiting the **Task** doesn't produce a result value.

Async subroutines are used primarily to define event handlers where a **Sub** procedure is required. The caller of an async subroutine can't await it and can't catch exceptions that the method throws.

For more information and examples, see Async Return Types (C# and Visual Basic).

Example

The following examples show an async event handler, an async lambda expression, and an async method. For a full example that uses these elements, see Walkthrough: Accessing the Web by Using Async and Await (C# and Visual Basic). You can download the walkthrough code from Developer Code Samples.

VB

```
' An event handler must be a Sub procedure.
Async Sub button1_Click(sender As Object, e As RoutedEventArgs) Handles button1.Click
    textBox1.Clear()
    ' SumPageSizesAsync is a method that returns a Task.
    Await SumPageSizesAsync()
    textBox1.Text = vbCrLf & "Control returned to button1_Click."
End Sub


' The following async lambda expression creates an equivalent anonymous
' event handler.
AddHandler button1.Click, Async Sub(sender, e)
                              textBox1.Clear()
                              ' SumPageSizesAsync is a method that returns a Task.
                              Await SumPageSizesAsync()
                              textBox1.Text = vbCrLf & "Control returned to
button1_Click."
                          End Sub


' The following async method returns a Task(Of T).
' A typical call awaits the Byte array result:
'      Dim result As Byte() = Await GetURLContents("http://msdn.com")
Private Async Function GetURLContentsAsync(url As String) As Task(Of Byte())

    ' The downloaded resource ends up in the variable named content.
    Dim content = New MemoryStream()
```

```
    ' Initialize an HttpWebRequest for the current URL.
    Dim webReq = CType(WebRequest.Create(url), HttpWebRequest)

    ' Send the request to the Internet resource and wait for
    ' the response.
    Using response As WebResponse = Await webReq.GetResponseAsync()
        ' Get the data stream that is associated with the specified URL.
        Using responseStream As Stream = response.GetResponseStream()
            ' Read the bytes in responseStream and copy them to content.
            ' CopyToAsync returns a Task, not a Task<T>.
            Await responseStream.CopyToAsync(content)
        End Using
    End Using

    ' Return the result as a byte array.
    Return content.ToArray()
End Function
```

See Also

AsyncStateMachineAttribute
Await Operator (Visual Basic)
Asynchronous Programming with Async and Await (C# and Visual Basic)
Walkthrough: Accessing the Web by Using Async and Await (C# and Visual Basic)

Narrowing (Visual Basic)

Visual Studio 2015

Indicates that a conversion operator (**CType**) converts a class or structure to a type that might not be able to hold some of the possible values of the original class or structure.

Converting with the Narrowing Keyword

The conversion procedure must specify **Public Shared** in addition to **Narrowing**.

Narrowing conversions do not always succeed at run time, and can fail or incur data loss. Examples are **Long** to **Integer**, **String** to **Date**, and a base type to a derived type. This last conversion is narrowing because the base type might not contain all the members of the derived type and thus is not an instance of the derived type.

If **Option Strict** is **On**, the consuming code must use **CType** for all narrowing conversions.

The **Narrowing** keyword can be used in this context:

Operator Statement

See Also

Operator Statement
Widening (Visual Basic)
Widening and Narrowing Conversions (Visual Basic)
How to: Define an Operator (Visual Basic)
CType Function (Visual Basic)
Option Strict Statement

Shadows (Visual Basic)

Visual Studio 2015

Specifies that a declared programming element redeclares and hides an identically named element, or set of overloaded elements, in a base class.

Remarks

The main purpose of shadowing (which is also known as *hiding by name*) is to preserve the definition of your class members. The base class might undergo a change that creates an element with the same name as one you have already defined. If this happens, the **Shadows** modifier forces references through your class to be resolved to the member you defined, instead of to the new base class element.

Both shadowing and overriding redefine an inherited element, but there are significant differences between the two approaches. For more information, see Shadowing in Visual Basic.

Rules

- **Declaration Context.** You can use **Shadows** only at class level. This means the declaration context for a **Shadows** element must be a class, and cannot be a source file, namespace, interface, module, structure, or procedure.

  You can declare only one shadowing element in a single declaration statement.

- **Combined Modifiers.** You cannot specify **Shadows** together with **Overloads**, **Overrides**, or **Static** in the same declaration.

- **Element Types.** You can shadow any kind of declared element with any other kind. If you shadow a property or procedure with another property or procedure, the parameters and the return type do not have to match those in the base class property or procedure.

- **Accessing.** The shadowed element in the base class is normally unavailable from within the derived class that shadows it. However, the following considerations apply.

  - If the shadowing element is not accessible from the code referring to it, the reference is resolved to the shadowed element. For example, if a **Private** element shadows a base class element, code that does not have permission to access the **Private** element accesses the base class element instead.

  - If you shadow an element, you can still access the shadowed element through an object declared with the type of the base class. You can also access it through **MyBase**.

The **Shadows** modifier can be used in these contexts:

Class Statement

Const Statement

Declare Statement

Delegate Statement

Dim Statement

Enum Statement

Event Statement

Function Statement

Interface Statement

Property Statement

Structure Statement

Sub Statement

See Also

Shared (Visual Basic)
Static (Visual Basic)
Private (Visual Basic)
Me, My, MyBase, and MyClass in Visual Basic
Inheritance Basics (Visual Basic)
MustOverride (Visual Basic)
NotOverridable (Visual Basic)
Overloads (Visual Basic)
Overridable (Visual Basic)
Overrides (Visual Basic)
Shadowing in Visual Basic

Unicode (Visual Basic)

Visual Studio 2015

Specifies that Visual Basic should marshal all strings to Unicode values regardless of the name of the external procedure being declared.

When you call a procedure defined outside your project, the Visual Basic compiler does not have access to the information it must have in order to call the procedure correctly. This information includes where the procedure is located, how it is identified, its calling sequence and return type, and the string character set it uses. The Declare Statement creates a reference to an external procedure and supplies this necessary information.

The *charsetmodifier* part in the **Declare** statement supplies the character set information to marshal strings during a call to the external procedure. It also affects how Visual Basic searches the external file for the external procedure name. The **Unicode** modifier specifies that Visual Basic should marshal all strings to Unicode values and should look up the procedure without modifying its name during the search.

If no character set modifier is specified, **Ansi** is the default.

Remarks

The **Unicode** modifier can be used in this context:

Declare Statement

Smart Device Developer Notes

This keyword is not supported.

See Also

Ansi (Visual Basic)
Auto (Visual Basic)
Keywords (Visual Basic)