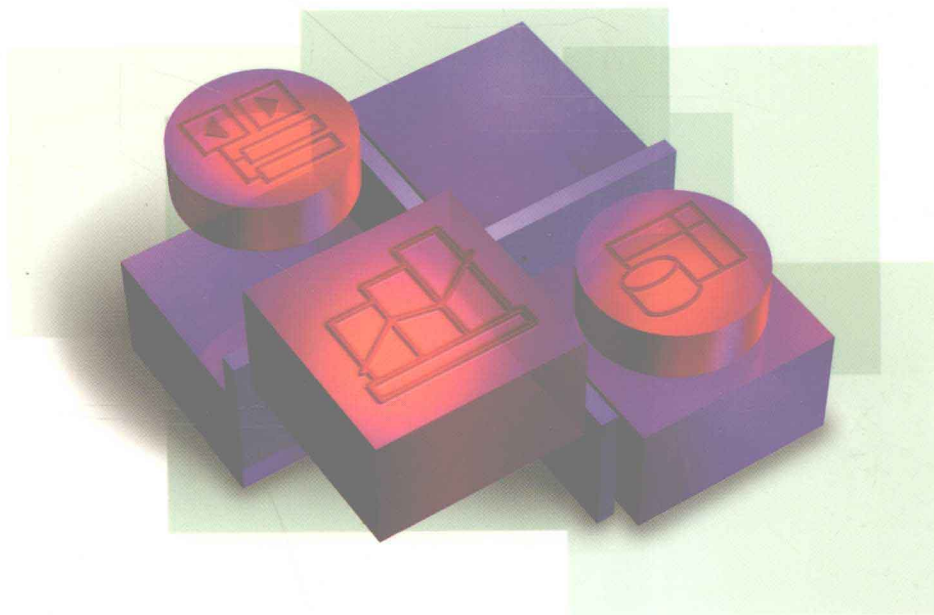


The Essential Guide to Microsoft Visual Basic 6.0



Microsoft® **Visual Basic® 6.0** Programmer's Guide

Microsoft®

Visual Basic 6.0

Programmer's Guide

Microsoft

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Copyright © 1998 by Microsoft Corporation

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Library of Congress Cataloging-in-Publication Data
Microsoft Visual Basic 6.0 Programmer's Guide / Microsoft Corporation.

p. cm.

Includes index.

ISBN 1-57231-863-5

1. Microsoft Visual BASIC. 2. BASIC (Computer program language)

I. Microsoft Corporation.

QA76.73.B3M5589 1998

005.26'8--dc21

98-14786

CIP

Printed and bound in the United States of America.

7 8 9 QWT 7 6 5 4 3 2 1

Distributed in Canada by Penguin Books Canada Limited.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to mspinput@microsoft.com.

Macintosh and TrueType fonts are registered trademarks of Apple Computer, Inc. ActiveX, Microsoft, Microsoft Press, MSDN, Visual Basic, Visual C++, Visual SourceSafe, Visual Studio, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.

The names of example companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted.

Acquisitions Editor: Eric Stroo

Project Editor: Maureen Williams Zimmerman

Document Conventions

This manual uses the following typographic conventions.

Example of convention	Description
Sub, If, Case Else, Print, True, BackColor, Click, Debug, Long	Words with initial letter capitalized indicate language-specific keywords.
setup	Words you're instructed to type appear in bold.
<i>event-driven</i>	In text, italic letters can indicate defined terms, usually the first time they occur in the book. Italic formatting is also used occasionally for emphasis.
<i>variable</i>	In syntax and text, italic letters can indicate placeholders for information you supply.
[<i>expressionlist</i>]	In syntax, items inside square brackets are optional.
{ While Until }	In syntax, braces and a vertical bar indicate a choice between two or more items. You must choose one of the items unless all of the items also are enclosed in square brackets.
<pre>Sub HelloButton_Click() Readout.Text = _ "Hello, world!" End Sub</pre>	This font is used for code.
ENTER	Capital letters are used for the names of keys and key sequences, such as ENTER and CTRL+R.
ALT+F1	A plus sign (+) between key names indicates a combination of keys. For example, ALT+F1 means to hold down the ALT key while pressing the F1 key.
DOWN ARROW	Individual direction keys are referred to by the direction of the arrow on the key top (LEFT, RIGHT, UP, or DOWN). The phrase "arrow keys" is used when describing these keys collectively.
BACKSPACE, HOME	Other navigational keys are referred to by their specific names.
c:\Vb\Samples\Calldlls.vbp	Paths and file names are given in mixed case.

Programming Style in This Manual

The following guidelines are used in writing programs in this manual. For more information, see Chapter 5, “Programming Fundamentals.”

- Keywords appear with initial letters capitalized:

```
' Sub, If, ChDir, Print, and True are keywords.
Print "Title Page"
```

- Line labels are used instead of line numbers. The use of line labels is restricted to error-handling routines:

```
ErrorHandler:
    Power = conFailure
End Function
```

- An apostrophe (') introduces comments:

```
' This is a comment; these two lines
' are ignored when the program is running.
```

- Control-flow blocks and statements in Sub, Function, and Property procedures are indented from the enclosing code:

```
Private Sub cmdRemove_Click ()
    Dim Ind As Integer

    Ind = lstClient.ListIndex    ' Get index.
    ' Make sure list item is selected.
    If Ind >= 0 Then
        lstClient.RemoveItem Ind ' Remove it
                                ' from list box.
        ' Display number.
        lblDisplay.Caption = lstClient.ListCount
    Else
        Beep                    ' If nothing selected, beep.
    End If
End Sub
```

- Lines too long to fit on one line (except comments) may be continued on the next line using a line-continuation character, which is a single leading space followed by an underscore (_):

```
Sub Form_MouseDown (Button As Integer, _
Shift As Integer, X As Single, Y As Single)
```

- Intrinsic constant names are in a mixed-case format, with a two-character prefix indicating the object library that defines the constant. Constants from the Visual Basic (VB) and Visual Basic for applications (VBA) object libraries are prefaced with a “vb”; constants from the data access (DAO) object library are prefaced with a “db”; constants from the Microsoft Excel object library are prefaced with an “xl.” For example:

```
vbTileHorizontal
dbAppendOnly
xlDialogBorder
```

Throughout this book, user-defined constants are usually prefaced with “con” and are mixed case. For example:

```
conYourOwnConstant
```

When using Windows API constants, however, the code examples still follow the same conventions as in previous versions of Visual Basic. For example:

```
EM_LINESCROLL
```

Programmer's Guide

Welcome to the *Microsoft Visual Basic 6.0 Programmer's Guide*, a comprehensive manual on programming with Visual Basic. To accommodate the wealth of features and capabilities in Visual Basic, the *Microsoft Visual Basic 6.0 Programmer's Guide* is divided into two parts.

The first part covers the basic concepts, providing a foundation for programmers new to Visual Basic. Part 2 covers more advanced programming concepts and techniques. Additional information helpful in using the product is presented in the appendices.

Part 1 Visual Basic Basics

An introduction to programming in Visual Basic.

Part 2 What You Can Do with Visual Basic

Advanced topics on Visual Basic programming.

Appendix A Visual Basic Specifications, Limitations, and File Formats

Technical details for Visual Basic.

Appendix B Visual Basic Coding Conventions

Suggested guidelines for consistent and readable code.

Appendix C Native Code Compiler Switches

Details on command line switches for compiling to native code.

Appendix D Adding Help to Your Application

Guidelines for adding online Help to a Visual Basic application.

Contents

Document Conventions	xi
Programmer's Guide	xv
Part 1 Visual Basic Basics	1
Chapter 1 Introducing Visual Basic	3
Welcome to Visual Basic	3
Installing Visual Basic	5
Getting Assistance While You Work	7
Chapter 2 Developing an Application in Visual Basic	11
Visual Basic Concepts	11
Elements of the Integrated Development Environment	13
Your First Visual Basic Application	18
Chapter 3 Forms, Controls, and Menus	31
Understanding Properties, Methods, and Events	32
Designing a Form	33
Clicking Buttons to Perform Actions	36
Controls for Displaying and Entering Text	37
Controls That Present Choices to Users	42
Controls That Display Pictures and Graphics	50
Additional Controls	56
Understanding Focus	57
Setting the Tab Order	59
Menu Basics	60
Prompting the User with Dialog Boxes	63
Chapter 4 Managing Projects	65
Working with Projects	65
The Structure of a Visual Basic Project	67
Creating, Opening, and Saving Projects	70
Adding, Removing, and Saving Files	71

Adding Controls to a Project	72
Making and Running an Executable File	76
Setting Project Options	77
Using Wizards and Add-Ins	77
Chapter 5 Programming Fundamentals	79
The Structure of a Visual Basic Application	80
Before You Start Coding	82
Code Writing Mechanics	84
Introduction to Variables, Constants, and Data Types	92
Introduction to Procedures	120
Introduction to Control Structures	131
Working with Objects	139
Part 2 What You Can Do with Visual Basic	159
Chapter 6 Creating a User Interface	161
Interface Styles	162
Multiple-Document Interface (MDI) Applications	164
More About Forms	173
Using Menus in Your Application	175
Toolbars	192
Dialog Boxes	195
Designing for Different Display Types	204
Designing with the User in Mind	205
Chapter 7 Using Visual Basic's Standard Controls	219
Introduction to Visual Basic Controls	220
Validating Control Data by Restricting Focus	227
Working with Control Arrays	227
Using the ADO Data Control	233
Using the Check Box Control	239
Using the Combo Box Control	241
Using the Command Button Control	246
Using the Common Dialog Control	248
Using the Data Control	257
Using the DataCombo and DataList Controls	258
Using the DataGrid Control	264
Using the File-System Controls (Directory List Box, Drive List Box, and File List Box)	280
Using the Frame Control	289

Using the Microsoft Hierarchical FlexGrid Control	291
Using the Horizontal and Vertical Scroll Bar Controls	321
Using the Image Control.	326
Using the Label Control	329
Using the Line Control.	330
Using the List Box Control.	332
Using the OLE Container Control	342
Using the Option Button Control.	343
Using the Picture Box Control	345
Using the RemoteData Control	348
Using the Shape Control.	350
Using the Text Box Control	352
Using the Timer Control.	355
Chapter 8 More About Programming.	359
Working with Multiple Projects	359
Managing Application Settings	362
Using Conditional Compilation.	365
Working with Resource Files	366
Working with Templates	368
Working with Command Line Switches.	370
Compiling Your Project to Native Code.	371
Creating Your Own Data Types	373
Using Enumerations to Work with Sets of Constants	377
Using Collections as an Alternative to Arrays	383
Chapter 9 Programming with Objects.	385
What You Need to Know About Objects in Visual Basic	386
Finding Out About Objects.	402
Creating Your Own Classes	409
Adding Properties and Methods to a Class	429
Adding Events to a Class	444
Creating Data-Aware Classes	453
Naming Properties, Methods, and Events.	460
Polymorphism.	461
Programming with Your Own Objects.	473
Object Models.	475
Creating Your Own Collection Classes	480
ActiveX Designers.	493

Chapter 10 Programming with Components	499
Types of ActiveX Components	500
In-Process and Out-of-Process Servers	501
Working with ActiveX Components	502
Creating a Reference to an Object	503
Using an Object's Properties, Methods, and Events	513
Releasing an ActiveX Component	516
Navigating Object Models	517
Handling Run-Time Errors in ActiveX Components	521
Handling Requests Pending to an ActiveX Component	523
Using a Component's Visual Interface	529
Chapter 11 Responding to Mouse and Keyboard Events	543
Responding to Mouse Events	544
Detecting Mouse Buttons	549
Detecting SHIFT, CTRL, and ALT States	555
Dragging and Dropping	556
OLE Drag and Drop	562
Customizing the Mouse Pointer	582
Responding to Keyboard Events	583
Interrupting Background Processing	591
Chapter 12 Working with Text and Graphics	595
Working with Fonts	596
Displaying Text on Forms and Picture Boxes	600
Formatting Numbers, Dates, and Times	604
Working with Selected Text	606
Transferring Text and Graphics with the Clipboard Object	607
Understanding the Coordinate System	612
Using Graphical Controls	618
Using Graphics Methods	634
Working with Color	648
Using the Picture Object	657
Printing	658
Chapter 13 Debugging Your Code and Handling Errors	667
How to Handle Errors	668
Designing an Error Handler	670
Error Handling Hierarchy	676

Testing Error Handling by Generating Errors	679
Inline Error Handling	681
Centralized Error Handling	684
Turning Off Error Handling	688
Error Handling with ActiveX Components	690
Approaches to Debugging	693
Avoiding Bugs	696
Design Time, Run Time, and Break Mode	697
Using the Debugging Windows	699
Using Break Mode	701
Running Selected Portions of Your Application	711
Monitoring the Call Stack	714
Testing Data and Procedures with the Immediate Window	716
Special Debugging Considerations	724
Tips for Debugging	728
Chapter 14 Processing Drives, Folders, and Files	731
Introduction to the File System Object Model	731
Programming in the FSO Object Model	733
Processing Files with Older File I/O Statements and Functions	742
Chapter 15 Designing for Performance and Compatibility	751
Understanding Optimization	752
Optimizing for Speed	754
Optimizing for Size	765
Optimizing Objects	774
Compiled vs. Interpreted Applications	777
Compatibility with Other Microsoft Applications	778
Chapter 16 International Issues	781
International Software Definitions	781
Designing International Software	782
Using Resource Files for Localization	785
Designing an International-Aware User Interface	788
General Considerations When Writing International Code	790
Writing International Code in Visual Basic	792
Issues Specific to the Double-Byte Character Set (DBCS)	801
Visual Basic Bidirectional Features	810

Chapter 17 Distributing Your Applications	811
The Package and Deployment Wizard	812
Application Packaging with the Wizard	816
Application Deployment with the Wizard	825
Managing Wizard Scripts	827
The Setup Toolkit	828
Manually Editing a Setup.lst File	834
Manually Creating Distribution Media	840
Using the Package and Deployment Wizard with the Setup Toolkit	842
Testing Your Setup Program	843
Allowing the User to Remove Your Application	844
Distributing Localized ActiveX Controls	846
Appendix A Visual Basic Specifications, Limitations, and File Formats	847
System Requirements for Visual Basic Applications	847
Project Limitations	848
Project File Formats	852
Form Structures	854
Appendix B Visual Basic Coding Conventions	869
Why Coding Conventions?	869
Object Naming Conventions	870
Constant and Variable Naming Conventions	874
Structured Coding Conventions	877
Appendix C Native Code Compiler Switches	881
Optimize for Fast Code	882
Optimize for Small Code	882
No Optimizations	882
Favor Pentium Pro	882
Create Symbolic Debug Info	883
Assume No Aliasing	883
Remove Array Bounds Checks	883
Remove Integer-Overflow Checks	884
Remove Floating-Point Error Checks	884
Remove Safe Pentium FDIV Checks	884
Allow Unrounded Floating-Point Operations	885

Appendix D Adding Help to Your Application 887

Adding Support for Help 887

Adding Support for What's This Help 889

Distributing Help with Your Application 890

Creating Your Own Constants

The syntax for declaring a constant is:

[Public|Private] Const *constantname* [**As** *type*] = *expression*

The argument *constantname* is a valid symbolic name (the rules are the same as those for creating variable names), and *expression* is composed of numeric or string constants and operators; however, you can't use function calls in *expression*.

A Const statement can represent a mathematical or date/time quantity:

```
Const conPi = 3.14159265358979
Public Const conMaxPlanets As Integer = 9
Const conReleaseDate = #1/1/95#
```

The Const statement can also be used to define string constants:

```
Public Const conVersion = "07.10.A"
Const conCodeName = "Enigma"
```

You can place more than one constant declaration on a single line if you separate them with commas:

```
Public Const conPi = 3.14, conMaxPlanets = 9, _
conWorldPop = 6E+09
```

The expression on the right side of the equal sign (=) is often a number or literal string, but it can also be an expression that results in a number or string (although that expression can't contain calls to functions). You can even define constants in terms of previously defined constants:

```
Const conPi2 = conPi * 2
```

Once you define constants, you can place them in your code to make it more readable. For example:

```
Static SolarSystem(1 To conMaxPlanets)
If numPeople > conWorldPop Then Exit Sub
```

Scoping User-Defined Constants

A Const statement has scope like a variable declaration, and the same rules apply:

- To create a constant that exists only within a procedure, declare it within that procedure.
- To create a constant available to all procedures within a module, but not to any code outside that module, declare it in the Declarations section of the module.

- To create a constant available throughout the application, declare the constant in the Declarations section of a standard module, and place the **Public** keyword before **Const**. Public constants cannot be declared in a form or class module.

For More Information For more information regarding scope, see “Understanding the Scope of Variables” earlier in this chapter.

Avoiding Circular References

Because constants can be defined in terms of other constants, you must be careful not to set up a *cycle*, or circular reference between two or more constants. A cycle occurs when you have two or more public constants, each of which is defined in terms of the other.

For example:

```
' In Module 1:
Public Const conA = conB * 2      ' Available throughout
                                   ' application.

' In Module 2:
Public Const conB = conA / 2      ' Available throughout
                                   ' application.
```

If a cycle occurs, Visual Basic generates an error when you attempt to run your application. You cannot run your code until you resolve the circular reference. To avoid creating a cycle, restrict all your public constants to a single module or, at most, a small number of modules.

Data Types

Variables are placeholders used to store values; they have names and data types. The data type of a variable determines how the bits representing those values are stored in the computer's memory. When you declare a variable, you can also supply a data type for it. All variables have a data type that determines what kind of data they can store.

By default, if you don't supply a data type, the variable is given the Variant data type. The Variant data type is like a chameleon — it can represent many different data types in different situations. You don't have to convert between these types of data when assigning them to a Variant variable: Visual Basic automatically performs any necessary conversion.

If you know that a variable will always store data of a particular type, however, Visual Basic can handle that data more efficiently if you declare a variable of that type. For example, a variable to store a person's name is best represented as a string data type, because a name is always composed of characters.

Data types apply to other things besides variables. When you assign a value to a property, that value has a data type; arguments to functions also have data types. In fact, just about anything in Visual Basic that involves data also involves data types.

You can also declare arrays of any of the fundamental types.

For More Information For more information, see the section, “Arrays,” later in this chapter. Selecting data types to improve your application’s performance is discussed in Chapter 15, “Designing for Performance and Compatibility.”

Declaring Variables with Data Types

Before using a non-Variant variable, you must use the `Private`, `Public`, `Dim` or `Static` statement to declare it *As type*. For example, the following statements declare an `Integer`, `Double`, `String`, and `Currency` type, respectively:

```
Private I As Integer
Dim Amt As Double
Static YourName As String
Public BillsPaid As Currency
```

A Declaration statement can combine multiple declarations, as in these statements:

```
Private I As Integer, Amt As Double
Private YourName As String, BillsPaid As Currency
Private Test, Amount, J As Integer
```

Note If you do not supply a data type, the variable is given the default type. In the preceding example, the variables `Test` and `Amount` are of the Variant data type. This may surprise you if your experience with other programming languages leads you to expect all variables in the same declaration statement to have the same specified type (in this case, `Integer`).

Numeric Data Types

Visual Basic supplies several numeric data types — `Integer`, `Long` (long integer), `Single` (single-precision floating point), `Double` (double-precision floating point), and `Currency`. Using a numeric data type generally uses less storage space than a variant.

If you know that a variable will always store whole numbers (such as 12) rather than numbers with a fractional amount (such as 3.57), declare it as an `Integer` or `Long` type. Operations are faster with integers, and these types consume less memory than other data types. They are especially useful as the counter variables in `For...Next` loops.

For More Information To read more about control structures, see “Introduction to Control Structures” later in this chapter.

If the variable contains a fraction, declare it as a `Single`, `Double`, or `Currency` variable. The `Currency` data type supports up to four digits to the right of the decimal separator and fifteen digits to the left; it is an accurate fixed-point data type suitable for monetary calculations. Floating-point (`Single` and `Double`) numbers have much larger ranges than `Currency`, but can be subject to small rounding errors.