Microsoft Visual Basic for Windows: Tips & Techniques
Prepared 03/29/94

VB Programming Using Standard Controls

VB Programming Using Custom & Third-Party Controls

Optimization, Memory Management, & General VB Programming

Advanced VB Programming -- Networks, APIs, DLLs, Graphics

Data Access & VB Database Programming

VB Design Environment

Running VB Applications

General VB References & Documentation Corrections

VB Setup, Installation, CDK, Help Compiler, DDE, & OLE

Microsoft Visual Basic for Windows: Tips & Techniques

VB Programming Using Standard Controls

- How to Trap VB Form Lost Focus with GetActiveWindow API
- How to Set Tab Stops in a List Box in Visual Basic
- How to Create Scrollable Viewports in Visual Basic
- Why Output Might Not Display from VB Form_Load Procedure
- How to Create Pop-up Menus on a Visual Basic Form
- How to Create Rubber-Band Lines/Boxes in Visual Basic
- Determining Number of Lines in VB Text Box; SendMessage API
- How to Scroll VB Text Box Programmatically and Specify Lines
- Overlapping Controls Not Supported in Visual Basic
- PRB: Access Key Causes Different Event Order than Mouse Click
- Carriage Return+Linefeed to Wrap Lines in Text Box Control
- Program Example for COM Port Support in Visual Basic
- VB Procedure Form_Load Not Executed when Unload Not Used
- VB Forms with Menus Cannot Have Fixed Double BorderStyle
- PRB: Long String Assigned to Multiline Text Box Seems to Hang
- DEL Key Behavior Depends on Text Box MultiLine Property
- PRB: Clipboard.SetData Gives Invalid Format Message with Icon
- Disabling the ENTER Key BEEP in a Visual Basic Text Box
- Scope of Line Labels/Numbers in Visual Basic for Windows
- How to Make a Push Button with a Bitmap in Visual Basic
- No New Timer Events During Visual Basic Timer Event Processing
- Creating Nested Control Arrays in Visual Basic
- Parameter Mismatch Error When Pass Properties by Reference
- Double-Clicking the Control Box Causes MouseUp Event in VB
- How to Place Animated Graphics on a Minimized Form in VB
- How to Convert Units to Pixels for DrawWidth in VB
- How to Move Controls Between Forms in VB for Windows
- How to Drop Item into Specified Location in VB List Box
- How to Draw a Line or Box on a Form Using a Label in Ver 1.0
- Form Global (Static) Data Is Preserved After Form Unload
- PRB: End Task from Windows Task List Doesn't Invoke VB Unload
- How to Rotate a Bitmap in VB for Windows
- How to Clear VB Picture Property at Run Time Using LoadPicture
- How to Print Multiline Text Box Using Windows API Functions
- Common Dialog Custom Control: FilterIndex Can Be Negative
- Common Dialog Control: Pipe (|) Optional in Filter Property
- How to Use More than One Type of Font in Picture Box
- Visual Basic SendKeys Statement Is Case Sensitive
- Task List Switch to VB Application Fails After ALT+F4 Close
- Overflow Error Plotting Points Far Outside Bounds of Control
- PRB: MDI Child: Child Window May Adopt Image of Other Control
- 'Text' Property is Read-Only Error as Set Combo Box Text Prop

- How to Close VB Combo Box with ENTER key
  - How to Edit Grid Cells in VB Using Overlapped Text Box
  - How to Make ENTER Key Move Focus Like TAB Key for VB Controls
  - PRB: GotFocus Event Fails If MsgBox Invoked in LostFocus Event
  - PRB: Can TAB in Error if Value of Option Button Set to False
  - How to Programmatically Display or Hide a VB Combo Box List
- PRB: ChDir or ChDrive Won't Change File / Directory List Boxes
  - Visual Basic Can Load RLE4 and RLE8 Bitmap Format Files
  - Example to Evaluate Basic Numeric Expressions
  - How to Right Justify Top-Level Menus in Visual Basic
  - Unable to Display Uppercase W in Small Text Box
  - PRB: SendKeys May Return Illegal Function Call Error
  - PRB: SetFocus During Form Load May Cause Illegal Function Call
- PRB: Click Event Invoked When Option Button Receives Focus
  - How to Detect when the Active Form Changes in Visual Basic
  - How to Determine Which Option Button is Selected in VB
  - How to Make a Spreadsheet-Style Grid that Allows Editing
  - PRB: DropDown Combo Box May Display Partial String
  - Visual Basic 3.0 Programming Questions & Answers
  - Name Property Cannot Be Set When Using Implicit Property
  - Making Enter Key in Directory List Box Act Like Double-Click
- How to Change the Size of the Text Cursor in a Text Box
  - Explanation of the Control Box Menu
  - Validating Text Box Data Causes Extra LostFocus Events
  - How to Use the Forms Collection to Unload All MDI Child Forms
  - How to Trap Keystrokes in the Form Instead of Form's Controls
  - Non-Menu Keys Disabled When Menu Pulled Down
  - If Invoked by Access Key, Click Event Handled Before LostFocus
- Using UP ARROW and DOWN ARROW Keys to Move the Focus
  - PRB: Can't Use ActiveForm to Reference Data Control in VB 3.0
  - How to Prevent Multiple Instances of a VB Application
  - How to Move Controls at Run Time By Using Drag and Drop
  - PRB: Invalid picture Error When Try to Bind Picture Control
  - PRB: Out of Stack Space When One Modal Form Shows Another
  - How to Program Two List Boxes to Scroll Together
- Adjusting VB FontSize at Run Time for Different Video Drivers
  - PRB: PrintForm Blank Page or GPF Due to Video Color Depth >256
  - PRB: Problem Changing Control's Picture to (None) in VB 3.0
  - Category Keywords for All Visual Basic KB Articles
  - How to Display Multiple Foreground Text Colors in VB List Box
  - BackColor Erases Existing Graphics on Form or Picture Control
  - PRB: MDI Child Form Painted Twice When Moved Before Loaded
- How to Distinguish a DblClick from a Click Event
  - How to Automatically Select or Highlight Text Box Upon Focus
  - How to Start a Visual Basic Screen Saver Using SendMessage API

- Selected Prop of List Box Can Cause Click Event & Out of Stack
- How to Right Justify Items in List Box w/ Tabs & SendMessage
- How to Right Justify/Center Text in Single-Line Text Control
- PRB: How to Prevent Flicker in the Repaint of a Label

VB Programming Using Custom & Third-Party Controls

Optimization, Memory Management, & General VB Programming

Advanced VB Programming -- Networks, APIs, DLLs, Graphics

Data Access & VB Database Programming

VB Design Environment

Running VB Applications

General VB References & Documentation Corrections

VB Setup, Installation, CDK, Help Compiler, DDE, & OLE

Visual Basic for Windows: Tips & Techniques

VB Programming Using Standard Controls

VB Programming Using Custom & Third-Party Controls

- VB Custom Controls Support only Certain Picture Formats
  - PRB: Grid Custom Control: Surprising Results when FillStyle=1
  - PRB: Grid Control's Cell Blank When Using Str$
  - VB Grid Custom Control: Text Limited to 255 Characters
  - PRB: Grid Custom Control: LeftCol/TopRow Valid Values
  - 3-D Group Push Button: AutoSize Takes Effect Only on PictureUp
- VB Graph Control Displays Maximum of 80 Characters Per Title
  - VB.EXE Error: License File for Custom Control Not Found
  - How to Use HORZ1.BMP with Professional Toolkit Gauge Control
  - HOME Key in VB.EXE Moves to Beginning of Code, Not Column 1
  - PRB: Animated Button Control: Refresh Won't Redraw Border
  - BUG: Graph Custom Control Text Disappears in EGA Video Mode
- VB Key Status: Autosize Property Affects Height and Width
  - VB Graph Control: ThisPoint, ThisSet Reset to 1 at Run Time
  - VB AniButton Control: Cannot Resize if PictDrawMode=Autosize
  - PRB: Can't Change Minimized/Maximized MDIChild's Position/Size
  - "Device Is Not Open or Is Not Known" Running VB MCITEST Sample
  - "Cannot Find MMSYSTEM.DLL" Loading VB MCI.VBX in Windows 3.0
  - PENCNTRL.VBX Err: Requires Microsoft Windows for Pen Computing
- PRB: MDI Child Cannot Be Maximized/Minimized While Invisible
  - PRB: MDI Child Custom Control: ScaleMode Defaults to Twips
  - VB Graph Custom Control: DataReset Property Resets to 0 (Zero)
  - How to Use VB Graph Control to Graph Data from Grid Control
  - How to Read Flag Property of VB Common Dialog Custom Controls
  - How to Create Column and Row Labels in VB Grid Custom Control
  - VB MCI Control Does Not Support PC Speaker Driver
- VB MCI Control Does Not Support Recording of MIDI Data
  - VB Grid Custom Control Refreshes on All Cell Change Events
  - VB Graph: Use XPosData to Plot Fractional X-Axis Values
  - Toolkit 3-D Control (THREED.VBX) Default Property Values
  - Using a Linked Sound Recorder Object with OLECLIEN.VBX
  - PRB: THREED Check Box Is Not Grayed Out When Value = 2 in VB
- How to Clear All or Part of Grid in Visual Basic
  - How to Make a Spreadsheet-Style Grid that Allows Editing
  - Masked Edit Control, Mask Property Clarification
  - Name Property Cannot Be Set When Using Implicit Property
  - New Features Added to Graph Control in Versions 2.0 and 3.0
  - Create .MMM Movie Files with Macromedia Director for Macintosh
  - MaxFileSize Property Range in CMDDIALOG.VBX Can Be 1 to 2048
- Maximum Length of Name Property Depends on Events Supported
  - Set DrawMode to 2 Or 3 to Update Changes to Graph at Run Time

- How to Right Justify Standard Numbers in a Masked Edit Field
- Playing an .AVI File with the MCITEST Example
- PRB: Some ATI Video Drivers Hang When Using MSOUTLIN.VBX
- International and U.S. Support for Crystal Reports
- How to Fill (Populate) a Grid with Database Data -- 4 Methods
- Error Listing for MCI.VBX Control
- How to Include Return Receipt Functionality w/ MAPI Control
- PRB: Default Extension Ignores File Type in VB Common Dialog
- PRB: Out of Memory Error Using VB Outline Control
- VB CDK Cannot Access the Properties of the VB System Objects
- VB ver 3.0 CDK TN002.TXT: Custom Control Version Management
- Windows 3.1 VERSIONINFO - Version-Information Resource Example
- Category Keywords for All Visual Basic KB Articles
- How to Display Multiple Foreground Text Colors in VB List Box
- PRB: Using RecordCount with VB Dynasets, Snapshots, and Tables
- Using Table Objects Versus Dynaset/Snapshot Objects in VB
- PRB: Common Dialog Open: Err=20476 Buffer lpstrFile Too Small
- VB Crystal Reports Files to Distribute with Your .EXE Program
- How to Use SizeMode Property of OLE Control to Size Display
- How to View Microsoft Word Toolbars Using OLE Control
- How to Print an Embedded Word Document in Visual Basic
- PRB: Serial Port Driver for WFW 3.11 Sends Extra Byte
- How to Save an Embedded Word Document in Visual Basic
- How to Create a Gantt Chart in VB Using a Graph Custom Control
- How to Size the Rows and Columns of a Grid to Fit Exactly
- How to Send a Mail Message Using Visual Basic MAPI Controls

Optimization, Memory Management, & General VB Programming

Advanced VB Programming -- Networks, APIs, DLLs, Graphics

Data Access & VB Database Programming

VB Design Environment

Running VB Applications

General VB References & Documentation Corrections

VB Setup, Installation, CDK, Help Compiler, DDE, & OLE

Visual Basic for Windows: Tips & Techniques

VB Programming Using Standard Controls

VB Programming Using Custom & Third-Party Controls

Optimization, Memory Management, & General VB Programming

- VB Out of Stack Space Error w/ LoadPicture in Form_Paint Event
- Comments and Blank Lines Increase Size of VB 1.0 .EXE File
- How to Optimize Size and Speed of Visual Basic Applications
- How to Determine Display State of a VB Form, Modal or Modeless
- Example of Sharing a Form Between Projects in VB for Windows
- Limit of 15 or 31 Timer Controls in Visual Basic for Windows
- Redim: Array Already Dimensioned Msg After Dim w/ Subscripts
- LONG: List of VB Version 1.0 for Windows Trappable Errors
- Differences Between QuickBasic and Visual Basic Statements
- PRB: For Loop w/ Integer Counter & Increment <=.5 Causes Hang
- How to Emulate MKI$ and CVI in VB Using Windows HMemCpy
- Diagnosing General Protection Fault / UAE in VB for Windows
- Visual Basic 3.0 General Information Questions & Answers
- How to Break Long Statements into Multiple Lines
- Basic Products Can Create and Use Non-Standard File Names
- Obtaining Date or Serial Result from DateSerial or DateValue
- FileDatetime Doesn't Include Time If File Time Is Midnight
- PRB: File Not Found Error When Running .EXE on Other Computer
- Sum Of VB Strings Can Exceed 64K in Certain Circumstances
- How to Retrieve Hidden/System Files Using Dir[$]() Function
- PRB: Can't Set Formal Parameter When Setting Object Vars
- Expected Expression Error: Dynamic Array Not OK in User-Type
- Category Keywords for All Visual Basic KB Articles
- How to Capitalize the First Letter of Each Word in a String
- How to Convert a Decimal Number to a Binary Number in a String
- How to Use TABs in a VB Text Box Without Changing the Focus
- PRB: VB 3.0 AppActivate Fails on 32-Bit Windows NT Application
- How to Find Num of Days Between Dates Outside of Normal Range
- How to Scroll a Form When VB Forms Are Limited to Screen Size
- How to Speed Up Data Access by Using BeginTrans & CommitTrans
- LONG: Microsoft Consulting Services Naming Conventions for VB
- How to Encrypt a String with Password Security in VB
- Searchable Electronic VB Info: VB Help Files & MSDN CD-ROM
- How to Remove Menu Items from a Form's Control-Menu Box
- PRB: Week Starts Sunday and Ends Saturday for Format Function
- How to Get a Handle to MS-DOS Application and Change Title
- Example of NPV and IRR Financial Functions in VB for Windows
- How to Mimic HIWORD, LOWORD, HIBYTE, LOBYTE C Macros in VB
- How to Determine If a File Exists by Using DIR$

- How To Seek a CD Track by Using the MCI Control
- How To Get the Total Playing Time of an Audio CD
- How to Get or Create a Unique Audio CD Volume Label
- General Memory Management in Visual Basic Vers 3.0 for Windows
- PRB: Error When Assign DB Value to Var: Invalid Use of Null

Advanced VB Programming -- Networks, APIs, DLLs, Graphics

Data Access & VB Database Programming

VB Design Environment

Running VB Applications

General VB References & Documentation Corrections

VB Setup, Installation, CDK, Help Compiler, DDE, & OLE

Visual Basic for Windows: Tips & Techniques

VB Programming Using Standard Controls

VB Programming Using Custom & Third-Party Controls

Optimization, Memory Management, & General VB Programming

Advanced VB Programming -- Networks, APIs, DLLs, Graphics

- How to Clear a VB List Box with a Windows API Function
  - How to Emulate QuickBasic's SOUND Statement in Visual Basic
  - How to Flood Fill (Paint) in VB using ExtFloodFill Windows API
  - How to Use Windows BitBlt Function in Visual Basic Application
- How to Pass One-Byte Parameters from VB to DLL Routines
  - How to Send an HBITMAP to Windows API Function Calls from VB
  - How to Create a Flashing Title Bar on a Visual Basic Form
  - How to Implement a Bitmap Within a Visual Basic Menu
  - How to Create Rubber-Band Lines/Boxes in Visual Basic
  - How to Create Flashing/Rotating Rubber-Band Box in VB
  - Declare Currency Type to Be Double When Returning from DLL
- How to Create a System-Modal Program/Window in Visual Basic
  - VB Out of Stack Space Error w/ LoadPicture in Form_Paint Event
  - How to Limit User Input in VB Combo Box with SendMessage API
  - Determining Number of Lines in VB Text Box; SendMessage API
  - How VB Can Determine if a Specific Windows Program Is Running
  - How to Scroll VB Text Box Programmatically and Specify Lines
  - WINAPI.TXT: Windows API Declarations and Constants for VB
  - PRB: Duplicate PostScript Font Names in VB Printer.Fonts List
- Determining Whether TAB or Mouse Gave a VB Control the Focus
  - How to Access Windows Initialization Files Within Visual Basic
  - How to Print the ASCII Character Set in Visual Basic
  - How to Clear a VB Combo Box with a Windows API Function
  - BUG: Bad Text in Long Right-Aligned Labels in Windows ver 3.0
  - Using Windows API Functions to Better Manipulate Text Boxes
  - PRB: No Events Generated When Msgbox Active
- How to Create and Use a Custom Cursor in Visual Basic; Win SDK
  - Terminating Windows from a Visual Basic Application
  - How to Print a VB Picture Control Using Windows API Functions
  - How to Invoke GetSystemMetrics Windows API Function from VB
  - Examples of Copying a Disk File in Visual Basic for Windows
  - How to Determine Display State of a VB Form, Modal or Modeless
  - Example of How to Read and Write Visual Basic Arrays to Disk
- How to Get Windows Master List (Task List) Using Visual Basic
  - Use Common Dialog or Escape() API to Specify Number of Copies
  - Lstrcpy API Call to Receive LPSTR Returned from Other APIs
  - PRB: Format$ Using # for Digit Affects Right Alignment

- Use SetHandleCount to Open More than 15 Files at Once in VB
- How to Set Landscape or Portrait for Printer from VB App
- How to Kill an Application with System Menu Using Visual Basic
- How to Reset the Parent of a Visual Basic Control
- How to Add a Horizontal Scroll Bar to Visual Basic List Box
- How to Print VB Form Borders and Menus
- How to Clear VB Picture Property at Run Time Using LoadPicture
- How to Get Windows Version Number in VB with GetVersion API
- How to Copy Entire Screen into a Picture Box in Visual Basic
- VB Custom Controls Support only Certain Picture Formats
- How to Print Multiline Text Box Using Windows API Functions
- How to Use FillPolygonRgn API to Fill Shape in Visual Basic
- How to Set Windows System Colors Using API and Visual Basic
- VB AniButton Control: Cannot Resize if PictDrawMode=Autosize
- How to Disable Close Command in VB Control Menu (System Menu)
- PRB: Can't Change Minimized/Maximized MDIChild's Position/Size
- How to Create a Form with no Title Bar in VB for Windows
- How to Call LoadModule() API Function from Visual Basic
- How to Draw an Ellipse with Circle Statement in VB
- UCase$/LCase$ in Text Box Change Event Inverts Text Property
- How to Print Entire VB Form and Control the Printed Size
- Creating TOPMOST or "Floating" Window in Visual Basic
- Property or Control Not Found When Use Form/Control Data Type
- PRB: DateValue Argument Gives "Illegal Function Call" Error
- How VB Can Get Windows Status Information via API Calls
- How to Determine the Number of VB Applications Running at Once
- VB "Bad DLL Calling Convention" Means Stack Frame Mismatch
- Print Form or Client Area to Size on PostScript or PCL Printer
- How to Play a Waveform (.WAV) Sound File in Visual Basic
- VB for Windows Line Method Does Not Paint Last Pixel
- How to Invoke Search in Windows Help from Visual Basic Program
- How to Use LZCOPYFILE Function to Decompress or Copy Files
- How to Hide a Non-Visual Basic Window or Icon
- How to Compare User-Defined Type Variables in Visual Basic
- How to Extract a Windows Program Icon -- Running or Not
- Diagnosing "Error in loading DLL" with LoadLibrary
- Converting an Icon (.ICO) to Bitmap (.BMP) Format
- Visual Basic 3.0 Programming Questions & Answers
- How to Get Windows 3.1 Version Number in VB with GetVersion
- How to Establish a Network DDE Link Using Visual Basic
- Form Cannot Be Larger Than the Screen
- How to Connect to a Network Drive by Using WNetAddConnection
- Using Lstrcpy() API Function to Get Far Address of a Variable
- How to Pass Numeric Variables to a C DLL
- How to Create a Transparent Bitmap Using Visual Basic

- How Windows Versions 3.0 and 3.1 Activate Apps Differently
- How to Use Windows 3.1 APIs to Play Videos in Visual Basic
- Using PASSTHROUGH Escape to Send Data Directly to Printer
- Using an Escape to Obtain and Change Paper Size for Printer
- How to Obtain & Change the Paper Bins for the Default Printer
- How to Determine When a Shelled Process Has Terminated
- Using the Printer Object to Print a Grid Control's Contents
- How to Use SystemParametersInfo API for Control Panel Settings
- Example of calling EnumFontFamilies from a DLL
- How to Print Text Sideways in Picture Control with Windows API
- How to Play MIDI Files Using API Calls from Visual Basic
- How to Read a Large File into Memory by Calling API Functions
- How to Find Next Available Drive Letter (for Network Connect)
- How to Set the Formatting Rectangle of a TextBox
- Adjusting Form Size for Different Video Screen Resolutions
- How to Play an .AVI Video File in Full Screen in Visual Basic
- Windows Debugging Tools for Use with Visual Basic
- How to Get Control Dimensions from VBGetControlProperty
- PRB: GP Fault if Uninitialized String Passed to API Function
- Changing WIN.INI Printer Settings from VB using Windows API
- How to Create a Screen Saver in Visual Basic
- How to Write C DLLs and Call Them from Visual Basic
- How to Pass User-Defined Structure Containing Strings to DLL
- PRB: Printer.FontSize Return Value Is Not Requested Value
- Category Keywords for All Visual Basic KB Articles
- Popular Windows API Functions Used from Visual Basic 3.0
- How to Invoke MessageBeep API to Play System Alert .WAV Sounds
- Using MSGBLAST.VBX Control to Process Windows Messages from VB
- LONG: How to Call Windows API from VB - General Guidelines
- How to Create a Read-Only Text Box Using SendMessage API
- How to Add Items into Control Menu Box of Visual Basic Form
- How to Turn on Mouse Trails with Visual Basic
- How to Make Mouse Pointer (Cursor) Maintain Hourglass Shape
- How to Right Justify Items in List Box w/ Tabs & SendMessage
- How to Get a Window's Class Name and Other Window Attributes
- How To Add a Scalable Font to Windows From Visual Basic
- How to Pass & Return Unsigned Integers to DLLs from VB
- How to use functions in VER.DLL -- a Sample Application

Data Access & VB Database Programming

VB Design Environment

Running VB Applications

General VB References & Documentation Corrections

VB Setup, Installation, CDK, Help Compiler, DDE, & OLE

Visual Basic for Windows: Tips & Techniques

VB Programming Using Standard Controls

VB Programming Using Custom & Third-Party Controls

Optimization, Memory Management, & General VB Programming

Advanced VB Programming -- Networks, APIs, DLLs, Graphics

Data Access & VB Database Programming

- ODBC Setup Program Gives Error: Could not open file...
  - How to Keep the Current Record the Same After Using Refresh
  - How to Copy Current Database Record into a Record Variable
  - How to Use Data Control to Scroll Up and Down in a Recordset
- ODBC Setup & Connection Issues for Visual Basic Version 3.0
  - How to Implement the DLookup Function in Visual Basic
  - PRB: Can't Use ActiveForm to Reference Data Control in VB 3.0
  - PRB: Visual Basic 3.0 ODBC Does Not Support OpenTable Method
  - Transactions on ODBC Data Sources in Visual Basic Version 3.0
  - How to Open dBASE Table with Nonstandard File Extension
  - PRB: Error When Updating Fields in Dynaset That Has 2+ Tables
- How to Build Access DB & Load Data from Btrieve for Windows DB
  - How to Make Access DB & Transfer Data from Btrieve for MS-DOS
  - Differences Between the Object Variables in VB Version 3.0
  - How to Convert a Text File into a New Access Database
  - Limitations of the Data Control in Visual Basic Version 3.0
  - How to Create an Access DB & Transfer Data from dBASE III DB
- Examples Show How to Query BIBLIO.MDB Database
  - PRB: TableDefs Not Updated When SQL Statement Creates Table
  - PRB: Error 3219 When Updating Record Set Created w/ Distinct
  - How to Encrypt a Microsoft Access Database in Visual Basic
  - Differences Among the Installable ISAMs
  - Referential Integrity Enforced for DBs Created in Access
  - How to Query for Dates Using a SQL Statement in VB 3.0
- How to Use VB Control Property or Variable in SQL Statement
  - PRB: Error or GP Fault When Pass Data Control as Control
  - PRB: Invalid Property Value When Binding Masked Edit Control
  - How Visual Basic Handles Security Set by Microsoft Access
  - PRB: Illegal to Use Find Methods w/ SQL PASSTHROUGH & ODBC DB
  - PRB: Illegal to Use Find Method with Table Object Variable
  - How to Call SQL Stored Procedures from Visual Basic
- PRB: Invalid Database Object after Rollback without BeginTrans
  - PRB: No Current Record Error In VB When Database is Empty
  - How VB Can Determine If Table Is Locked By Other Processes
  - How to Change Read-Only Access of a Data Control at Run Time

- Possible Reasons for Couldn't Find Installable ISAM Error
- How to Create a Parameter Query in Visual Basic for Windows
- LONG: PERFORM.TXT - Performance Tuning Tips for VB and Access
- Microsoft Access Database RAM Cache Is Faster Data File Method
- VISDATA Example of Every Data Access Function in VB Prof 3.0
- How to Create a Microsoft Access Database using VB Prof 3.0
- How to Copy Table from One Database to Another in VB Prof 3.0
- How to Delete a Field from a Populated Table
- Comparison of Seek Versus Find Methods, for VB Data Access
- How to Count Rows Affected Before Query in VB Prof ver 3.0
- DOC: Revised Index Property (Data Access)
- LONG: Overview of Data Access in Visual Basic Version 3.0
- PRB: Find Methods Don't Use Indexes to Speed Up VB Data Access
- How to Attach an External Database Table to a VB 3.0 Database
- How to Request Exclusive Use of a Table in VB Prof 3.0
- Category Keywords for All Visual Basic KB Articles
- Basic Cannot Get Description Shown in Access Table Design View
- How to List the Fields in a Table & the Tables in a Database
- How to Use SQL Outer Join to Find All Table B Records Not in A
- How to Set VB Data Control to External ODBC Database Dynaset
- How to Speed Up Data Access by Using BeginTrans & CommitTrans
- PRB: Dynaset Loses Contents After Transaction Rollback
- How to Use Wildcards in SQL Query to Make Dynasets & Snapshots
- PRB: Closed ODBC Database Stays Open Until Time-Out or VB Ends
- How to Use Seek and MoveNext to Find a Group/Range of Records
- How to Copy a Record from One Table to Another in VB
- PRB: Couldn't Open PARADOX.NET When Opening Paradox 3.x Table
- PRB: Object Variable Not Set When Referencing Data Control
- How to Determine the Restored State of a Minimized Form
- PRB: Commit or Rollback without BeginTrans Error and VB Forms
- Documentation and Features for Visual Basic's Data Manager
- PRB: Error: Couldn't Lock File SHARE.EXE Hasn't Been Loaded
- How to Use SQL SELECT Statement Without Field Syntax Error
- Create Database with Data Manager & View w/ Text/Data Control
- How to Delete a Table from a Database Using Visual Basic
- PRB: Novell Btrieve Unexpected Error from External DB Driver
- PRB: VB Record Too Large When Add or Update Record > 2K
- How to Create Database with Memo Fields Up to 32000 Bytes
- Hitchhiker's Guide to VBSQL -- VBSQL vs ODBC API Data Access
- How to Implement ToolTips Help in Visual Basic Applications
- How to Read Database Fields Into and Out of a List Box
- PRB: Can't find Installable ISAM When Run Two DB Apps in VB
- How to Perform Microsoft Access Macro Action Via DDE from VB

VB Design Environment

Running VB Applications

General VB References & Documentation Corrections

VB Setup, Installation, CDK, Help Compiler, DDE, & OLE

- Visual Basic for Windows: Tips & Techniques
- **VB Programming Using Standard Controls**
- **VB Programming Using Custom & Third-Party Controls**
- **Optimization, Memory Management, & General VB Programming**
- **Advanced VB Programming -- Networks, APIs, DLLs, Graphics**
- **Data Access & VB Database Programming**
- **VB Design Environment**
  - Clicking Toolbox/Color Palette Menu Doesn't Leave Menu Open
    - No Edit Menu Access for Property Entry; Use Edit Shortcut Keys
    - Deleting VB Control Moves Associated Code to Object: (General)
  - PRB: VB Help Misleading Error: Unable to Find Windows Help.EXE
    - Using PAGE DOWN and PAGE UP Keys Within VB.EXE Environment
    - CTRL+HOME Commits Current Line to VB Syntax Checking/Parsing
    - VB Forms with Menus Cannot Have Fixed Double BorderStyle
    - PRB: Invalid in Immediate Window Error When Creating Variable
    - PRB: ToolBox/Color Palette Menus Lose Focus After Single ESC
  - PRB: Compatibility Problems with Adobe Type Manager
    - Restart in VB Break Mode if Delete Blank Line Above End Sub
    - PRB: Printer Error When Printing VB Form to Text-Only Printer
    - PRB: Printing with HPPCL5A.DRV to HP LaserJet III Cuts Line
    - High Granularity Setting Affects Windows/VB Form Resizing
    - Helv and Tms Rmn FontNames Not Available in Windows 3.1
  - VB Uses Bitmap Fonts when TrueType FontSize Less Than 7 Points
    - VB for Windows Trappable Errors List of Changes/Additions
    - How to Use Visual Basic Vers 1.0, 2.0, & 3.0 on Same Computer
    - Add Graph Causes Err: GSW.EXE and GSWDLL.DLL Version Mismatch
    - PRB: Placing Controls inside Container Controls
    - Category Keywords for All Visual Basic KB Articles
- **Running VB Applications**
- **General VB References & Documentation Corrections**
- **VB Setup, Installation, CDK, Help Compiler, DDE, & OLE**

Visual Basic for Windows: Tips & Techniques

VB Programming Using Standard Controls

VB Programming Using Custom & Third-Party Controls

Optimization, Memory Management, & General VB Programming

Advanced VB Programming -- Networks, APIs, DLLs, Graphics

Data Access & VB Database Programming

VB Design Environment

Running VB Applications

- Can't Use Multiple & (for Access Keys) in a VB Menu Control
  - Cannot Tile or Cascade Programs Created with Visual Basic
- Some VB.EXE Main Menu Commands Can Be Invisible at Run Time
  - UAE or GP Fault with VB .EXE Acting as Windows 3.0 Shell
  - F5 in Run Mode with Focus on Main Menu Bar Acts as CTRL+BREAK
  - PRB: Access Key Causes Different Event Order than Mouse Click
  - Determining Whether TAB or Mouse Gave a VB Control the Focus
  - How to Use CodeView for Windows (CVW.EXE) with Visual Basic
  - Simulating ON KEY and Key Trapping by Using the KeyDown Event
- Sending Keystrokes from Visual Basic to an MS-DOS Application
  - "Error Loading DLL" if VB Compiled .EXE Has Same Name as DLL
  - VB Error Using Shell: Cannot Find DLL, Insert in Drive A
  - VB CURDIR$ Function Not Reliable to Determine Program Location
  - How to Get Windows Version Number in VB with GetVersion API
  - PRB: Device Unavailable Msg When Change Path & Drive Door Open
  - How to Right Justify Numbers Using Format$
- Programming a Delay Using the Timer Function
  - How to Emulate Overtype Mode in a Visual Basic Text Box
  - 'Error in loading DLL' When LIBRARY Name Not Same as Filename
  - PRB: Some ATI Video Drivers Hang When Using MSOUTLIN.VBX
  - Category Keywords for All Visual Basic KB Articles
  - PRB: Making .EXE Gives Error: Wrong Version of Runtime DLL

General VB References & Documentation Corrections

VB Setup, Installation, CDK, Help Compiler, DDE, & OLE

Visual Basic for Windows: Tips & Techniques

VB Programming Using Standard Controls

VB Programming Using Custom & Third-Party Controls

Optimization, Memory Management, & General VB Programming

Advanced VB Programming -- Networks, APIs, DLLs, Graphics

Data Access & VB Database Programming

VB Design Environment

Running VB Applications

General VB References & Documentation Corrections

- (Complete) Tutorial to Understand IEEE Floating-Point Errors
  - How to Contribute Visual Basic Articles to the Microsoft KB
- Why Cooper Software Is Listed in Visual Basic's Copyright
  - Technical Data Sheets Available for Visual Basic for Windows
  - Visual Basic Online Help Example Errors
  - List of Visual Basic Companion Products and Services Available
  - LONG: Visual Basic Companion Products & Services (Complete)
  - Cobb Group's "Inside Visual Basic" Journal Article Titles
- Visual Basic 3.0 Support Service Questions & Answers
  - Name Property Cannot Be Set When Using Implicit Property
  - Visual Basic MCI Control TimeFormat Property Information
  - Corrections for Errors in Visual Basic Version 2.0 Manuals
  - Visual Basic User Groups in the U.S.A. and Other Countries
  - Differences Between VCP Version 1.0 and VB Version 2.0 or 3.0
  - Data Manager Source Code Available on CompuServe
- International and U.S. Support for Crystal Reports
  - LONG: Corrections for Errors in VB Version 3.0 Manuals
  - README.TXT for Standard Edition of VB ver 3.0 for Windows
  - README.TXT for Professional Edition of VB 3.0 for Windows
  - PACKING.LST for Standard Edition of VB 3.0 for Windows
  - PACKING.LST for Professonal Edition of VB 2.0 for Windows
  - README.TXT for Professional Edition of VB Ver 2.0 for Windows
- PACKING.LST for Professional Edition of VB 3.0 for Windows
  - Developer Services Offers Solution Provider Packages
  - How to Get Entire VB KB in 2 Help Files with Full-Text Search
  - How to Write C DLLs and Call Them from Visual Basic
  - LONG: VB Pro 3.0 SAMPLES.TXT: Descriptions of Sample Programs
  - DOC: WinHelp Declaration Incorrect in Windows Ver 3.1 API Ref
- LONG: List of Trappable Errors for Visual Basic 3.0
  - LONG: VB 3.0 EXTERNAL.TXT: Using External Database Tables

- VB 3.0 CONSTANT.TXT Gives Values for Named Constants
- VB 3.0 DATACONS.TXT: Const Constant Values for Data Access
- Category Keywords for All Visual Basic KB Articles
- LONG: How to Call Windows API from VB - General Guidelines
- LONG: Microsoft Consulting Services Naming Conventions for VB
- How to Add Items into Control Menu Box of Visual Basic Form
- DOCERR: GetPrivateProfileString Declaration Incorrect in API

Fixlist for Visual Basic for Windows as of 14-Feb-1994

- Buglist for Visual Basic for Windows as of 14-Feb-1994

VB Setup, Installation, CDK, Help Compiler, DDE, & OLE

Visual Basic for Windows: Tips & Techniques

VB Programming Using Standard Controls

VB Programming Using Custom & Third-Party Controls

Optimization, Memory Management, & General VB Programming

Advanced VB Programming -- Networks, APIs, DLLs, Graphics

Data Access & VB Database Programming

VB Design Environment

Running VB Applications

General VB References & Documentation Corrections

VB Setup, Installation, CDK, Help Compiler, DDE, & OLE

- PRB: Insufficient Disk Space Error When Setup Copies Files
- Example of Client-Server DDE Between Visual Basic Applications
  - DDE Example Between Visual Basic and Word for Windows
  - DDE from Visual Basic for Windows to Excel for Windows
  - Using DDE Between Visual Basic and Q+E for Windows
  - DDE Example Between Visual Basic and Windows Program Manager
  - Visual Basic and DDE/OLE with Other Windows Applications
  - PRB: Workaround for Not Enough Memory to Load Tutorial Error
- VB CDK VBAPI.LIB Contains CodeView Information
  - How to Subclass a VB Form Using VB CDK Custom Control
  - VB CDK Custom Property Name Cannot Start with Numeric Value
  - PRB: SETUP.EXE Error: Insufficient Disk Space on: C:\WINDOWS
  - Call VBSetErrorMessage() In Response to VBM_ Messages Only
  - Getting Program Manager Group Names into Combo Box in VB
- VB DDE to Excel with Embedded TAB Can Truncate String in Excel
  - VB Example of Using DDE LinkExecute to Word for Windows 2.0
  - VB CDK: Example of Subclassing a Visual Basic Form
  - VB Example of Using DDE to Run a Word 2.0 for Windows Macro
  - How to Use a Linked Paintbrush Object with OLECLIEN.VBX
  - How to Obtain a Listing of Classes for OLE Client Control
- Visual Basic 3.0 Setup & Installation Questions & Answers
  - Visual Basic 3.0 Programming Questions & Answers
  - How to Establish a Network DDE Link Using Visual Basic
  - Use COMPRESS-r to Avoid Error: Could not execute: SETUP1.EX 2
  - DDE Conversation Can Cause Error Message: DDE Channel Locked
  - How to Use DDE to Display Microsoft Access Data in VB
  - OLE Embedding & Linking Word for Windows Objects into VB Apps
- PRB: Error: Setup could not be completed due to system errors

- PRB: GP Fault with Visual Basic DDE Sample & Word for Windows
- How to Change the Setup Application Name in SETUP1.EXE
- Additions to 'Determining the Files You Need to Distribute'
- How to Run a WinHelp Macro from a Help File
- How to Manipulate Groups & Items in Program Manager Using DDE
- How to Use DDE Between Excel and Visual Basic

How to Copy and Paste DDE Links Using CF_LINK in Visual Basic

- Sample .MAK for Compiling VB Custom Control in Borland C++ 3.1
- VB Ver 3.0 CDK TN001.TXT: Support for DT_OBJECT Properties
- PRB: VB.LIC License File Not Found, Can't Load MSOUTLIN.VBX
- How VB Can Use OLE Automation with Word Version 6.0
- PRB: DDE Error When Running Setup on Norton Desktop

PRB: Extra Repaint of VB CDK Graphical Custom Control

- Category Keywords for All Visual Basic KB Articles
- How to View Microsoft Word Toolbars Using OLE Control
- How to Navigate Excel Objects from Visual Basic Version 3.0
- How to Print an Embedded Word Document in Visual Basic
- Retrieving Groups & Items from Program Manager Using DDE in VB
- How to Find Articles On Visual Basic For Applications
- How to Create Excel Chart w/ OLE Automation from Visual Basic

How to Save an Embedded Word Document in Visual Basic

- How VB Can Use OLE Automation with Excel Version 5.0
- POSITION.HLP File for VB OLE Automation w/ Word for Windows
- How to Perform Microsoft Access Macro Action Via DDE from VB

How to Trap VB Form Lost Focus with GetActiveWindow API

Article ID: Q69792

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

The LostFocus event in Microsoft Visual Basic is useful when transferring control within an application, and you can use the form deactivate and activate events in versions 2.0 and 3.0 to see if the entire form has lost the focus. However, in version 1.0, no global routine exists to check for the entire form losing the focus. To check whether your version 1.0 application has lost the focus, periodically check the Windows API function GetActiveWindow in a Visual Basic timer event, as explained below.

MORE INFORMATION

The only way that version 1.0 provides a check for loss of focus on a form or control is by triggering the LostFocus event. A form does support a LostFocus event; however, a form will only get focus if there are no controls on that form. Focus goes to the controls on a form, and when you click any other visible form, the control's LostFocus procedure will be called. A control's LostFocus procedure will also be called when another control on the form is activated. To perform a routine that occurs only when the form loses focus requires careful management of what generated a LostFocus event on each control (such as setting a flag if another control's Click event was called).

For a simpler method to check if a whole form has lost the focus, you can call the Windows API function GetActiveWindow, located in USER.EXE (a DLL provided with Windows 3.0). The GetActiveWindow API call returns the window handle of the currently active window, which is the new window that you last clicked anywhere in Microsoft Windows. In a timer event procedure for the form, call GetActiveWindow and compare the handle of the currently active Window with the handle of the form window (Form1.hWND). If the handle differs, you know the form has lost the focus. The following program example demonstrates this technique:

Program Example

This single-form example will print "Lost Focus" on the form when you click a different window (such as when you click another program running in Windows).

In Visual Basic, draw one timer control (Timer1) and one command button (Command1) on a single form (Form1).

From the VB.EXE Code menu, choose View Code, and enter the following

code for Form1, using (general) from the Object box, and
(declarations) from the Procedure box:

```
   Declare Function GetActiveWindow Lib "User" () As Integer
   Dim FOCUS As Integer
   Const TRUE = -1
   Const FALSE = 0
```

From the Object box, choose Timer1, and from the Procedure box, choose
Timer, and then put the following code in the Timer1_Timer procedure:

```
Sub Timer1_Timer ()
   If FOCUS = TRUE Then
    ' Compare the handle of the currently active Window with the handle
    ' of the Form1 window:
    If GetActiveWindow() <> Form1.hWND Then
       'Do form's lost-focus routines here.
       Print "Lost Focus"
       FOCUS = FALSE
    End If
   End If
End Sub
```

You must set FOCUS=TRUE in the Click event procedure of every control
on the form, as follows:

From the Object box, choose Command1, and from the Procedure box, choose
Click, then put the following code in the Command1_Click procedure:

```
   Sub Command1_Click ()
      FOCUS = TRUE
   End Sub
```

Double-click Form1 (at design time) and enter the following code
for the Form_Click procedure:

```
   Sub Form_Click ()
      FOCUS = TRUE
      Timer1.Interval = 10
   End Sub
```

You can now run the program.

Reference(s):

"Programming Windows: the Microsoft Guide to Writing Applications for
Windows 3," Charles Petzold. Microsoft Press, 1990.

"Microsoft Windows Software Development Kit: Reference Volume 1,"
version 3.0.

WINSDK.HLP file shipped with Microsoft Windows 3.0 Software
Development Kit.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Set Tab Stops in a List Box in Visual Basic

Article ID: Q71067

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Visual Basic does not have any intrinsic function for creating multiple-column list boxes. To create multiple-column list boxes, you must call a Windows API function to set tab stops within the control. The tab stops create the multiple-column effect.

For this technique to work, the values in the tab stop array must be cumulative. That is, for three successive tabs to occur, you need to load the tab stop array with 100, 150, 200.

MORE INFORMATION

To create the multiple-column effect in list boxes, call the Windows API SendMessage function. After you set the focus to the list box, you must send a message to the window's message queue that will reset the tab stops of the list box. Using the argument LB_SETTABSTOPS as the second parameter to SendMessage will set the desired tab stops for the multicolumn effect based on other arguments to the function. The SendMessage function requires the following parameters to set tab stops:

```
SendMessage (hWnd%,LB_SETTABSTOPS, wParam%, lparam)
```

where

wParam%	is an integer that specifies the number of tab stops.
lParam	is a long pointer to the first member of an array of integers containing the tab stop position in dialog units.

A dialog unit is a horizontal or vertical distance. One horizontal dialog unit is equal to 1/4 of the current dialog base-width unit. The dialog base units are computed based on the height and the width of the current system font. The GetDialogBaseUnits function returns the current dialog base units in pixels.) The tab stops must be sorted in increasing order; back tabs are not allowed.

After setting the tab stops with the SendMessage function, return the focus to the control that had the focus before the procedure call. PutFocus is the Alias for the Windows API SetFocus function. The Windows API SetFocus needs to be redefined using the "Alias" keyword because SetFocus is a reserved word within Visual Basic.

Example Code to Create Multicolumn List Box

For example, to create a multiple-column list box in Visual Basic:

1. Start a new project in Visual Basic, and add a list box (List1) to Form1.

2. Declare the following Windows API function at the module level or in the Global section of your code as follows:

```
' Enter the Declare statement on one, single line:
Declare Function SendMessage Lib "user" (ByVal hwnd As Integer,
   ByVal wMsg As Integer, ByVal wp As Integer, lp As Any) As Long
```

3. Declare the following constants:

```
Const WM_USER = &H400
Const LB_SETTABSTOPS = WM_USER + 19
```

4. Add the following code to the Form_Load Sub procedure:

```
Sub Form_Load ()
   Const LB_SETTABSTOPS = &H400 + 19
   Static tabs(1 To 3) As Integer

   'Set up the array of defined tab stops.
   tabs(1) = 100
   tabs(2) = 150
   tabs(3) = 200

   'Send a message to the message queue.
   retVal& = SendMessage(List1.hWnd, LB_SETTABSTOPS, 3, tabs(1))
   'Enter the following two lines as one, single line:
   list1.AddItem "1" & Chr$(9) & "2" & Chr$(9) & "3" & Chr$(9)
      & "4" & Chr$(9) & "5"
End Sub
```

REFERENCES

"Programming Windows: the Microsoft Guide to Writing Applications for Windows 3," Charles Petzold, Microsoft Press, 1990

"Microsoft Windows Software Development Kit: Reference Volume 1," version 3.0

WINSDK.HLP file shipped with Microsoft Windows 3.0 Software Development Kit

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Create Scrollable Viewports in Visual Basic

Article ID: Q71068

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can create scrollable viewports in Visual Basic by using standard Basic calls. The viewports can include bitmaps, graphics, or other controls.

MORE INFORMATION

This information is included with the Help file provided with Microsoft Professional Toolkit for Visual Basic version 1.0, Microsoft Visual Basic version 2.0, and Microsoft Visual Basic version 3.0.

To create a scrollable picture with clipping, you must have two picture controls. The first picture control is called the stationary parent picture control. Within the parent picture control, you need to create a movable child picture control.

It is the child picture control that will be moved within the parent picture control. Moving the child picture within the parent picture control creates the clipping effect. During run time when you move the child picture, it will be clipped by the boundaries of the parent picture.

To create these two picture controls, do the following:

1. Choose the picture box control from the Toolbox window in Visual Basic.

2. Draw a picture on the form. This is the parent picture.

3. Again choose the picture box control from the Toolbox window.

4. Draw the second picture on top of and within the boundaries of the first picture control. This is the child picture.

The sample application below shows how to create a scrollable bitmap within a viewport. Perform the sequence above to create a parent/child picture control. Add a horizontal scroll bar and a vertical scroll bar to the form.

Make sure that the path to your bitmap is correct. Several of the properties are set during run time, which could have been set during design time as well.

Moving the thumb of the two scroll bars will move the child picture

within the parent picture. The handle (upper-left corner of the picture) to the child picture will be located either at (0,0) of the parent picture or to the left and/or right of the parent picture. Because the clipping region is that of the parent picture, the child picture will appear to move across the parent picture viewport.

Add the following code to the appropriate event procedures:

```
Sub Form_Load ()
   Const PIXEL = 3
   Add the following constant only in Visual Basic 1.0:
   '  Const TRUE = -1
   Const NONE = 0

   ' Set design properties, included here for simplicity.
   Form1.ScaleMode = PIXEL
   Picture1.ScaleMode = PIXEL

   ' AutoSize is set to TRUE so that the boundaries of
   ' Picture2 are expanded to the size of the actual bitmap.
   Picture2.AutoSize = TRUE

   ' Get rid of annoying borders.
   Picture1.BorderStyle = NONE
   Picture2.BorderStyle = NONE

   ' Load the picture that you want to display.
   Picture2.Picture = LoadPicture("c:\win\party.bmp")

   ' Initialize location of both pictures.
   Picture1.Move 0, 0, ScaleWidth - VScroll1.Width,_
   ScaleHeight - HScroll1.Height
   Picture2.Move 0, 0

   ' Position the horizontal scroll bar.
   HScroll1.Top = Picture1.Height
   HScroll1.Left = 0
   HScroll1.Width = Picture1.Width

   ' Position the vertical scroll bar.
   VScroll1.Top = 0
   VScroll1.Left = Picture1.Width
   VScroll1.Height = Picture1.Height

   ' Set the Max value for the scroll bars.
   HScroll1.Max = Picture2.Width - Picture1.Width
   VScroll1.Max = Picture2.Height - Picture1.Height

   ' Determine if child picture will fill up screen.
   ' If so, then there is no need to use scroll bars.

   VScroll1.Enabled = (Picture1.Height < Picture2.Height)
   HScroll1.Enabled = (Picture1.Width < Picture2.Width)
End Sub

Sub HScroll1_Change ()
  ' Picture2.Left is set to the negative of the value because
```

```
  ' as you scroll the scroll bar to the right, the display
  ' should move to the Left, showing more of the right
  ' of the display, and vice-versa when scrolling to the
  ' left.

   Picture2.Left = -HScroll1.Value

End Sub

Sub VScroll1_Change ()
  ' Picture2.Top is set to the negative of the value because
  ' as you scroll the scroll bar down, the display
  ' should move up, showing more of the bottom
  ' of the display, and vice-versa when scrolling up.

  Picture2.Top = -VScroll1.Value

End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Why Output Might Not Display from VB Form_Load Procedure

Article ID: Q71101

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Any graphics or output done within a Form_Load procedure will not display on the form unless you first make the form visible with the Form1.Show method or if you set the form's AutoRedraw property to be true (non-zero).

MORE INFORMATION

When the Form_Load procedure executes (at the beginning of the program), by default the form is not yet displayed. Therefore, during the Form_Load event, no graphics are displayed to the nonexistent form unless you first Show the form (at run time) or set the form's AutoRedraw property (at design time or run time).

A better approach to drawing graphics to the form is to have the graphics drawn to the form during a Sub Form_Paint procedure. This allows the Form.AutoRedraw property to be set to FALSE, increasing the speed performance of your program. Visual Basic does not have to refresh the screen image of your form as it does when a form is overlapped with another window. You (as the programmer) are responsible for refreshing the form, and Sub Form_Paint is the most logical place to handle this situation.

Listed below are three examples of drawing graphics to your form. The first example shows how the graphics fail to be displayed to the form when drawn from within a Form_Load event procedure. The second example shows how you could draw a circle to the form, but the Form.AutoRedraw property must be set to TRUE for the circle to be retained in the event the form needs to be refreshed. The third example is the best approach; it is the fastest and most efficient of the three.

For each example below, add the following Function procedure as a code procedure to Form1.

```
Function Minimum! (n1!, n2!)
    If n1! < n2! Then
        Minimum! = n1!
    Else
        Minimum! = n2!
    End If
End Function
```

Example 1

No graphic is displayed to the form in the following:

```
Sub Form_Load
   Row = Form1.ScaleHeight / 2
   Col = Form1.ScaleWidth / 2
   Radius = Minimum(Row, Col)   ' Function that returns smaller number.
   Form1.Circle (Col, Row), Radius
End Sub
```

Example 2

This example will work, but the AutoRedraw property of Form1 must be TRUE for the screen to refresh properly:

```
Sub Form_Load
   Form1.Show
   Form1.AutoRedraw = -1
   Row = Form1.ScaleHeight / 2
   Col = Form1.ScaleWidth / 2
   Radius = Minimum(Row, Col)   ' Function that returns smaller number.
   Form1.Circle (Col, Row), Radius
End Sub
```

Example 3

This is the best example. AutoRedraw should be set to FALSE for better speed and efficiency.

```
Sub Form_Paint
   Row = Form1.ScaleHeight / 2
   Col = Form1.ScaleWidth / 2
   Radius = Minimum(Row, Col)   ' Function that returns smaller number.
   Form1.Circle (Col, Row), Radius
End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Create Pop-up Menus on a Visual Basic Form

Article ID: Q71279

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Microsoft Visual Basic for Windows can call the Windows API function TrackPopupMenu to display a specified menu at the location on the screen where the user clicks with the mouse.

This information applies only to versions 1.0 and 2.0 because the new PopupMenu command was introduced in Visual Basic version 3.0 for Windows.

MORE INFORMATION

The TrackPopupMenu function displays a "floating" pop-up menu at the specified location and tracks the selection of items on the pop-up menu. A floating pop-up menu can appear anywhere on the screen. The hMenu parameter specifies the handle of the menu to be displayed; the application obtains this handle by calling GetSubMenu to retrieve the handle of a pop-up menu associated with an existing menu item.

TrackPopupMenu is defined as follows:

```
TrackPopupMenu (hMenu%,wFlags%, X%, Y%, rRes%, hwnd%, lpRes&)
```

where:

```
hMenu%  - Identifies the pop-up menu to be displayed.
wFlags% - Is not used and must be set to zero.
x%      - Specifies the horizontal position in screen coordinates
          of the left side of the menu on the screen.
y%      - Specifies the vertical position in screen coordinates
          of the top of the menu on the screen.
nRes%   - Is reserved and must be set to zero.
hWnd%   - Identifies the window that owns the pop-up menu.
lpRes&  - Is reserved and must be set to NULL.
```

The supporting Windows API functions needed to support the arguments to TrackPopupMenu are:

1. GetMenu(hWnd%)

   ```
   hWnd%   - Identifies the window whose menu is to be examined.
   ```

   GetMenu returns a value that identifies the menu. The return value is NULL if the given window has no menu. The return value is

undefined if the window is a child window.

2. GetSubMenu(hMenu%, nPos%)

   hMenu% - Identifies the menu.
   nPos%   - Specifies the position in the given menu of the pop-up
             menu. Position values start at zero for the first
             menu item.

   GetSubMenu returns a value that identifies the given pop-up menu.
   The return value is NULL if no pop-up menu exists at the given
   position.

To create a pop-up menu within Visual Basic for Windows, define a menu system with the Menu Design window. The following is an example of a menu system:

Caption	Name	Indented
File	M_File	No
New	M_New	Once
Open	M_Open	Once
Close	M_Close	Once
Exit	M_Exit	Once
Help	M_Help	No

(In Visual Basic version 1.0 for Windows, set the CtlName Property for the above objects instead of the Name property.)

Within the general-declaration section of your Code window, declare the following:

```
' Enter each Declare statement on one, single line:
Declare Function TrackPopupMenu% Lib "user"(ByVal hMenu%, ByVal wFlags%,
   ByVal X%, ByVal Y%, ByVal r2%, ByVal hwnd%, ByVal r1&)
Declare Function GetMenu% Lib "user" (ByVal hwnd%)
Declare Function GetSubMenu% Lib "user" (ByVal hMenu%, ByVal nPos%)
```

Place the following code in the form's MouseUp event procedure:

```
Sub Form1_MouseUp (Button As Integer, Shift As Integer, X As Single,
      Y As Single)
   ' The above Sub statement must be concatenated onto one line.
   Const PIXEL = 3
   Const TWIP = 1
   ScaleMode = PIXEL
   InPixels = ScaleWidth
   ScaleMode = TWIP
   IX = (X + Left) \ (ScaleWidth \ InPixels)
   ' Enter the following IY statement on one, single line:
   IY = (Y + (Top + (Height - ScaleHeight -
        (Width - ScaleWidth)))) \ (ScaleWidth \ InPixels)
   hMenu% = GetMenu(hwnd)
   hSubMenu% = GetSubMenu(hMenu%, Button - 1)
   R = TrackPopupMenu(hSubMenu%, 0, IX, IY, 0, hwnd, 0)
End Sub
```

When you run the program, clicking anywhere in Form1 to display the first menu on your menu bar at that location.

Additional reference words: 1.00 2.00 3.00 pop up popup
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Create Rubber-Band Lines/Boxes in Visual Basic

Article ID: Q71488

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Creating rubber bands within Visual Basic can be done using the DrawMode property. Rubber bands are lines that stretch as you move the mouse cursor from a specified point to a new location. This can be very useful in graphics programs and when defining sections of the screen for clipping routines.

MORE INFORMATION

The theory of drawing a rubber-band box is as follows:

1. Draw a line from the initial point to the location of the mouse cursor using:

   ```
   [form].DrawMode = 6. {INVERT}
   ```

2. Move the mouse cursor.

3. Save the DrawMode.

4. Set the [form].DrawMode to 6. {INVERT}

5. Draw the same line that was drawn in step 1. This will restore the image underneath the line.

6. Set the [form].DrawMode back to the initial DrawMode saved in step 3.

7. Repeat the cycle again.

DrawMode equal to INVERT allows the line to be created using the inverse of the background color. This allows the line to be always displayed on all colors.

The sample below will demonstrate the rubber-band line and the rubber-band box. Clicking the command buttons will allow the user to select between rubber-band line or a rubber-band box. The user will also be able to select a solid line or a dashed line.

Create and set the following controls and properties:

Control	Name	Caption	Picture

```
   Form1          Form1        c:\windows\chess.bmp
   Command1       RubberBand
   Command2       RubberBox
   Command3       Dotted
   Command4       Solid
```

In the general section of your code, define the following constants:

```
Const INVERSE = 6       '*Characteristic of DrawMode property(XOR).
Const SOLID = 0         '*Characteristic of DrawStyle property.
Const DOT = 2           '*Characteristic of DrawStyle property.
Const TRUE = -1
Const FALSE = 0
Dim DrawBox As Integer  '*Boolean-whether drawing Box or Line
Dim OldX, OldY, StartX, StartY As Single  '* Mouse locations
```

In the appropriate procedures, add the following code:

```
Sub Form_MouseDown (Button As Integer, Shift As Integer, X As
                                       Single, Y As Single)
   '* Store the initial start of the line to draw.
   StartX = X
   StartY = Y

   '* Make the last location equal the starting location
   OldX = StartX
   OldY = StartY
End Sub

Sub Form_MouseMove (Button As Integer, Shift As Integer, X As
                     Single, Y As Single)
   '* If the button is depressed then...
   If Button Then
      '* Erase the previous line.
      Call DrawLine(StartX, StartY, OldX, OldY)

      '* Draw the new line.
      Call DrawLine(StartX, StartY, X, Y)

      '* Save the coordinates for the next call.
      OldX = X
      OldY = Y
   End If
End Sub

Sub DrawLine (X1, Y1, X2, Y2 As Single)
   '* Save the current mode so that you can reset it on
   '* exit from this sub routine. Not needed in the sample
   '* but would need it if you are not sure what the
   '* DrawMode was on entry to this procedure.
   SavedMode% = DrawMode

   '* Set to XOR
   DrawMode = INVERSE

   '*Draw a box or line
   If DrawBox Then
```

```
      Line (X1, Y1)-(X2, Y2), , B
   Else
      Line (X1, Y1)-(X2, Y2)
   End If

   '* Reset the DrawMode
   DrawMode = SavedMode%
End Sub

Sub Form_MouseUp (Button As Integer, Shift As Integer, X As Single,
                  Y As Single)
   '* Stop drawing lines/boxes.
   StartEvent = FALSE
End Sub

Sub Command2_Click ()
   '* Boolean value to determine whether to draw a line or box.
   DrawBox = TRUE
End Sub

Sub Command1_Click ()
   '* Boolean value to determine whether to draw a line or box.
   DrawBox = FALSE
End Sub

Sub Command3_Click ()
   '* Create a dotted line
   Form1.DrawStyle = DOT
End Sub

Sub Command4_Click ()
   '* Create a solid line.
   Form1.DrawStyle = SOLID
End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap PrgCtrlsStd

Determining Number of Lines in VB Text Box; SendMessage API

Article ID: Q72719

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

To determine the number of lines of text within a text box control, call the Windows API function SendMessage with EM_GETLINECOUNT(&H40A) as the wMsg argument.

Calling SendMessage with the following parameters will return the amount of lines of text within a text box:

```
hWd%   - Handle to the text box.
wMsg%   - EM_GETLINECOUNT(&H40A)
wParam% - 0
lParam% - 0
```

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

For example, to determine the amount of lines within a text box, perform the following steps:

1. Create a form with a text box and a command button. Change the MultiLine property of the text box to TRUE.

2. Declare the API SendMessage function in the global-declarations section of your code window (the Declare statement must be on just one line):

```
Declare Function SendMessage% Lib "user" (ByVal hWd%,
                                          ByVal wMsg%,
                                          ByVal wParam%,
                                          ByVal lParam&)
```

3. In Visual Basic version 1.0 for Windows, you will need to declare another API routine to get the handle of the text box. Declare this routine also in your global declarations section of your code window. The returned value will become the hWd% argument to the SendMessage function. For example:

```
Declare Function GetFocus% Lib "user" ()
```

4. Within the click event of your button, add the following code:

```
Sub Command1_Click ()
   Const EM_GETLINECOUNT = &H40A  ' Defined within Windows SDK
                                  ' file, WINDOWS.H.

   ' Command button has focus, give focus to text box.
   Text1.SetFocus

   ' For Visual Basic 1.0 for Windows get the handle of the text box.
   ' hWd% = GetFocus()

   ' Print the amount of lines to the immediate window.
   Debug.Print SendMessage(Text1.hWnd, EM_GETLINECOUNT, 0, 0)
   ' For Visual Basic 1.0 for Windows use hWd% instead of Text1.hWnd.
End Sub
```

5. Run the program. Add several lines of text to the text box. Click the command button to see the number of lines printed out to the immediate window.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther PrgCtrlsStd

How to Scroll VB Text Box Programmatically and Specify Lines

Article ID: Q73371

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

By making a call to the Windows API function SendMessage, you can scroll text a specified number of lines or columns within a Microsoft Visual Basic for Windows text box. By using SendMessage, you can also scroll text programmatically, without user interaction. This technique extends Visual Basic for Windows' scrolling functionality beyond the built-in statements and methods. The sample program below shows how to scroll text vertically and horizontally a specified number of lines.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Note that Visual Basic for Windows itself does not offer a statement for scrolling text a specified number of lines vertically or horizontally within a text box. You can scroll text vertically or horizontally by actively clicking the vertical and horizontal scroll bars for the text box at run time; however, you do not have any control over how many lines or columns are scrolled for each click of the scroll bar. Text always scrolls one line or one column per click the scroll bar. Furthermore, no built-in Visual Basic for Windows method can scroll text without user interaction. To work around these limitations, you can call the Windows API function SendMessage, as explained below.

Example

To scroll the text a specified number of lines within a text box requires a call to the Windows API function SendMessage using the constant EM_LINESCROLL. You can invoke the SendMessage function from Visual Basic for Windows as follows:

```
r& = SendMessage& (hWd%, EM_LINESCROLL, wParam%, lParam&)
```

hWd%	The window handle of the text box.
wParam%	Parameter not used.
lParam&	The low-order 2 bytes specify the number of vertical lines to scroll. The high-order 2 bytes specify the number of horizontal columns to scroll. A positive value for lParam& causes text to scroll upward or to the left. A negative value causes text to scroll downward or to the right.

r& Indicates the number of lines actually scrolled.

The SendMessage API function requires the window handle (hWd% above) of the text box. To get the window handle of the text box, you must first set the focus on the text box using the SetFocus method from Visual Basic. Once the focus has been set, call the GetFocus API function to get the window handle for the text box. Below is an example of how to get the window handle of a text box.

```
' The following appears in the general declarations section of
' the form:
Declare Function GetFocus% Lib "USER" ()

' Assume the following appears in the click event procedure of a
' command button called Scroll.
Sub Command_Scroll_Click ()
     OldhWnd% =  Screen.ActiveControl.Hwnd
     ' Store the window handle of the control that currently
     ' has the focus.

     ' For Visual Basic 1.0 for Windows use the following line:
     ' OldhWnd% =  GetFocus ()

     Text1.SetFocus
     hWd% = GetFocus()
End Sub
```

To scroll text horizontally, the text box must have a horizontal scroll bar, and the width of the text must be wider than the text box width. Calling SendMessage to scroll text vertically does not require a vertical scroll bar, but the length of text within the text box should exceed the text box height.

Below are the steps necessary to create a text box that will scroll five vertical lines or five horizontal columns each time you click the command buttons labeled "Vertical" and "Horizontal":

1. From the File menu, choose New Project (press ALT, F, N).

2. Double-click Form1 to bring up the code window.

3. Add the following API declaration to the General Declarations section of Form1. Note that you must put all Declare statements on a separate and single line. Also note that SetFocus is aliased as PutFocus because there already exists a SetFocus method within Visual Basic for Windows.

   ```
   Declare Function GetFocus% Lib "user" () ' For Visual Basic 1.0 only.
   Declare Function PutFocus% Lib "user" Alias "SetFocus" (ByVal
                                                          hWd%)
   Declare Function SendMessage& Lib "user" (ByVal hWd%,
                                             ByVal wMsg%,
                                             ByVal wParam%,
                                             ByVal lParam&)
   ```

4. Create a text box called Text1 on Form1. Set the MultiLine property to True and the ScrollBars property to Horizontal (1).

5. Create a command button called Command1 and change the Caption to "Vertical".

6. Create a another command button called Command2 and change the Caption to "Horizontal".

7. From the General Declarations section of Form1, create a procedure to initialize some text in the text box as follows:

```
Sub InitializeTextBox ()
   Text1.Text = ""
   For i% = 1 To 50
      Text1.Text = Text1.Text + "This is line " + Str$(i%)

      ' Add 15 words to a line of text.
      For j% = 1 to 10
         Text1.Text = Text1.Text + " Word "+ Str$(j%)
      Next j%

      ' Force a carriage return (CR) and linefeed (LF).
      Text1.Text = Text1.Text + Chr$(13) + Chr$(10)

      x% = DoEvents()
   Next i%
End Sub
```

8. Add the following code to the load event procedure of Form1:

```
Sub Form_Load ()
   Call InitializeTextBox
End Sub
```

9. Create the actual scroll procedure within the General Declarations section of Form1 as follows:

```
' The following two lines must appear on a single line:
Function ScrollText& (TextBox As Control, vLines As Integer, hLines
                      As Integer)
   Const EM_LINESCROLL = &H406

   ' Place the number of horizontal columns to scroll in the high-
   ' order 2 bytes of Lines&. The vertical lines to scroll is
   ' placed in the low-order 2 bytes.
   Lines& = Clng(&H10000 * hLines) + vLines

   ' Get the window handle of the control that currently has the
   ' focus, Command1 or Command2.
   SavedWnd%  = Screen.ActiveControl.Hwnd
   ' For Visual Basic 1.0 use the following line instead of the one
   ' used above.
   ' SavedWnd% = GetFocus%()

   ' Set the focus to the passed control (text control).
   TextBox.SetFocus

   ' For Visual Basic 1.0, get the handle to current focus (text
```

```
        ' control).
        ' TextWnd% = GetFocus%()

        ' Scroll the lines.
        Success& = SendMessage(TextBox.HWnd, EM_LINESCROLL, 0, Lines&)
        ' For Visual Basic 1.0 use the following line instead of the one
        ' used above.
        ' Success& = SendMessage(TextWnd%, EM_LINESCROLL, 0, Lines&)

        ' Restore the focus to the original control, Command1 or
        ' Command2.
        r% = PutFocus% (SavedWnd%)

        ' Return the number of lines actually scrolled.
        ScrollText& = Success&

    End Function
```

10. Add the following code to the click event procedure of Command1 labeled "Vertical":

```
    Sub Command1_Click ()
       ' Scroll text 5 vertical lines upward.
       Num& = ScrollText&(Text1, 5, 0)
    End Sub
```

11. Add the following code to the click event procedure of Command2 labeled "Horizontal":

```
    Sub Command2_Click ()
        ' Scroll text 5 horizontal columns to the left.
        Num& = ScrollText&(Text1, 0, 5)
    End Sub
```

12. Run the program. Click the command buttons to scroll the text five lines or columns at a time.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd APrgWindow

Overlapping Controls Not Supported in Visual Basic

Article ID: Q73651

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Overlapping Visual Basic controls may not respond as expected to mouse events. For example, the bottom control will receive the mouse event even when it appears that you have selected the top control. The use of overlapping Controls is not supported in Visual Basic version 1.0, however, in versions 2.0 and 3.0, overlapping Controls are supported.

MORE INFORMATION

Although the Visual Basic design editor allows you to overlap controls, when you run the application the region of the controls that overlap may not function as you would expect.

For example, if two Command buttons, Command1 and Command2, overlap so that Command1 is partially on top of Command2, when you select Command1 within the region of overlap you would expect a Click event to be issued for Command1. However, the Click event may occur on Command2 even though it is underneath Command1 in the overlapping region.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

PRB: Access Key Causes Different Event Order than Mouse Click

Article ID: Q74905

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Visual Basic for MS-DOS, version 1.0

SYMPTOMS

In Visual Basic, events may be generated in a different order if you choose a control (such as a button, a check box, or an option box) using an access key rather than with the mouse. The events that occur in a different order are Click, LostFocus, and GotFocus.

WORKAROUND

By inserting the DoEvents statement as the very first statement in the Click event handler, you can cause the LostFocus and GotFocus events to be handled before the body of the Click event handler.

STATUS

This behavior is by design. It is not a bug in Visual Basic.

MORE INFORMATION

You can create an access key at design time by changing the Caption property of a control to include an ampersand (&). The access key is the character after the ampersand, and at run time you press ALT+character to choose the control. (See page 120 of the "Microsoft Visual Basic: Programmer's Guide" version 1.0. manual.)

When you press an access key (ALT+character) to choose a control, the Click event is generated before the LostFocus and GotFocus event; however, when you choose a control by clicking the mouse, the LostFocus and GotFocus events are generated before the Click event.

The example below shows this different order of events. The example uses command buttons, but also applies to Check and Option boxes:

1. Open a new form and create two command buttons.

2. Enter the code as shown further below.

3. Change the Caption property of Command2 to "Command&2"

4. Run the program.

5. a. When Command1 has the focus and you click Command2, the
      following events are generated in the following order:

      Command1_LostFocus
      Command2_GotFocus
      Command2_Click

   b. When Command1 has the focus and you press the access key, ALT+2,
      the following events are generated in the following order:

      Command2_Click
      Command1_LostFocus
      Command2_GotFocus

Sample Code:

```
Sub Command1_Click ()
   Print "Command1_click"
End Sub

Sub Command1_LostFocus ()
   Print "Command1_lostfocus"
End Sub

Sub Command1_GotFocus ()
    Print "Command1_gotfocus"
End Sub

Sub Command2_Click ()
    Print "Command2_click"
End Sub

Sub Command2_LostFocus ()
   Print "Command2_lostfocus"
End Sub

Sub Command2_GotFocus ()
   Print "Command2_gotfocus"
End Sub
```

Additional reference words: 1.00 2.00 3.00 vbmsdos
KBCategory:
KBSubcategory: PrgCtrlsStd EnvtRun

Carriage Return+Linefeed to Wrap Lines in Text Box Control

Article ID: Q74906

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Under Microsoft Windows, version 3.0, using the carriage return character, Chr$(13), alone to create a line wrap to the next line in a Visual Basic text box control will cause the character following the carriage return to be removed from a multiline text box. Under Microsoft Windows, version 3.1, it will cause a pipe character '|' to be displayed in the place of the CHR$(13).

To correctly wrap to the next line, you must instead use both a carriage return and a linefeed, Chr$(10). This requirement is by design.

MORE INFORMATION

The correct method to create a line wrap is to use a carriage return character followed by a linefeed character, Chr$(13) + Chr$(10). The Windows text box expects to find this sequence and assumes that the character following the carriage return is a linefeed, thus removing the following character as if it were a linefeed.

The following steps show the results of using just the carriage return, and the results of using both carriage return and linefeed characters in a text box.

1. In a new project, click the text box icon from the Toolbox (second tool down in the right column).

2. Click anywhere on the form and drag diagonally to create a text box large enough to hold more then one line of text.

3. From the Properties bar (below the main menu) scroll down to Multiline, then choose the Settings box for that Multiline property (also on the Properties bar below the menu) and choose True. The text box can now accommodate several lines of text.

4. Double-click anywhere in the form outside of the text box to bring up the Form_click code window (or use the F7 function key).

5. On the line below Sub Form_click (), type the following:

```
Text1.text = "Hello" + Chr$(13) + "World"
```

6. Press F5 to run the newly created application, then click

anywhere in the form outside the text box. The following text will appear.

For Windows, version 3.0:

```
Hello
orld
```

Note that the W of "World" is missing.

For Windows, version 3.1:

```
Hello|World
```

7. To obtain the desired result, you must add a linefeed following the carriage return character, as follows:

```
Text1.text = "Hello" + Chr$(13) + Chr$ (10) + "World"
```

This statement will display the expected result of:

```
Hello
World
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Program Example for COM Port Support in Visual Basic

Article ID: Q75856

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

A sample program is available to show how a Visual Basic program can use Windows API functions for serial port communications. This program may not be necessary in Visual Basic versions 2.0 and 3.0 for Windows because COM support is already built into the Communications Control.

VBCOMDEM can be found in the Microsoft Software Library by searching for the word VBCOMDEM, the Q number of this article, or S13150. VBCOMDEM was archived using the PKware file-compression utility. When you decompress VBCOMDEM, you will have the following four files:

SIMPCOMM.EXE, SIMPCOMM.FRM, SIMPCOMM.GLB, SIMPCOMM.MAK

MORE INFORMATION

In the Visual Basic environment (VB.EXE), you can load the files in this sample program by choosing Open Project from the File menu and selecting the SIMPCOMM.MAK file.

You can also run SIMPCOMM.EXE in Windows as a separate program that requires the Visual Basic run-time file VBRUN100.DLL.

This sample program is only a starting point. It does not use all of the serial communications API functions available through Windows. This simple example uses Windows API Comm functions, such as OpenComm, CloseComm, ReadComm, and WriteComm. You are free to modify and extend the program to suit your specific needs.

The SIMPCOMM program has no error trapping, and makes no allowances for noisy communication lines or handshaking errors. Should an error occur, Windows will suspend all reading from the communications port until you clear the error by calling the Windows API function GetCommError.

To modify or understand this program example, you must have a reference manual for the Windows API routines.

REFERENCES

"Microsoft Windows Programmer's Reference," Microsoft Press, 1990
Microsoft Windows 3.0 Software Development Kit

Additional reference words: 1.00 2.00 3.00 COM1 COM2 asynchronous

KBCategory:
KBSubcategory: PrgCtrlsStd

VB Procedure Form_Load Not Executed when Unload Not Used

Article ID: Q76629

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Code inside a Form_Load event procedure will not execute under the circumstances described below. The example below helps clarify the behavior of the Load event procedure.

A Load event procedure will only execute when a form is loaded, either with the Load statement or an implicit load. An implicit load is caused when a form is currently not loaded, and a property or method accesses the form or associated control.

This behavior is by design in Microsoft Visual Basic programming system version 1.0 for Windows.

MORE INFORMATION

Below is a demonstration of this behavior:

1. From the File menu, choose New Project.

2. From the File menu, choose New Form.

3. Place a command button on each form. Place command button 1 on form 1 and command button 2 on form 2.

4. Place the following code in the event procedure Command1_Click in form 1:

```
Sub Command1_Click ()
 Form1.MousePointer = 11  'Hourglass pointer
 Form2.Show
End Sub
```

5. Add the following code in the event procedure Form_Load in form 1:

```
Sub Form_Load ()
 Form1.MousePointer = 0   'Default pointer
End Sub
```

6. Add the following code in the event procedure Command2_Click in form 2:

```
Sub Command2_Click ()
```

```
        Form2.MousePointer = 11  'Hourglass pointer
        Form1.Show
       End Sub
```

7. Add the following code in the event procedure Form_Load in form 2:

```
       Sub Form_Load ()
        Form2.MousePointer = 0   'Default pointer
       End Sub
```

8. Run the program with the F5 key. You will see Form1 load up with the Command1 button on it. If you click the Command1 button, you will see the mouse cursor change to an hourglass until Form2 is loaded. With Form2 loaded, you can see that the mouse cursor is back to the default arrow. Click the Command2 button and see the mouse cursor change back to an hourglass until Form1 is loaded.

   This is where the behavior starts; the hourglass continues to be displayed instead of going back to the default arrow. This is because the code Form1.MousePointer = 0 in the Form_Load event procedure of Form1 is not being executed. You can continue by clicking the Command1 button again to go to Form2 and the hourglass continues to be displayed.

The easiest way to work around this behavior is to add an Unload statement after each .Show statement, as shown below:

```
   Sub Command1_Click ()
    Form1.MousePointer = 11
    Form2.Show
    Unload Form1           'new line of code to be added
   End Sub

   Sub Command2_Click ()
    Form2.MousePointer = 0
    Form1.Show
    Unload Form2           'new line of code to be added
   End Sub
```

Note: This method may slow the painting of forms at run-time, but this method will guarantee that the Form_Load event procedure is executed when the Show method is executed.

Another workaround is to place the code

```
   .MousePointer = 0 statements
```

into the Form_Paint event procedures. Note that this method will only work when one form is being painted over another. Use the Cut and Paste routines from the Edit menu of Visual Basic. Cut the following line of code

```
   Form1.MousePointer = 0
```

from the event procedure Form_Load in Form1 and paste the code into the Form1 Form_Paint event procedure. Repeat the same Cut and Paste task in Form2, placing the code

```
   Form2.MousePointer = 0
```

in the Form2 Form_Paint event procedure.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

VB Forms with Menus Cannot Have Fixed Double BorderStyle

Article ID: Q76630

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Windows versions 3.0 and 3.1

SUMMARY

Because of Windows version 3.0 and 3.1 limitations, forms with menus cannot have the BorderStyle property set to Fixed Double. To have menus, a form's BorderStyle property must be either None, Fixed Single, or Sizable.

MORE INFORMATION

Steps to Reproduce Problem

1. Run Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. In the Menu Design window, create a menu on Form1.

3. Set the BorderStyle of Form1 to Fixed Double.

4. Run the program.

Note that the border style is fixed single.

Because of a Windows problem with menus on forms with fixed double borders, Visual Basic does not paint the menus correctly. For this reason, Visual Basic does not allow this particular combination of a menu on a form with a fixed double border.

For more information on this limitation, query on the following words in the Microsoft Knowledge Base:

   visual basic and menu and caption and bar

Additional reference words: 1.00 3.00 3.10
KBCategory:
KBSubcategory: PrgCtrlsStd EnvtDes

PRB: Long String Assigned to Multiline Text Box Seems to Hang

Article ID: Q76635

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

On some computers, when you assign a long text string to a multiline text box, it takes a long time (1 to 2 minutes) to update. This may give the impression that Visual Basic is hung, when in fact it is not.

MORE INFORMATION

Windows has a problem inserting line breaks in multiline text boxes. The amount of time needed to complete the process grows exponentially as the length of the string increases.

Steps to Reproduce Problem

1. In Visual Basic, place a text box (Text1) on a new form, and change the MultiLine property of Text1 to True.

2. Place the following statement in the Form_Click event procedure:

   ```
   text1.text=string$(32767,"X")
   ```

3. From the Run menu, choose Start.

4. Click the form.

The application may now take up to two minutes to respond to any other events because it is still executing the text1.text assignment.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

DEL Key Behavior Depends on Text Box MultiLine Property

Article ID: Q77737

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Pressing the DEL key in a multiline text box generates a KeyPress event for that text box with an ASCII code of 8 for the key. In a standard text box, no KeyPress event is generated for the DEL key. This behavior is inherent to Windows and is not specific to Microsoft Visual Basic for Windows.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Steps to Reproduce Problem

1. Place a text box on a form.

2. Set the MultiLine property for the text box to True.

3. Add the following code to the text box KeyPress event:

```
Sub Text1_KeyPress (keyAscii as Integer)
     debug.print keyAscii  ' This will print the generated ASCII
                           ' code to VB's Immediate window.
End Sub
```

4. Execute the program and press the DEL key while the focus is on the text box. An "8" will be printed in the Immediate window.

If the text box's MultiLine property is set to false, no KeyPress event occurs and nothing is printed to the Immediate window when you press the DEL key. This behavior is standard for Windows multiline text boxes.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

PRB: Clipboard.SetData Gives Invalid Format Message with Icon

Article ID: Q78073

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

If you use the Visual Basic LoadPicture function to load an icon file (.ICO) into a picture control, and then attempt to copy that picture control's picture to the Clipboard by using the SetData method, the following error message is displayed regardless of the format specified in SetData method:

```
Invalid Clipboard Format
```

This error also occurs if you attempt to load an icon file directly onto the Clipboard by using this code:

```
ClipBoard.SetData LoadPicture("c:\vb\icons\arrows\arw01rt.ico")
```

CAUSE

The Microsoft Windows Clipboard in Windows has no CF_ICON format, so the Clipboard cannot be assigned Icons.

WORKAROUND

To work around the problem, set the picture control's Autoredraw property to True (-1) and use the Picture control's Image property in the SetData method rather than the Picture control's picture property.

```
'*** This code will fail with the error "Invalid Clipboard Format" ***
Picture1.Picture = LoadPicture("c:\vb\icons\arrows\arw01rt.ico")
Clipboard.SetData Picture1.Picture, 2

'*** This code will avoid the error ***
Picture1.Autoredraw = -1
Picture1.Picture = LoadPicture("c:\vb\icons\arrows\arw01rt.ico")
Clipboard.SetData Picture1.Image, 2

'*** This code will also work ***
Picture1.Picture = LoadPicture("c:\vb\icons\arrows\arw01rt.ico")
Picture1.Picture = Picture1.Image
Clipboard.SetData Picture1.Picture, 2
```

STATUS

Microsoft has confirmed this to be a limitation of the Microsoft Windows Clipboard.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Disabling the ENTER Key BEEP in a Visual Basic Text Box

Article ID: Q78305

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- The Standard and Professional Editions of Microsoft Visual Basic for MS-DOS, version 1.0

SUMMARY

In a Microsoft Visual Basic for Windows text box, the ENTER key causes a warning beep to sound only if the MultiLine property is set to False (the default) and the Warning Beep option is selected in the Sound dialog box of the Windows Control panel. To disable the beep, in the KeyPress event procedure for the text box, set the value of KeyAscii (which is a parameter passed to KeyPress) equal to zero (0) when the user presses the ENTER key.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Specifically, use an IF statement to trap the ENTER key and the set KeyAscii to zero (0). Setting the value to zero before the event procedure ends prevents Windows from detecting that the ENTER key was pressed and prevents the warning beep. This behavior is by design and is due to the fact that a non-multiline text box is a Windows default class of edit box.

Example

The following code will prevent the beep.

```
' (Set Multiline property to False).

Sub Text1_KeyPress (KeyAscii as Integer)
  If KeyAscii=13 Then
    KeyAscii=0
  End If
End Sub
```


Additional reference words: 1.00 return 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Scope of Line Labels/Numbers in Visual Basic for Windows

Article ID: Q78335

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY
=======

Line labels (and line numbers) do not follow the same scoping rules as variables and constants in Visual Basic for Windows. Line labels must be unique within each module and form. However, you can only transfer control to a line label or line number within the current Sub or Function.

MORE INFORMATION
================

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

When you attempt to define the same line label twice within a module or form, you receive the error message "Duplicate label". This message means that the label is already defined in another procedure within the current module.

When you use a GOTO or GOSUB statement that names a line label defined in another procedure, you receive the error message "Label not defined." This message means that the label is not defined in the current Sub or Function.

For more information about line labels, see the description of the GOTO and GOSUB statements in the "Microsoft Visual Basic: Language Reference" or in the Visual Basic for Windows online Help system.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Make a Push Button with a Bitmap in Visual Basic

Article ID: Q78478

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Command buttons in Visual Basic for Windows are limited to a single line of text and one background color (gray). The 3D command button shipped in the Professional Editions of Visual Basic version 2.0 and 3.0 for Windows does have the capability of displaying bitmaps within a command button in Visual Basic for Windows. However, there is no way to alter the background or border colors to change its appearance. You can create the look and feel of a command button by using a picture control and manipulating the DrawMode in conjunction with the Line method. Using a picture control also allows you to display the "command button" in any color with multiple lines of caption text.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The technique (demonstrated further below) simulates the effect of pressing a command button by using the Line method with the BF option (Box Fill) in invert mode each time a MouseUp or MouseDown event occurs for the picture control. To add multiline text to the "button," either print to the picture box or add the text permanently to the bitmap.

The steps to create a customized "command button" are as follows:

1. Start Visual Basic for Windows, or choose New Project from the File menu (press ALT, F, N) if Visual Basic for Windows is already running. Form1 will be created by default.

2. Put a picture control (Picture1) on Form1.

3. Set the properties for Picture1 as given in the chart below:

Property	Value
AutoRedraw	True
AutoSize	True
BorderStyle	0-None
DrawMode	6-Invert

4. Assign the Picture property of Picture1 to the bitmap of your

choice. For example, choose ARW01DN.ICO from the ARROWS subdirectory of the ICONS directory shipped with Visual Basic for Windows. This is a good example of a bitmap with a three dimensional appearance.

5. Enter the following code in the Picture1_DblClick event procedure of Picture1:

```
Sub Picture1_DblClick ()
    Picture1.Line (0, 0)-(Picture1.width, Picture1.height), , BF
End Sub
```

Note: This code is necessary to avoid getting the bitmap stuck in an inverted state because of Mouse messages being processed out of order or from piling up due to fast clicking.

6. Enter the following code in the Picture1_MouseDown event procedure of Picture1:

```
Sub Picture1_MouseDown (Button As Integer, Shift As Integer, X As
                        Single, Y As Single)  ' Append to above line
    Picture1.Line (0, 0)-(Picture1.width, Picture1.height), , BF
End Sub
```

7. Enter the following code in the Picture1_MouseUp event procedure of Picture1:

```
Sub Picture1_MouseUp (Button As Integer, Shift As Integer,
                      X As Single, Y As Single) ' Append to above line.
    Picture1.Line (0, 0)-(Picture1.width, Picture1.height), , BF
End Sub
```

8. Add the following code to the Picture1_KeyUp event procedure for Picture1:

```
Sub Picture1_KeyUp (KeyCode As Integer, Shift As Integer)
    '* Check to see if the ENTER key was pressed.  If so, restore
    '* the picture image.
    If KeyCode = 13 Then
      Picture1.Line (0, 0)-(Picture1.width, Picture1.height), , BF
    End If
End Sub
```

9. Add the following code to the Picture1_KeyDown event procedure for Picture1:

```
Sub Picture1_KeyDown (KeyCode As Integer, Shift As Integer)
    '* Check to see if the ENTER key was pressed.  If so, invert
    '* the picture image.
    If KeyCode = 13 Then
      Picture1.Line (0, 0)-(Picture1.width, Picture1.height), , BF
    End If
End Sub
```

10. From the Run menu, choose Start. Click the picture box. The image of the picture should be inverted while the mouse button is down, giving the visual effect of a button press.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

No New Timer Events During Visual Basic Timer Event Processing

Article ID: Q78599

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY
=======

Timer controls can be used to automatically generate an event at predefined intervals. This interval is specified in milliseconds, and can range from 0 to 65535 inclusive.

Timer event processing will not be interrupted by new timer events. This is because of the way that Windows notifies an application that a timer event has occurred. Instead of interrupting the application, Windows places a WM_TIMER message in its message queue. If there is already a WM_TIMER message in the queue from the same timer, the new message will be consolidated with the old one.

After the application has completed processing the current timer event, it checks its message queue for any new messages. This queue may have new WM_TIMER messages to process. There is no way to tell if any WM_TIMER messages have been consolidated.

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Creating Nested Control Arrays in Visual Basic

Article ID: Q79029

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

This article explains how to create an array of picture controls or frames with an array of child controls (such as command buttons) within each element of the parent array by using the Windows API functions SetParent and GetFocus. This is not possible in Visual Basic for Windows without using the Windows API functions because Visual Basic does not support overlapping controls. In other words, in Visual Basic alone, you cannot create controls and then simply move them into position within previously created controls.

By using the Windows API functions in conjunction with the Load and Unload methods, you can circumvent this problem and allow dynamic, flexible structures to be created during execution.

MORE INFORMATION

This article explains how to get the following control structure:

Parent Control	Child Controls on Parent
Picture1(1)	Command1(1), Command1(2), Command1(3), Command1(4)
Picture1(2)	Command1(5), Command1(6), Command1(7), Command1(8)
Picture1(3)	Command1(9), Command1(10), Command1(11), Command1(12)
Picture1(4)	Command1(13), Command1(14), Command1(15), Command1(16)

(where Picture1 and Command1 are control arrays).

The following example uses the two API functions (GetFocus and SetParent) to establish the correct parent/child relationships between an array of parents (such as picture controls) and an array of children (such as command buttons). Each child array is placed within one element of the parent control.

The GetFocus function requires no parameters. The SetParent function requires two parameters as follows:

Parameter	Type and Description
hWndChild	HWND Identifies the child window
hWndNewParent	HWND Identifies the new parent window

The return value identifies the previous parent window.

This example also demonstrates how Windows handles a drag and drop when parentage was set at run time. If you drag a control to another control of the same type as the previous parent and drop it, the released control assumes the same relative position it had in the previous parent.

The example program demonstrates that before unloading controls nested using SetParent and moved during the run, parentage should be returned to the original hierarchy. This avoids the possibility of general protection (GP) faults or Unrecoverable Application Errors (UAEs) that could occur due to conflicting messages to Windows.

Step-by-Step Example
---

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a picture box and five command buttons to Form1, giving them the property settings shown here:

Control	Control Name	Property Setting
Form	Form1	
Picture	Picture1()	Index=1
Command button	Command1()	Index=1
Command button	Loadall	TabIndex=0, Caption="Load All"
Command button	Loadsome	TabIndex=1, Caption="Load Four"
Command button	Changemode	TabIndex=2, Caption="DragMode=0"
Command button	Unloadall	TabIndex=3, Caption="Unload All"

   NOTE: The placement of the original picture box and command button is important. The picture should be created first and the command button drawn within the picture box. The placement of the other controls at design time is not critical. They are resized and moved at run time.

3. Add code to the examaple application. Here is a summary of each of the Sub procedures you will create:

Code	Purpose
Loadall_Click	Loads all parent controls and all child controls
Loadsome_Click	Loads all parent controls and four child controls
Unloadall _Click	Unloads all controls. Must reset parentage to the original first!
Changemode_Click	Changes DragMode between auto and manual modes
Resetparent	General procedure to reset parentage for Unload and Close
Cplace	General procedure to place the four command buttons
Form_Resize	Code to maintain original size
Form_Load	Sets initial size and properties of controls
Form_Unload	Calls Unloadall_Click to avoid conflicting Windows messages
Picture1_DragDrop	Handles setting of parentage to user actions
Command1_Click	Sets captions to reflect change in position and state

4. Add the following code to general declarations section of Form1:

```
Declare Function setparent% Lib "user" (ByVal h%, ByVal h%)
Declare Function getfocus% Lib "user" ()
Global Handle2Child As Integer
Global Handle2Parent As Integer
Global dragstate, loadstate, toggle, innernum As Integer
Global I, N, K As Integer
Global xoffset, yoffset, cmdnum, childsize, parentsize As Integer
Global Const maxouter = 4
Global Const maxinner = 4
Option Base 1
Global storecaption(16) As String     ' array=maxinner*maxouter
```

5. Add the following Sub procedures to the application in apprpriate events in Form1. Each code statement must be entered as one, single line.

```
' Enter the following two lines as one, single line:
Sub Picture1_DragDrop (index As Integer, source As Control, X As Single,
   Y As Single)
   picture1(index).SetFocus
   ' Procedure for control array of parent Picture Boxes:
   Handle2Parent = getfocus()
   source.SetFocus
   Handle2Child = getfocus()
   ret% = setparent(Handle2Child, Handle2Parent)
   source.caption = Mid$(source.caption, 1, 1) + "/" +
   LTrim$(RTrim$(Str$(index)))
End Sub

Sub Form_Load ()
   form1.width = screen.width - screen.width \ 8
   form1.height = screen.height \ 2
   form1.backcolor = &HFFFF00
   form1.caption = "Nested Control Arrays"
   picture1(1).visible = 0
   command1(1).visible = 0
   parentsize = CInt((form1.width \ maxouter) * .8)
   childsize = CInt((2 * parentsize \ maxinner) * .6)
   picture1(1).height = parentsize
   picture1(1).width = parentsize
   command1(1).height = childsize
   command1(1).width = childsize
   cplace loadall, 1
   cplace loadsome, 2
   cplace changemode, 3
   cplace unloadall, 4
End Sub

Sub resetparent ()      ' Function to clean up parentage before unload.
   picture1(1).SetFocus
   Handle2Parent = getfocus()
   For I = innernum To 1 Step -1
      command1(I).SetFocus
      Handle2Child = getfocus()
      ret% = setparent(Handle2Child, Handle2Parent)
   Next I
End Sub
```

```
Sub Command1_Click (index As Integer)    ' Procedure for control array
                                         ' of Buttons.
   If toggle Then
      command1(index).caption = storecaption(index)
      toggle = 0
   Else
      storecaption(index) = command1(index).caption  ' Change caption to
                                                     ' reflect state.
      command1(index).caption = "ON"
      toggle = -1
   End If
End Sub

Sub Form_Unload (Cancel As Integer)  ' Cleans up before program exits.
   unloadall_click
End Sub

Sub changemode_Click () ' Toggles between automatic & manual dragmodes.
   If loadstate Then
      If Not dragstate Then
         For I = 1 To innernum
            command1(I).dragmode = 1   ' Automatic
            dragstate = -1             ' Reset flag.
         Next I
         changemode.caption = "DragMode=1"
      Else
         For I = 1 To innernum
            command1(I).dragmode = 0   ' Manual
            dragstate = 0              ' Reset flag.
         Next I
         changemode.caption = "DragMode=0"
      End If
   End If
End Sub

Sub unloadall_click () ' Unloads all dynamically created controls only.
   Select Case loadstate
   Case 1
      resetparent  ' Must call prior to unload to avoid GP fault or UAE

      For I = maxouter To 1 Step -1
         For N = maxinner To 1 Step -1
            cmdnum = ((I - 1) * 4) + N
            If cmdnum <> 1 Then Unload command1(cmdnum)
         Next N
            If I <> 1 Then Unload picture1(I)
      Next I
      command1(1).visible = 0     ' Can't unload controls
      picture1(1).visible = 0     ' created at design time so hide!
      loadstate = 0               ' Reset flag for load routines.
      changemode.enabled = 0

   Case 2
      resetparent  ' Must call prior to unload to avoid GP fault or UAE.
      For I = maxouter To 1 Step -1
         If I = 1 Then
            For N = maxinner To 2 Step -1
```

```
            Unload command1(N)
         Next N
      End If
      If I <> 1 Then Unload picture1(I)
   Next I
   command1(1).visible = 0      ' Can't unload controls
   picture1(1).visible = 0      ' created at design time so hide!
   loadstate = 0                ' Reset flag for load routines.
   changemode.enabled = 0
End Select
End Sub

Sub loadsome_click ()  ' Loads all parents and one set of children
   If loadstate = 0 Then  ' to demonstrate drag and drop.
      changemode.enabled = -1
      command1(1).Move 0, 0
      For I = 1 To maxouter
         If I <> 1 Then    ' Can't load control created at design time.
            Load picture1(I)
         End If
         ' Enter the following two lines as one, single line:
         picture1(I).Move -picture1(1).width, -picture1(1).height,
            parentsize, parentsize
         picture1(I).visible = -1     ' Load off-screen /\.
         Picture1(I).SetFocus
         Handle2Parent = getfocus()   ' Get handle by API call.
         If I = 1 Then
            For N = 1 To 4
               If N <> 1 Then
                  Load command1(N)  ' Can't load control created at
               End If               ' design time.
               ' Enter the following two lines as one, single line:
               xoffset = picture1(I).scalewidth \ 4 - command1(N).width
                  \ 2 + ((N - 1) Mod 2) * (picture1(I).scalewidth \ 2)
               If N > 2 Then
                  ' Enter the following three lines as one, single line:
                  yoffset = picture1(I).scaleheight \ 2 +
                     (picture1(I).scaleheight \ 4 -
                     command1(N).height \ 2)
               Else
                  ' Enter the following two lines as one, single line:
                  yoffset = (picture1(I).scaleheight \ 4 -
                     command1(N).height\2)
               End If
               command1(N).Move xoffset, yoffset
               command1(N).visible = -1
               command1(N).SetFocus
               Handle2Child = getfocus()  ' Get handle by API call.
               ' Call API function.
               ret% = setparent(Handle2Child, Handle2Parent)
               ' Enter the following two lines as one, single line:
               command1(N).caption = LTrim$(RTrim$(Str$(N))) + "/" +
                  LTrim$(RTrim$(Str$(I)))
            Next N
         End If
      Next I
      xoffset = ((form1.scalewidth \ maxouter) - picture1(1).width) \ 2
```

```
      picture1(1).Move xoffset, 0
      For I = 2 To maxouter
         ' Enter the following line as one, single line:
         picture1(I).Move (I - 1) * (form1.scalewidth \ maxouter) +
            xoffset, picture1(I - 1).top ' **
      Next I
      innernum = 4  ' Set global loop maximum.
      loadstate = 2
   End If
End Sub

Sub loadall_click ()         ' Loads all parents and children in
   If loadstate = 0 Then     ' nested structure.
      changemode.enabled = -1
      command1(1).Move 0, 0
      For I = 1 To maxouter  ' size command button proportionally.
         If I <> 1 Then Load picture1(I) ' Can't load control created at
                                         ' design time.
         ' Load off-screen:
         picture1(I).Move -picture1(1).width, -picture1(1).height
         picture1(I).visible = -1
         picture1(I).SetFocus
         Handle2Parent = getfocus()      ' Get handle by API call.
         For N = 1 To maxinner
            cmdnum = ((I - 1) * 4) + N
            If cmdnum <> 1 Then
               Load command1(cmdnum) ' Can't load control created at
                                     ' design time.
            End If
            xoffset=((N-1) Mod 2)*(picture1(I).scalewidth\(maxinner\2))
            If N > 2 Then
               yoffset = picture1(I).scaleheight \ 2
            Else yoffset = picture1(I).scaletop
            End If
            ' Enter the following command as one, single line:
            command1(cmdnum).Move picture1(I).scalewidth \ 8 +
                xoffset, picture1(I).scaleheight \ 8 + yoffset
            command1(cmdnum).visible = -1
            command1(cmdnum).SetFocus
            Handle2Child = getfocus()      ' Get handle by API call.
            ' Call API function.
            ret% = setparent(Handle2Child, Handle2Parent)
         Next N
      Next I          ' Caption the control array buttons.
      For K = 1 To (maxinner * maxouter)
         command1(K).caption = LTrim$(RTrim$(Str$(K)))
      Next K
      xoffset = ((form1.scalewidth \ maxouter) - picture1(1).width) \ 2
      picture1(1).Move xoffset, 0
      For I = 2 To maxouter
              picture1(I).Move (I - 1) * (form1.scalewidth \ maxouter) +
                                 xoffset, picture1(I - 1).top ' **
      Next I
      innernum = 16   ' Set global loop maximum.
      loadstate = 1
   End If
End Sub
```

```
   Sub Form_Resize ()
      form1.width = screen.width - screen.width \ 8
      form1.height = screen.height \ 2
   End Sub

   Sub cplace (dummy As Control, num As Integer)  ' Size static controls.
      theheight% = parentsize + childsize * 2
      ' Enter the following two lines as one, single line:
      dummy.Move (form1.width \ 4) * (num - 1) + parentsize \ 10,
         theheight%, parentsize, childsize ' **
   End Sub
```

4. Run the example and try all the buttons. Toggle the DragMode on and off and drag the command buttons from one picture to another.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Parameter Mismatch Error When Pass Properties by Reference

Article ID: Q79597

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Control property values in Visual Basic are stored in a formatted form whose location is periodically changed as part of Windows memory management. The values are accessed by handles, not addresses. Although the values behave like their prescribed types when used directly, they cannot be passed by reference to a SUB or FUNCTION. Any attempt to do so will generate a "PARAMETER MISMATCH" error.

MORE INFORMATION

Passing by reference, the default parameter passing method in Visual Basic, places the address of the variable on the stack. The SUB or FUNCTION then accesses the address on the stack and uses it to refer to that variable. Sending a control property as a parameter to a SUB or FUNCTION will place its handle on the stack instead of an address. Because the handle uses a different form than an address, the SUB or FUNCTION accesses a value that it is not expecting, and will generate a "PARAMETER MISMATCH" error.

As a workaround, pass the property by value instead of by reference. To pass by value, place a set of parentheses around the property variable in the SUB or FUNCTION call. This syntax will place the actual value of the property on the stack and tell the SUB or FUNCTION to treat it as such. Because an actual memory location is not transferred to the SUB or FUNCTION, any changes to the value of the property are localized to that SUB or FUNCTION.

Another workaround is to assign the property value to a temporary variable. The temporary variable has an actual address and can be passed to a SUB or FUNCTION in the usual manner. Because an actual address is sent, any change to the temporary variable will be permanent. In order for the actual property variable to reflect this change, the value of the temporary variable must be assigned to the property variable upon return from the SUB or FUNCTION.

Example

Create a project with one form (Form1), two command buttons (Command1 and Command2), and one text box (Text1). Add the two command Click events as follows:

```
Sub Command1_Click ()
```

```
    Text1.text = "passed by value"
    CALL Mysub ((Text1.text))
        ' Notice Text1.text did not change.
End Sub

Sub Command2_Click()
        Text1.text = "passed temporary variable"
        temp$ = Text1.text
        CALL Mysub (temp$)
        Text1.text = temp$
        ' Notice Text1.text did change when assigned to temp$.
End Sub
```

In the General section of Form1, add the following:

```
Sub Mysub(A$)
  A$ = "Changed"
End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Double-Clicking the Control Box Causes MouseUp Event in VB

Article ID: Q79599

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Double-clicking the control box of a form to close it will cause a MouseUp event on an enabled form or control if it is lying beneath the control box. This is a standard behavior inherent to Windows, and is not an error in Visual Basic.

MORE INFORMATION

You can prevent the above behavior in several ways:

- Set a global flag in the MouseDown event and check the flag in the MouseUp event. If the flag is set, perform the event and set the flag to FALSE. If the flag is not set, exit the MouseUp event.

- Set a global flag in the overlapping form's Form_Unload event, and then test this flag in the underlying form or control's MouseUp event.

- Restrict the placement or movement of a form so that it's control box does not appear above an enabled form or control.

- Avoid coding the MouseUp event of any enabled form or control over which a control box may appear.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Place Animated Graphics on a Minimized Form in VB

Article ID: Q79601

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can place animated graphics onto a minimized form in Visual Basic. Normally, when a form is minimized, the form is replaced with an icon that had been previously set using the Icon property of that form. This icon is an actual bitmap that cannot be manipulated. Using the method below, the icon can be replaced with a set of graphics methods that will draw to the minimized form.

MORE INFORMATION

To place animated graphics onto a minimized form, you must use a timer event. This will allow the program to continue its animation when the form is minimized. A minimized form is just like a non-minimized form, except its size is decreased and certain rules apply. The following guidelines should be followed when creating animated graphics on a form:

- The AutoRedraw property must be set to 0 (False).
- The user must place routines in the Paint event procedure to handle cases when the Paint event occurs in the maximized form. In the minimized form, a Paint event never occurs, and you must depend upon the timer event to refresh the icon representing the minimized form.
- The user must handle the painting of the background because a minimized form has no background, only foreground.
- Adjust your animation to the size of the minimized form by using either the Scale method or the ScaleWidth and ScaleHeight property.

The following example creates an animated icon that displays random circles every 500 milliseconds:

1. From the File menu, choose New Project.

2. Remove the icon from the Icon property. (You can do this by selecting the Icon property and pressing the DELETE key.)

3. Place a new timer control on the form.

4. Change the timer interval to 500.

5. Type the following code into the new timer event:

```
Static prevx!, prevy!
If windowstate = 1 Then           'Checks to see if form is
                                  'minimized.
    form1.Scale (0, 0)-(100, 100) 'Sets the max height and
                                  'width of the form.
    fillcolor = QBColor(0)
    Circle (prevx!, prevy!), scalewidth / 10, QBColor(0)
    fillstyle = 0
    fillcolor = QBColor(1)
    prevx! = Int(Rnd(1) * scalewidth) + 1
    prevy! = Int(Rnd(1) * scaleheight) + 1
    Circle (prevx!, prevy!), scalewidth / 10, QBColor(1)
End If
```

6. From the Run menu, choose Start.

7. Minimize the form by choosing Minimize from the control box menu, or click the minimize arrow (the minimize arrow is the down arrow) on the form.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Convert Units to Pixels for DrawWidth in VB

Article ID: Q79604

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The DrawWidth property controls line thickness for the graphics methods Circle, Line, and PSet. You can only set DrawWidth in units of pixels. Pixel size and density vary among video and printer devices.

This article describes how to set DrawWidth to the number of pixels to correspond with measurements in units other than pixels.

MORE INFORMATION

The following steps describe how to calculate DrawWidth from units other than pixels, referred to as "target units."

1. Determine the form (or printer) width in target units by setting the ScaleMode property to one of the values listed below and then retrieving the ScaleWidth property.

   ScaleMode Settings

   0 -- user-defined
   1 -- twips
   2 -- points
   4 -- characters
   5 -- inches
   6 -- millimeters
   7 -- centimeters

   For example:

   ```
   Form1.ScaleMode = 7  ' centimeters
   cm = Form1.ScaleWidth
   ```

2. Determine the form (or printer) width in pixels by setting the ScaleMode property to 3 (PIXELS in CONSTANT.TXT) and then retrieving the ScaleWidth property.

   For example:

   ```
   Form1.ScaleMode = 3
   pixel = Form1.ScaleWidth
   ```

3. Calculate the ratio of pixels per target unit by dividing the form (or printer) width in target units by the form (or printer) width in pixels.

   For example:

      pixel_per_cm = pixel / cm

4. Set DrawWidth to the number of target units multiplied by the ratio of pixels per target unit.

   For example:

      Form1.DrawWidth = 5 * pixel_per_cm ' 5cm thick lines

The following code example demonstrates how to calculate the DrawWidth property in inches, for a form and the printer:

```
'*** In the global module: ***

' ScaleMode (form, picture box, Printer)
Global Const TWIPS = 1
Global Const POINTS = 2      ' 20 twips
Global Const PIXELS = 3
Global Const CHARACTERS = 4  ' x: 120 twips, y: 240 twips
Global Const INCHES = 5      ' 1440 twips
Global Const MILLIMETERS = 6 ' 5669 twips
Global Const CENTIMETERS = 7 ' 566.9 twips

' *** In the form: ***

Sub Form_Click ()
   Dim ptr_inch  As Integer ' printer width in inches
   Dim ptr_pixel As Long    ' printer width in pixels
   Dim ptr_dpi   As Single  ' printer dots (pixels) per inch
   Dim scn_inch  As Integer ' screen width in inches
   Dim scn_pixel As Long    ' screen width in pixels
   Dim scn_dpi   As Single  ' screen dots (pixels) per inch

   ' Determine printer pixels-per-inch ratio
   save% = Printer.ScaleMode
   Printer.ScaleMode = INCHES: ptr_inch = Printer.ScaleWidth
   Printer.ScaleMode = PIXELS: ptr_pixel= Printer.ScaleWidth
   Printer.ScaleMode = save%
   ptr_dpi = ptr_pixel / ptr_inch

   ' Determine form (screen) pixels-per-inch ratio
   save% = Form1.ScaleMode
   Form1.ScaleMode = INCHES: scn_inch = Form1.ScaleWidth
   Form1.ScaleMode = PIXELS: scn_pixel = Form1.ScaleWidth
   Form1.ScaleMode = save%
   scn_dpi = scn_pixel / scn_inch

   ' Set printer and form DrawWidth to 0.25 inches
   ' and draw a 0.25 inch thick line
   Printer.DrawWidth = .25 * ptr_dpi
   Form1.DrawWidth = .25 * scn_dpi
```

```
    Printer.Line (0, 0)-(Form1.ScaleWidth, Form1.ScaleHeight)
    Form1.Line (0, 0)-(Form1.ScaleWidth, Form1.ScaleHeight)

    ' Set printer.DrawWidth to match screen pixel size
    ' and draw a 5 screen-pixel thick line
    Form1.DrawWidth = 5
    Printer.DrawWidth = Form1.DrawWidth * ptr_dpi / scn_dpi
    Form1.Line (0, Form1.ScaleHeight)-(Form1.ScaleWidth, 0)
    Printer.Line (0, Form1.ScaleHeight)-(Form1.ScaleWidth, 0)

    Printer.EndDoc

End Sub
```

When run, the above sample program will cause two lines in the form of an X to be printed to the form and printer simultaneously. The width of the thicker diagonal line should be 0.25 inches wide on the printed page. The other diagonal line represents a line five pixels wide.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Move Controls Between Forms in VB for Windows

Article ID: Q79884

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Microsoft Visual Basic for Windows does not support the actual movement of controls between forms. Attempting to change the parent/child relationship of a control from one form to another can result in unpredictable behavior.

However, by creating a control array of the same control type on each form, and by creating a subroutine or function in a Visual Basic for Windows module, you can simulate the movement of a control from one form to another. An example of how to do this is listed below.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

This example uses the Windows API functions GetFocus and GetParent to determine the origin of the control dropped onto a form. For more information on GetFocus and GetParent, query separately on the following words in the Microsoft Knowledge Base:

```
GetFocus
GetParent
```

The following steps demonstrate how to simulate the movement of controls between two forms. Note that you can improve this example by Loading and Unloading the controls as they are needed.

1. Start Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 will be created by default.

2. From the File menu, choose New Form (press ALT, F, F). Form2 will be created.

3. From the File menu, choose New Module (press ALT, F, M). Module1 will be created.

4. Create the following controls for both Form1 and Form2:

Control	Name	Property Setting

```
    Command button   Command1()   Index = 0
    Command button   Command2     Caption = "Enable Drag"
```

(In Visual Basic version 1.0 for Windows, set the CtlName Property for the above objects instead of the Name property.)

5. Add the following code to the Module1 (or GLOBAL.BAS in Visual Basic version 1.0 for Windows):

```
' Windows API function declarations.
Declare Function GetFocus Lib "USER" () As Integer
Declare Function GetParent Lib "USER" (ByVal hWnd As Integer) As Integer
```

6. Add the following code to the General Declarations section of Form1:

```
    Dim EnableDrag As Integer
```

7. Add the following code to the Form_Load event procedure of Form1:

```
Sub Form_Load ()

   ' Move the form to the left half of the screen.
   Move 0, Top, Screen.Width \ 2
   Form2.Show
   EnableDrag = 0
   Command1(0).Top = 0
   Command1(0).Left = 100

   For i% = 1 To 4                              ' Load Control Array.
      Load Command1(i%)
      Command1(i%).Left = Command1(i% - 1).Left
      Command1(i%).Top = Command1(i% - 1).Top + Command1(i% - 1).Height
   Next i%

   For i% = 0 To 4                          ' Define Control Properties.
      Command1(i%).Caption = "Button" + Str$(i%)
      Command1(i%).Visible = -1
   Next i%
End Sub
```

8. Add the following code to the Command1_Click event procedure of Form1:

```
Sub Command1_Click (Index As Integer)
   Button_Clicked Command1(Index)        ' Call Routine in MODULE1.BAS.
End Sub
```

9. Add the following code to the Command2_Click event procedure of Form1:

```
Sub Command2_Click ()
   If EnableDrag = 0 Then                       ' Toggle DragMode.
      EnableDrag = 1
      Command2.Caption = "Disable Drag"
   Else
      EnableDrag = 0
```

```
      Command2.Caption = "Enable Drag"
   End If

   For i% = 0 To 4                          ' Set DragMode for Controls.
      Command1(i%).DragMode = EnableDrag
   Next i%
End Sub
```

10. Add the following code to the Form_DragDrop event procedure of Form1:

```
Sub Form_DragDrop (Source As Control, X As Single, Y As Single)
   Source.SetFocus                       ' Get Parent of Source Control.
   CtrlHnd% = GetFocus()
   Parent% = GetParent(CtrlHnd%)

   If Parent% <> Form1.hWnd Then         ' If Parent is other Form.
      Index% = Source.Index
      Command1(Index%).Caption = Source.Caption
      Command1(Index%).Left = Source.Left
      Command1(Index%).Top = Source.Top
      Command1(Index%).Width = Source.Width
      Command1(Index%).Height = Source.Height
      Command1(Index%).Visible = -1
      Source.Visible = 0
   End If
End Sub
```

11. Add the following code to the General Declarations section of Form2:

```
Dim EnableDrag As Integer
```

12. Add the following code to the Form_Load event procedure of Form2:

```
Sub Form_Load ()
   ' Move the form to the right half of the screen.
   Move Screen.Width \ 2, Top, Screen.Width \ 2

   EnableDrag = 0
   Command1(0).Visible = 0
   For i% = 1 To 4                          ' Load Control Array.
      Load Command1(i%)
      Command1(i%).Top = Command1(i% - 1).Top + Command1(i% - 1).Height
      Command1(i%).Visible = 0
   Next i%
End Sub
```

13. Add the following code to the Command1_Click event procedure of Form2:

```
Sub Command1_Click (Index As Integer)
   Button_Clicked Command1(Index)
End Sub
```

14. Add the following code to the Command2_Click event procedure of Form2:

```
Sub Command2_Click ()
   If EnableDrag = 0 Then
      EnableDrag = 1
      Command2.Caption = "Disable Drag"
   Else
      EnableDrag = 0
      Command2.Caption = "Enable Drag"
   End If

   For i% = 0 To 4
      Command1(i%).DragMode = EnableDrag
   Next i%
End Sub
```

15. Add the following code to the Form_DragDrop event procedure of Form2:

```
Sub Form_DragDrop (Source As Control, X As Single, Y As Single)
   Source.SetFocus           ' Determine Parent of Source.
   CtrlHnd% = GetFocus()
   Parent% = GetParent(CtrlHnd%)
   If Parent% <> Form2.hWnd Then
      Index% = Source.Index
      Command1(Index%).Caption = Source.Caption
      Command1(Index%).Left = Source.Left
      Command1(Index%).Top = Source.Top
      Command1(Index%).Width = Source.Width
      Command1(Index%).Height = Source.Height
      Command1(Index%).Visible = -1
      Source.Visible = 0
   End If
End Sub
```

16. Add the following code to Module1:

```
Sub Button_Clicked (Source As Control) ' Generic Click routine.
    MsgBox "Button" + Str$(Source.Index) + " Clicked!!!"
End Sub
```

17. From the Run menu, choose Start (press ALT, R, S) to run the program.

To drag controls from one form to the other, choose the Enable Drag button. Once this button has been activated on a form, you can drag any of the command buttons from one form to the other. The drag mode can be disabled by choosing the Disable Drag button. When drag mode has been disabled, clicking any of the command buttons on the form will cause a message box to be displayed.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Drop Item into Specified Location in VB List Box

Article ID: Q80187

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can drag an item and drop it into a list box by using the Visual Basic TextHeight method and the Windows API SendMessage() function to calculate where to drop the item.

MORE INFORMATION

There is no standard way to determine the exact position where you are dropping an item into a Visual Basic list box when you perform a drag and drop operation. You must calculate the position using the TextHeight method and the Windows API SendMessage() function with the constant LB_GETTOPINDEX.

Using TextHeight, determine the height of each row of a list box. Divide this by the Y value that is passed as an argument in the List_DragDrop event procedure to determine how many lines from the top of the list box that the Drag.Icon is located over. The SendMessage constant LB_GETTOPINDEX gives you the index of the first visible item in the list box. Adding these two numbers shows you the index location for the insertion point -- the spot where you want to insert the item in the list box.

Step-by-Step Example to Demonstrate Dropping Items into List box

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a Picture control (Picture1) to Form1 and set its DragMode property to Automatic. Then add a List box (List1) to Form1 and set its DragMode property to Manual.

3. Add the following code to the global module:

```
'============= Global.Bas ===============
'NOTE: Enter the following Declare statement as one, single line:
Declare Function SendMessage& Lib "User" (ByVal hWnd%, ByVal wMsg%,
   ByVal wParam%, lParam As Any)
Declare Function GetFocus Lib "User" () As Integer
Global Const LB_GETTOPINDEX = &H400 + 15
```

3. Add the following code to the DragDrop event procedure of List1:

```
'============== Form1.frm ==================
```

```
Sub List1_DragDrop (Source As Control, X As Single, Y As Single)

   'get the first visible index in the list box
   List1.SetFocus
   ListHwnd = GetFocus()
   TopI& = SendMessage(ListHwnd, LB_GETTOPINDEX, 0&, 0&)
   ColumnHeight = TextHeight("A ")
   InsertI& = Y \ ColumnHeight
   If InsertI& <= List1.ListCount Then
      ' Enter the following two lines on one, single line:
      List1.AddItem "This is inserted @" + Format$(InsertI&
         + TopI&, "0"), InsertI& + TopI&
   Else
      List1.AddItem "This is inserted"
   End If

End Sub
```

4. From the Run menu, choose Start (ALT, R, S) to run the program.

Drag and drop the picture box over the list box and an item should be added to the list box. An item will be added to the list box each time you drag and drop the picture box over the list box.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Draw a Line or Box on a Form Using a Label in Ver 1.0

Article ID: Q80285

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Visual Basic version 1.0 Help topic "Drawing a Line or Box on a Form" contains incorrect information regarding how to use a label control to draw a line or box on a form. This article corrects and supplements that information.

To draw a line or box on a form in Visual Basic version 2.0, use the Shape control instead of a Label control.

MORE INFORMATION

You can use the label tool in the Toolbox in VB.EXE to draw simple lines or solid (filled-in) boxes on forms. By using a label instead of the Line method, you can see the line or box in design mode and you can easily animate the line or box with the Move method.

To draw a line or box with a label control, do the following:

1. Place a label on a form.

2. Set the Caption property to null.

3. Set the BackColor property to black, or some other color.

4. Size the label. To make a line, set either the Height or Width property to the minimum value (1 pixel).

To find the incorrect Help topic, search Visual Basic version 1.0 Help for "line." The topic states the following:

> To add color, set the BackColor property to the color you want.

Do not set the BorderStyle property to -1 (True) as it states in Help.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Form Global (Static) Data Is Preserved After Form Unload

Article ID: Q80287

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article lists documentation errors for Visual Basic version 1.0 manuals and provides additional information about static data, arrays and variables.

If you have version 2.0, see page 394 of the Language Reference for more information on the Static statement.

MORE INFORMATION

Visual Basic version 1.0 Documentation Errors

The information on page 226 of the "Microsoft Visual Basic: Programmer's Guide" version 1.0 manual in the section titled "Unloading Forms" implies that all data in a form is lost after the form is unloaded using the Unload statement. However, this does not apply to these types of data:

- Any variable or array that is dimensioned in the general Declarations section of the form.
- Any static variable or array that has been declared within a Sub or Function procedure.
- Any local variable or array that has been allocated in a static Sub or Function procedure.

The following statement on page 226 of the "Visual Basic: Programmer's Guide" is incorrect:

> Any data stored in the form is lost unless you have saved it to a file.

This statement should be changed to read as follows:

> Any data stored in the form, with the exception of static variables and arrays, is lost unless you have saved it to a file. The values of static arrays and variables are preserved after the form is unloaded.

More Information on Static Data

Static data stored in a form consists of the following:

- Arrays or variables dimensioned in the general Declarations section of a form.
- Variables or arrays declared in a Sub or Function procedure using

the Static keyword.
- All local variables and arrays allocated in a Sub or Function procedure where the procedure name is preceded by the keyword Static.

All static data is allocated in a global area of memory managed by Visual Basic. Unloading the form does not cause this memory to be deallocated; rather, the data is preserved by Visual Basic until the program terminates. Although the data is maintained after the form is unloaded, you cannot access this data from any other form or module. You must reload the form to access the data.

Static Variables and Arrays Are Not Deallocated

To demonstrate how static variables and arrays allocated from a form do not get deallocated, do the following:

1. Start Visual Basic or if Visual Basic is already running, choose New Project from the File menu (ALT, F, N). Form1 is created by default.

2. Add a command button (Command1) to Form1.

3. Change the caption of Command1 to "Show Form Global Vars" (without the quotation marks).

4. Add the following statements to the general Declarations section of Form1:

```
Dim varX As Integer
Dim arrayX(10) As String
```

5. Add the following code to the Command1_Click event procedure of Command1:

```
Sub Command1_Click ()

   Static StaticX

   StaticX = 1           'Initialize the form global variables.
   varX = 10
   For i = 0 To 10
      arrayX(i) = Format$(i, "#0")
   Next i

   Unload Form1
   Form1.Show            'Reload and show form.
                         'Values of varX and arrayX will still be
                         'preserved.
   Print StaticX         'Print the values to the form.
   Print varX
   For i = 0 To 10
      Print arrayX(i)
   Next i

End Sub
```

6. Run the program (F5) and choose the Commmand1 "Show Form Global Vars" button.

7. The values of StaticX, varX, and arrayX will print, even though the form has been unloaded.

There is no way to cause static data in the general Declarations section to be deallocated when the form is unloaded. For example, the Erase statement will not cause memory to be deallocated for arrays dimensioned in the general Declarations section.

To deallocate arrays, you must use the ReDim statement to dynamically allocate the array when needed. To unload variables, use local variables instead of static variables. If you use local variables instead of static variables, the local variables are deallocated upon exit from the procedure in which they were allocated.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: PrgCtrlsStd

PRB: End Task from Windows Task List Doesn't Invoke VB Unload

Article ID: Q80292

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

Terminating a Visual Basic program from the Windows Task List (by choosing the End Task button) does not generate a Visual Basic Unload event in version 1.0 but does generate a Form_Unload in versions 2.0 and 3.0. Therefore, any code attached to the Form_Unload event procedure executes in versions 2.0 and 3.0 but not in version 1.0 when the program terminates.

RESOLUTION

This behavior is by design. The version 1.0 design is different from the versions 2.0 and 3.0 designs. In versions 2.0 and 3.0, the form_unload event executes when the program is closed from the task list. In version 1.0, Form_Unload does not execute when the program is closed from the task list.

MORE INFORMATION

If a Visual Basic program has code in an Form_Unload event procedure, and you exit the program by choosing Close from the system menu, the Unload event occurs and the code in the Form_Unload event procedure executes.

If instead, you exit the program from the Windows Task List, the Unload event occurs only in Visual Basic version 2.0, not in 1.0. The code in the Form_Unload event procedure executes only when the Unload event occurs.

Steps to Reproduce Behavior

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Place a command button (Command1) on Form1.

3. Add the following code to the Command1_Click event procedure for Command1:

```
Sub Command1_Click()
   Unload Form1
End Sub
```

4. Add the following code to the Form1_Unload event procedure:

```
Sub Form1_Unload(Cancel as integer)
   If MsgBox("Continue to unload the form?",1) = 2 Then
      cancel = -1
   Else
      cancel = 0
   End If
End Sub
```

5. From the File menu, choose Make EXE File (ALT, F, K). Enter a filename for the .EXE program and choose the OK button.

6. From the Windows Program Manager File menu, choose Run (ALT, F, R), and enter the name of the .EXE file created in step 5.

While the program is running, exit by double-clicking the system menu or by pressing the command button. You will see the message box appear and you will be able to abort unloading the form if you choose the Cancel button.

Now try running the program (step 6 above) and bring up the Windows Task List by pressing CTRL+ESC. If you select the name of the program from the Task List and then choose the End Task button, your program will terminate without the message box ever being displayed in Visual Basic version 1.0.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Rotate a Bitmap in VB for Windows

Article ID: Q80406

The information in this article applies to:

- Microsoft Visual Basic Programming System for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

This article contains a program example that uses Visual Basic for Windows statements and functions to rotate a bitmap.

MORE INFORMATION

Steps to Create Example Program

1. Run Visual Basic for Windows, or from the File menu, choose New Project (ALT, F, N) if Visual Basic for Windows is already running. Form1 will be created by default.

2. Place two picture boxes named Picture1 and Picture2 on Form1. Assign a bitmap to the Picture property of Picture1.

3. Set the ScaleMode property of both picture boxes to 3 - Pixel.

4. Set the AutoSize property of Picture1 to True (-1).

5. Set the AutoRedraw property of Picture1 and Picture2 to True (-1).

6. Place a command button named Command1 on Form1.

7. Enter the following code in the Command1_Click event procedure:

```
' Example of how to call bmp_rotate.
Sub Command1_Click ()
    Const Pi = 3.14159265359

    For angle = Pi / 6 To 2 * Pi Step Pi / 6
       picture2.Cls
       Call bmp_rotate(picture1, picture2, angle)
    Next
End Sub
```

8. Enter the following code in the general Declarations section:

```
' bmp_rotate(pic1, pic2, theta)
' Rotate the image in a picture box.
'   pic1 is the picture box with the bitmap to rotate
'   pic2 is the picture box to receive the rotated bitmap
'   theta is the angle of rotation
```

```
'
Sub bmp_rotate (pic1 As Control, pic2 As Control, ByVal theta!)
  Const Pi = 3.14159265359
  Dim c1x As Integer  ' Center of pic1.
  Dim c1y As Integer  '   "
  Dim c2x As Integer  ' Center of pic2.
  Dim c2y As Integer  '   "
  Dim a As Single     ' Angle of c2 to p2.
  Dim r As Integer    ' Radius from c2 to p2.
  Dim p1x As Integer  ' Position on pic1.
  Dim p1y As Integer  '   "
  Dim p2x As Integer  ' Position on pic2.
  Dim p2y As Integer  '   "
  Dim n As Integer    ' Max width or height of pic2.

  ' Compute the centers.
  c1x = pic1.scalewidth / 2
  c1y = pic1.scaleheight / 2
  c2x = pic2.scalewidth / 2
  c2y = pic2.scaleheight / 2

  ' Compute the image size.
  n = pic2.scalewidth
  If n < pic2.scaleheight Then n = pic2.scaleheight
  n = n / 2 - 1
  ' For each pixel position on pic2.
  For p2x = 0 To n
     For p2y = 0 To n
        ' Compute polar coordinate of p2.
        If p2x = 0 Then
          a = Pi / 2
        Else
          a = Atn(p2y / p2x)
        End If
        r = Sqr(1& * p2x * p2x + 1& * p2y * p2y)

        ' Compute rotated position of p1.
        p1x = r * Cos(a + theta)
        p1y = r * Sin(a + theta)

        ' Copy pixels, 4 quadrants at once.
        c0& = pic1.Point(c1x + p1x, c1y + p1y)
        c1& = pic1.Point(c1x - p1x, c1y - p1y)
        c2& = pic1.Point(c1x + p1y, c1y - p1x)
        c3& = pic1.Point(c1x - p1y, c1y + p1x)
        If c0& <> -1 Then pic2.PSet (c2x + p2x, c2y + p2y),c0&
        If c1& <> -1 Then pic2.PSet (c2x - p2x, c2y - p2y),c1&
        If c2& <> -1 Then pic2.PSet (c2x + p2y, c2y - p2x),c2&
        If c3& <> -1 Then pic2.PSet (c2x - p2y, c2y + p2x),c3&
     Next
     ' Allow pending Windows messages to be processed.
     t% = DoEvents()
  Next
End Sub
```

9. Assign a bitmap image to the Picture1 Picture property.

10. To start the program, press F5, then click the Command1 button. The program rotates the image of Picture1 by 30 degrees and places the rotated image in Picture2. It continues to draw the image rotated at successive multiples of 30 degrees until it has rotated the picture by 360 degrees.

To save the new bitmap created in Picture2, you can use the following statement:

```
SavePicture Picture2.Image, "filename.bmp"
```

Additional reference words: 1.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Clear VB Picture Property at Run Time Using LoadPicture

Article ID: Q80488

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

During execution of a Visual Basic program, you can clear the Picture property of a form or picture control by using the LoadPicture function. Calling LoadPicture with no parameters and assigning the result to the Picture property of a form or control will clear the Picture property.

MORE INFORMATION

This information is documented in the Visual Basic Help menu under the LoadPicture function.

Code Example

To clear the picture property at run time, do the following:

1. Start Visual Basic.

2. Make a picture box called Picture1.

3. Assign a bitmap or icon the picture1.picture property.

4. Add the following code to the form1.click event by double-clicking the form:

```
Sub Form_Click ()
   picture1.picture = LoadPicture()
End Sub
```

5. Run the program.

6. Click the form.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd APrgGrap

How to Print Multiline Text Box Using Windows API Functions

Article ID: Q80867

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Printing the Text property of a multiline text box while maintaining the line structure requires attention to word wrapping and carriage return/line feeds. The programmer can either track the number of characters and lines in code or use Windows API functions to manipulate the Text property. This article demonstrates these techniques in a Visual Basic example.

MORE INFORMATION

The example below demonstrates how to use the API function SendMessage() to track the number of lines in a multiline text box and to select and print the lines the way they appear -- with line breaks or word wrapping intact. This code will work without modification even if the form and controls are resized at run time. The actual position of word wrapping will change.

For more information about API functions relating to text boxes, query on the following words in the Microsoft Knowledge Base:

```
API and text and box and manipulate
```

Step-by-Step Example

1. Create a form and place a label, text box, and command button on it.

2. Set the following properties at design time:

Control	Property	Setting
Text box	TabIndex	0 (zero, or first in tab order)
Text box	MultiLine	True
Label	AutoSize	True
Label	Name	aGetLineCount

3. Add the following code to the Global module:

```
Declare Function GetFocus% Lib "user" ()
' Enter the following Declare statement on one, single line:
Declare Function SendMessage% Lib "user" (ByVal hWnd%, ByVal wMsg%,
   ByVal wParam%, ByVal lParam As Any)
Global Buffer As String
```

```
Global resizing As Integer
Global Const EM_GETLINE = &H400 + 20
Global Const EM_GETLINECOUNT = &H400 + 10
Global Const MAX_CHAR_PER_LINE = 80  ' Scale this to size of text box
```

4. Add the following code to the Form_Load procedure:

```
Sub Form_Load ()
   ' Size form relative to screen dimensions.
   ' Could define all in move command but recursive definition causes
   ' extra paints.
   form1.width = screen.width * .8
   form1.height = screen.height * .6
   ' Enter the following form1.Move method on one, single line:
   form1.Move screen.width\2-form1.width\2,
      screen.height\2-form1.height\2
End Sub
```

5. Add the following code to the Form_Resize procedure:

```
Sub Form_Resize ()
   resizing = -1      ' Global  flag for fGetLineCount function call
   ' Dynamically scale and position the controls in the form.
   ' This code also is executed on first show of form.
   Text1.Move 0, 0, form1.width, form1.height \ 2
   Text1.SelStart = Text1.SelStart    ' To avoid UAE -see Q80669
   ' Enter the following two lines as one, single line:
   command1.Move form1.width\2-command1.width\2,
      form1.height-form1.height\4
   ' Enter the following two lines as one, single line:
   aGetLineCount.Move form1.width \ 2 - command1.width \ 2,
      Text1.height
   X% = fGetLineCount() ' Update to reflect change in text box size
   resizing = 0
End Sub
```

5. Add the following code to the Command1_Click event:

```
Sub Command1_Click ()
   '* Pop up an inputbox$ to allow user to specify which line
   '* in the text box to print or print all lines.
   '* Also check bounds so that a valid line number is printed
   OK = 0                     ' Zero the Do Loop flag
   NL$ = Chr$(13) + Chr$(10)
   prompt$ = "Which line would you like to print?"
   prompt1$ = prompt$ + NL$ + "Enter -1 for all"
   prompt2$ = "Too many lines" + NL$ + "Try again!" + NL$ + prompt1$
   prompt$ = prompt1$
   Do
      response$ = InputBox$(prompt$, "Printing", "-1")
      If response$ = "" Then Exit Sub  ' if user hits cancel then exit
      If Val(response$) > fGetLineCount&() Then
         prompt$ = prompt2$
      Else
         OK = -1        ' Line chosen is in valid range so exit DO
      End If
   Loop Until OK
```

```
   If Val(response$) = -1 Then  ' Print all lines
      ndx& = fGetLineCount&()
      For N& = 1 To ndx&
         Buffer = fGetLine(N& - 1)
         printer.Print Buffer       ' or print to the screen
      Next N&
   Else                          ' Print a line
      Buffer = fGetLine(Val(response$) - 1)
      printer.Print Buffer       ' or print to the screen
   End If
End Sub
```

6. Add the following code to the general Declarations section of the form's code:

```
Function fGetLine$ (LineNumber As Long)
  ' This function fills the buffer with a line of text
  ' specified by LineNumber from the text box control.
  ' The first line starts at zero.
   byteLo% = MAX_CHAR_PER_LINE And (255)  '[changed 5/15/92]
   byteHi% = Int(MAX_CHAR_PER_LINE / 256) '[changed 5/15/92]
   Buffer$ = chr$(byteLo%) + chr$(byteHi%)+Space$(MAX_CHAR_PER_LINE-2)
   ' [Above line changed 5/15/92 to correct problem.]
   text1.SetFocus 'Set focus for API function GetFocus to return handle
   x% = SendMessage(GetFocus(), EM_GETLINE, LineNumber, Buffer$)
   fGetLine$ = Left$(Buffer$,X%)
End Function

Function fGetLineCount& ()
   ' This function will return the number of lines
   ' currently in the text box control.
   ' Setfocus method illegal while in resize event
   ' so use global flag to see if called from there
   ' (or use setfocus prior to this function call in general case).
   If Not resizing Then
      Text1.SetFocus   ' Set focus for following function GetFocus
      resizing = 0
   End If
   lcount% = SendMessage(GetFocus(), EM_GETLINECOUNT, 0&, 0&)
   aGetLineCount.caption = "GetLineCount = " + Str$(lcount%)
   fGetLineCount& = lcount%
End Function
```

7. Add the following code to the Text1_Change event:

```
Sub Text1_Change ()
   X% = fGetLineCount() '* Update label to reflect current line
End Sub
```

8. Save the project. Then run the application.

9. Enter text into the text box and either let it wrap or use the ENTER key to arrange lines.

10. Choose the button or TAB and press ENTER.

11. Choose the default (which prints all lines) or enter the line

desired. If you choose Cancel, nothing will print.

12. Resize the form and repeat steps 9 to 11 above. The text will appear on the printed page as you saw it in the text box. Modify the example to print to the screen, write to a file, and so forth.

Reference(s):

"Microsoft Windows Programmer's Reference Book and Online Resource"
(Visual Basic Add-on kit number 1-55615-413-5)

Additional reference words: 1.00 2.00 3.00 textbox
KBCategory:
KBSubcategory: PrgCtrlsStd APrgWindow

Common Dialog Custom Control: FilterIndex Can Be Negative

Article ID: Q80934

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The FilterIndex property of the Common Dialog custom control (COMMDLG.DLL) can be any long integer value, including negative numbers and 0 (zero).

MORE INFORMATION

Normally, the smallest value to which you would set the FilterIndex property is 1, because the first filter is defined as 1. If you use a number less than 1, such as 0 or any negative number (within the range of a long integer), you will get the same result as if you set it equal to 1.

Likewise, if you use a number greater that the total number of filters, the FilterIndex property will function as if you set it to the last of the filters.

For example, if you have three filters and you set FilterIndex to -2, it will function as if you set FilterIndex to 1. If you set FilterIndex to 23, it will function as if you set FilterIndex to 3.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Common Dialog Control: Pipe (|) Optional in Filter Property

Article ID: Q80935

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The pipe character (|), which is placed at the end of each selection in the Filter property of the Common Dialog custom control, is optional on the last item in the string.

This information applies to the Common Dialog custom control supplied with Microsoft Professional Toolkit for Microsoft Visual Basic programming system version 1.0 for Windows and with the Professional Edition of Visual Basic version 2.0 for Windows.

MORE INFORMATION

Filter is a property of the Common Dialog custom control (COMMDLG.DLL). The Filter property is assigned a string that contains sets of "description" and "filter". Each set represents one entry in the List Files of Type list box.

Each of these items in the string are followed by a pipe character (|). The last item in the list need not be followed by a pipe, although it is allowed.

The syntax for using the Filter property is as follows:

```
CommonDialog.Filter[= desc1$|filter1$|desc2$|filter2$]
```

Either of the following code examples will work:

```
CMDialog1.Filter = "Text Files (*.txt)|*.txt"
```

-or-

```
CMDialog1.Filter = "Text Files (*.txt)|*.txt|"
```

The Microsoft Visual Basic Professional Toolkit documentation uses both of these methods in its code examples.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Use More than One Type of Font in Picture Box

Article ID: Q81220

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The text box control in Visual Basic for Windows displays the entire text box with either the FontUnderline, FontBold, FontItalic, or FontStrikethru fonts, but with only one font at a time. This behavior is by design.

However, you may want to display a box with all four fonts at the same time with separate words displayed in different fonts. Below is an example of displaying the fonts FontBold, FontItalic, FontStrikethru, and FontUnderline fonts in a picture box control in Visual Basic for Windows to work around the limitation in text boxes.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The example below is one way of simulating a text box's contents in a variety of fonts.

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Place a picture box on Form1, and double-click the picture box to open the Code window. Add the following code to the Click event. Notice that the font properties are a Boolean type (that is, -1 = True and 0 = False).

```
Sub Picture1_Click ( )

'** The word "Hello, " will be in FontBold.
        temp$ = "Hello, "
        Picture1.FontBold = -1
        Picture1.FontItalic = 0
        Picture1.FontStrikethru = 0
        Picture1.FontUnderline = 0
        Picture1.Print temp$

'** Need to program the next location to print in FontItalic.
        Picture1.Currentx = 500
        Picture1.Currenty = 0
```

```
        Picture1.FontBold = 0
        Picture1.FontItalic = -1
        Picture1.FontStrikethru = 0
        Picture1.FontUnderline = 0
        temp$ = " there!"
        Picture1.Print temp$

'** Need to program location to print in FontStrikethru.
        Picture1.Currentx = 1100
        Picture1.Currenty = 0
        Picture1.FontBold = 0
        Picture1.FontItalic = 0
        Picture1.FontUnderline = 0
        Picture1.FontStrikethru = -1
        temp$ = "This"
        Picture1.Print temp$

'** Need to program location to print in FontUnderline.
        Picture1.Currentx = 0
        Picture1.Currenty = 200
        Picture1.FontBold = 0
        Picture1.FontItalic = 0
        Picture1.FontStrikethru = 0
        Picture1.FontUnderline = -1
        temp$ = "is a test."
        Picture1.Print temp$

End Sub
```

Notice that the CurrentX and CurrentY properties are used to place the text at a certain location in the picture box. This example is rather simple, but its purpose is to give you an idea on how to simulate a text box in Visual Basic for Windows to be more flexible with a mix of the different types of fonts available.

The same method can be used to print more than one type of font to a printer. To do this, modify the Picture1_Click Sub procedure by changing Picture1. to Printer.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Visual Basic SendKeys Statement Is Case Sensitive

Article ID: Q81466

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The SendKeys statement in Microsoft Visual Basic for Windows is case sensitive with regards to the keystrokes sent. Sending an uppercase letter may be interpreted by the receiving application differently than the lowercase version of a letter.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The following line of code sends an ALT+F key combination to the application that currently has the focus:

```
SendKeys "%(F)"
```

Note that this is different than ALT+f:

```
SendKeys "%(f)"
```

This can be a problem because some applications distinguish between an uppercase F and lower case f when sent by the SendKeys statement.

For example, Microsoft Word versions 1.0b and earlier for Windows (WINWORD.EXE) do not distinguish the difference. However, Microsoft Word version 2.0 for Windows does distinguish the lowercase f sent by SendKeys.

When SendKeys (from Visual Basic for Windows) sends the ALT+F key combination, WINWORD.EXE version 2.0 interprets the keystroke as ALT+Shift+f, at which Word for Windows will simply beep. However, SendKeys using ALT+f will correctly activate the File menu.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Task List Switch to VB Application Fails After ALT+F4 Close

Article ID: Q81469

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Selecting the Close command from the Control menu (ALT+F4) to quit a Visual Basic for Windows application will not necessarily unload any other forms that have been loaded. If other forms have been loaded but are not visible, the application may still be running under Windows. If this is the case, the Windows Task List will still contain the name of the application. Attempting to switch to the application from the Windows Task List will be unsuccessful.

If you want the application to terminate as a result of unloading a particular form, place an End statement in the Form_Unload event procedure for the form, or use the Unload statement to unload all forms that are loaded. This will cause all forms (visible and invisible) to be unloaded, and the application to terminate.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Even if the form that is closed is the designated startup form in your application, it will not automatically unload previously loaded forms. Therefore, the application can in fact still be running and appear in the Windows Task List. You can terminate the application by selecting the End Task button in the Windows Task List, but you will not be able to switch to the task.

Below are the steps necessary to cause an application to terminate when a particular form is closed from the Control menu (ALT+F4).

With the application loaded in VB.EXE (the Visual Basic for Windows development environment), do the following:

1. Double-click the form to open the Code window.

2. Add an End statement to the Form_Unload event procedure for the form. For example:

   ```
   Sub Form_Unload (Cancel As Integer)

       ' Your code goes here.
   ```

```
      End  ' This unloads all the forms and terminates the application.

   End Sub
```

Adding an End statement to the Unload event procedure of a form will not cause the Unload event procedures for the other forms to be called. To cause the Unload event procedures for the other forms to be called, use the Unload statement to explicitly unload each form.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Overflow Error Plotting Points Far Outside Bounds of Control

Article ID: Q81953

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Visual Basic for Windows may give an Overflow error when you plot points on a form or picture box if a point's coordinates far exceed the borders and scale of the form or control. The point at which overflow occurs depends on the ScaleMode property value and the points plotted. In the case of ScaleMode = 0 (User Defined Scale), the size of the form or picture box and the scale chosen are also determinants.

A workaround is to trap the error and use a RESUME NEXT statement to exit the error handler. The example below contains the necessary code to trap the Overflow error.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Before Visual Basic for Windows can plot a point, it must first convert the coordinates into their absolute location in twips. If, after the conversion, one or both coordinates are greater than 32,767 or less than -32,768, an Overflow error is generated. The following chart lists the ScaleModes, their equivalence in twips, and the values that will cause a coordinate (z) to overflow:

ScaleMode	Equivalents in Twips (Tp)	Overflow Point (z)
0 (User defined)	User defined	User defined (see example)
1 (Twips)	1 twip = 1 twip	(z < -32768) or (z > 32767)
2 (Point)	1 point = 20 twips	(z < -1638) or (z > 1638)
3 (Pixel)	System dependent	System dependent
4 (Character)	x-axis=120 twips/char y-axis=240 twips/char	(x < -273) or (x > 273) (y < -136) or (y > 136)
5 (Inch)	1 Inch = 1440 twips	(z < -22) or (z > 22)
6 (Millimeter)	1 mm = 56.7 twips	(z < -577) or (z > 577)
7 (Centimeter)	1 cm = 567 twips	(z < -57) or (z > 57)

The example below can be used to determine the value that generates the Overflow error for ScaleMode 0 or 3.

Example

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Add the following controls to Form1:

Control	Name (use CtlName in Visual Basic 1.0 for Windows)
Text box	Text1
Command button	Command1

3. Set the MultiLine property for Text1 to True. With ScaleMode = 0 only, the overflow value is dependent upon the size of the picture box or form. If you are testing the overflow value with ScaleMode = 0, you must size the form appropriately.

4. Add the following code to the Form1 Form_Load event procedure:

```
Sub Form_Load ()
   Command1.Caption = "Find Ranges"

   '* Change ScaleMode to see different results.
   Form1.ScaleMode =  3   ' PIXEL.
End Sub
```

5. Add the following code to the Command1_Click event procedure:

```
Sub Command1_Click ()
   CR$ = Chr$(13) + Chr$(10)  ' Carriage return.

   X = FindValue("X")
   Y = FindValue("Y")

   Text1.Text = "Valid value when..."
   Text1.Text = Text1.Text + CR$ + "-" + Str$(X) + " < X < " + Str$(X)
   Text1.Text = Text1.Text + CR$ + "-" + Str$(Y) + " < Y < " + Str$(Y)
End Sub
```

6. Add the following general purpose function to the general Declarations section:

```
Function FindValue (Which$)
   On Error GoTo rlhandler

   HiValue = 100000
   LoValue = 0
   Errored = FALSE
   ' Do binary select.
   Do
      NewCheck = Value
      If Errored Then
         Value = HiValue - (HiValue - LoValue) \ 2
      Else
         Value = LoValue + (HiValue - LoValue) \ 2
      End If

      If Which$ = "X" Then
```

```
            Form1.PSet (Value, 0)
         Else
            Form1.PSet (0, Value)
         End If

         If ErrorNum = 6 Then
             HiValue = Value
            ErrorNum = 0
         Else
            LoValue = Value
         End If
      Loop Until NewCheck = Value
      FindValue = Value

      Exit Function

   rlhandler:
      ' Err = 6 is OverFlow error.
      If Err = 6 Then
         ErrorNum = Err
      Else
         Form1.Print Err
      End If
   Resume Next

End Function
```

7. In Visual Basic version 1.0 for Windows, add the following to the general declarations section of Form1:

```
    Const FALSE = 0
    Const TRUE = -1
```

8. From the Run menu, choose Start (or press the F5 key), and click the Command1 button to calculate the point at which the X and Y coordinates generate an Overflow error.

When the above Click event is triggered, Visual Basic for Windows will try to set a point on the form. Past the border, Visual Basic for Windows is plotting points that exceed the visual scope of the control. Once the program traps the Overflow error, the text box will display the valid range of coordinates you can use that will not generate the Overflow error.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

PRB: MDI Child: Child Window May Adopt Image of Other Control

Article ID: Q81956

The information in this article applies to:

- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

When an MDI Child custom control is placed on a form, no other non-MDI Child control(s) should be placed on the same form. If a non-MDI Child control is placed directly on the parent form, the MDI child window may appear to adopt, or "pick up," the control when the MDI child window is minimized then maximized (so that it covers the control on the form once maximized or sized).

RESOLUTION

This article does not apply to later versions of Visual Basic. The MDI Child custom control shipped only with version 1.0. Multiple-document interface (MDI) forms are built into Visual Basic version 2.0 and later, making the MDI custom control obsolete.

When using the MDI Child custom control, you should only place controls directly on the child windows you create. The "Microsoft Visual Basic Custom Control Reference" states on page 184 that no controls should be placed on the form (parent window) when using MDI child windows.

STATUS

The problem described in this article does not occur in Visual Basic version 2.0 for Windows.

MORE INFORMATION

Steps to Reproduce Problem in Visual Basic Version 1.0

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the MDICHILD.VBX custom control file. The MDI Child tool appears in the toolbox.

3. Place an MDI Child control on Form1.

4. Place another control (for example, a command button) directly on the form outside the MDI child window.

5. Press F5 to run the application.

6. Move the MDI child window so that it is covering the command button by clicking and dragging the title bar.

7. Click the Minimize button (the down arrow) on the MDI child window to minimize it.

8. Double-click the icon of the minimized MDI child window to restore it.

9. The MDI child window will now appear to have a command button on it.

The MDI child window does not actually have a fully functional copy of the control that was placed directly on the form--it has only an image of the control (in this example, a command button).

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: PrgCtrlsStd

'Text' Property is Read-Only Error as Set Combo Box Text Prop

Article ID: Q84056

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

If you assign a value to the Text property of a combo box with Style=2 (Dropdown list), you will receive a

    'Text' property is read-only

error message. To assign a default value to a combo box with the Style property set to 2, you need to set the ListIndex property of the combo box. If you assign a value to the Text property of a combo box with Style set to 0 (Dropdown Combo) or 1 (Simple Combo), you will not get the above error message.

MORE INFORMATION

A combo box combines the features of a text box and a list box. The user can make a selection by selecting an item from its list or by entering text in the text box portion of the combo box at the top of the combo box.

If the Style property is set to 0 (Dropdown Combo) or 1 (Simple Combo), the user can select an item in the list or enter text in the text portion of the combo box. The text entered may or may not be an item in the list. A default value for the combo box can be set either by assigning a value to the ListIndex property or by assigning a value to the Text property of the combo box.

However, if the Style property is set to 2 (Dropdown List), the user can only select an item from the list. The user cannot enter text directly in the text portion of the combo box. Therefore, a default item for the combo box can be set by assigning a value to the ListIndex property. The Text property of a combo box with Style set to 2 (Dropdown List) is read-only.

Reference(s):

"Microsoft Visual Basic: Language Reference," version 1.0, page 311

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Close VB Combo Box with ENTER key

Article ID: Q84474

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

If you open a combo box and then use the ARROW keys to scroll through it, pressing the ENTER key will not close the combo box like a mouse click will. This is normal behavior. The following example demonstrates how to make a combo box close when the ENTER key is pressed.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The following program makes use of the Windows API SendMessage function to send the combo box the message to close. This is done only after the ENTER key is detected in the KeyPress event for the combo box.

Two Windows API Declare statements must be added to your application. These can be added either in the GLOBAL.BAS module, or in the general Declarations section of the form containing the combo box.

Steps to Reproduce Behavior

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Add the following two declarations to the global module or the General Declarations for Form1:

   ```
   Declare Function SendMessage% Lib "user" (ByVal hWnd%, ByVal
                   wMsg%, ByVal wParam%, ByVal lParam&)
   Declare Function GetFocus Lib "user" () As Integer
   ```

   (Note that the first Declare statement must be on just one line, not split across two lines as it is here.)

3. Place a combo box on Form1.

4. Under the KeyPress event for the combo box, place the following code:

```
    If KeyAscii = 13 Then
        Const WM_USER = &h400
        Const  CB_SHOWDROPDOWN = WM_USER + 15

        Combo1.SetFocus
        BoxwHND% = GetFocus()
        r& = SendMessage(BoxwHND%, CB_SHOWDROPDOWN, 0, 0)
        KeyAscii = 0
    End If
```

5. Place a command button on Form1.

6. In the Click event for Command1, place the following code:

```
   ' This will add some data to the combo box.
   for i =1 to 10
      Combo1.AddItem STR$(i)
   Next i
```

7. Press the F5 key to run the application.

8. Choose the Command1 button to fill the combo box.

9. Open the combo box with the mouse, and scroll down with the ARROW keys.
   Pressing the ENTER key will close the Combo Box.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Edit Grid Cells in VB Using Overlapped Text Box

Article ID: Q85109

This information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

The Grid custom control does not provide any text editing capability. The example program below shows how you can use a text box to perform text editing in the current cell of a grid.

MORE INFORMATION

The example program shown below enables you to edit the contents of a grid cell. When you press a key, the grid moves a text box to the position of the current cell and sets the focus to the text box. When you press the ENTER key or change focus away from the current cell, the program transfers the text in the text box back to the grid.

Steps to Create Example Program

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File. Then select the GRID.VBX file. The Grid tool appears in the Toolbox.

3. Place a grid (Grid1) and a text box (Text1) on Form1. Set the Grid1 Cols and Rows properties both to 4. Then size the grid to show all the cells. Set the Text1 BorderStyle property to None (0) and the Visible property to False (0).

4. Enter the following declarations in the general Declarations section:

```
Const ASC_ENTER = 13        ' ASCII code of ENTER key.
Dim gRow As Integer
Dim gCol As Integer
```

5. Enter the following code in the Grid1_KeyPress procedure:

```
Sub Grid1_KeyPress (KeyAscii As Integer)
    ' Move the text box to the current grid cell:
    Call grid_text_move(Grid1, Text1)

    ' Save the position of the grids Row and Col for later:
    gRow = Grid1.Row
    gCol = Grid1.Col

    ' Make text box same size as current grid cell:
```

```
      Text1.Width = Grid1.ColWidth(Grid1.Col) - 2 * Screen.TwipsPerPixelX
      Text1.Height = Grid1.RowHeight(Grid1.Row) - 2 * Screen.TwipsPerPixelY

      ' Transfer the grid cell text:
      Text1.Text = Grid1.Text

      ' Show the text box:
      Text1.Visible = True
      Text1.ZOrder 0
      Text1.SetFocus

      ' Redirect this KeyPress event to the text box:
      If KeyAscii <> ASC_ENTER Then
         SendKeys Chr$(KeyAscii)
      End If
   End Sub
```

6. Add the following code to the Text1_KeyPress procedure:

```
   Sub Text1_KeyPress (KeyAscii As Integer)
      If KeyAscii = ASC_ENTER Then
         Grid1.SetFocus  ' Set focus back to grid, see Text_LostFocus.
         KeyAscii = 0    ' Ignore this KeyPress.
      End If
   End Sub
```

7. Add the following code to the Text1_LostFocus procedure:

```
   Sub Text1_LostFocus ()
      Dim tmpRow As Integer
      Dim tmpCol As Integer

      ' Save current settings of Grid Row and col. This is needed only if
      ' the focus is set somewhere else in the Grid.
      tmpRow = Grid1.Row
      tmpCol = Grid1.Col

      ' Set Row and Col back to what they were before Text1_LostFocus:
      Grid1.Row = gRow
      Grid1.Col = gCol

      Grid1.Text = Text1.Text  ' Transfer text back to grid.
      Text1.SelStart = 0       ' Return caret to beginning.
      Text1.Visible = False    ' Disable text box.

      ' Return row and Col contents:
      Grid1.Row = tmpRow
      Grid1.Col = tmpCol
   End Sub
```

8. In the general Declarations section or in a separate .BAS file, add the following Sub routine:

```
   Sub grid_text_move (Grid As Control, TextBox As Control)

      ' Move a text box to the position of the current cell in a grid:
      Dim X As Single   ' x position of current grid cell.
```

```
    Dim Y As Single   ' y position of current grid cell.
    Dim i As Integer  ' Column/row index.

    ' Skip grid border:
    X = Grid.Left
    Y = Grid.Top
    If Grid.BorderStyle = 1 Then
       X = X + Screen.TwipsPerPixelX
       Y = Y + Screen.TwipsPerPixelY
    End If

    ' Skip fixed columns and rows:
    For i = 0 To Grid.FixedCols - 1
       X = X + Grid.ColWidth(i)
       If Grid.GridLines Then
          X = X + Screen.TwipsPerPixelX
       End If
    Next
    For i = 0 To Grid.FixedRows - 1
       Y = Y + Grid.RowHeight(i)
       If Grid.GridLines Then
          Y = Y + Screen.TwipsPerPixelY
       End If
    Next

    ' Find current data cell:
    For i = Grid.LeftCol To Grid.Col - 1
        X = X + Grid.ColWidth(i)
        If Grid.GridLines Then
            X = X + Screen.TwipsPerPixelX
        End If
    Next
    For i = Grid.TopRow To Grid.Row - 1
        Y = Y + Grid.RowHeight(i)
        If Grid.GridLines Then
            Y = Y + Screen.TwipsPerPixelY
        End If
    Next

    ' Move the Text Box, and make small adjustments:
    TextBox.Move X + Screen.TwipsPerPixelX, Y + Screen.TwipsPerPixelY
End Sub
```

9. Press the F5 key to run the program. Press a key to begin entering text into a cell. Type in some text. Press the ENTER key to finish editing the cell. Use the arrow keys to move to another cell. You can press the ENTER key to begin editing a cell without changing the contents of the cell.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsStd

How to Make ENTER Key Move Focus Like TAB Key for VB Controls

Article ID: Q85562

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can cause the ENTER key to move the focus to the control with the next higher TabIndex property value, as the TAB key does.

However, using the Enter key to move the focus does not follow recommended Windows application design guidelines. The Enter key should be used to process the default command or to process entered information, not to move the focus.

MORE INFORMATION

You can detect when the user presses ENTER from the KeyPress event procedure by checking to see if the KeyAscii parameter is the character code for ENTER (13). Then you can move the focus to the next control in the TabIndex order with SendKeys "{tab}". You can move the focus to the previous control with SendKeys "+{tab}".

This technique works with most kinds of controls. It does not work with command button controls, because command buttons do not receive the KeyPress event when you press ENTER.

Steps to Create Example Program

1. Start Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Add two text boxes (Text1 and Text2) to Form1.

3. Enter the following code in the Text1 KeyPress procedure:

```
Sub Text1_KeyPress (KeyAscii As Integer)
   If KeyAscii = 13 Then  ' The ENTER key.
      SendKeys "{tab}"    ' Set focus to next control.
      KeyAscii = 0        ' Ignore this key.
   End If
End Sub
```

4. Enter the following code in the Text2 KeyPress procedure:

```
Sub Text2_KeyPress (KeyAscii As Integer)
```

```
      If KeyAscii = 13 Then  ' The ENTER key.
         SendKeys "{tab}"   ' Set focus to next control.
         KeyAscii = 0       ' Ignore this key.
      End If
   End Sub
```

5. Press the F5 key to run the program. When you press ENTER, the focus moves between the two controls.

Additional reference words: 1.00 2.00 3.00 return key
KBCategory:
KBSubcategory: PrgCtrlsStd

PRB: GotFocus Event Fails If MsgBox Invoked in LostFocus Event

Article ID: Q85856

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

Invoking a message box from a control's LostFocus event will prevent the GotFocus event of the next selected control from executing.

CAUSE

This happens because the GotFocus event is not executed. Removing the message box from the control's LostFocus will allow the GotFocus event to execute as expected.

WORKAROUND

To work around the problem, set a flag in the control's LostFocus event procedure. Then call a generic test routine from the next control's GotFocus event, as demonstrated in the following example:

1. Start Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Create the following controls and properties for Form1:

Control	CtlName (Name in 2.0 or 3.0)	Property Setting
Text Box	Text1	TabIndex = 0
Text Box	Text2	TabIndex = 1

3. Add the following code to the general Declarations section of Form1:

```
Dim Text1LostFocus As Integer
Sub CheckLostFocus()
   If Text1LostFocus Then
      MsgBox "Text1 has Lost the Focus"
      Text1LostFocus = 0
   End If
End Sub
```

4. Add the following code to the Text1_LostFocus event procedure:

```
Sub Text1_LostFocus ()
   Text1LostFocus = -1
```

```
   End Sub
```

5. Add the following code to the Text2_GotFocus event procedure:

```
   Sub Text1_GotFocus ()
      Call CheckLostFocus
      MsgBox "Text2 has Received the Focus"
   End Sub
```

6. Press F5 to run the program.

Now, both message boxes should appear as expected when the focus is changed by using the TAB key or by clicking the Text2 box.

STATUS

This behavior is by design. It is a limitation of Visual Basic's MsgBox statement.

MORE INFORMATION

Steps to Reproduce Behavior

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Create the following controls and properties for Form1:

Control	CtlName (Name in 2.0)	Property Setting
Text Box	Text1	TabIndex = 0
Text Box	Text2	TabIndex = 1

3. Add the following code to the Text1_LostFocus event procedure:

```
   Sub Text1_LostFocus ()
      MsgBox "Text1 has Lost the Focus"
   End Sub
```

4. Add the following code to the Text2_GotFocus event procedure:

```
   Sub Text2_GotFocus ()
      MsgBox "Text2 has Received the Focus"
   End Sub
```

5. Press F5 to run the program.

Notice that when you click the second text box (Text2), the message box specified in the GotFocus event fails to display. This also occurs if you try to tab between text boxes or set up labels and quick keys.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

PRB: Can TAB in Error if Value of Option Button Set to False

Article ID: Q85990

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

SYMPTOMS

Setting the Value of an option button to False (0) also sets its TabStop property to False. If you set the Value property of an option button to False without setting the Value property of another option button to True (-1), Visual Basic will allow the user to tab over the other option buttons because all the TabStops are set to False.

This is an invalid state for a group of option buttons. One of the option buttons should always be selected (that is, its Value property should be set to True, which also sets the TabStop property to True).

CAUSE

By default, the TabStop property of option buttons is set to True. Once an option button is selected at run time, the Value property for the other option buttons not selected is set to False, which also sets the TabStop property to False. If you just change the Value property of one option button to False, and do not set the Value property of another option button to True, none of the other option buttons will have their TabStop property set to True, and these option buttons will skipped when the user presses the TAB key to move through controls at run time.

WORKAROUND

To avoid this problem, ensure that one of the options in an option group is always selected.

MORE INFORMATION

Steps to Reproduce Problem

1. Start Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Place a command button (Command1), two option buttons (Option1 and Option2) and another command button (Command2) on Form1.

3. Set the Value property of Option1 to True.

4. Add the following code to the Command1 Click procedure:

```
Sub Command1_Click ()
```

```
        Const FALSE = 0
        Option1.Value = FALSE
    End Sub
```

Note that you do not need to setup a Const FALSE = 0 in Visual Basic version 2.0 because FALSE is already a keyword in version 2.0.

5. Press F5 to run the program.

If you do not click Command1, you can TAB through Option1. However, you will TAB over both option buttons if you click Command1.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Programmatically Display or Hide a VB Combo Box List

Article ID: Q85991

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The list of a Visual Basic drop-down combo box (Style=0 or 2) is usually opened and closed by using a mouse. However, you can also open and close the list of a combo box programmatically by using the Windows SendMessage function as described below.

However, there is an easy way. In visual basic, when a drop-down combo box has the focus, you can press ALT-DOWN to open it up. Therefore, you can use SendKeys to send these keys to the combo box, as in this example:

To do this programmatically, use the following code to change focus to the combo box and send the necessary keystroke:

```
combo1.SetFocus
SendKeys "%{Down}"
```

MORE INFORMATION

The CB_SHOWDROPDOWN constant can be used with the SendMessage function to programmatically open or close the list of a Visual Basic drop-down combo box of Style=0 or Style=2 (Style=1 always has the list displayed). The following steps demonstrate how to open the list of a drop-down combo box:

1. Start a new project in Visual Basic. Form1 is created by default.

2. Place a combo box (Combo1) and a command button (Command1) on Form1.

3. Add the following declarations and constants to the general Declarations section of Form1:

```
' Enter each Declare statement as one, single line:
Declare Function GetFocus Lib "User" () as Integer
Declare Function GetParent Lib "User" (ByVal hWnd as Integer)
   as Integer
Declare Function SendMessage Lib "User" (ByVal hWnd as Integer,
   ByVal wMsg as Integer, ByVal wParam as Integer,
   ByVal lParam as Any) as Long
Global Const WM_USER = &H400
Global Const CB_SHOWDROPDOWN = WM_USER + 15
```

4. Add the following code to the Form1 Load event procedure to put

some items in the combo box:

```
Sub Form_Load ()
   Combo1.AddItem "apple"
   Combo1.AddItem "orange"
   Combo1.AddItem "banana"
End Sub
```

5. Add the following code to the Command1_Click event procedure:

```
Sub Command1_Click ()
   Combo1.SetFocus
   cbhWnd% = GetFocus ()
   cblisthWnd% = GetParent (cbhWnd%)
   cbFunc% = -1           'cbFunc% = -1 displays the list
                          'cbFunc% = 0 hide the list
   retval& = SendMessage (cblisthWnd%, CB_SHOWDROPDOWN, cbFunc%, 0&)
End Sub
```

6. Press the F5 key to run the program. Click Command1 to display the list of the combo box.

If Style=2 for the combo box, there is no need to use the GetParent function. Use the return value of the GetFocus (cbhWnd% in the above example) call as the first parameter of the SendMessage function.

NOTE: The list of a combo box with Style=0 or 2 will close when the combo box loses focus.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

PRB: ChDir or ChDrive Won't Change File / Directory List Boxes

Article ID: Q86279

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

Using the ChDir or ChDrive statement to change the current directory or drive does not change the listing of a file list box or a directory list box. However, the list changes if you run the program a second time in the VB.EXE environment.

RESOLUTION

To change the contents of a file list box or directory list box, set its Path property instead of using the ChDir or ChDrive statement.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Reproduce Behavior:

1. Run Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Place a label (Label1), a file list box (File1), and a directory list box (Dir1) on to Form1.

3. In the Form_Load event procedure, add the following code:

```
Sub Form_Load ()
   ChDir "C:\DOS"
   Label1.Caption = CurDir$
End Sub
```

4. Press the F5 key to run the program. The label will display "C:\DOS", but the files listed are still those from the directory where Visual Basic was started.

5. From the Run menu, choose End. Press the F5 key to run the program again. This time, the files listed are from the C:\DOS subdirectory.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Visual Basic Can Load RLE4 and RLE8 Bitmap Format Files

Article ID: Q86283

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Microsoft Visual Basic can load icons (.ICO), Windows metafiles (.WMF), Windows bitmap files (.BMP), and Windows compressed bitmap files (.RLE), both RLE4 and RLE8.

MORE INFORMATION

You can load an .RLE bitmap file as you would any other bitmap. For example, at design time you can set the Picture property of a picture box or form to an .RLE file. You can also use the LoadPicture function at run time to load the .RLE file into a picture box or form.

Although Visual Basic can load these image formats, the SavePicture statement can only save images in the regular Windows bitmap (.BMP) or Windows icon (.ICO) file format.

To save an image as an icon (.ICO), you must first load it as an icon. You can change the image using graphics methods such as Line and PSet, and then save the Picture property of the picture box or form as an icon (.ICO). Otherwise, you can only save images as Windows bitmap files (.BMP).

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Example to Evaluate Basic Numeric Expressions

Article ID: Q86688

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0
- The Standard and Professional Editions of Microsoft Visual Basic for MS-DOS, version 1.0

SUMMARY

This article contains an example program that evaluates a numeric expression contained in a string, mimicking the operators, built-in functions, and order of evaluation used by Microsoft Basic language products. This article also explains the operator precedence rules in detail.

MORE INFORMATION

The example program listed below recognizes the following operators and subexpressions, listed by precedence from highest to lowest:

- Constants, function calls, parentheses
- Exponentiation ^
- Unary minus -
- Multiplication and division *, /
- Integer division \
- Integer modulus MOD
- Addition and subtraction +, -
- Relational operators =, <>, <, >, <=, >=
- NOT
- AND
- OR
- XOR
- EQV
- IMP

The precedence of unary minus "-" and operator "NOT" indicate the highest possible precedence of their operand. Unary minus and "NOT" may occur in an expression of any precedence. The following expressions illustrate the precedence rules for unary minus and "NOT".

Expression	Value
-1 ^ 2	-1
-(1 ^ 2)	-1
(-1) ^ 2	1
2 ^ -2	.25
NOT 0 = 1	-1
NOT (0 = 1)	-1

```
   (NOT  0) = 1        0
    NOT  0  AND 1      1
   (NOT  0) AND 1      1
    NOT (0  AND 1)    -1
```

The example program listed below accepts number constants written as decimal numbers with an optional fraction. For example, it accepts "123" and "123.4". It is possible to modify the program to recognize hexadecimal, scientific notation, or other formats.

This example program also recognizes the following functions: ABS, ATN, COS, EXP, FIX, INT, LOG, RND, SNG, SIN, SQR, and TAN.

Steps to Create Example Program

1. Run Visual Basic, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic is already running. Form1 will be created by default.

2. Add a text box (Text1) and a command button (Command1) to Form1.

3. Set the Text property for Text1 to the null string (empty).

4. Enter the following code in the Command1 Click event procedure:

```
Sub Command1_Click ()
    Dim n As Double

    If e_eval(Text1.Text, n) Then
      MsgBox Format$(n)
    End If
End Sub
```

5. Add the following code in the general Declarations section of Form1:

```
' To run this program in Visual Basic for MS-DOS, change the
' following Dim statements to DIM SHARED.
'
Dim e_input As String     ' Expression input string.
Dim e_tok As String       ' Current token kind.
Dim e_spelling As String  ' Current token spelling.
Dim e_error As Integer    ' Tells if syntax error occurred.

' e_eval
'   Evaluate a string containing an infix numeric expression.
'   If successful, return true and place result in <value>.
'   This is the top-level function in the expression evaluator.
Function e_eval (ByVal s As String, value As Double) As Integer
   ' Initialize.
   e_error = 0
   e_input = s
   Call e_nxt

   ' Evaluate.
   value = e_prs(1)
```

```
   ' Check for unrecognized input.
   If e_tok <> "" And Not e_error Then
      MsgBox "syntax error, token = '" + e_spelling + "'"
   e_error = -1
   End If

   e_eval = Not e_error
End Function

' e_prs
'   Parse an expression, allowing operators of a specified
'   precedence or higher. The lowest precedence is 1.
'   This function gets tokens with e_nxt and recursively
'   applies operator precedence rules.
Function e_prs (p As Integer) As Double
   Dim n As Double    ' Return value.
   Dim fun As String  ' Function name.

   ' Parse expression that begins with a token (precedence 12).
   If e_tok = "num" Then
      ' number.
      n = Val(e_spelling)
      Call e_nxt
   ElseIf e_tok = "-" Then
      ' unary minus.
      Call e_nxt
      n = -e_prs(11)    ' Operand precedence 11.
   ElseIf e_tok = "not" Then
      ' logical NOT.
      Call e_nxt
      n = Not e_prs(6)  ' Operand precedence 6.
   ElseIf e_tok = "(" Then
      ' parentheses.
      Call e_nxt
      n = e_prs(1)
      Call e_match(")")
   ElseIf e_tok = "id" Then
      ' Function call.
      fun = e_spelling
      Call e_nxt
      Call e_match("(")
      n = e_prs(1)
      Call e_match(")")
      n = e_function(fun, n)
   Else
      If Not e_error Then
         MsgBox "syntax error, token = '" + e_spelling + "'"
         e_error = -1
      End If
   End If

   ' Parse binary operators.
Do While Not e_error
  If 0 Then  ' To allow ElseIf .
  ElseIf p <= 11 And e_tok  = "^"  Then Call e_nxt: n = n ^   e_prs(12)
  ElseIf p <= 10 And e_tok  = "*"  Then Call e_nxt: n = n *   e_prs(11)
  ElseIf p <= 10 And e_tok = "/"   Then Call e_nxt: n = n /   e_prs(11)
```

```
  ElseIf p <= 9  And e_tok = "\"   Then Call e_nxt: n = n \   e_prs(10)
  ElseIf p <= 8  And e_tok = "mod" Then Call e_nxt: n = n Mod e_prs(9)
  ElseIf p <= 7  And e_tok = "+"   Then Call e_nxt: n = n +    e_prs(8)
  ElseIf p <= 7  And e_tok = "-"   Then Call e_nxt: n = n -    e_prs(8)
  ElseIf p <= 6  And e_tok = "="   Then Call e_nxt: n = n =    e_prs(7)
  ElseIf p <= 6  And e_tok = "<"   Then Call e_nxt: n = n <    e_prs(7)
  ElseIf p <= 6  And e_tok = ">"   Then Call e_nxt: n = n >    e_prs(7)
  ElseIf p <= 6  And e_tok = "<>"  Then Call e_nxt: n = n <>   e_prs(7)
  ElseIf p <= 6  And e_tok = "<="  Then Call e_nxt: n = n <=   e_prs(7)
  ElseIf p <= 6  And e_tok = ">="  Then Call e_nxt: n = n >=   e_prs(7)
  ElseIf p <= 5  And e_tok = "and" Then Call e_nxt: n = n And  e_prs(6)
  ElseIf p <= 4  And e_tok = "or"  Then Call e_nxt: n = n Or   e_prs(5)
  ElseIf p <= 3  And e_tok = "xor" Then Call e_nxt: n = n Xor  e_prs(4)
  ElseIf p <= 2  And e_tok = "eqv" Then Call e_nxt: n = n Eqv  e_prs(3)
  ElseIf p <= 1  And e_tok = "imp" Then Call e_nxt: n = n Imp  e_prs(2)
  Else
       Exit Do
  End If
 Loop

   e_prs = n
End Function

' e_function.
'   Evaluate a function. This is a helper function to simplify
'   e_prs.
Function e_function (fun As String, arg As Double) As Double
   Dim n As Double

   Select Case LCase$(fun)
   Case "abs": n = Abs(arg)
   Case "atn": n = Atn(arg)
   Case "cos": n = Cos(arg)
   Case "exp": n = Exp(arg)
   Case "fix": n = Fix(arg)
   Case "int": n = Int(arg)
   Case "log": n = Log(arg)
   Case "rnd": n = Rnd(arg)
   Case "sgn": n = Sgn(arg)
   Case "sin": n = Sin(arg)
   Case "sqr": n = Sqr(arg)
   Case "tan": n = Tan(arg)
   Case Else
      If Not e_error Then
         MsgBox "undefined function '" + fun + "'"
         e_error = -1
      End If
   End Select

   e_function = n
End Function

' e_nxt
'   Get the next token into e_tok and e_spelling and remove the
'   token from e_input.
'   This function groups the input into "words" like numbers,
'   operators and function names.
```

```
Sub e_nxt ()
   Dim is_keyword As Integer
   Dim c As String  ' Current input character.

   e_tok = ""
   e_spelling = ""

   ' Skip whitespace.
   Do
      c = Left$(e_input, 1)
      e_input = Mid$(e_input, 2)
   Loop While c = " " Or c = Chr$(9) Or c = Chr$(13) Or c = Chr$(10)

   Select Case LCase$(c)

   ' Number constant. Modify this to support hexadecimal, etc.
   Case "0" To "9", "."
      e_tok = "num"
      Do
         e_spelling = e_spelling + c
         c = Left$(e_input, 1)
         e_input = Mid$(e_input, 2)
      Loop While (c >= "0" And c <= "9") Or c = "."
      e_input = c + e_input

   ' Identifier or keyword.
   Case "a" To "z", "_"
      e_tok = "id"
      Do
         e_spelling = e_spelling + c
         c = LCase$(Left$(e_input, 1))
         e_input = Mid$(e_input, 2)
         is_id% = (c >= "a" And c <= "z")
         is_id% = is_id% Or c = "_" Or (c >= "0" And c <= "9")
      Loop While is_id%
      e_input = c + e_input

      ' Check for keyword.
      is_keyword = -1
      Select Case LCase$(e_spelling)
         Case "and"
         Case "eqv"
         Case "imp"
         Case "mod"
         Case "not"
         Case "or"
         Case "xor"
         Case Else: is_keyword = 0
      End Select
      If is_keyword Then
         e_tok = LCase$(e_spelling)
      End If

   ' Check for <=, >=, <>.
   Case "<", ">"
      e_tok = c
      c = Left$(e_input, 1)
```

```
      If c = "=" Or c = ">" Then
         e_tok = e_tok + c
         e_input = Mid$(e_input, 2)
      End If

   ' Single character token.
   Case Else
      e_tok = c
   End Select

    If e_spelling = "" Then
       e_spelling = e_tok
   End If
End Sub

' e_match
'   Check the current token and skip past it.
'   This function helps with syntax checking.
Sub e_match (token As String)
   If Not e_error And e_tok <> token Then
      MsgBox "expected " + token + ", got '" + e_spelling + "'"
      e_error = -1
   End If
   Call e_nxt
End Sub
```

6. Press F5 to run the program. Type an expression into Text1 such as "1+2*3^4". Click Command1.

The program displays the result, 163 in this case.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Right Justify Top-Level Menus in Visual Basic

Article ID: Q86772

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

It is possible to right justify top-level menu items by including a CHR$(8) (backspace) as the first character of the caption for the left-most menu that you want to appear on the right side of the menu bar.

MORE INFORMATION

Microsoft publishes a book called "The Windows Interface: An Application Design Guide," which contains standards and guidelines for Windows 3.1. The following guidelines for right-justified Help menu items is taken from that book:

> "Some applications currently place the Help menu at the extreme right of the menu bar. The current recommendation is that the Help menu immediately follow the next-to-last menu item, for three reasons: (1) to increase the accessibility of the help menu; (2) to decrease the likelihood of pressing the Maximize or Minimize buttons by mistake when trying to access Help; and (3) to make the space at the extreme right of the menu bar available in MDI applications for a Restore icon for maximized child windows."
>
> Source: "The Windows Interface: An Application Design Guide"
> by Microsoft Corporation
> 248 pages, plus two 3.5-inch disks
> ISBN: 1-55615-439-9
> Publication Date: October 9, 1992

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the Window menu, choose Menu Design Window.

3. Create a menu item with One as both caption and control name.

4. Choose the Next button, and repeat step 3 with using Two as the name.

5. Choose Done to exit the Menu Design Window. You now have two top-level menus.

6. Add the following code to the Form1 Form_Load event procedure:

```
Sub Form_Load ()
   Two.Caption = Chr$(8) + Two.Caption
End Sub
```

7. Press the F5 key to run the program.

The Two menu item appears on the right side of the menu bar. If you add the backspace to the One menu caption, both menus will be right-justified.

Special Trick for Design Time

The technique given above can be done only at run time. However, here's a trick you can use to do the same thing at design time:

1. Make a text file with ASCII character 8 in it.

2. Load the text file into NOTEPAD.EXE and copy the character to the clipboard.

3. Then, when you're in the Menu Design window, Press SHIFT-INS (Paste) in the caption field to place CHR$(8) character in it.

Additional reference words: 1.00 2.00 3.00 alignment right-aligned align
KBCategory:
KBSubcategory: PrgCtrlsStd

Unable to Display Uppercase W in Small Text Box

Article ID: Q87770

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

SYMPTOMS

An uppercase W character may fail to display in the smallest possible size text box on a Microsoft Visual Basic for Windows form with the default FontSize (8.25) and FontName (Helv or Helvetica) selected. All other uppercase letters display correctly.

CAUSE

Because the uppercase letter W is the widest of the uppercase letters in the Helv 8.25 font, and it is slightly wider then the width of the smallest possible Visual Basic text box, there is not enough room to display the letter. Therefore, nothing appears in the text box.

RESOLUTION

This situation is limitation of the text box control in Microsoft Visual Basic for Windows. Avoid the problem by using a different font for the text box or by increasing the size of the text box.

MORE INFORMATION

Steps to Reproduce Behavior

1. Start Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add the smallest possible text box to display the letter T in the word Text1 of the text box to Form1.

3. Press F5 to run the example. Activate the CAPS LOCK key and type an uppercase W.

In Windows version 3.0, the letter W does not appear in the text box. Instead, a space appears as if you had pressed the SPACEBAR.

In Windows version 3.1, the letter W appears not to be entered. It appears as if it is ignored.

For further testing, try other uppercase letters. There should be no problem with displaying other uppercase letters.

Additional reference words: 1.00 2.00 3.00
KBCategory:

KBSubcategory: PrgCtrlsStd

PRB: SendKeys May Return Illegal Function Call Error

Article ID: Q87773

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

SYMPTOMS

The SendKeys statement reports the error "Illegal function call" when its argument contains an incorrectly formatted string. This article describes specific circumstances that cause this error, and contains a code example that shows how to send any string with SendKeys.

CAUSE

The following characters have special meaning to the SendKeys statement:

```
+ ^ % ~ ( ) [ ] { }
```

The SendKeys statement reports an "Illegal function call" error if its argument contains one of the following, not enclosed in braces:

- An unmatched parenthesis () or bracket {}
- A bracket []
- Braces containing an undefined character sequence, such as {abc}

RESOLUTION

To prevent the SendKeys statement from interpreting a character, enclose the character in braces {}. For example, to send the string

```
The interest rate is 5% (annually).
```

Use the following SendKeys syntax:

```
SendKeys "The interest rate is 5{%} {(}annually{)}."
```

MORE INFORMATION

Step-by-Step Example

The following example demonstrates how to use the SendKeys statement to send strings that would normally cause an "Illegal function call" error:

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Place a text box (Text1) on Form1.

3. Place a command button (Command1) on Form1.

4. Enter the following code:

```
Sub Command1_Click ()
   Text1.SetFocus
   SendKeys sendkeys_prepare("1+2^5% ")
   SendKeys sendkeys_prepare("[] ")
   SendKeys sendkeys_prepare("{abc}")
End Sub

' The following function puts braces {} around characters that
' are special to the SendKeys statement.
Function sendkeys_prepare (in As String) As String
   For i% = 1 To Len(in)
      ' Get the next character into c$.
      c$ = Mid$(in, i%, 1)
      ' If c$ is one of the special characters.
      If InStr("+^%~()[]{}", c$) Then
         out$ = out$ + "{" + c$ + "}"
      Else
         out$ = out$ + c$
      End If
   Next
   sendkeys_prepare = out$
End Function
```

5. Press F5 to run the program. Click Command1. Some example text containing characters special to SendKeys appear in Text1.

Reference(s):

"Microsoft Visual Basic: Language Reference," version 1.0, pages 283-284

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

PRB: SetFocus During Form Load May Cause Illegal Function Call

Article ID: Q88477

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

SYMPTOMS

Using the SetFocus method to set the focus to a specific control on a form during the form load event procedure may result in an "Illegal Function Call" error.

CAUSE

This error occurs because the form that the control is on is not yet visible.

RESOLUTION

To prevent this error from occurring, execute Form.Show before executing the SetFocus method.

MORE INFORMATION

Steps to Reproduce Problem

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Place a text box (Text1) on Form1.

3. Add the following code to the Form_Load procedure for Form1:

```
Sub Form_Load ()
   Text1.SetFocus
End Sub
```

4. Press F5 to run the application. The line of code in the load event will be highlighted with the error "Illegal Function Call."

5. If you show the form before executing SetFocus, the program will run as expected.

```
Sub Form_Load ()
   Form1.Show
   Text1.SetFocus
End Sub
```

Additional reference words: 1.00 2.00 3.00

KBCategory:
KBSubcategory: PrgCtrlsStd

PRB: Click Event Invoked When Option Button Receives Focus

Article ID: Q88792

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

When an option button or group of option buttons is initially placed on a form in Microsoft Visual Basic for Windows, the buttons will remain unselected until the focus is shifted to one of the option buttons. This can cause unexpected results, because shifting the focus to one of the option buttons will invoke a Click event for that option button.

WORKAROUND

To work around this feature, you must manually, in design mode, set the Value property for one of the option buttons to True.

STATUS

This behavior is by design.

MORE INFORMATION

In a group of option buttons, one of the buttons should be selected unless the option buttons apply to only certain selected objects on the current form. In this case, the option buttons, when initially displayed, would not be selected.

Visual Basic for Windows will allow option buttons to be placed on a form without selecting any of the option buttons. If you desire to create a group of option buttons with none of them selected, there is no way to prevent a Click event from being invoked when the focus is shifted to one of them. A problem may occur when an unselected option button is first in the tab order. The option button will automatically get selected when the form is shown. In all cases, to prevent the Click event from occurring, in design mode, you must set the Value property of one of the option buttons to True.

The following steps demonstrate this feature, as well as a way to work around this feature:

Steps to Reproduce Behavior

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Set the AutoRedraw property for Form1 to True. This prevents any text printed to the screen from being overwritten when Windows redraws the form.

3. Create one or more option buttons on Form1.

4. In the Click event procedure for the first option button, insert the following code:

```
PRINT "Option Button Clicked"
```

5. From the Run menu, choose Start (ALT, R, S) to run the program.

Note that the Click event is invoked when the form is shown and the focus is given to the first option button. To work around this problem, after creating the option buttons in step 2, set the Value property of one of the option buttons to True.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Detect when the Active Form Changes in Visual Basic

Article ID: Q88909

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

A program can detect when the active form changes by monitoring Screen.ActiveForm with a timer control. You generally cannot use the Form_GotFocus event to detect when the active form changes because GotFocus only occurs on forms that contain no active controls.

MORE INFORMATION

To determine when the user activates a new form, put a timer on one of your active forms that continuously checks to see if a property on Screen.ActiveForm changes. For example, Screen.ActiveForm.Caption gives the caption of the current form. If each form has a unique caption, Screen.ActiveForm.Caption changes as the active form changes.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose New Form to create Form2.

3. Add a timer (Timer1) to Form1 and set its Interval to 100.

4. Enter the following code into the Timer1_Timer event procedure:

```
Sub Timer1_Timer ()
   Static last_caption As String
   If Screen.ActiveForm.Caption <> last_caption Then
      MsgBox Screen.ActiveForm.Caption + " activated"
      last_caption = Screen.ActiveForm.Caption
   End If
End Sub
```

5. Enter the following code into the Form_Load event procedure of Form1:

```
Sub Form_Load ()
   Form2.Show
End Sub
```

6. Press F5 to run the program. When you run the program and each time you activate a new form, a message box displays the caption of the newly activated form.

Additional reference words: 1.00 2.00 3.00 VBMSDOS
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Determine Which Option Button is Selected in VB

Article ID: Q88910

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article describes a suggested method for determining which one of a group of option buttons is selected.

MORE INFORMATION

You can do the following to determine which option button is selected:

1. Make the group of option buttons a control array.

2. In the Click event handler for the control array, save the index of the selected option button into a global variable.

3. When you want to know which option button is selected, check the global variable.

An alternative method to check which option button was selected is to examine the Value property of each option button in a sequence of If-Then statements, or in a Select Case statement.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Place a command button (Command1) and option button (Option1) on Form1.

3. With Option1 selected, from the Edit menu, choose Copy.

4. From the Edit menu, choose Paste. A dialog box asks you if you want to create a control array. Choose Yes.

5. Change the Caption property of the two option buttons to "option 1" and "option 2".

6. Enter the following code into the general Declarations section of Form1:

   ```
   Dim option_index As Integer
   ```

7. Enter the following code into the Option1 control array Click event procedure:

```
   Sub Option1_Click (Index As Integer)
      option_index = Index
   End Sub
```

8. Enter the following code into the Command1 Click event procedure:

```
   Sub Command1_Click ()
      MsgBox Option1(option_index).Caption
   End Sub
```

9. Press F5 to run the program. When you click Command1, a dialog box displays the caption of the selected option button.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Make a Spreadsheet-Style Grid that Allows Editing

Article ID: Q88912

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.00
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY
=======

The Grid custom control does not provide any text editing capability. However, you can create a spreadsheet-style grid that allows editing by using a picture box and a text box.

MORE INFORMATION
================

We do not recommend creating a spreadsheet-style grid with a large matrix of text box controls because doing so will slow down your program, and use excessive system resources.

An efficient way to create a grid is to draw vertical and horizontal lines to represent the cells of the grid. Use a single text box to allow editing of the active cell. Check for MouseDown events to move the text box to the currently active cell position, and use the Print method to draw text in a cell when the text box moves away from the cell. Then, store the grid cell values in a two dimensional array, indexed by the column and row.

Code can be added to allow for highlighting areas, using ARROW keys to move between cells, and so on.

Step-by-Step Example
----

1. Start Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Place a picture (Picture1) on Form1, and set its properties as follows:

Property	Value
AutoRedraw	True
ScaleMode	3 - Pixel
Height	2000
Width	3000

3. Place a text box (Text1) in Picture1 by clicking the text box tool. The mouse pointer turns to cross-hairs. Click and drag inside Picture1 to place a gray rectangle appears in Picture1.

4. Add the following code to the general Declarations section of Form1:

```
' Maximum grid size.
Const grid_col_max = 10
Const grid_row_max = 20

' Current grid size.
Dim grid_cols As Integer
Dim grid_rows As Integer

' Current cell position.
Dim grid_col As Integer
Dim grid_row As Integer

' Grid string contents.
Dim grid_text(grid_col_max, grid_row_max) As String

' Grid line positions.
Dim grid_line_col(grid_col_max) As Integer
Dim grid_line_row(grid_col_max) As Integer

' grid_edit_move.
'   Moves the grid edit text box to a new position.
'
Sub grid_edit_move (col As Integer, row As Integer)
   Dim x1 As Integer  ' Picture box positions.
   Dim y1 As Integer
   Dim x2 As Integer
   Dim y2 As Integer

   ' Save text box contents to grid array.
   grid_text(grid_col, grid_row) = Text1.Text

   ' Clear current cell.
   x1 = grid_line_col(grid_col) + 1
   y1 = grid_line_row(grid_row) + 1
   x2 = grid_line_col(grid_col + 1) - 1
   y2 = grid_line_row(grid_row + 1) - 1
   Picture1.Line (x1, y1)-(x2, y2), Picture1.BackColor, BF

   ' Print text box contents to current cell.
   Picture1.CurrentX = x1 + 3
   Picture1.CurrentY = y1 + 3
   Picture1.Print Text1.Text

   ' Set new grid current cell.
   grid_col = col
   grid_row = row

   ' Move text box to new cell.
   x1 = grid_line_col(grid_col)
   y1 = grid_line_row(grid_row)
   w! = grid_line_col(grid_col + 1) - x1
   h! = grid_line_row(grid_row + 1) - y1
   Text1.Move x1 + 1, y1 + 1, w! - 1, h! - 1

   ' Copy contents of new cell to text box.
   Text1.Text = grid_text(grid_col, grid_row)
```

```
End Sub
```

5. Add the following code to form Load event procedure:

```
Sub Form_Load ()
   ' Set grid size.
   grid_cols = 4
   grid_rows = 6

   ' Remove border.
   Picture1.BorderStyle = 0

   ' Set column widths and row heights.
   Dim i As Integer
   Dim d As Integer
   d = 0
   For i = 0 To UBound(grid_line_col)
      grid_line_col(i) = d
      d = d + 40
   Next
   d = 0
   For i = 0 To UBound(grid_line_row)
      grid_line_row(i) = d
      d = d + 20
   Next

   ' Draw grid lines.
   For i = 0 To grid_cols
      x2% = grid_line_col(i)
      y2% = grid_line_row(grid_rows)
      Picture1.Line (grid_line_col(i), 0)-(x2%, y2%)
   Next
   For i = 0 To grid_rows
      x2% = grid_line_col(grid_cols)
      y2% = grid_line_row(i)
      Picture1.Line (0, grid_line_row(i))-(x2%, y2%)
   Next

   Call grid_edit_move(0, 0)
End Sub
```

6. Add the following code to the Picture1 GotFocus event procedure:

```
Sub Picture1_GotFocus ()
   Text1.SetFocus
End Sub
```

7. Add the following code to the Picture1 MouseDown event procedure:

```
' The following line should appear on one line.
Sub Picture1_MouseDown (Button As Integer, shift As Integer,
      x As Single, y As Single)
   Dim col As Integer
   Dim row As Integer
   Dim i As Integer

   ' Find the cell clicked in.
```

```
    col = grid_col
    row = grid_row
    For i = 0 To grid_cols - 1
      If x>=grid_line_col(i) And x<grid_line_col(i+1) Then
        col = i
        Exit For
      End If
    Next
    For i = 0 To grid_rows - 1
      If y>=grid_line_row(i) And y<grid_line_row(i+1) Then
        row = i
        Exit For
      End If
    Next

    ' Move the text box there.
    Call grid_edit_move(col, row)
  End Sub
```

8. Press F5 to run the program. Click a cell and edit the text.

This example is very limited in functionality. Text can be edited in each cell but you must click a cell to move to that particular cell. This article shows a method of creating a grid without tying up a large amount of system resources. Feel free to add code to increase its functionality.

Additional reference words: 1.00 2.00 3.00 optimize
KBCategory:
KBSubcategory: PrgCtrlsStd PrgCtrlsCus

PRB: DropDown Combo Box May Display Partial String

Article ID: Q89219

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Windows versions 3.0 and 3.1

SYMPTOMS

If the Visual Basic for Windows Combo Box Style is set to "0 - Dropdown Combo," you may see only the rightmost portion (right-aligned string) of the text displayed in the text box portion of the combo box at run time.

This problem only occurs if the combo box is too narrow to display the entire string.

WORKAROUND

To work around the problem, set the combo box Style property to "2 - Dropdown List" to change the Style property displays to the left part of the string.

Another alternative is to design the combo box with a wider dimension by increasing the width to greater than 1440 twips (the equivalent of one inch). For example, if you set the Width property to 4320 twips, the width increases to approximately three inches -- a size of combo box that would hold the entire string in the example shown in the More Information section below.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Demonstrate Behavior

1. Start Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a one-inch wide combo box (that is, the width is equal to 1440 twips by default for a combo box) to Form1.

3. Double-click the form or press F7 to open the Form_Click event procedure. Add the following code to the Form_Click event procedure:

```
   Sub Form_Click ()
      Combo1.AddItem "12345678900000000000"  '** 10 zeros
   End Sub
```

4. Press F5 to run the example, or from the Run menu, choose Start.

5. Click the form a couple of times.

6. Select the down arrow on the combo box, and click one of the entries. You should see the entry being placed in the text box portion of the combo box, but instead the entry only displays zeros. The digits 1 through 9 are not displayed.

7. If you change the Style property of the combo box to "2 - Dropdown List," and try the example, the left-aligned string displays in the combo box.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Visual Basic 3.0 Programming Questions & Answers

Article ID: Q92550

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, version 3.0

---

1. Q. I use the picture control to group other controls. However when I select the picture control, the other controls do not remain on top of the picture control. How can I correct this problem?

   A. This problem occurs if you place the controls on the form in the same place as the picture control but not in the picture control. itself. To group the controls in a picture control, you must first select the Picture control and then draw the desired control within the Picture control. For more information, please see Chapter 3 of the "Microsoft Visual Basic version 3.0 Programmer's Guide."

2. Q. How can I make calls from Visual Basic to the functions in the Windows Application Programming Interface (API) or other dynamic link libraries (DLLs)?

   A. To call a subroutine or function from one of the Windows APIs or any other DLL, you need to first provide a Declare statement for that subroutine or function in your Visual Basic application. The exact syntax for the declaration for each Windows API function can be found in the WIN31API.HLP help file included with the Professional Edition of Visual Basic. For more information, please see Chapter 24 of the "Microsoft Visual Basic version 3.0 Programmer's Guide."

3. Q. Is there a reference available that lists the correct Visual Basic declarations for the Multimedia API functions?

   A. Yes, the file is called WINMMSYS.TXT. It comes with the Professional edition of Visual Basic. You can find it in the \VB\WINAPI directory.

4. Q. Is there a reference available that lists the correct Visual Basic declarations for the Windows for Workgroups API functions?

   A. No, at this time such a file is not available from Microsoft. However, you can obtain a copy of the Windows for Workgroups SDK from the WINEXT forum on CompuServe.

5. Q. I followed the examples in the manuals and in the help file on how to use Domain functions such as DSum and DCount, but I keep receiving this error:

      Reference to undefined function or array.

      Why?

   A. The examples provided for the Domain Aggregate functions are

incorrect. These functions must be used within an SQL Statement just as SQL Aggregate functions such as Sum and Count are used. Please look at the SQL Aggregate examples to see how to use these functions within an SQL Statement. For more information, query on the following words in the Microsoft Knowledge Base:

DOMAIN and FUNCTION and SQL

6. Q. I want to sort the records referenced by the Data Control in my application. I tried to use the Index Property as described in the example in the manual and in the help file, but I receive the following error message:

   Property 'Index' not found

   Why?

   A. The examples provided in the Index Property are incorrect. The Index property does not apply to the Data Control. To sort the records referenced by the Data Control, use the ORDER BY Clause within an SQL Statement in the RecordSource property of the Data Control.

7. Q. Is there a better way than the Print Form method to print Forms and Controls in a program?

   A. Yes, it is possible to print forms and/or controls and specify the printed size by using various Windows API function calls. This process is documented in Microsoft Knowledge Base article Q85978. You can also find this article in the top 10 Microsoft Knowledge Base articles that are in the Visual Basic help file. To view these articles, select "Technical Support" from the Contents screen in the Visual Basic help file. Then select "Knowledge Base Articles on Visual Basic."

Additional reference words: 3.00 ivrfax fasttips
KBCategory:
KBSubcategory: PrgCtrlsStd APrgOther TlsCDK

Name Property Cannot Be Set When Using Implicit Property

Article ID: Q93214

The information in this article applies to:

- Microsoft Visual Basic for Windows, version 2.0

SUMMARY

On Page 126 of the Visual Basic Programmer's Guide, it incorrectly states that all controls have an implicit property you can use for storing or retrieving values. Some controls supplied with the Professional Edition of Visual Basic for Windows use the Name property as their implicit property, which you cannot use at run-time.

MORE INFORMATION

The following controls from the Visual Basic Professional Edition use the Name property as their implicit property:

   Common dialog
   MAPI session
   MAPI message
   Spin button

Attempting to access the implicit property of these controls results in one of the following errors:

   'Name' property cannot be read at run time
   'Name' property cannot be set at run time

You access the implicit property of a control (also known as the "value of a control" or the "default value of a control") by writing the control name with no property. For example, with a text box named Text1, you can write the following statement to assign a value to the Text property:

```
Text1 = "hello world"
```

The following list shows the implicit properties for all the controls in both the Standard and Professional Editions:

Standard Control	Implicit Property
Check box	Value
Combo box	Text
Command button	Value
Directory list box	Path
Drive list box	Drive
File list box	FileName
Frame	Caption
Grid	Text
Image	Picture

Label	Caption
Line	Visible
List box	Text
Menu	Enabled
OLE client	Action
Option button	Value
Picture box	Picture
Scroll bar vertical	Value
Scroll bar horizontal	Value
Shape	Shape
Text box	Text
Timer	Enabled

Professional Control	Implicit Property
3D check box	Value
3D command button	Value
3D frame	Caption
3D group push button	Value
3D option button	Value
3D panel	Caption
Animated button	Value
Common dialog	Name (not usable)
Communications	Input
Gauge	Value
Graph	QuickData
Key status	Value
MAPI session	Name (not usable)
MAPI message	Name (not usable)
Masked edit	Text
Multimedia MCI	Command
Pen BEdit	Text
Pen HEdit	Text
Pen ink on bitmap	Picture
Pen on-screen keyboard	Visible
Picture clip	Picture
Spin button	Name (not usable)

Additional reference words: 2.00 docerr
KBCategory:
KBSubcategory: RefsDoc PrgCtrlsStd PrgCtrlsCus

Making Enter Key in Directory List Box Act Like Double-Click

Article ID: Q93215

The information in this article applies to:

- Microsoft Visual Basic for Windows, versions 1.0, 2.0, and 3.0
- The Standard and Professional Editions of Microsoft Visual Basic for MS-DOS, version 1.0

SUMMARY

When you double-click an item in a directory list box control, it opens the directory and displays its subdirectories.

The directory list box control ignores the Enter key by default. To cause a directory list box to treat the Enter key the same way as a double-click, set the Path property to List(ListIndex) from within the KeyPress event handler.

MORE INFORMATION

The following code shows how to cause a directory list box to open the selected directory when the user presses the ENTER key. This code causes a Change event, just as when you double-click an item.

```
Sub Dir1_KeyPress (KeyAscii As Integer)
   If KeyAscii = 13 Then
      Dir1.Path = Dir1.List(Dir1.ListIndex)
   End If
End Sub
```

If your form contains a command button with the Default property set to True, pressing the ENTER key clicks the command button instead of firing the KeyPress event. In this case, set the Path property to List(ListIndex) from within the button Click event handler. For example:

```
Sub Command1_Click ()
   Dir1.Path = Dir1.List(Dir1.ListIndex)
End Sub
```

Additional reference(s):

Chapter 18, "Using the File-System Controls" of the "Microsoft Visual Basic for Windows Programmer's Guide"

Chapter 10, "Using the File-System Controls" of the "Microsoft Visual Basic for MS-DOS Programmer's Guide"

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Change the Size of the Text Cursor in a Text Box

Article ID: Q94318

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

Although there is no property that will allow you to change the appearance of the text cursor (text caret) in a Visual Basic text box, you can use the Windows API call CreateCaret() function to do so.

MORE INFORMATION

API Calls

In the example below, API calls change the size of the text cursor. The CreateCaret() function creates a new shape for the system caret and assigns ownership of the caret to the given window. The caret shape can be a line, block, or bitmap. Here's the syntax:

```
Void CreateCaret(hwnd, hbmp,nwidth,nheight)

HWND hwnd      - handle of owner window
HBITMAP hbmp   - handle of bitmap for caret shape
int nwidth     - caret width
int nheight    - caret height
```

The ShowCaret() function shows the caret on the screen at the caret's current position. Once shown, the caret begins flashing automatically.

```
Void ShowCaret(hwnd)

HWND hwnd     - handle of window with caret
```

The GetFocus() function retrieves the handle of the window that currently has the input focus.

```
HWND GetFocus(void)
```

Example Code

To see these API calls in action do the following:

1. Start Visual Basic or start a new project (ALT, F, N).

2. Add two text boxes to Form1.

3. Add the following declarations to the General Declarations section of Form1. Note that you must enter each declaration on a single line even though, for readability, the first declaration is shown on two lines:

```
Declare Sub CreateCaret Lib "user" (ByVal w%, ByVal x%,
      ByVal y%, ByVal z%)
Declare Function showcaret% Lib "user" (ByVal x%)
Declare Function getfocus% Lib "user" ()
```

4. Add the following code to the Command1_Click event:

```
Sub Text1_GotFocus ()
   h% = getfocus%()                ' get the handle to the text box
   Call createcaret(h%, 0, 3, 24)  ' create new caret size
   x% = showcaret%(h%)             ' show the new caret
End Sub
```

5. Run the program.

You will see that while the focus is on Text1 the size of the text caret in the text box appears larger than normal.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Explanation of the Control Box Menu

Article ID: Q94936

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- The Standard and Professional Editions of Microsoft Visual Basic for MS-DOS, version 1.0

SUMMARY

This article describes how to cause the control box menu to drop down and retract as well as how to use the Microsoft Windows versions 3.0 and 3.1 Size and Move options with Visual Basic for MS-DOS.

In Windows, the control box in the upper left corner of a window has a drop-down menu that appears when you click the control box. The drop-down menu contains items such as Move, Size, Minimize, and Maximize. Depending on the position of the mouse, when you hold down the mouse button, the drop-down menu either remains down or retracts to its original position.

MORE INFORMATION

If you click a window's control box, the menu remains down. To create the same effect in Visual Basic for Windows, press ALT+SPACE on the keyboard. In Visual Basic for MS-DOS, press ALT+- (the ALT and minus keys).

If you simply depress the mouse button over the control box instead of clicking the control box, the resulting behavior depends on where you release the mouse button:

- If the button is released over a grayed item, the menu remains dropped to indicate that the selection is not currently active.

- If the button is released outside of the menu or control box, the menu is dismissed and no action is taken

- If the button is released over an enabled menu item, then the menu is dismissed and the appropriate action is invoked.

The purpose of the Size and Move menu commands located in the control box on a window (form) can be confusing. They provide a way to move or size a window by using the keyboard. When you click the control box and choose either Move or Size, the outline of the window is displayed to indicate that you can now move or size the window by using the arrow keys on the keyboard. You must use the arrow keys; if you try to drag the outline with the mouse, you will cancel out the Move or Size operation and resume normal mouse operations. Once you have sized or moved the window to the correct position while it still has an outline, you can lock the window into its new position by pressing the ENTER key.

Additional reference words: B_VBmsdos
KBCategory:
KBSubcategory: PrgCtrlsStd

Validating Text Box Data Causes Extra LostFocus Events

Article ID: Q96846

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Standard and Professional Editions of Microsoft Visual Basic for MS-DOS, version 1.0

SUMMARY

Using the LostFocus event to validate data in a text box can cause excess LostFocus events after the data is determined invalid and focus is set back to the text box. Setting the focus back to the text box, as is the custom when data is invalid, causes a LostFocus event to occur in the control that just received the focus. If that control is also validating data in its LostFocus event and no data (or invalid data) is entered, that control could set the focus back to itself, triggering a LostFocus event in the text box.

MORE INFORMATION

To work around the problem, you need to handle the intended LostFocus event and ignore those generated as a side-effect of handling invalid data. Using a Dim Shared variable in Visual Basic for Windows or Visual Basic for MS-DOS, you can use the LostFocus event to validate text box data. A Dim Shared variable holding either the TabIndex of the next control to be validated or a flag indicating that any control can be validated next, allows you to ignore unintended LostFocus events in other controls.

The example below demonstrates how to use a Dim Shared variable to validate Text box data in the LostFocus event. The example gives step-by-step instructions for Visual Basic for Windows, but you can use the exact same code and controls in Visual Basic for MS-DOS without modification.

Steps to Create Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add two text boxes (Text1 and Text1) to Form1.

3. Add the following code to the General Declarations section of Form1. (In Visual Basic for MS-DOS, add the code to the form-level code.)

```
Dim Shared Focus As Integer

Function IsValid (t1 As TextBox) As Integer
   If t1.Text = "" Then
      IsValid = False
```

```
      Else              ' add other data restrictions here
         IsValid = True
      End If
   End Function
```

4. Add the following code to the Form_Load event procedure of Form1:

```
   Sub Form_Load ()
      Focus = -1
   End Sub
```

5. Add the following code to the Text1_LostFocus event procedure:

```
   Sub Text1_LostFocus ()
      If Not IsValid(Text1) And (Focus = -1 Or Focus = Text1.TabIndex) Then
         MsgBox "Text in Text1 invalid"
         Focus = Text1.TabIndex
         Text1.SetFocus
      Else
          Focus = -1
      End If
   End Sub
```

6. Add the following code to the Text2_LostFocus event procedure:

```
   Sub Text2_LostFocus ()
      If Not IsValid(Text2) And (Focus = -1 Or Focus = Text2.TabIndex) Then
         MsgBox "Text in Text2 invalid"
         Focus = Text2.TabIndex
         Text2.SetFocus
      Else
         Focus = -1
      End If
   End Sub
```

7. From the Run menu, choose Start (ALT, R, S) to run the program. Text boxes Text1 and Text2 both contain the default text, their Name property.

8. Delete the text in Text1.

9. Press the Tab key to move the focus to Text2. The Text1_LostFocus event detects that there is no text in the text box, displays a message box stating that the text in the Text1 box is invalid, and sets the focus back to the Text1 box.

Additional reference words: 1.00 2.00 3.00 b_vbmsdos
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Use the Forms Collection to Unload All MDI Child Forms

Article ID: Q97620

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

You can use Visual Basic code to close all MDI children by using the forms collection. The forms collection contains references to all forms -- the MDI parent form, MDI children forms, and non-MDI forms. To unload or close all MDI forms, loop through the forms collection testing the value of the MDIChild property on each form. If the MDIChild property is true, unload the form.

MORE INFORMATION

Steps to Create Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Change the MDIChild property of Form1 to True.

3. From the File menu, choose New MDI Form (Alt+F+I). This creates the MDIForm1 MDI parent form.

4. From the Window menu, choose Menu Design (ALT+W+M), and create the following menu with two menu items on MDIForm1:

Caption	Name	Indent
File	mFile	no
New	mNew	once
Close All	mCloseAll	once

5. Add the following code to the general declarations section of MDIForm1.

```
Dim ChildCount As Integer
```

6. Add the following code to the mNew event handler.

```
Sub mNew_Click ()
   Dim newWindow As New Form1
   ChildCount = ChildCount + 1
   newWindow.Caption = "Child " & Str$(ChildCount)
   newWindow.Show
End Sub
```

7. Add the following code to the mCloseAll_Click event handler.

```
Sub mCloseAll_Click ()
   i = 1
   Do While i < Forms.Count
      If forms(i).MDIChild Then
         ' *** Do not increment i% since a form was unloaded
         Unload forms(i)
      Else
         ' Form isn't an MDI child so go to the next form
         i = i + 1
     End If
   Loop
   ChildCount = 0
End Sub
```

8. From the Options menu, choose Project. Make MDIForm1 the Start Up Form.

9. From the Run menu, choose Start (ALT, R, S) to run the program.

10. From the File menu on MDIForm1, choose New. Repeat this several times.

11. From the File menu on MDIForm1, choose Close All to unload all the MDI children.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Trap Keystrokes in the Form Instead of Form's Controls

Article ID: Q99688

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.00 and 3.00

SUMMARY

To trap most keystrokes (see NOTE below) at the form level instead of passing them to the form's controls, set the form's KeyPreview property to True and use KeyAscii=0 in the Form_KeyPress event. This prevents keystrokes from going to the form's controls.

NOTE: the technique described in this article will not intercept the ENTER key on command buttons. Command buttons are subclassed Windows push button controls and the ENTER key is an accelerator key that is passed to the superclass; Visual Basic never receives it.

Also note that KeyCode=0 in the Form_KeyDown event won't prevent keystrokes going to the form's controls. This behavior is by design.

MORE INFORMATION

A form's KeyPreview property determines whether form keyboard events are invoked before control keyboard events. The keyboard events are KeyDown, KeyUp, and KeyPress.

You can use the KeyPreview property to create a keyboard-handling procedure for a form. For example, when an application uses function keys, it's likely that you'll want to process the keystrokes at the form level rather than writing code for each control that might receive keystroke events. If a form has no visible and enabled controls, it automatically receives all keyboard events.

To handle keyboard events only at the form level and not allow controls to receive keyboard events, set KeyAscii to 0 in the form's KeyPress event.

Using Form_KeyPress Versus Form_KeyDown to Prevent Text Box Input

This example demonstrates the difference between Form_KeyPress and Form_KeyDown to attempt to trap and prevent all keyboard input for a text box.

1. Start Visual Basic or from the File menu, choose New Project if Visual Basic is already running. Form1 is created by default.

2. Set the KeyPreview property of Form1 to True.

3. Add a text box (Text1) to Form1.

4. Add the following code to the Form_KeyDown event of Form1:

```
   KeyCode = 0
```

5. From the Run menu, choose Start or press the F5 key.

The Text1 box still accepts input, which you may not have expected. This behavior is by design.

To prevent the Text1 box from accepting input, add KeyAscii = 0 to the Form_KeyPress event of Form1. This traps and disables all input to all the controls on the form, as desired. The Form_KeyPress event enables you to handle the keystrokes the way you want.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Non-Menu Keys Disabled When Menu Pulled Down

Article ID: Q99811

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

When a Visual Basic menu is pulled down, all non-menu keystrokes are disabled and keystrokes cannot be detected. This behavior is by design. When a menu is down, all keystrokes just beep or do nothing, except for the keystrokes that control the menu.

You cannot determine which menu item the user chose, until after the user clicks the menu item or presses ENTER. The Click event for the menu item will then give you the chosen menu item.

MORE INFORMATION

Access Keys Give a User Keyboard Access to Menu Items

To give a user keyboard access to a menu item, insert an ampersand (&) immediately in front of a letter in the Caption by using the Menu Design Window. At run time, this letter (called the access key) is underlined. The user can change the focus to a menu or command by pressing ALT plus the letter (access key).

You can use an access key such as ALT+F to give focus quickly to a menu, command, or control by using the keyboard as an alternative to the mouse.

Unlike shortcut keys (such as F10 or CTRL+T, which are also assigned in the Menu Design Window), access keys do not execute commands when pressed, until the ENTER key is pressed. If you open a menu with an access key, then all non-menu keystrokes are disabled until you press a menu-control key such as ENTER, ESC, or ALT.

Step-by-Step Example

To trap all keystrokes in the form instead of the form's controls, you can set the form's KeyPreview property to True. However, because menu controls disable non-menu key presses when the menu is down, you won't be able to preview or trap keys that are pressed when the menu is down, as this example demonstrates:

1. Set the KeyPreview property to True for Form1.

2. Using the Menu Design window, add a Main menu with two submenus:

   Mainmenu
      Submenu1
      Submenu2

3. In the KeyPress event for the form, print trapped key values as follows:

```
Sub Form_KeyPress (KeyAscii As Integer)
   Print Str$(KeyAscii)
End Sub
```

4. Run the program. Press any alphanumeric key, and its ASCII value will be trapped by Form_KeyPress and be printed.

5. Click the Mainmenu menu to drop it down. While the menu is down, non-menu key strokes cause the computer to either beep or do nothing. For example, pressing the x key causes the computer to beep; pressing the F1 key causes the computer to do nothing; but pressing the s key toggles the menu selection between Submenu1 and Submenu2 because both begin with the letter s.

6. Press ESC, ALT, ENTER, or click the menu to make the menu go away and reenable trapping of keystrokes.

While the menu is down, the Form_KeyPress event detects no keys.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

If Invoked by Access Key, Click Event Handled Before LostFocus

Article ID: Q99875

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Standard and Professional Editions of Microsoft Visual Basic for MS-DOS, version 1.0

SUMMARY

Below is an example showing that the Click and LostFocus events occur in different order depending upon whether you cause the click event with the mouse or the keyboard (with an access key). This behavior is by design.

When the focus changes between controls, the Click event can occur before the LostFocus event in some situations. This is mainly because certain events (including GotFocus, LostFocus, and clicking the button with the mouse) are posted to a message queue and other events, such as ALT+V from the keyboard, are issued directly.

To force the code for the LostFocus event to always execute before the Click event code, place a DoEvents statement at the beginning of the Click event code.

MORE INFORMATION

Steps to Reproduce Behavior

1. Draw a text box (Text1) and a command button (Command1) on the default Form1.

2. Set the Caption property of Command1 to &Valid. The &V sets up the ALT+V as a way to execute the Command1 button from the keyboard.

3. Add a Beep statement to the Text1_LostFocus event procedure.

4. Add an End statement to the Command1_Click event procedure.

5. Press F5 to run the program. The focus starts by default on the Text1 box. Click the Command1 button, and notice that the LostFocus event occurs and you hear a Beep before the program ends.

6. Press F5 to run the program again. The focus starts by default on the Text1 box. Type ALT+V to activate the Command1 button. Notice that the program ends with no LostFocus event (no beep).

The difference in behavior is not a bug. It is by design.

In order make the Text1_LostFocus event occur first, place a DoEvents

statement (or function) at the beginning of the Click event code for the Command1 button.

Additional reference words: 1.00 2.00 3.00 B_VBMSDOS
KBCategory:
KBSubcategory: PrgCtrlsStd

Using UP ARROW and DOWN ARROW Keys to Move the Focus

Article ID: Q100413

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Standard and Professional Edition of Microsoft Visual Basic for MS-DOS, version 1.0

SUMMARY

You can trap for the UP ARROW and DOWN ARROW extended keyboard keys in some Visual Basic controls by placing code in the KeyDown event procedure. The code uses KeyCode values to trap the UP ARROW and DOWN ARROW keys. You cannot, however, trap the keys in all Visual Basic controls because some controls already have built-in functionality for the UP ARROW and DOWN ARROW keys, so there is no KeyDown event generated.

MORE INFORMATION

The information in this article is provided to show that it is possible to trap the UP ARROW and DOWN ARROW keys, however Microsoft does not recommend that you implement it because the UP ARROW and DOWN ARROW keys have standard, predefined behavior on some controls. Microsoft recommends that you use the standard method for using the keyboard to move the focus; that is, use the TAB and SHIFT+TAB keys or use the access keys.

Step-by-Step Example for Moving the Focus Using UP ARROW and DOWN ARROW

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running.

2. Add a Picture box and two Text boxes to Form1.

3. In the Picture1_KeyDown event procedure, add this code:

```
Sub Picture1_KeyDown(KeyCode AS INTEGER, Shift AS INTEGER)
   IF KeyCode = 38 Then    '* 38 = up arrow key
      Text2.SetFocus
   Text2.SelStart = 0   '* set the cursor to the start
   END IF

   IF KeyCode = 40 Then    '* 40 = down arrow key
      Text1.SetFocus
   Text1.SelStart = 0   '* set the cursor to the start
   END IF
END SUB
```

4. In the Text1_KeyDown event procedure, add this code:

```
Sub Text1_KeyDown(KeyCode AS INTEGER, Shift AS INTEGER)
   If KeyCode = 38 Then    '* 38 = UP ARROW key
      Picture1.SetFocus
   End If

   If KeyCode = 40 Then    '* 40 = DOWN ARROW key
      Text2.SetFocus
      Text2.SelStart = 0   '* set the cursor to the start
   End If
End Sub
```

5. In the Text2_KeyDown event procedure, add this code:

```
Sub Text2_KeyDown(KeyCode AS INTEGER, Shift AS INTEGER)
   If KeyCode = 38 Then    '* 38 = UP ARROW key
      Text1.SetFocus
      Text1.SelStart = 0   '* set the cursor to the start
   End If

   If KeyCode = 40 Then    '* 40 = DOWN ARROW key
      Picture1.SetFocus
   End If
End Sub
```

6. Choose Start from the Run menu or press F5 to run the example. Press the UP ARROW or DOWN ARROW key to see the focus move to a different control.

If you use the LEFT ARROW or RIGHT ARROW keys, you can scroll in a Text box, but these keys are ignored in the Picture box in this example.

Additional reference words: 2.00 3.00 B_VBmsdos 1.00
KBCategory:
KBSubcategory: PrgCtrlsStd

PRB: Can't Use ActiveForm to Reference Data Control in VB 3.0

Article ID: Q101252

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SYMPTOMS

Using the ActiveForm Property of the Screen control or an MDI Parent form to reference a Data control causes a "Type Mismatch" error in Visual Basic.

CAUSE

This behavior is by design. This is not a bug in Visual Basic. The Visual Basic environment does not know in advance that the Active form will actually contain a Data control, so it generates a "Type mismatch" error.

WORKAROUND

To avoid the error message, use global objects to reference the local controls. The "More Information" section below demonstrates one method for doing this.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Correct Problem

This example shows how to correct the problem. First, create the problem by following the steps listed in "Steps to Reproduce Problem." Then correct the problem with these steps:

1. Add the following code to the Form_Activate Event:

```
Sub Form_Activate ()
   Set CurrentDS = Data1.Recordset
End Sub
```

2. Change two lines of code into comments by adding a single quotation mark to the beginning of the line. Change the Set CurrentDS statement in the Set_CurrentDS Sub in Module1 to a comment, and do the same to the Call Set_CurrentDS statement in the Form_Click event of Form1.

Steps to Reproduce Problem
---

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a data control (Data1) to Form1.

3. Set the DatabaseName Property of Data1 to BIBLIO.MDB.

4. From the File menu, choose New Module (ALT, F, M). Module1 is created.

5. Add the following code to the General section of Module1:

```
Global CurrentDS As DynaSet
```

6. Add the following code to Module1:

```
Sub Set_CurrentDS ()
   Set CurrentDS = Screen.ActiveForm.Data1.Recordset
End Sub
```

7. Add the following code to the Form_Click event procedure of Form1:

```
Sub Form_Click ()
   Call Set_CurrentDS
End Sub
```

8. From the Run menu, choose start (ALT, R, S) or press the F5 key.

A "Type mismatch" error will occur on the Set statement.

Additional reference words: 3.00 errmsg
KBCategory:
KBSubcategory: APrgDataIISAM PrgCtrlsStd

How to Prevent Multiple Instances of a VB Application

Article ID: Q102480

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

This article describes how to avoid loading a second instance of an application when the user already has one instance running. It also sets the focus to the first instance of the Visual Basic .EXE application when you attempt to start a second instance of the same application.

MORE INFORMATION

With Microsoft Windows applications, you usually want only one instance of each application running at the same time. If, for example, you try to start the Windows File Manager when an instance is already running, the first instance of File Manager is activated and its window is opened. By using the following example, you can achieve the same effect with a Visual Basic application.

Step-by-Step Example

1. On the startup form (Form1), put the following code in the Form_Load event:

```
Sub Form_Load ()
  If App.PrevInstance Then
    SaveTitle$ = App.Title
    App.Title = "... duplicate instance."
    Form1.Caption = "... duplicate instance."
    AppActivate SaveTitle$
    SendKeys "% R", True
    End
  End If
End Sub
```

2. From the File menu, choose Make EXE File.

3. Exit Visual Basic for Windows.

4. Start your program through Program Manager or double-click the .EXE file name under Windows File Manager.

5. Minimize the program you started in step 4.

6. Attempt to start another instance of the program by repeating step 4.

When you try to launch a second instance of the program, the Visual Basic application executes the following logic:

1. It checks the App object property PrevInstance to see if there is a previous instance of an application with the same App.Title property.

2. If there is, the new instance of the program saves its App.Title property to a local string to be used to activate the first instance of the same name.

3. Then it changes its own name to avoid an ambiguous reference in the AppActivate call.

4. Next, it performs AppActivate which causes the first instance of the application to be the current window.

5. Now that the first instance of the application has the focus, the second instance uses SendKeys to send the equivalent keystrokes to restore the first instance's window state.

6. Finally, the second instance of the application Ends itself leaving the first instance with the focus.

Additional reference words: 2.00 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsStd

How to Move Controls at Run Time By Using Drag and Drop

Article ID: Q103062

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

You can move controls at run time by using manual dragging with the Drag method. Automatic dragging (DragMode = 1) does not work well for repositioning controls at run time.

MORE INFORMATION

The key points to remember when using drag and drop to move controls at run time are:

- In the MouseDown event, save the X and Y parameters. This position is relative to the upper left corner of the control to drag. Note that the MouseDown event only occurs when DragMode is set to Manual (0).

- In the DragDrop event, move the control to the position of the mouse pointer adjusted by the position saved in MouseDown.

Example Program

The following example program demonstrates how to reposition a picture box at run time. Place the pieces of the program in the appropriate event procedures.

```
Dim Save_X As Single
Dim Save_Y As Single

' Enter the following Sub as one, single line:
Sub Picture1_MouseDown (Button As Integer, Shift As Integer, X As
   Single, Y As Single)
   Save_X = X       ' save mouse position (relative to this control)
   Save_Y = Y
   Picture1.Drag 1  ' begin dragging
End Sub

' Enter the following Sub as one, single line:
Sub Picture1_MouseUp (Button As Integer, Shift As Integer, X As
   Single, Y As Single)
   Picture1.Drag 2  ' end dragging, do DragDrop event
End Sub

Sub Form_DragDrop (Source As Control, X As Single, Y As Single)
   ' Move the control to the position of the mouse pointer.
```

```
   ' Adjust it by the distance the mouse pointer to the upper
   ' left corner of the control.
   Source.Move X - Save_X, Y - Save_Y
End Sub

Sub Picture1_DragDrop (Source As Control, X As Single, Y As Single)
   ' This handles the case when the control is dropped on itself
   ' as would happen if it was only moved a small amount.
   ' This is similar to Form_DragDrop except that the X and Y
   ' parameters are relative to this control, not the form.
   Source.Move Picture1.Left + X - Save_X, Picture1.Top + Y - Save_Y
End Sub
```

Additional reference words: 1.00 2.00 3.00 runtime run-time
KBCategory:
KBSubcategory: PrgCtrlsStd

PRB: Invalid picture Error When Try to Bind Picture Control

Article ID: Q103115

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

If you try to bind a picture control to a Microsoft Access database field that contains an OLE object such as a PaintBrush bitmap, you will correctly receive the error "Invalid picture."

CAUSE

This error occurs because the picture control can only bind to a bitmap, metafile, or icon stored in the database field -- not to an OLE object.

RESOLUTION

Using the method described below, you can simulate the binding of a picture control to a PaintBrush OLE (or bitmap) object.

MORE INFORMATION

If you use Microsoft Access to store a PaintBrush picture in an OLE field, there is no way to bind any control provided with Visual Basic version 3.0 to the OLE field. Ideally you could bind the MSOLE2 control to the data control, but no features were added to the MSOLE2 control to allow you to bind to a database field.

From Visual Basic, you can use a bound picture control to store and retrieve bitmaps, metafiles, and icons directly in a long binary or OLE database field. However, Microsoft Access will not be able to display the bitmap, metafile, or icon that you've stored.

Step-by-Step Example

The following example demonstrate how you can create an application that retrieves and displays a bitmap from an OLE field containing a PaintBrush object. To get it to work, you need to have the NWIND.MDB sample database provided with Microsoft Access.

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose New Module (ALT, F, M). Module1 is created.

3. Add a picture control (Picture1) to Form1.

4. Add a data control (Data1) to Form1.

5. Set the Data1.Databasename property to NWIND.MDB and include the full path to this file. This file is a sample database that ships with Microsoft Access versions 1.0 and 1.1. Look for it in your Microsoft Access directory -- for example, C:\ACCESS.

6. Set the Data1.RecordSource property to Employees.

7. Add the following code to the Data1_Reposition event procedure in Form1:

```
'*********************************************************************
'*  Title
'*       Data1_Reposition ()
'*
'*  Description
'*       Each time the data control is being repositioned to a new
'*       record, the bitmap contained in the "Photo" is displayed
'*       in Picture1. Therefore, it simulates binding the picture
'*       control to an OLE field containing a Microsoft Paint Brush
'*       picture object.
'*
'*       The code requires a field named Photo, and it requires that
'*       the embedded OLE object be a Microsoft Paint Brush picture.
'*********************************************************************
Sub Data1_Reposition ()

   Screen.MousePointer = 11

   'Make sure this is the current record:
   If Not (Data1.Recordset.EOF And Data1.Recordset.BOF) Then

      'Change Photo to the name of the OLE field
      'for the record set you are using:
      DisplayOleBitmap Picture1, Data1.Recordset("Photo")

   End If

   Screen.MousePointer = 0

End Sub
```

8. Add the following code to Module1:

```
'*********************************************************************
'* OLEACCES.BAS
'*
'* general-declarations section
'*********************************************************************
Option Explicit

Global Const LENGTH_FOR_SIZE = 4
Global Const OBJECT_SIGNATURE = &H1C15
Global Const OBJECT_HEADER_SIZE = 20
Global Const CHECKSUM_SIGNATURE = &HFE05AD00
Global Const CHECKSUM_STRING_SIZE = 4
```

```
'PT : Window sizing information for object
'     Used in OBJECTHEADER type
Type PT
   Width As Integer
   Height As Integer
End Type

'OBJECTHEADER : Contains relevant information about object
'
Type OBJECTHEADER
   Signature As Integer         'Type signature (0x1c15)
   HeaderSize As Integer        'Size of header (sizeof(struct
                                'OBJECTHEADER) + cchName +
                                'cchClass)
   ObjectType As Long           'OLE Object type code (OT_STATIC,
                                'OT_LINKED, OT_EMBEDDED)
   NameLen As Integer           'Count of characters in object
                                'name (CchSz(szName) + 1)
   ClassLen As Integer          'Count of characters in class
                                'name (CchSz(szClass) + 1)
   NameOffset As Integer        'Offset of object name in
                                'structure (sizeof(OBJECTHEADER))
   ClassOffset As Integer       'Offset of class name in
                                'structure (ibName + cchName)
   ObjectSize As PT             'Original size of object (see
                                'code below for value)
   OleInfo As String * 256
End Type

Type OLEHEADER
   OleVersion As Long
   Format As Long
   OleInfo As String * 512
End Type

'Enter the following Declare statement as one, single line:
Declare Function GetTempFileName Lib "Kernel" (ByVal cDriveLetter
   As Integer, ByVal lpPrefixString As String, ByVal wUnique As
   Integer, ByVal lpTempFileName As String) As Integer

'Enter the following Declare statement as one, single line:
Declare Sub hmemcpy Lib "Kernel" (dest As Any, source As Any,
   ByVal bytes As Long)

'*********************************************************************
'* Title
'*      DisplayOleBitmap
'*
'* Description
'*      Causes the OLE bitmap in the given data field to be
'*      copied to a temporary file. The bitmap is then
'*      displayed in the given picture.
'*
'* Parameters
'*      ctlPict         Picture control in which to display the
'*                      bitmap image
```

```
'*      OleField        Database field containing the OLE
'*                      embedded Microsoft Paint Brush bitmap
'*********************************************************************
Sub DisplayOleBitmap (ctlPict As Control, OleField As Field)

   Const DT_LONGBINARY = 11

   Dim r As Integer
   Dim Handle As Integer
   Dim OleFileName As String

   If OleField.Type = DT_LONGBINARY Then

      OleFileName = CopyOleBitmapToFile(OleField)

      If OleFileName <> "" Then

         'Display the bitmap:
         ctlPict.Picture = LoadPicture(OleFileName)

         'Delete the temporary file:
         Kill OleFileName

      End If

   End If

End Sub

'*********************************************************************
'* Title
'*      CopyOleBitmapToFile
'*
'* Description
'*      Copies the bitmap contained in a OLE field to a file.
'*********************************************************************
Function CopyOleBitmapToFile (OleField As Field) As String

   Const BUFFER_SIZE = 8192

   Dim tempFileName As String
   Dim Handle As Integer
   Dim Buffer As String

   Dim BytesNeeded As Long

   Dim Buffers As Long
   Dim Remainder As Long

   Dim OLEHEADER As OBJECTHEADER
   Dim sOleHeader As String

   Dim ObjectOffset As Long
   Dim BitmapOffset As Long
   Dim BitmapHeaderOffset As Integer

   Dim r As Integer
```

```
Dim i As Long

tempFileName = ""
If OleField.FieldSize() > OBJECT_HEADER_SIZE Then

   'Get the Microsoft Access OLE header:
   sOleHeader = OleField.GetChunk(0, OBJECT_HEADER_SIZE)
   hmemcpy OLEHEADER, ByVal sOleHeader, OBJECT_HEADER_SIZE

   'Calculate the offset where the OLE object starts:
   ObjectOffset = OLEHEADER.HeaderSize + 1

   'Get enough bytes after the OLE header so that we get the
   'bitmap header
   Buffer = OleField.GetChunk(ObjectOffset, 512)

   'Make sure the class of the object is a Paint Brush object
   If Mid(Buffer, 12, 6) = "PBrush" Then

      BitmapHeaderOffset = InStr(Buffer, "BM")

      If BitmapHeaderOffset > 0 Then

         'Calculate the beginning of the bitmap:
         BitmapOffset = ObjectOffset + BitmapHeaderOffset -1

         'Calculate the size of the bitmap:
         'Enter the following BytesNeeded statement as a single line:
         BytesNeeded = OleField.FieldSize() - OBJECT_HEADER_SIZE -
            BitmapHeaderOffset - CHECKSUM_STRING_SIZE + 1

         'Calculate the number of buffers needed to copy
         'the OLE object based on the bitmap size:
         Buffers = BytesNeeded \ BUFFER_SIZE
         Remainder = BytesNeeded Mod BUFFER_SIZE

         'Get a unique, temp filename:
         tempFileName = Space(255)
         r = GetTempFileName(0, "", -1, tempFileName)

         'Copy the bitmap to the temporary file chunk by chunk:
         Handle = FreeFile
         Open tempFileName For Binary As #Handle

         For i = 0 To Buffers - 1
            'Enter the following Buffer statement as a single line:
            Buffer = OleField.GetChunk(BitmapOffset + i *
               BUFFER_SIZE, BUFFER_SIZE)
            Put #Handle, , Buffer
         Next

         'Copy the remaining chunk of the bitmap to the file:
         'Enter the following Buffer statement as a single line:
         Buffer = OleField.GetChunk(BitmapOffset + Buffers *
            BUFFER_SIZE, Remainder)
         Put #Handle, , Buffer
```

```
                Close #Handle

            End If

         End If

      End If

      CopyOleBitmapToFile = Trim(tempFileName)

   End Function
```

9. From the Run menu, choose Start (ALT, R, S) or press the F5 key to run the program.

You should see the photo of the first employee displayed in the picture box. By clicking the directional arrows on the data control, you can view the other employee photos.

Additional reference words: 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

PRB: Out of Stack Space When One Modal Form Shows Another

Article ID: Q103461

The information in this article applies to:

- The Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 1.0, 2.0, and 3.0
- The Standard and Professional Editions of Microsoft Visual Basic for MS-DOS, version 1.0

SYMPTOMS

Any of the following error messages can occur when two or more forms in a program repeatedly show each other modally (SHOW 1).

- Out of stack space.
- Out of memory.
- Out of overlay stack space.

CAUSE

This can happen even if you unload the form, which in turn shows the next form. A form is not actually unloaded by the Unload statement until all its event procedures return (End Sub or Exit Sub). Showing a form modally suspends execution and, like a procedure call, maintains information on the stack. Further explanation is given in the MORE INFORMATION section below.

WORKAROUNDS

- Show the forms non-modally (SHOW 0). It is acceptable practice to have forms show each other non-modally.

- Do not have modal forms call each other continually. Instead, have an initial form call all the other forms. Think of this initial form (probably your startup form) as your foundation with all other forms called from the foundation.

MORE INFORMATION

The following example gives an Out of stack space error message. Remove the apostrophe from (uncomment) the MsgBox statements in Visual Basic for MS-DOS to see the amount of remaining stack space.

```
' Form1:
Sub Form_Click ()
   ' MsgBox STR$(FRE(-2))
   Unload Form1
   Form2.Show 1
End Sub
```

```
' Form2:
Sub Form_Click ()
   ' MsgBox STR$(FRE(-2))
   Unload Form2
   Form1.Show 1
End Sub
```

When a function or a subroutine is called, the variables in the calling procedure get pushed onto the stack. This way these values are preserved. When the function or subroutine ends on an End Function, End Sub, or Exit Sub statement, these variables get popped off the stack, and program execution returns to the statement that follows the call. Only then are the variables once again usable.

If a subroutine or function calls another function, program execution is halted within that subroutine or function, and the stack used is not cleared up until an End Function, End Sub, or Exit Sub is encountered. This is why you should not have two subroutines repeatedly call each other with no stopping condition.

The behavior of event procedures within forms is similar to subroutines in that when a form is shown, information is pushed onto the stack, and when forms are unloaded, information is popped off the stack. Modal forms halt program execution of all other events. However, a form is not actually unloaded by the Unload statement until all of its event procedures return with an End Sub or Exit Sub. When a modal form displays a second modal form, the second modal form puts a hold on program execution, so the first modal form cannot proceed to the rest of its code, thus making it impossible to ever reach the End Sub or Exit Sub statement. This is why you should not have modal forms show each other repeatedly.

Additional reference words: 1.00 2.00 3.00 B_VBMSDOS
KBCategory:
KBSubcategory: PrgCtrlsStd

How to Program Two List Boxes to Scroll Together

Article ID: Q103809

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 2.0 and 3.0

SUMMARY

You can give two list boxes the ability to scroll together in unison. In other words, you can program your Visual Basic application so that when the user scrolls the List1 box, the contents of the List2 box will scroll in the same direction automatically -- without using the List2 scroll bar.

MORE INFORMATION

The example below uses two list boxes, side by side, to demonstrate this technique to simulate the appearance of two list boxes scrolling together.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add two list boxes, one timer control, and one command button to Form1. For the best visual effect, place the list boxes side by side with the List1 box on the left.

3. Add the following code to the (general) (declarations) section of Form1:

```
DefInt A-Z
```

4. Add the following code to the Form Load event procedure of Form1:

```
Sub Form_Load ()
   'Initialize two list boxes with the alphabet
   For i = 1 To 26
      list1.AddItem Chr$(i + 64)
   Next i
   For i = 1 To 26
      list2.AddItem Chr$(i + 64)
   Next i
   Timer1.Interval = 1
   Timer1.Enabled = True
End Sub
```

5. Add the following code to the Command1 Click event procedure of Form1:

```
Sub Command1_Click ()
   End
```

```
End Sub
```

6. Add the following code to the Timer1_Timer event procedure of Form1:

```
Sub Timer1_Timer ()
   Static PrevTI_List1
   'Get the index for the first item in the visible list
   TopIndex_List1 = list1.TopIndex
   'See if the top index has changed
   If TopIndex_List1 <> PrevTI_List1 Then
      'Set the top index of List2 equal to List1 so that the list boxes
      'scroll to the same relative position
      list2.TopIndex = TopIndex_List1
      'Keep track of the current top index
      PrevTI_List1 = TopIndex_List1
   End If
   'Select the item in the same relative position in both list boxes
   If list1.ListIndex <> list2.ListIndex Then
      list2.ListIndex = list1.ListIndex
   End If
End Sub
```

7. Press the F5 key to run the program. Select a letter in the List1 box. Then try the scroll bar of the List1 box. You should see the same letter highlighted in the List2 box when you select a letter from the List1 box. Then when you try the scroll bar of the List1 box, you should see the List2 box scroll in unison with the List1 box.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

Adjusting VB FontSize at Run Time for Different Video Drivers

Article ID: Q106164

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

Text on controls and labels may have a different size or appearance at run time depending on the video device and driver. For example, many video drivers have a large-fonts option and a small-fonts option. This article describes how to automatically adjust text at run time to fit the design-time label size, independent of the video driver.

MORE INFORMATION

Predicting how fonts will appear on different display devices is not easy. However, you can calibrate the appropriate FontSize to use at run time by using the following example. This example adjusts the form's FontSize so that a particular label caption fits best inside its Width. The label's Width can be set at design time to adjust how big the fonts ought to appear at run time. This example assumes a True Type FontName setting, which can be set to almost any size needed.

Step-by-Step Example

1. Start Visual Basic, and begin a new project.

2. Draw a label on a form.

3. Add the following code to the form's Load event:

```
Sub Form_Load ()

    ' This procedure determines the appropriate FontSize to display
    ' text in a label that was sized at design time.

    Const SMALLESTFONTSIZE = 3  'Enter smallest available font size

    ' 1. Assign the FontName properties of the label to the Form:
    '    use Me (a reserved word describing the current form) instead of
    '    the name of the label control because you are comparing sizes
    '    using the TextWidth property. TextWidth returns the width as if
    '    it was printed directly on an object (the form). The TextWidth
    '    property does not apply to controls.
    Me.FontName = Label1.FontName
    Me.FontSize = Label1.FontSize

    ' 2. Increase FontSize until Caption is too wide too fit in label:
```

```
      i = Me.FontSize
      Do Until Me.TextWidth(Label1.Caption) >= Label1.Width
         i = i + 1
         Me.FontSize = i
         'Debug.Print Me.FontSize
      Loop

      ' 3. Decrease FontSize until Caption fits width-wise in label.
      '    NOTE: If the fontsize becomes less than SMALLESTFONTSIZE below,
      '    the Caption is too big for the current label size, even with
      '    the smallest available fontsize.

      i = Me.FontSize
      Do Until Me.TextWidth(Label1.Caption) <= Label1.Width
         i = i - 1
         If i < SMALLESTFONTSIZE Then
            MsgBox "Caption width truncated to fit label - smallest font."
            Exit Sub
         End If
         Me.FontSize = i
         'Debug.Print "width:" & i; Me.FontSize
      Loop

      ' 4. Decrease FontSize until Caption fits height-wise in label:
      i = Me.FontSize
      Do Until Me.TextHeight(Label1.Caption) <= Label1.Height
         i = i - 1
         If i < SMALLESTFONTSIZE Then
            MsgBox "Caption height truncated to fit label - smallest font."
            Exit Sub
         End If
         Me.FontSize = i
         'Debug.Print "height" & i; Me.FontSize
      Loop

      ' 5. Assign Font properties from the Form back to the label:
      Label1.FontName = Me.FontName
      Label1.FontSize = Me.FontSize

   End Sub
```

If you need several different font sizes, set up a label to calibrate each font size needed. The labels used to make this adjustment do not have to be visible. Optionally, you can set the Visible property to False.

You can also have the program size label controls depending on the Screen.Height and Screen.Width properties at run time. Once you determine the correct size of the label, size the text inside the label.

Example of How Fonts May Differ on Different Hardware

You can call Windows API functions to obtain the enumerated FontSize list. This is useful to know for fixed, non-TrueType fonts. Visual Basic also offers font properties (Fonts, FontName, FontSize, and FontCount) to determine font information.

The enumerated FontSize list for non-TrueType fonts may vary from one screen resolution to another. This can happen because the number of logical pixels per inch can vary between resolutions. This means that the number of points per pixel can also vary. The point size of a font is adjusted to the nearest pixel.

The point size on a screen is based on logical inches. Logical inches are somewhat arbitrary because Windows has no way of really knowing how big a pixel is on your screen. For example, you could be hooked up to a projection TV or a tiny monitor. Usually the logical inch is overly large; tiny text is often difficult to read on a video display.

Because the point size is based on the logical pixels per inch of a device, not all point sizes can be represented. For example, on a standard VGA, Windows will tell you that the device has 96 pixels per logical inch (according to the Windows API GetDeviceCaps(hDC, LOGPIXELSY) function). A 96-pixel tall glyph is 72 points because each point is about 1/72 of an inch. This means each pixel is 72/96 point or 0.75 point per pixel. The system could theoretically represent fonts of the following heights:

```
1 pixel = 0.75 point
2 pixels = 1.50 point
3 pixels = 2.25 point
4 pixels = 3.00 point
```

Rounding errors are unavoidable in this scheme. Even if the device displayed exactly 96 dots per inch (DPI), it could not represent a font that was exactly 2 points. The closest it could come would be 2.25 points. Usually, this small difference is not noticeable. However, if two screen drivers have different logical pixels-per-inch, you might see different point sizes in the enumerated list in Windows.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsStd

PRB: PrintForm Blank Page or GPF Due to Video Color Depth >256

Article ID: Q108470

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

The PrintForm method in Visual Basic is designed to send a bit-for-bit image of a non-MDI form to the printer.

Using the PrintForm method to print a form to a Hewlett-Packard (HP) LaserJet Series II can give a general protection (GP) fault from Windows due to the cause described below. An HP LaserJet Series IV can print a blank page due to the cause described below. Similar behavior may occur with other printers.

The problem can occur even on a simple form, such as a form with a PrintForm method in the click event of a single command button.

CAUSE

This memory problem can be caused by using a video driver with a color depth setting greater than 256 colors, or greater than 8 bits per pixel.

For example, a color depth setting of 65,000 colors, or 16 bits per pixel, can cause this memory management problem. A color depth setting of 24 bits per pixel for the video driver can also cause this memory problem.

WORKAROUND

To eliminate the GP fault or blank page problem, reset the color depth of the video driver on your system to 256 colors or less, or 8 bits per pixel or less.

STATUS

This behavior is caused by memory management limitations between the video driver and Windows.

Additional reference words: 3.00 lock freeze GP Fault hang fail IIp laser
KBCategory: Prg
KBSubcategory: PrgCtrlsStd

PRB: Problem Changing Control's Picture to (None) in VB 3.0

Article ID: Q108602

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

You can attach a graphic to a form or picture control using the Picture property in Visual Basic. By default, Visual Basic gives a value of (none) for the Picture property in the Properties window. After you attach a graphic to a form or picture control, the Picture property has a value of (bitmap), (icon), or (metafile).

However, Visual Basic can prevent your attempts to detach the graphic. Highlighting the (bitmap), (icon), or (metafile) value and pressing DEL can fail to change the value to (none).

NOTE: The three dots on the right of the Settings box for the Picture property do not pop up a choice for (none). Visual Basic requires you to use the DEL key to change the Picture property value to (none).

CAUSE

Visual Basic requires you to click the Picture property in the Properties list immediately before highlighting the (bitmap), (icon), or (metafile) value for deletion. Visual Basic requires that click even if the Picture property has the focus from a previous operation.

WORKAROUND

To work around this behavior, click the Picture property in the Properties list before highlighting the (bitmap), (icon), or (metafile) value. Or, change the focus to any other property on the Properties window, then reselect the Picture property. The DEL key then correctly deletes the (bitmap), (icon), or (metafile) value and changes the value to (none). The graphic correctly disappears from the form or picture control.

STATUS

This behavior is under review and will be considered for correction in a future release.

MORE INFORMATION

Steps to Reproduce the Behavior

The following steps set the Picture property to (bitmap), (icon), or (metafile):

1. Start a new project in Visual Basic. Form1 is created by default.

2. Press the F4 key. Select the Picture property from the list in the Properties window.

3. Click the three dots on the right of the Settings box. Visual Basic displays a dialog box from which you select a picture file. The graphic from the picture file displays on the form.

4. Click the form to see the graphic. This changes the focus to the form.

5. Press the F4 key to give focus back to the Properties window.

   NOTE: The Picture property is still highlighted on the list in the Properties window. To duplicate the behavior, don't click the Picture property until the workaround is explained further below.

6. In the Settings box in the Properties window, highlight or double-click the word (bitmap), (icon), or (metafile). Press the DEL key. This fails to change the value to (none).

To work around this behavior, click the Picture property in the Properties list before highlighting the (bitmap), (icon), or (metafile) value. Or, change the focus to any other property on the Properties window, then reselect the Picture property. Then highlight the (bitmap), (icon), or (metafile) value. Press the DEL key to delete the value. This successfully changes the value to (none) and removes the graphic from the form.

REFERENCES
==========

- "Microsoft Visual Basic Version 3.0: Programmer's Guide", Page 324.

Additional reference words: 3.00 bitmap bitmaps .BMP .WMF .ICO .DIB
KBCategory: Prg
KBSubcategory: PrgCtrlsStd

Category Keywords for All Visual Basic KB Articles

Article ID: Q108753

The information in this article applies to:

- Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Each article in the Visual Basic for Windows collection contains at least one keyword (called a KBSubcategory keyword) that places the article in an appropriate category. This article lists all the KBSubcategory keywords.

MORE INFORMATION

Category & Subcategory Description	KBSubcategory Keyword
Setup / Installation (Setins)	Setins
Environment-specific Issues (Envt)	
VB Design Environment	EnvtDes
Run-Time Environment	EnvtRun
Programming (Prg)	
Visual Basic Forms and Controls	
Standard Controls / Forms	PrgCtrlsStd
Custom Controls	PrgCtrlsCus
Third-Party Controls	PrgCtrlsThird
Optimization	
Memory Management	PrgOptMemMgt
General Optimization Tips	PrgOptTips
General VB Programming	PrgOther
Advanced programming (APrg)	
Network	APrgNet
Windows Programming (APIs / DLLs)	
Printing	APrgPrint
Graphics	APrgGrap
Windowing	APrgWindow
INI Files	APrgINI
Other API / DLL Programming	APrgOther
Data Access	
ODBC	APrgDataODBC
IISAM	APrgDataIISAM
Access	APrgDataAcc
General Database Programming	APrgDataOther
3rd Party DLL's	APrgThirdDLL

Inter-Application Programmability (IAP)

OLE	IAPOLE
DDE	IAPDDE
3rd Party Interoperability	IAPThird

Tools (Tls)

Setup Toolkit / Wizard	TlsSetWiz
Control Development Kit (CDK)	TlsCDK
Help Compiler (HC)	TlsHC

References (Refs)

Documentation / Help File Fixes	RefsDoc
Product Information	RefsProd
Third-Party Information	RefsThird
PSS-Only Information	RefsPSS

Using Keywords to Query the KB

At Microsoft, we use the subcategory keywords to organize the articles for Help files and for the FastTips Catalog. You can use them to query the Microsoft Knowledge Base for Visual Basic articles that apply to that category or subcategory. For example, you can find all the general database programming articles by querying on the following words in the Microsoft Knowledge Base:

```
visual and basic and APrgDataOther
```

Use the asterisk (*) wildcard to find articles that fall into the general categories or into an intermediate subcategory. The first element in each keyword is the category. For example, to find all the articles that apply to Visual Basic Forms and Controls regardless of whether they are standard, custom, or third-party controls, use the following words to query the Microsoft Knowledge Base:

```
visual and basic and PrgCtrls*
```

To find all advanced programming articles, query on these words:

```
visual and basic and APrg*
```

Add KBSubcategory Keyword to Each Article

When contributing an article to the Visual Basic Knowledge Base, add the appropriate KBSubcategory keyword to the bottom of the article on the KBSubcategory line. Each article in the Visual Basic for Windows collection contains the following section at the bottom of the article:

```
Additional reference words:
KBCategory:
KBSubcategory: <keyword>
```

An article usually has only one subcategory keyword, but it may have more.

If you are interested in contributing, please obtain the guidelines by

querying on the following words in the Microsoft Knowledge Base:

   visual and basic and kbguide and kbartwrite

Additional reference words: 3.00 dskbguide subcatkey
KBCategory:
KBSubcategory: RefsPSS

How to Display Multiple Foreground Text Colors in VB List Box

Article ID: Q108811

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

To set the foreground and background color of a list box control, set the ForeColor and BackColor properties at either design time or run time. All text in a list box uses the color set by the ForeColor property. The text is printed against a background color set by the BackColor property.

Visual Basic doesn't directly support the display of text of different colors simultaneously in the list box. This article describes how to display words of different colors simultaneously in a list box by using an indirect technique.

MORE INFORMATION

You can display lines or words of different colors simultaneously in a list box by using one of the following indirect techniques.

1. Simulate the list box with a picture box control. You can store the desired text strings in an array of strings, and use the Print method to write the array entries into the picture box with different ForeColor properties. For example:

   ```
   picture1.BackColor = QBColor(14)   ' 14=Light yellow
   picture1.ForeColor = QBColor(4)    '  4=Red
   picture1.Print "in living red"
   picture1.ForeColor = QBColor(2)    '  2=Green
   picture1.Print "in living green"
   ```

   You can also add a vertical scroll bar next to the picture box. When the scroll bar is scrolled, your code needs to redraw the picture box. The ForeColor property of the picture box controls the current color used by the Print method. The picture box will not let you highlight text. NOTE: The BackColor method erases any pre-existing text on the picture control.

2. For coloring list box entries with multiple foreground colors, the Desaware company provides two solutions:

   a. The MLIST2.VBX control file is included with the Custom Control Factory product from Desaware. MLIST2.VBX allows each line in a list box to be colored independently. All words on the same line must be the same color. MLIST2.VBX comes with full source code.

   b. A more flexible and advanced solution is to turn Visual Basic's list

box into an owner-draw list box. Desaware says that you can make true owner-draw list boxes with their SpyWorks-VB product. SpyWorks-VB allows you to color each entry of the list box with the full power of the Windows API drawing functions. SpyWorks-VB comes with sample source code for an owner-draw list box and command button, along with explanations of how to turn the standard controls into owner-draw controls. See the section on owner-draw controls further below.

How to Contact Desaware

NOTE: Desaware products are manufactured independent of Microsoft. Microsoft makes no warranty, implied or otherwise, regarding these products' performance or reliability.

Desaware
5 Town & Country Village #790
San Jose, CA 95128
Contact: Gabriel Appleman (213) 943-3305
          Dan Appleman (408) 377-4770
Fax: (408) 371-3530

The Desaware company offers the following products:

1. Custom Control Factory -- an interactive development tool for creating custom controls including Animated Pushbuttons, Multistate Buttons, enhanced buttons, check box, and option button controls for Windows applications.

2. CCF-Cursors -- provides you with complete control over cursors (mouse pointers) in Visual Basic applications. Create your own cursors or convert icons to cursors, and much more. Includes over 50 cursors.

3. SpyWorks-VB -- an advanced development tool for use with Visual Basic.

Owner-Draw Controls in Windows

The owner-draw list capability is appropriate for advanced programmers for Microsoft Windows. You will need a good reference for the Windows API to learn the required drawing functions.

Owner-draw controls were introduced in Windows version 3.0. Because your application does all the drawing of the contents of the controls, you can customize them any way you like. Owner-draw controls are similar to predefined controls in that Windows will handle the control's functionality and mouse and keyboard input processing. However, you are responsible for drawing the owner-draw control in its normal, selected, and focus states.

You can create owner-draw controls from the menu, button, and list-box classes. You can create owner-draw combo boxes, but they must have the CBS_DROPDOWNLIST style, which equates to a static text item and a list box. The elements of an owner-draw control can be composed of strings, bitmaps, lines, rectangles, and other drawing functions in any combination, in your choice of colors.

REFERENCES
==========

 - "Microsoft Windows Programmer's Reference"

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsCus PrgCtrlsStd PrgCtrlsThird

BackColor Erases Existing Graphics on Form or Picture Control

Article ID: Q108812

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Setting the BackColor property on forms or picture boxes at run time erases all previous printed graphics and output, including a persistent bitmap. This behavior is by design. Setting the ForeColor property does not affect the graphics or print output that are already drawn.

Setting the BackColor property on other controls, such as list boxes, text boxes, or combo boxes, does not erase previously added items.

MORE INFORMATION

NOTE: A persistent bitmap is a bitmap, accessed with the Image property, that stores output from graphics methods in memory. For more information, see the following items in the Visual Basic Help menu:

- AutoRedraw property
- persistent bitmap

Steps to Reproduce Behavior

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a list box and a picture box to Form1.

3. Double-click the form to open the code window. Add the following code to the Form Load event:

```
Sub Form_Load ()
   form1.Show 'In Load event, must show form before Print works

   picture1.ForeColor = QBColor(1)
   picture1.BackColor = QBColor(11)
   picture1.Print "color1"
   form1.ForeColor = QBColor(1)
   form1.BackColor = QBColor(12)
   form1.Print "color1"
   list1.ForeColor = QBColor(1)
   list1.BackColor = QBColor(11)
   list1.AddItem "color1"

   MsgBox "click to see next color"
   picture1.ForeColor = QBColor(2)
```

```
    picture1.BackColor = QBColor(10)  'This BackColor erases picture1
    picture1.Print "color1"
    form1.ForeColor = QBColor(2)
    form1.BackColor = QBColor(13)  'This BackColor erases form1
    form1.Print "color1"
    list1.ForeColor = QBColor(2)
    list1.BackColor = QBColor(10)  'This BackColor doesn't erase List1
    list1.AddItem "color2"

    MsgBox "click to see next color"
    picture1.ForeColor = QBColor(4)
    picture1.BackColor = QBColor(14)  'This BackColor erases picture1
    picture1.Print "color1"
    form1.ForeColor = QBColor(4)
    form1.BackColor = QBColor(8)  'This BackColor erases form1
    form1.Print "color1"
    list1.ForeColor = QBColor(4)
    list1.BackColor = QBColor(14)  'This BackColor doesn't erase List1
    list1.AddItem "color3"

End Sub
```

4. Start the program or press the F5 key. Click OK to see the next color. To end the program, close the form.

Here are the results:

- Changing the Form1.BackColor property erases the text that you previously printed on Form1.
- Changing the Picture1.BackColor property erases the text that you previously printed on Picture1.
- Changing the List1.BackColor property does not erase the text that you previously added to the List1 list box.

All of this behavior is by design.

Additional reference words: 2.00 3.00 delete remove blank out
KBCategory: Prg
KBSubcategory: PrgCtrlsStd

PRB: MDI Child Form Painted Twice When Moved Before Loaded

Article ID: Q109801

The information in this article applies to:

- The Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

If you Load an MDI child form and then change the position or size of the MDI child form, it gets painted twice -- once in a default starting position and then in its final position.

CAUSE

When you access any properties of a form that is not currently loaded, the form is loaded immediately. Note that the Move method simply sets the Left, Top, Width, and Height form properties. Since MDI child forms cannot have their Visible property False, they cannot be loaded without being Visible. Therefore, when you try to set the position or size properties of an MDI child form before showing it, the MDI child form appears in a default position before your new settings take effect.

RESOLUTION

Initialize the position of MDI child forms from within their own Form_Load event handler. The Form_Load event handler is executed before the form actually becomes visible.

MORE INFORMATION

Steps to Reproduce Behavior

1. Start a new project. Form1 is created by default.

2. Set the Form1.MDIChild property to True.

3. From the File menu, choose New MDI Form.

4. From the Options menu, choose Project, and set the Start Up Form to MDIForm1.

5. Add the following code to the MDIForm1 Load event procedure:

```
Sub MDIForm_Load ()
   Form1.Move 0, 0, MDIForm1.ScaleWidth, MDIForm1.ScaleHeight
End Sub
```

6. Run the program. Form1 appears briefly in the upper-left region of MDIForm1, then resizes to fill MDIForm1.

7. To fix this problem, remove the code from the MDIForm1 Load event and place the following code in the MDIForm1 and Form1 event procedures:

```
' MDIForm1:
Sub MDIForm_Load ()
   Form1.Show
End Sub

' Form1:
Sub Form_Load ()
   Form1.Move 0, 0, MDIForm1.ScaleWidth, MDIForm1.ScaleHeight
End Sub
```

8. Run the program. Now Form1 appears once, in its final position and size.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsStd

How to Distinguish a DblClick from a Click Event

Article ID: Q109865

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

Usually when you issue two consecutive mouse clicks on a form or object, you will receive a click event for the first mouse click and another click event or a double-click event for the second mouse click -- depending on the period of time between the mouse clicks.

At times, you may want to receive only the double-click event without the preceding click event. This article describes how to write code to accomplish this.

MORE INFORMATION

The following example demonstrates how to receive only a DblClick event on a form rather than a Click and then a DblClick event.

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a Timer control (Timer1) to Form1.

3. Add the following code to the (general) (declarations) section of Form1:

```
' The following Function gets the DoubleClickSpeed form the WIN.INI
' file. Windows uses this setting to determine how close together two
' consecutive mouse clicks must occur for it to be interpreted as
' a double-click.
' Enter the following Declare statement on one, single line:
Declare Function GetProfileInt% Lib "Kernel" (ByVal lpAppName$,
   ByVal lpKeyName$, ByVal nDefault%)
```

4. Add the following code to the specified event procedures:

```
Sub Form_Load ()
   clickSpeed% = GetProfileInt("Windows", "DoubleClickSpeed", 0)
   Timer1.Enabled = False          ' Timer should be off to begin with.
   Timer1.Interval = clickSpeed%   ' After the timer is turned on
                                   ' it will trigger the Timer Event
                                   ' after a specific amount of time
                                   ' equal to that of DoubleClickSpeed.
End Sub
```

```
Sub Form_Click ()
   Timer1.Enabled = True          ' Turn the timer on. If another mouse
                                  ' click does not occur within the
                                  ' DoubleClickSpeed interval, the
                                  ' Timer1_Timer Event will fire.
End Sub

Sub Form_DblClick ()
   Timer1.Enabled = False         ' Turn off the timer. This
                                  ' prevents the Timer1_Timer event
                                  ' from firing and thus the code
                                  ' for a single click will not be
                                  ' processed.
   Print "This is a double-click" ' Code for double-click goes here.
End Sub

Sub Timer1_Timer ()
   ' If this event occurs then there has not been another mouse
   ' click since the previous one within the DoubleClickSpeed
   ' time interval. Thus there will be two Click events rather
   ' than a DblClick Event.
   Timer1.Enabled = False         ' Turn off the timer so the
                                  ' Timer1_Timer Event does not
                                  ' continue to fire.
   Print "This is a Single Click" ' Code for single click goes here.
End Sub
```

5. Run the program.

6. Click once, wait a while, click again to receive two single clicks.

7. Click twice quickly to receive a double-click with no preceding single click.

NOTE: The single click code will not occur until the DoubleClickSpeed interval passes and the Timer1_Timer event is fired.

Additional reference words: 1.00 2.00 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsStd

How to Automatically Select or Highlight Text Box Upon Focus

Article ID: Q110394

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The sample program below automatically selects (highlights) all text in a text box whenever the text box gets the focus. This is done by using the SelStart and SelLength properties in the GetFocus event for the text box.

Automatic highlighting is useful in data entry boxes where you want to give the user the option to quickly overwrite the existing contents by pressing any character key.

To avoid overwriting the highlighted text after giving focus to the text box, the user can single-click the text, or press an arrow (direction) key or cursor-movement key (such as END or HOME) to remove the highlighting from the text and allow editing.

MORE INFORMATION

Example How to Automatically Select Text Whenever Given the Focus

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add two text boxes to Form1.

3. Add the following code to the Form Load event:

```
Sub Form_Load ()
   text1.Text = "This sentence is highlighted whenever given the focus."
   text2.Text = "This is not highlighted."
End Sub
```

4. Add the following code to the Text1 GotFocus event:

```
Sub Text1_GotFocus ()
   text1.SelStart = 0          ' Start selection at beginning.
   text1.SelLength = Len(text1.Text)  ' Length of text in Text1.
End Sub
```

5. Start the program, or press the F5 key. Click the text in Text1 to remove its highlighting. Click Text2 to change the focus. Click Text1 again and notice that the complete contents of Text1 are automatically highlighted again. Close the form to end the program.

Syntax of SelLength, SelStart, and SelText Properties

The SelLength, SelStart, and SelText properties apply to combo boxes and text boxes, and behave as follows:

- SelLength determines the number of characters selected.

- SelStart determines the starting point of text selected. SelStart indicates the position of the insertion point if no text is currently selected.

- SelText determines the string containing the currently selected text; consists of an empty string ("") if no characters are currently selected.

The SelLength, SelStart, SelText properties are not available at design time. They are only available at run time. They have the following syntax:

```
[form.]{combobox|textbox}.SelLength[ = length ]
[form.]{combobox|textbox}.SelStart[ = index ]
[form.]{combobox|textbox}.SelText[ = stringexpression ]
```

For SelLength and SelStart, the valid range of settings is 0 to the text length -- the total number of characters in the edit area of a combo box or text box.

Use these properties for tasks such as setting the insertion point, establishing an insertion range, selecting substrings in a control, or clearing text. Used in conjunction with the Clipboard object, these properties are useful for copy, cut, and paste operations. When working with these properties:

- Setting SelLength less than 0 causes a run-time error.

- Setting SelStart greater than the text length sets the property to the existing text length; changing SelStart changes the selection to an insertion point and sets SelLength to 0.

- Setting SelText to a new value sets SelLength to 0 and replaces the selected text with the new string.

For example, to position the insertion point at the end of a text box whenever the box gets the focus, use the following code:

```
text1.SelStart = Len(text1.Text)
text1.SelLength = 1  ' Length of selection
```

SelLength and SelStart have a Long data type. SelText has a String data type.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsStd

How to Start a Visual Basic Screen Saver Using SendMessage API

Article ID: Q110589

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The sample code below shows how to start a Visual Basic screen saver by sending a Windows message to the Control-menu box on the form.

MORE INFORMATION

Microsoft Windows starts screen savers through the System-menu box on a form. The System-menu box is also known as the Control-menu box in Visual Basic. You can send Windows messages to the Control-menu box by using the SendMessage Windows API (application programming interface) function.

Add the following to the general declarations section of Form1, or to a .Bas module file, in Visual Basic:

```
' The following Declare statement must be on one, single line:
Declare Function SendMessage Lib "User" (ByVal hWnd As Integer,
   ByVal wMsg As Integer, ByVal wParam As Integer, lParam As Any) As
   Long
```

In the following example, a command button starts the Form1 screen saver:

```
Sub Command1_Click ()
   Dim result As Long
   Const WM_SYSCOMMAND = &H112
   Const SC_SCREENSAVE = &HF140
   result = SendMessage(Form1.hWnd, WM_SYSCOMMAND, SC_SCREENSAVE, 0&)
End Sub
```

You can find two sample programs and a complete explanation showing how to write your own screen savers in Visual Basic in the following book:

"Visual Basic Workshop 3.0" by John C. Craig, published by Microsoft Press.

Additional reference words: 3.00 .SCR TOPMOST SETWINDOWPOS SCRNSAVE timer
KBCategory: Prg
KBSubcategory: PrgCtrlsStd

Selected Prop of List Box Can Cause Click Event & Out of Stack

Article ID: Q110957

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The Selected property applies to list box and file list box controls. When you change the selected state of a list box item by using code to set the Selected property at run time, a Click event occurs -- just as if you had actually clicked with the mouse. A Click event does not occur if you set the item's Selected property to a value that keeps the item's current selected or unselected state.

You need to take care when changing the Selected property within the Click event procedure for a list or file list box. Clicking an item in a list control causes a Click event and can change a selection. That changed selection can affect whether your code's subsequent change to an item's Selected property causes another Click event. The resulting second click event can cause recursion and "Out of Stack Space" or other errors if you designed your Click event procedure incorrectly. The More Information section below provides examples showing how to avoid recursion problems.

This behavior is by design.

MORE INFORMATION

The Selected property determines the selection status of an item in a list box or file list box control. The Selected property is an array of Boolean values with the same number of items as the List property. The Selected property is available at run time, but not at design time. Here is the syntax:

```
[form.]{filelistbox|listbox}.Selected(index)[ = {True|False}]
```

The Selected property settings are:

```
True  = The item is selected.
False = (Default) The item is not selected.
```

The Selected property is particularly useful where users can make multiple selections. You can quickly check which items in a list are selected. Your code can use this property to select or deselect items in a list.

If only one item is selected, you can use the ListIndex property to get the index of the selected item. However, in a multiple selection, the ListIndex property returns the index of the item contained within the focus rectangle, whether or not the item is actually selected. Multiple selection mode can be set with the MultiSelect property.

Changing Item's Selected Property Can Cause Second Click Event

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a list box (list1) to Form1. (Or you can use a file list box.)

3. Add the following code to the Form Load event:

```
Sub Form_Load ()
   For i = 1 To 5
      list1.AddItem Str$(i)  'Add more than 2 items to the list box.
   Next
End Sub
```

4. Add the following code to the List1 Click event:

```
Sub List1_Click ()
   Static x   ' Preserve the value of x between Click events.
   x = x + 1  ' Increment the count of Click events.
   Print x    ' Print the cumulative number of click events.
   ' The following statement only causes a second event when an item
   ' other than the first item is clicked. Clicking the first item
   ' (item 0) does not cause a second event, because list1.Selected(0)
   ' is already True:
   list1.Selected(0) = True   ' Selects the first item.
End Sub
```

5. Start the program, or press the F5 key. Click any list box item other than the first. That causes two Click events. Click the first item in the list box. That causes just one Click event. Click more items to repeat the same behavior. Close the form to end the program.

How to Make File List Box Items That Can Be Scrolled, But Not Selected

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a file list box to Form1.

3. a. Add the following code to the File1 Click event:

```
Sub File1_Click ()
   ' If no item is selected in the file list box, exit the sub:
   If File1.ListIndex = -1 Then Exit Sub
   File1.Selected(File1.ListIndex) = False
End Sub
```

NOTE: If you left out the above If statement, then the File1.Selected(File1.ListIndex) statement would give the following error when you click the file list box:

```
Invalid property array index (Error 381)
```

The index value of the File1.Selected() property must be greater than 0. By default, File1.ListIndex starts as -1.

b. Instead, you can add the following code to the File1 Click event. In this example, the focus always reverts to the first item when you click any other item. The first item remains unselected (without highlight). The focus is indicated by a dotted box around the item.

```
Sub File1_Click ()
   ' NOTE: If you delete the following line, you get "Out of Stack
   ' Space" due to recursion (about 30 iterations):
   If File1.ListIndex = -1 Then Exit Sub
   File1.Selected(0) = True
   File1.Selected(0) = False  'Resets first item to nonselected.
End Sub
```

4. Start the program, or press the F5 key. Now you can click items in the file list box, but they won't be selected or highlighted.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsStd

How to Right Justify Items in List Box w/ Tabs & SendMessage

Article ID: Q110958

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The sample program below shows how to right justify items in a list box.

MORE INFORMATION

This program calls the SendMessage Windows API function to set a tab stop at every character position in the list box. The program prefixes the appropriate number of tabs to right justify each string in the list box. You need to set the maximum allowed string length in the program.

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a large list box (List1) to Form1.

3. Add the following to the Form Load event code:

```
Sub Form_Load ()

    Const WM_USER = &H400
    Const LB_SETTABSTOPS = WM_USER + 19
    Const maxlen = 10 ' Maximum expected string length in list box.
    tabchar = Chr$(9) ' ASCII code for a tab
    ReDim a$(maxlen)  ' String array to right justify in list box.
    form1.Show        ' Must Show form in Load event before Print
                      ' will become visible.

    ' GetDialogBaseUnits() API function lets you calculate the average
    ' width of characters in the system font.
    bu& = GetDialogBaseUnits()
    hiword = bu& \ (2 ^ 16)    ' 16 pixels high in default system font.
    loword = bu& And &HFFFF&   ' 8 pixels wide in default system font.
    Print "System font width and height, in pixels:   " & loword, hiword

    'Assign the array of defined tab stops.
    Static tabs(1 To maxlen) As Integer
    For j = 1 To maxlen  ' Set tabs every 4 dialog units (one character):
       tabs(j) = (loword * j) / 2
       ' On most Windows systems, you need only this: tabs(j) = j * 4
    Next
```

```
      'Send message to the List1 control through the Windows message queue:
      retVal& = SendMessage(List1.hWnd, LB_SETTABSTOPS, maxlen, tabs(1))

      For j = 1 To maxlen
         a$(j) = String$(j, "a") ' Assign an arbitrary character string.
         ' Add the appropriate number of tabstops to right justify:
         tabstring = String$(maxlen + 1 - Len(a$(j)), Chr$(9))
         List1.AddItem tabstring & a$(j)
      Next

   End Sub
```

4. Add the following Windows API declarations to the General Declarations section:

```
   Declare Function GetDialogBaseUnits Lib "User" () As Long
   ' Enter the following Declare statement on one, single line:
   Declare Function SendMessage Lib "user" (ByVal hWnd As Integer,
      ByVal wMsg As Integer, ByVal wp As Integer, lp As Any) As Long
```

5. Start the program, or press the F5 key. All strings are right-justified in the list box. Close the form to end the program.

Additional reference words: 3.00 alignment right-align align
KBCategory:
KBSubcategory: APrgOther PrgCtrlsStd

How to Right Justify/Center Text in Single-Line Text Control

Article ID: Q111952

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

To center or right justify (align) the text in a text control that contains a single line of text, set the multiline property to True and the Alignment property to the desired value. Then trap the KeyPress event and use the multiline and MaxLength properties to trap the carriage return and convert it to another character.

MORE INFORMATION

Step-by-Step Example to Demonstrate a Right-Justified Text Control

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a text control (Text1) to Form1. Set its Alignment property to 1 - Right Justify. Set the Multiline property to True. Set the MaxLength property to some arbitrary value equal to the width of the text box in characters (15). Be sure to make the text controls height property large enough to display the first line of text. The actual height of the text box may need to be a little bigger than normal.

3. Add the following code to the KeyDown event procedure of Text1:

```
'============== Form1.frm ==================

Sub Text1_KeyPress (KeyAscii As Integer)
   If KeyAscii = 13 Then
      KeyAscii = 7            ' Beep - no effect on text
   End If
End Sub
```

4. From the Run menu, choose Start (ALT, R, S) to run the program.

Type text into the text control, and press the ENTER key. The keystroke is trapped, and the text does not change.

Additional reference words: 3.00 align format
KBCategory:
KBSubcategory: PrgCtrlsStd

PRB: How to Prevent Flicker in the Repaint of a Label

Article ID: Q112675

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

When a label control fires a change event, the Caption tends to flicker as it is repainted within the control -- more so when the font size is large.

WORKAROUND

The flicker can be avoided if a picture control (with its AutoRedraw property set to true) is used instead of the label control. However, note that a picture control uses considerably more resources than a label. If you're going to replace a lot of labels, replace them all with one picture control (and multiple print statements) rather than with multiple picture controls. Otherwise, you'll run out of system resources.

The following example demonstrates the use of the picture control to prevent the flicker.

1. Start a new project in Visual Basic, Form1 is created by default.

2. Add a picture box (Picture1) to the form, and set its autoredraw property to True.

3. Add a timer control (Timer1) to the form and set its interval property to 100.

4. Add the following code to the Timer1_Timer event:

```
Sub Timer1_Timer ()
   ' Reset the picture in Picture1:
   Picture1.Cls
   ' Specify the top and left coordinates for the text:
   Picture1.CurrentX = 100
   Picture1.CurrentY = 100
   ' Enter the text (in this case the current time):
   Picture1.Print = Format$(Now, "h:mm:ss AM/PM")
End Sub
```

5. Run the program.

An Alternative to the Picture Control

Flicker in a label can be reduced considerably by only updating it when absolutely necessary. For example, update it at the end by using code

similar to this:

```
   Sub Timer1_Timer ()
      Dim TimeStr As String
      TimeStr = Format$(Now, "h:mm:ss AM/PM")
      If Label1.Caption <> TimeStr Then Label1.Caption = TimeStr
   End Sub
```

Or by using code similar to this slightly more efficient code:

```
   Sub Timer1_Timer ()
      Static LastSecond As Integer
      If Seconds(Now) <> LastSecond Then
         LastSecond = Seconds(Now)
         Label1.Caption = Format$(Now, "h:mm:ss AM/PM")
      End If
   End Sub
```

MORE INFORMATION
================

Steps to Reproduce Behavior
----

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a label control (Label1) to Form1.

3. Add a timer control (Timer1) to Form1.

4. Add the following code to the Timer1_Timer event:

```
   Sub Timer1_Timer ()
      Label1.Caption = Format$(Now, "h:mm:ss AM/PM")
   End Sub
```

5. Run the program.

You will notice a flicker of the caption of the label control. If you have trouble seeing the flicker, try increasing the fontsize.

Additional reference words: 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd

VB Custom Controls Support only Certain Picture Formats

Article ID: Q80779

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Load Picture dialog box for the 3-D Command Button, 3-D Group Push Button, Gauge, and Picture Clip custom controls include the extensions for picture formats that are invalid formats for these controls.

MORE INFORMATION

The 3-D Command Button, 3-D Group Push Button, Gauge, and Picture Clip custom controls use the same dialog box that Visual Basic uses to assign pictures to certain properties. However, not all .BMP, .ICO, and .WMF files are valid picture formats for the properties of these controls.

The following table lists the valid formats for the picture properties of custom controls and the error messages displayed if an invalid picture format is used:

Control	Property	Valid Formats	Error Message if Invalid Format
3-D Command Button	Picture	.BMP, .ICO	"Only Picture Formats '.BMP' and '.ICO' supported."
3-D Group Push Button	PictureUp, PictureDn, PictureDisabled	.BMP	"Only Picture Format '.BMP' supported."
Gauge	Picture	.BMP, .ICO	"Invalid Picture."
Picture Clip	Picture	.BMP	"Only Picture Format '.BMP' supported."

For additional information on Visual Basic version 2.0 custom controls, review the Professional Features manual.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus APrgGrap

PRB: Grid Custom Control: Surprising Results when FillStyle=1

Article ID: Q80849

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic Programming System for Windows, version 1.0

SYMPTOMS

When the Grid custom control has its FillStyle property set to 1 (repeat), assignments to the Text and Picture properties store a value in all the cells within the selected region (determined by SelStartCol, SelStartRow, SelEndCol, and SelEndRow). However, the value returned from Text and Picture comes from the current cell (determined by the Col and Row properties).

This behavior can produce surprising results when the current cell is located outside the selected region.

When FillStyle is 0 (single), the Text and Picture properties store to the current cell and retrieve from the current cell.

RESOLUTION

To cause the Text property to return the same value assigned when FillStyle=1, set the current cell location to a cell inside the selected region. For example, use this code:

```
Grid1.Text = "hello"
Grid1.Col = Grid1.SelColStart
Grid1.Row = Grid1.SelRowStart
' Length of Text is 5
MsgBox "Len(Text)=" + Format$(Len(Grid1.Text))
```

STATUS

This behavior is by design.

MORE INFORMATION

The CellSelected property returns True (-1) if the current cell is within the grid's selected region; otherwise, CellSelected returns False (0).

Steps to Reproduce Problem

1. Start Visual Basic, or if Visual Basic is already running, choose New Project from the File menu (ALT, F, N). Form1 is created by default.

2. From the File menu, choose Add File, and select GRID.VBX. The Grid tool will appear in the Toolbox.

3. Select the Grid tool from the Toolbox, and place a grid (Grid1) on Form1.

4. On the Properties bar (Properties window in Visual Basic version 2.0), set the grid properties Cols and Rows each to 4. In Visual Basic 2.0, you will need to press the F4 key to display the Properties Window, so you can set the Cols and Rows properties.

5. Size the grid so that you can see all the cells.

6. Double-click the form to bring up the Code window. In the Procedure box, select Load. Enter the following code:

```
Sub Form_Load ()
   Grid1.FillStyle = 1     ' Repeat.

   ' Set selected region.
   Grid1.SelStartCol = 2
   Grid1.SelStartRow = 2
   Grid1.SelEndCol = 3
   Grid1.SelEndRow = 3

   ' Set current cell, outside of selected region.
   Grid1.Col = 1
   Grid1.Row = 1

   ' Assign to Text.
   Grid1.Text = "hello"

   ' Length of Text is 0, not 5.
   Show
   MsgBox "Len(Text)=" + Format$(Len(Grid1.Text))
End Sub
```

7. Press the F5 key to run the program.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

PRB: Grid Control's Cell Blank When Using Str$

Article ID: Q80904

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Visual Basic for Windows, version 1.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

With the Microsoft Professional Toolkit for Visual Basic Grid control, when you use the Str$ function to store numeric values in a grid cell, the cell appears blank if it is not wide enough to completely display the value.

CAUSE

This behavior occurs because of word wrapping. The Str$ function returns a string that begins with a space character. When this string does not fit in a grid cell, it wraps to the next line, breaking on the leading space so that no text remains on the first line of the cell.

WORKAROUND

To avoid the problem, use Format$ instead of Str$, or Ltrim$ with Str$. To work around the problem, change the assignment to Grid1.Text to one of the following:

```
Grid1.Text = Format$(123456)

   -or-

Grid1.Text = Ltrim$(Str$(123456))
```

This will eliminate the leading space, and the information in the cell will be displayed up to the width of the cell. You can also increase the width of the cell to allow all characters to be visible.

MORE INFORMATION

Steps to Reproduce Problem

1. Start Visual Basic or, if Visual Basic is already running, choose New Project from the File menu (ALT, F, N). Form1 is created by default.

2. From the File menu, choose Add File, and select GRID.VBX to add the Grid tool. The Grid tool appears in the Toolbox.

3. Place a grid named Grid1 on Form1.

4. Set the grid properties Cols and Rows each to 4.

5. Size the grid so that you can see all the cells.

6. Enter the following code. To enter the code, double-click Grid1, select Click in the Procedure box, and enter the code into the code template.

```
Sub Grid1_Click ()
   Grid1.Text = Str$(123456)
   Debug.Print Grid1.Text
End Sub
```

7. Press F5 to run the program.

8. Each time you click a cell in Grid1, this code prints "123456" in the Immediate window, but the cell remains blank.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

VB Grid Custom Control: Text Limited to 255 Characters

Article ID: Q80906

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Text property for the Grid custom control can hold a string of up to 255 characters. If you assign a string longer than 255 characters to the grid Text property, the string is truncated.

This behavior is by design in Visual Basic. This behavior is similar to a text box control with the MultiLine property set to FALSE (0).

This information applies to Microsoft Professional Toolkit for Microsoft Visual Basic programming system version 1.0 for Windows.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

PRB: Grid Custom Control: LeftCol/TopRow Valid Values

Article ID: Q80911

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

The LeftCol and TopRow grid properties control the position of the scrollable region of a Grid custom control. When you attempt to set the LeftCol or TopRow grid properties to display the lower right region of a grid, you may receive the error "Invalid Column Value" (30010) or "Invalid Row Value" (30009), respectively.

WORKAROUND

The example given below in the More Information section shows how to determine the range of values that do not give errors.

STATUS

This behavior is by design.

MORE INFORMATION

A program can determine the maximum values allowed for LeftCol and TopRow by setting these properties to each valid column and row number, respectively, and then determining if the assignment caused an error. The example code below shows how to use this method.

The minimum values allowed for LeftCol and TopRow are always given by the values of the Grid custom control properties FixedCols and FixedRows, respectively.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File, and select the GRID.VBX custom control file. The Grid tool will appear in the Toolbox.

3. Select the Grid tool from the Toolbox, and add a grid (Grid1) to Form1.

4. Size the Grid and choose values for the Cols and Rows properties of

the Grid.

5. Place a command button (Command1) on Form1.

6. Enter the following code in the Command1_Click event procedure:

```
' Example of how to call Grid_Scroll_Range to determine the
' range of values for Grid properties LeftCol and TopRow.
Sub Command1_Click ()
   Dim msg As String          ' message string
   Dim max_LeftCol As Single  ' maximum LeftCol
   Dim max_TopRow As Single   ' maximum TopRow

   Call grid_scroll_range(grid1, max_LeftCol, max_TopRow)

   msg = "Valid Grid.LeftCol: "
   msg = msg + Format$(grid1.FixedCols) + ".."
   msg = msg + Format$(max_LeftCol) + Chr$(13) + Chr$(10)
   msg = msg + "Valid Grid.TopRow: "
   msg = msg + Format$(grid1.FixedRows) + ".."
   msg = msg + Format$(max_TopRow)

   MsgBox msg
End Sub
```

7. Enter the following code in the general Declarations section:

```
' grid_scroll_range
'   Determines the maximum values allowable for grid LeftCol
'   and TopRow. Minimum values are FixedCols and FixedRows.
'   Parameters:
'       grid    -- a Grid control
'       LftMax  -- return value, maximum LeftCol value
'       TopMax  -- return value, maximum TopRow value
'
Sub grid_scroll_range (grid As Control, LftMax!, TopMax!)
   Dim save As Integer     ' for restoring grid properties

   ' Calculate LftMax
   ' Try each column number to see if it causes a run-time
   ' error. Go in reverse order to minimize the number of
   ' tries to the number of columns displayed in the grid.
   save = grid.LeftCol
   On Error Resume Next
   For LftMax = grid.Cols - 1 To grid.FixedCols + 1 Step -1
      Err = 0
      grid.LeftCol = LftMax
      If Err = 0 Then
         Exit For
      End If
   Next
   grid.LeftCol = save

   ' Calculate TopMax
   ' Try each row number to see if it causes a run-time
   ' error. Go in reverse order to minimize the number of
   ' tries to the number of rows displayed in the grid.
```

```
      save = grid.TopRow
      On Error Resume Next
      For TopMax = grid.Rows - 1 To grid.FixedRows + 1 Step -1
         Err = 0
         grid.TopRow = TopMax
         If Err = 0 Then
            Exit For
         End If
      Next
      grid.TopRow = save
   End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

3-D Group Push Button: AutoSize Takes Effect Only on PictureUp

Article ID: Q80938

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The 3-D Group Push Button (THREED.VBX) custom control will not automatically size itself to the bitmap assigned to the PictureDown property. When the AutoSize property is set to "2 - Adjust Button Size to Picture," the 3-D Group Push Button custom control will automatically size itself to the bitmap assigned to the PictureUp property. This behavior is by design.

This information applies to Microsoft Professional Toolkit for Microsoft Visual Basic programming system version 1.0 for Windows.

MORE INFORMATION

The Visual Basic 3-D Group Push Button custom control can have a bitmap assigned to the button when it is in the down position, and another bitmap when the button is in the up position (the PictureUp and PictureDown properties are set to different .BMP files at design time). However, the 3-D Group Push Button control will not automatically size itself to the size of the picture assigned to the PictureDown property even if the AutoSize property is set to "2 - Adjust Button Size to Picture."

When the AutoSize property is set to "2 - Adjust Button Size to Picture," the 3-D Group Push Button custom control will automatically size itself to the bitmap assigned to the PictureUp property. This means that the button will size itself to the picture it is supposed to display only when it is in the up position. If there is a bitmap assigned to the PictureDown property and this picture is bigger than the 3-D Group Push Button control, this picture will appear clipped when the button is pressed.

The following steps demonstrate how the 3-D Group Push Button custom control does not size itself to the bitmap assigned to the PictureDown property even when the AutoSize property for the control is set to "2 - Adjust Button Size to Picture."

Example

1. Run Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the THREED.VBX custom control file.

3. Click the Toolbox to select the 3-D Group Push Button control.

4. Click and drag on the form to place a 3-D Group Push Button control.

5. Change the AutoSize property in the Properties Bar to "2 - Adjust Button Size to Picture" (this is the default setting).

6. Change the PictureDown property in the Properties Bar by choosing a bitmap file from the Properties list box. Note that "(none)" is first displayed, because no picture is assigned by default. You can click the button with three dots on the right of the Properties list box to choose a bitmap file.

7. From the Run menu, choose Start to run the application.

8. Click the 3-D Group Push Button to push it into the down position.

The picture that is assigned to the PictureDown property is displayed. If the picture happens to be larger than the 3-D Group Push Button control, the picture will appear clipped. If the picture chosen was smaller, the background of the 3-D Group Push Button control will show. In either case, the 3-D Group Push Button control did not resize itself to the picture once the button was pressed.

If you assign a picture to the PictureUp property of the 3-D Group Push Button control, the button will automatically size itself to this picture. Nonetheless, the 3-D Group Push Button control will not size itself to the picture assigned to the PictureDown property once the button is pressed.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

VB Graph Control Displays Maximum of 80 Characters Per Title

Article ID: Q81450

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Graph custom control has an 80 character maximum limit on all displayed strings such as labels and legends. However, the combined length of the actual string may be longer than 80 characters.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The Graph custom control can display strings by using several different properties. For example, the BottomTitle and LeftTitle properties may be set from the Properties bar in the programming environment.

The following example sets the BottomTitle property of a Graph to 90 characters:

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the GRAPH.VBX custom control file. The Graph tool will appear in the toolbox.

3. Select the Graph icon on the toolbox and place it on Form1, and expand it to the largest size possible.

4. Double-click the Graph control to open the Code window for the Click event.

5. Add the following code to the Click event:

```
Graph1.BottomTitle = String$(79, "i") + "*"
Debug.Print Len(Graph1.BottomTitle)
Graph1.DrawMode = 2          ' Update Graph.
```

6. Run the program and click on the graph control. If your Graph is expanded to the largest possible size, you should be able to see the string of 80 characters.

7. Change the code as follows:

```
   Graph1.BottomTitle = String$(80, "i") + "*"
   Debug.Print Len(Graph1.BottomTitle)
   Graph1.DrawMode = 2                ' Update Graph.
```

You should not be able to see the last character, the asterisk (*).

In this example, 80 characters at most will show on the screen even though you set the BottomTitle property to a larger character string. The actual BottomTitle property, however, contains more characters. Whether or not the actual strings are displayed also depends on other factors, such as the width and height of the control, or the strings that are placed in the other properties of the control.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

VB.EXE Error: License File for Custom Control Not Found

Article ID: Q81458

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

If you distribute the source code (.FRM) of a program that uses a custom control, you must also distribute the necessary custom control files for that control (.VBX, .DLL, and/or .EXE support files).

If a user has not purchased the Professional Edition of Microsoft Visual Basic versions 2.0 or 3.0 for Windows, or the Microsoft Professional Toolkit for Microsoft Visual Basic programming system version 1.0 for Windows, and the user receives a program containing an .FRM file written with the Professional Edition or Professional Toolkit, then the Visual Basic for Windows programming environment (VB.EXE) will not be able to load the program, and will display the following error message:

> License file for custom control not found. You do not have an appropriate license to use this custom control in the design environment.

Note that anyone who acquires a program in the form of an executable (.EXE) file that uses the custom controls from versions 2.0 or 3.0 of the Professional Edition of Visual Basic for Windows, or from version 1.0 of the Professional Toolkit for Visual Basic for Windows, will be able to run that program with no error.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The licensing file, VB.LIC is installed by the SETUP.EXE program included in the Professional Edition of Visual Basic for Windows, or the SETUP.EXE included in the Visual Basic for Windows Professional Toolkit. This licensing file is installed into the Windows' \SYSTEM subdirectory. You are NOT allowed to distribute this file with any application that you develop and distribute.

A custom control's startup code checks to see if this VB.LIC licensing file exists when the control is loaded into the environment. If the file does not exist, or is corrupt, the control aborts the loading process and displays the following Alter message box:

License file for custom control not found. You do not have an appropriate license to use this custom control in the design environment.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

How to Use HORZ1.BMP with Professional Toolkit Gauge Control

Article ID: Q81459

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article contains a program example of using the Visual Basic for Windows Gauge custom control (GAUGE.VBX) with the HORZ1.BMP bitmap file.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Note: The GAUGE.VBX custom control file can be found in the \Windows\System subdirectory. The HORZ1.BMP bitmap file can be found in the \BITMAPS\GUAGE subdirectory that was created during installation.

Example Program

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the GAUGE.VBX custom control file. The Gauge tool will appear in the toolbox.

3. Create the following controls for Form1:

Control	Name	Property Setting
Timer	Timer1	Interval = 1
Gauge	Gauge1	Picture = "Horz1.BMP"
		Max = 50
		InnerBottom = 16
		InnerLeft = 38
		InnerRight = 2
		InnerTop = 14
		ForeColor = &HFF&

   (In Visual Basic version 1.0 for Windows, set the CtlName Property for the above objects instead of the Name property.)

4. Add the following line to the General Declarations section:

```
   Dim YoYo As Integer
```

5. Add the following code to the Form_Load event procedure:

```
Sub Form_Load ()
   Form1.Caption = "YoYo Gauge Demo"
   Gauge1.Value = Gauge1.Min
End Sub
```

6. Add the following code to the Timer1_Timer event procedure:

```
Sub Timer1_Timer ()
   If Gauge1.Value = Gauge1.Max Then YoYo = -1
   If Gauge1.Value = Gauge1.Min Then YoYo = 1
   Gauge1.Value = Gauge1.Value + YoYo
End Sub
```

When run, this program example will alternately fill and empty the gauge control's fill area, as controlled by the Timer event procedure.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

HOME Key in VB.EXE Moves to Beginning of Code, Not Column 1

Article ID: Q81465

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOM

Pressing the HOME key while in the Code window in the VB.EXE environment will move the insertion point (cursor) to the beginning of the code on a line instead of to the first column of the line

STATUS

This behavior is by design even though it does differ from most other Windows-based products.

MORE INFORMATION

If the insertion point is on a line of code indented with spaces, and you press HOME, the insertion point will not be moved to the beginning of the line, but will instead move to the beginning of the code.

In many Windows-based applications, including Microsoft Word for Windows, Notepad, and Write, pressing the HOME key moves the insertion point to the beginning of the line, not to the beginning of the characters on the line.

Steps to Reproduce Behavior

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Double-click Form1 to open the Code window.

3. In the Form1_Click event, press TAB to indent the next statement, and add the following code:

```
    Print "Hello"
```

4. Press the HOME key. The insertion point moves to the P in Print.

5. Press the HOME key again.

You might expect the insertion point will move to the beginning of the line, but it remains on the P.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

PRB: Animated Button Control: Refresh Won't Redraw Border

Article ID: Q81471

The information in this article applies to:

- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

When two Animated Button (ANIBUTON.VBX) custom controls are overlapped and the BorderStyle is set to 1 - Single, then when one of the controls is refreshed, the border of that control is not redrawn.

WORKAROUND

To work around this behavior, set the BorderStyle property to 0 - None for the controls. Avoid using overlapped controls.

STATUS

This behavior is by design. However, it does not apply to Microsoft Visual Basic version 2.0 or 3.0 because these later versions support overlapping controls.

MORE INFORMATION

The standard picture control shows the same behavior when overlapped. Either using the Refresh method or causing an implicit refresh by clicking the control being overlapped will demonstrate the behavior.

Note that Visual Basic versions 1.0 and 2.0 do not support overlapping controls. If you want to overlap controls, set BorderStyle to 0 - None.

Steps to Reproduce Behavior

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files list box, select the ANIBUTON.VBX custom control file. The Animated Button tool will appear in the toolbox.

3. Add two Animated Button controls to the form with one overlapping the other, and set the BorderStyle property for both to 1 - Single.

4. From the VB.EXE Run menu, choose Start, then choose Break.

5. In the Immediate window, enter Anibutton1.Refresh or Anibutton2.Refresh,

depending on which control is overlapped by the other.

The formerly overlapped border is not redrawn when the control is repainted.

Alternatively, in run mode, click the overlapped control; the result is the same.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

BUG: Graph Custom Control Text Disappears in EGA Video Mode

Article ID: Q81949

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

When using the Visual Basic Graph custom control in an EGA video mode with the Graph control Background property value set to dark gray and the Foreground property value set to light gray, the text on the graph will disappear.

CAUSE

This is a known problem with Windows versions 3.0 and 3.1. This is not a problem with the Graph custom control or with Visual Basic.

STATUS

Microsoft has confirmed this to be a problem with Microsoft Windows versions 3.0 and 3.1. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

Windows defines dark gray as the color created when red=128, blue=128, and green=128. Windows defines light gray as the color created when red=192, blue=192, and green=192.

Windows, when given light gray text on a dark gray background in EGA video mode, alters the value of the text color to dark gray, which is the closest representation it can make in that video mode. The subsequent dark gray text on a dark gray background makes it appear as though the text has disappeared.

The Visual Basic Graph custom control allows you to set the background and foreground colors to 16 predefined colors. Colors 7 and 8 are light gray and dark gray, respectively. Graph uses Windows values for dark gray and light gray, and so displays the same video problems as Windows itself.

Steps to Reproduce Problem

1. Set the video mode of Windows to EGA.

2. Re-enter Windows if necessary and start Visual Basic.

3. In the Visual Basic environment with the VB Graph custom control loaded, create a form (Form1).

4. Add a Graph custom control (Graph1).

5. Set Graph1.DrawMode=2 (draw).

6. Set Graph1.Background=8 (dark gray) and Graph1.Foreground=7 (light gray).

The text disappears, leaving colored bars on a dark gray background.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

VB Key Status: Autosize Property Affects Height and Width

Article ID: Q81952

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

In the products listed above, you can use the Key Status control (KEYSTAT.VBX) to show and set the current status of the CAPS LOCK, NUM LOCK, SCROLL LOCK, and INSERT keys. One of the features of the Key Status control is its ability to size itself (the Autosize property) to its original dimensions.

If the Autosize property is set to True (the default setting), the control's Height and Width properties will remain at, or be reset to its predetermined values. The size of the control cannot be changed if Autosize is set to True. If the Autosize property is set to False, the Height and Width properties can be changed to reflect the desired control size. Autosize can be set at both design time and run time.

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

VB Graph Control: ThisPoint, ThisSet Reset to 1 at Run Time

Article ID: Q82155

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Graph version 1.2 custom control in the Professional Edition of Microsoft Visual Basic versions 2.0 or 3.0 for Windows, and in the Microsoft Professional Toolkit for Visual Basic version 1.0 for Windows, allows you to set the values of the ThisPoint and ThisSet properties at design time to aid in the development of your graphs. However, when you run the project, the Graph custom control resets the property values of ThisPoint and ThisSet to 1.

This behavior is a design feature of the Graph custom control to help avoid logic errors in your code. If your program requires ThisPoint and ThisSet to be a value other than 1 upon execution of the project, you will need to specifically set these property values in the program's code.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The example below demonstrates that ThisPoint and ThisSet are reset to 1 at run time.

Example

1. With Visual Basic for Windows running and Graph loaded, create a form (Form1).

2. On Form1 create a graph control (Graph1).

3. Change the following properties:

Control	Property	Value
Command1	Caption	Show values
Graph1	Top	2000
Graph1	NumSet	2
Graph1	ThisPoint	2
Graph1	ThisSet	2

4. Add the following code to the Command1 button Click event:

```
Sub Command1_Click ()
     Form1.Print "Graph1.ThisPoint = "; Graph1.ThisPoint
     Form1.Print "Graph1.ThisSet = "; Graph1.ThisSet
End Sub
```

5. Press the F5 key to run the program.

When you run the program and click the Command1 button, the program will display the current values of Graph1.ThisPoint and Graph1.ThisSet. These values should have changed from 2 to 1.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

VB AniButton Control: Cannot Resize if PictDrawMode=Autosize

Article ID: Q82159

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Resizing an Animated Button custom control by setting the Width or Height property at run time will not work if the PictDrawMode property is set to Autosize (1). This is by design. When the PictDrawMode property is in autosize mode, the size is determined by the size of the images loaded, not by the design time setting of Width or Height nor the run time setting of those values.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Steps to Reproduce Behavior

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. From the Files menu, choose Add File. In the Files box, select the ANIBUTON.VBX custom control file. The Animated Button tool appears in the toolbox.

3. Add the following code to the Form_Load procedure:

```
Sub Form_Load ()
  Form1.BackColor = &HFFFF00  ' To make the size of the control more
                              ' visible.
  AniButton1.Move Form1.Width \ 4, 0, 1600, 1600
  AniButton1.TextPosition = 3  ' Put caption at top for clarity.
End Sub
```

4. Add the following code to the Form_Click procedure:

```
   Sub Form_Click ()
       AniButton1.Caption = "This is a very very long caption"
       AniButton1.PictDrawMode = 1   ' Autosize control.
      'AniButton1.PictDrawMode = 0   ' As Defined.
      'AniButton1.PictDrawMode = 2   ' Stretches image to fit.
   End Sub
```

4. Add the following code to the Form_DoubleClick event:

```
Sub Form_DblClick ()
   Print AniButton1.Width
   AniButton1.Width = 400
   Print AniButton1.Width
   Print AniButton1.PictDrawMode
End Sub
```

5. Run the project with the PictDrawMode setting of 0 uncommented and the other two commented out.

6. Click once to see the effect of changing the mode. Then double-click the form to see the changes due to changing the Width property. Because the caption is the largest object in an unloaded Animated Button, the autosize adjusts to it.

7. Access the Frame property and load a bitmap into the first frame and an icon in the second, or vice versa.

8. Repeat steps 5 and 6. Notice that the larger object (the bitmap) causes the control to resize to it.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus APrgGrap

PRB: Can't Change Minimized/Maximized MDIChild's Position/Size

Article ID: Q82878

The information in this article applies to:

- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

SYMPTOMS

When a MDI Child custom control is minimized (reduced to an icon), attempting to change its position or size at run time by setting the Top, Left, Height, or Width property will generate the following Visual Basic error message:

Cannot Change MDIChild Position Or Size While Minimized Or Maximized.

This valid error message will also be generated if the MDI child window is maximized and you attempt to change the size of position of the MDI child.

RESOLUTION

This article does not apply to later versions of Visual Basic. The MDI Child custom control shipped only with version 1.0. Multiple-document interface (MDI) forms are built into Visual Basic version 2.0 and later, making the MDI custom control obsolete.

You cannot change the position or size of a Visual Basic version 1.0 MDI child window when it is minimized or maximized. These properties can be set at run time in code or at design time for any MDI child window that is not maximized or minimized to an icon.

However, you can set the properties in Visual Basic version 2.0 for Windows. You do not get an error. Note though that MDI is different in Visual Basic version 2.0 because it is built in to both the Standard and Professional Editions rather than being a separate custom control, as it is in Visual Basic version 1.0.

MORE INFORMATION

The following steps demonstrate that an error message is generated in Visual Basic version 1.0 when you attempt to change (at run time in code) the Left property of an MDI child window that has been either reduced to an icon or maximized (to the full size of the parent form).

Steps to Reproduce Problem

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the MDICHILD.VBX custom control file. The MDI Child tool appears in the toolbox.

3. Place an MDI Child window control on Form1.

4. Double-click the form outside the MDI child window to open the Code window.

5. Add the following code to the Form1 Click event:

```
Sub Form_Click ()
   MDIchild1.Left = 0
End Sub
```

6. Press F5 to run the application.

7. Click the Control-menu box (in the upper left corner) of the MDI child window, and choose Minimize.

8. Click directly on the form.

The following error message dialog box is generated:

```
Cannot Change MDIChild Position Or Size While Minimized Or Maximized
```

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: PrgCtrlsCus APrgGrap

"Device Is Not Open or Is Not Known" Running VB MCITEST Sample

Article ID: Q83756

The information in this article applies to:

- Professional Editions of Microsoft Visual Basic for Windows, versions 2.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

If you run the MCITEST sample program included with the Microsoft Visual Basic Professional Edition programming system version 2.0 for Windows; or the Professional Toolkit for Microsoft Visual Basic programming system version 1.0 for Windows, and receive the following error message:

   The device is not open or is not known

it is possible that you have not installed the multimedia movie player driver that is on Disk 1 of the Professional Edition, or Professional Toolkit. These drivers are not automatically installed because of their large sizes. If you need these files, you must install them using the Windows Control Panel.

MORE INFORMATION

The drivers needed to run the animation portion of the MCITEST are MCIMMP.DRV and MMP.DLL. These files are archived on Disk 1 of the Visual Basic for Windows Professional Edition or Professional Toolkit. To install these drivers, you must use the Windows 3.1 Control Panel. If you have Multimedia Extensions for Windows 3.0, you must rename the following files on Disk 1 before installing using the Control Panel:

MCIMMP.DR_	to	MCIMMP.DRV
MMP.DL_	to	MMP.DLL

To install the drivers, do the following:

1. Run Control Panel from the Windows Program Manager either by clicking on the icon or by choosing Run from the File menu.

2. In the Control Panel, double-click the Drivers icon.

3. In the Drivers dialog box, choose the Add button.

4. In the Add dialog box, select Unlisted or Updated Driver and choose the OK button.

5. In the Install Driver dialog box, specify the drive containing Disk 1 of the Visual Basic for Windows Professional Edition or Professional

Toolkit, and choose the OK button.

6. In the Add Unlisted or Updated Driver dialog box, select the "[MCI] Multimedia Movie Player" driver and choose the OK button.

The driver is now installed. You should now be able to run the animation portion of the MCITEST program.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: PrgCtrlsCus

"Cannot Find MMSYSTEM.DLL" Loading VB MCI.VBX in Windows 3.0

Article ID: Q83758

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY
=======

The Microsoft Professional Toolkit for Microsoft Visual Basic version 1.0 for Windows comes with a custom control that gives you easy access to writing applications for multimedia. Without Windows 3.1 or Multimedia Extensions for Windows 3.0, you will not be able to load the MCI.VBX custom control file into the Visual Basic programming environment.

If you try to load MCI.VBX into Windows 3.0 without Multimedia Extensions, you will receive the following error message:

```
   Cannot find MMSYSTEM.DLL, Please insert in drive A:
```

For more information about Multimedia Extensions for Windows 3.0, contact your local subsidiary, or call Microsoft End User Sales and Service at (800) 426-9400. A better solution is to upgrade to Windows version 3.1.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

PENCNTRL.VBX Err: Requires Microsoft Windows for Pen Computing

Article ID: Q83800

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Microsoft Professional Edition of Microsoft Visual Basic versions 2.0 and 3.0 for Windows, and Microsoft Professional Toolkit for Microsoft Visual Basic programming system version 1.0 for Windows, includes a custom control that gives you easy access to writing applications for Microsoft Windows for Pen Computing. Without Microsoft Windows for Pen Computing, PENCNTRL.VBX cannot be loaded into the Visual Basic for Windows programming environment.

If you try to load the PENCNTRL.VBX custom control without having installed Microsoft Windows for Pen Computing, the process will abort with the following message box:

> This program requires Microsoft Windows for Pen Computing

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

For more information about Microsoft Windows for Pen Computing, call Microsoft End User Sales and Service at (800) 426-9400. If calling from outside the United States, contact your local Microsoft subsidiary.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

PRB: MDI Child Cannot Be Maximized/Minimized While Invisible

Article ID: Q83803

The information in this article applies to:

- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

If you try to set an MDI Child custom control WindowState property to Maximized (2) or Minimized (1) while its Visible property is set to False (invisible), you will receive one of the following errors:

MDIChild cannot be maximized while invisible

-or-

MDIChild cannot be minimized while invisible

RESOLUTION

An MDI Child control with a WindowState property set to Maximized (2) or Minimized (1) will revert to Normal (0) if the control's Visible property is set to False. You must determine if you want the MDI Child control WindowState property set to Maximized when you make the MDI Child visible.

This article does not apply to later versions of Visual Basic. The MDI Child custom control shipped only with version 1.0. Multiple-document interface (MDI) forms are built into Visual Basic version 2.0 and later, making the MDI custom control obsolete.

STATUS

This behavior is by design.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: PrgCtrlsCus

PRB: MDI Child Custom Control: ScaleMode Defaults to Twips

Article ID: Q83905

The information in this article applies to:

- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

The MDI Child custom control (MDICHILD.VBX) does not have a ScaleMode property. Therefore, child controls of an MDI Child control will default to twips, regardless of whether the child control has a ScaleMode property. To confirm this, you can check the Height and Width properties of the child controls.

CAUSE

Some controls in Visual Basic have a ScaleMode property (for example, picture box), while other controls do not (for example, text boxes, label controls, and command buttons).

A child control (a control placed within other controls) takes many of its properties from the parent control. In addition, the default ScaleMode for Visual Basic is twips. Because the MDI Child control does not have its own ScaleMode property, it takes the default Visual Basic ScaleMode (twips). As a result, whenever you make a control a child of an MDI Child control, it uses twips as the ScaleMode for its dimensions (for example, Height and Width properties). However, if you place a control on a control that is already a child and whose ScaleMode is set to pixels, it will use pixels as the default ScaleMode. The example below illustrates this.

WORKAROUND

You can work around the problem by placing a control that has a ScaleMode property (such as a picture control) inside the MDI Child custom control, change its ScaleMode to something else (for example, pixels), then place subsequent controls inside it rather than in the MDI Child custom control.

RESOLUTION

This article does not apply to later versions of Visual Basic. The MDI Child custom control shipped only with version 1.0. Multiple-document interface (MDI) forms are built into Visual Basic version 2.0 and later, making the MDI custom control obsolete.

STATUS

This behavior is by design.

MORE INFORMATION
==========

Steps to Reproduce Problem
---

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the MDICHILD.VBX custom control file. The MDI Child tool appears in the Toolbox.

3. Place an MDI Child control (MDIChild1) on Form1.

4. Size the MDI control for a larger viewable area.

5. Place a picture control in the MDI control. To do this, click the picture control tool in the Toolbox. Place the mouse cursor in the MDI control. Notice that the cursor changes to a cross hair when you move it over Form1. Place the cross hair in MDIChild1, and size the picture control accordingly.

   Notice that the Height and Width properties of MDIChild1 and Picture1 are in twips.

6. Click Form1 to give it the focus. Set the ScaleMode property to 3 - Pixel.

Notice that the Height and Width properties of MDIChild1 are now expressed in pixels, while the Height and Width properties of the Picture control inside MDIChild1 are still expressed in twips.

If you change the ScaleMode of Picture1 to 3 - Pixel and place a command button control in Picture1, the Height and Width properties of Command1 will be expressed in pixels. The workaround above uses this method to work around the MDI Child control's limitation.

Additional reference words: 1.00 2.00 MDIChild MDI Child
KBCategory:
KBSubcategory: PrgCtrlsCus

VB Graph Custom Control: DataReset Property Resets to 0 (Zero)

Article ID: Q84058

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

When you assign a value to the DataReset property of the Graph version 1.2 custom control, the value of DataReset always resets to 0 - None. This is by design. Although DataReset is listed in the Properties box, it also has characteristics of a method. A value assigned to DataReset is transient, which means that it causes a one-time action and then resets to 0 - None.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

In Visual Basic for Windows, a property is an attribute of the control that you can set to define one of the object's characteristics. DataReset is a property because you can set its value which, depending on that value, defines one or more of the Graph control's characteristics. Because it defines a Graph's characteristics by resetting the chosen property array to its default values, DataReset is found in the Properties list box.

A method in Visual Basic for Windows behaves similarly to a statement in that it always acts on an object. DataReset can also be considered a method because it does perform an action on the graph. Namely, it resets the chosen property array to its default values. DataReset performs the assigned action as soon as its value does not equal 0. If it retained its assigned value, it would continually generate an endless loop and lock the system. To prevent this from occurring, it is automatically reset to 0 - None upon the first execution of its call.

The example below demonstrates the behavior of DataReset.

Example

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the GRAPH.VBX custom control file. The Graph tool will appear in the Toolbox.

3. Add a Graph control (Graph1) to Form1.

4. In the Properties list box, select the DataReset property. The value that appears in the Settings box will be 0 - None.

5. Change the value of DataReset to a number between 1 and 9. The values 1-9 refer to Graph property arrays that can be reset by using the DataReset property.

6. Graph1 will update to display the default values in the property array you chose in step 5.

7. In the Properties list box, select DataReset. The value of DataReset is 0 - None. It did not retain the value from step 5.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

How to Use VB Graph Control to Graph Data from Grid Control

Article ID: Q84063

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article contains an example of how to use a Graph custom control to graph the data contained in a Grid custom control.

In order to use either the Grid or the Graph control, you must add them to the Toolbox in the Visual Basic for Windows environment (in VB.EXE). You do this by selecting Add File from the File menu. From here select the Graph.VBX file, and then repeat the process for Grid.VBX. Graph.VBX and Grid.VBX should be found in your Windows\System directory.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

To create the example, do the following:

1. Run Visual Basic for Windows, or from the File menu, choose New Project (ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the GRAPH.VBX custom control file. The Graph tool appears in the Toolbox.

3. Repeat step 2 for the GRID.VBX custom control file.

4. Add a Grid control (Grid1), a Graph control (Graph1), and a command button (Command1) to Form1.

5. In the Load event for Form1, add the following code:

```
Sub Form_Load ()
    ' This Sub will do all the configuration for the Grid.
    ConfigureGrid
    ' This Sub will do all the configuration for the Graph.
    ConfigureGraph
End Sub
```

6. Create the following subroutine in the general Declarations section of Form1 to make it callable from anywhere in the form:

```
Sub ConfigureGrid ()

    ' Set the number of cols and rows for the grid.
    Grid1.Rows = 11
    Grid1.Cols = 4

    ' Set the alignment for the fixed col to centered.
    Grid1.FixedAlignment(0) = 2

    ' Set the alignment for the variable cols to centered.
    Grid1.ColAlignment(1) = 2
    Grid1.ColAlignment(2) = 2
    Grid1.ColAlignment(3) = 2

    Grid1.ScrollBars = 0

    ' Add the row labels.
    Grid1.Col = 0
    For i = 1 To 10
        Grid1.Row = i
        Grid1.Text = Str$(i)
    Next i

     ' Add the Col labels.
    Grid1.Row = 0
        Grid1.Col = 1
        Grid1.Text = "May"
        Grid1.Col = 2
        Grid1.Text = "June"
        Grid1.Col = 3
        Grid1.Text = "July"

  ' Set the starting cell on the Grid.
    Grid1.Row = 1
    Grid1.Col = 1
End Sub
```

7. Create the following subroutine in the general Declarations section of Form1 to make it callable from anywhere on the form:

```
Sub ConfigureGraph ()
  ' Set the Graph to auto increment.
    Graph1.AutoInc = 1
    Graph1.BottomTitle = "Months"
    Graph1.GraphCaption = "Graph Caption"
  ' Set the number of data groupings.
    Graph1.NumPoints = 10
  ' Set the number of data points per group.
    Graph1.NumSets = 3
End Sub
```

8. Place the following line of code into the KeyPress event for Grid1:

```
Sub Grid1_KeyPress (KeyAscii As Integer)
  ' This adds each keystroke to the data in the current cell.
    Grid1.Text = Grid1.Text + Chr$(KeyAscii)
```

```
    End Sub
```

9. For the Click event of Command1, enter the following code:

```
    Sub Command1_Click ()
   ' This Sub graphs the data in the Grid using the Graph control.
      ' Set the graph to the first point.
        Graph1.ThisSet = 1
        Graph1.ThisPoint = 1
    ' Load the GraphData array with all the values from the Grid,
    '    in order.
        For i = 1 To 3
            For j = 1 To 10
                Grid1.Row = j
                Grid1.Col = i
                Graph1.GraphData = Val(Grid1.Text)
            Next j
        Next i

      ' This actually graphs the array to the Graph control.
        Graph1.DrawMode = 2
    End Sub
```

This example will give you a grid with three columns (Months) and 10 rows. After you enter the data into the columns, choose the command button (with the mouse or keys). The data will be taken from the grid and graphed as a line graph.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

How to Read Flag Property of VB Common Dialog Custom Controls

Article ID: Q84068

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Flags property of a Common Dialog control can be read by examining individual bit values of the Flag property and comparing them with the predefined constant values in CONSTANT.TXT (or CONST2.TXT for Visual Basic version 1.0 for Windows). This applies to the following Visual Basic for Windows Common Dialogs:

- File Open Dialog
- File Save Dialog
- Color Dialog
- Choose Font Dialog
- Print Dialog

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The Flags property can be set at design time or run time.

To set the value of the Flags property, assign it a value. This is most commonly done using a predefined constant (found in CONSTANT.TXT or CONST2.TXT). For example, to set the PRINTTOFILE flag on the Print Dialog box, use the following code:

```
   CMDialog1.Flags = PD_PRINTTOFILE
```

To set more than one flag, OR the two flags (the pipe [|] character acts the same as the OR statement):

```
   CMDialog1.Flags = PD_PRINTTOFILE | PD_SHOWHELP
```

The settings of the Flags property can also be changed at run time by the user making various selections in the dialog box. When a selection is made, or the status of a check box or option button is changed, the Flags property reflects this change. You can then read the value of the Flags property and determine if a specific flag has been set.

For example, in the above sample code, two flags are set in the Flags property. The value of PD_PRINTTOFILE = &H00000020& and the value of PD_SHOWHELP = &H00000800&.

The binary equivalent of the two is the following:

```
   PD_PRINTTOFILE = 00000000000000000000000000100000
   PD_SHOWHELP    = 00000000000000000000100000000000
```

Thus the value:

```
   Flags          = 00000000000000000000100000100000
```

Note how each flag setting has its own bit setting within the Flags property.

To determine if a specific flag is set, you only need to AND the flag with the Flags property. If the result is 0, then the flag is not set; if the result is the same as the flag value, then the flag is set.

For example:

```
   Form1.Print (CMDialog1.Flags AND PD_PRINTTOFILE)
```

The output is decimal 32. Thus, broken down:

```
   Flags          = 00000000000000000000100000100000
                             AND
   PD_PRINTTOFILE = 00000000000000000000000000100000
                    ----------------------------------
   Result         = 00000000000000000000000000100000
```

Thus, the flag for PRINTTOFILE is one of the flags that are set in the Flags property:

```
If  (CMDialog1.Flags AND PD_PRINTTOFILE) Then
   ' Code for printing to file goes here.
Else
   ' Code for printing to printer goes here.
End If
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

How to Create Column and Row Labels in VB Grid Custom Control

Article ID: Q84113

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The example program below demonstrates how you can display labels in the top row and left column of the Grid custom control at run time. It is not possible to assign labels in a grid at design time.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The example program below assigns labels to a grid from the Form_Load event procedure. It puts numbers down the left, labeling the first non-fixed row as "1". It puts letters across the top, labeling the first 26 non-fixed columns as "A" through "Z" then subsequent columns with "AA", "AB", and so on.

Steps to Create Example Program

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the GRID.VBX. The Grid tool appears in the Toolbox.

3. Select the Grid tool from the Toolbox, and place a grid (Grid1) on Form1.

4. On the Properties bar, set the Grid Cols and Rows properties to 30.

5. Double-click the form to open the Code window. In the Procedure box, select Load. Enter the following code:

```
Sub Form_Load ()
    Dim i As Integer

    ' Make sure grid has at least one fixed column and row.
    If Grid1.FixedCols < 1 Or Grid1.FixedRows < 1 Then
        Stop
    End If
```

```
    ' Put letters across top.
    For i = 0 To Grid1.Cols - 2
        Grid1.Col = i + 1
        Grid1.Row = 0
        Grid1.Text = Chr$(i Mod 26 + Asc("A"))
        ' If more than 26 columns, use double letter labels.
        If i + Asc("A") > Asc("Z") Then
            Grid1.Text = Chr$(i \ 26 - 1 + Asc("A")) + Grid1.text
        End If
        Grid1.FixedAlignment(Grid1.Col) = 2  ' Centered.
    Next

    ' Put numbers down left edge.
    For i = 1 To Grid1.Rows - 1
        Grid1.Col = 0
        Grid1.Row = i
        Grid1.Text = Format$(i)
    Next
    Grid1.FixedAlignment(0) = 2  ' Centered.
End Sub
```

6. Press the F5 key to run the program.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

VB MCI Control Does Not Support PC Speaker Driver

Article ID: Q84268

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The MCI custom does not support playing wave (.WAV) sound files through a PC speaker driver such as SPEAKER.DRV. The MCI custom control (and the Windows Media Player application) uses the MCI sound drivers, which do not support the PC speaker. The Windows default sounds and the Sound Recorder application are the only way to play sounds through the SPEAKER.DRV PC speaker driver.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The MCI control manages the recording and playback of multimedia files on Media Control Interface (MCI) devices, such as audio boards, MIDI sequencers, CD-ROM drives, audio CD players, video disc players, and videotape recorders and players.

Although the MCI control will not allow you to play .WAV files through the PC speaker, you can use the Object Linked and Embedding (OLE) Client custom control provided with the Professional Edition of the Microsoft Visual Basic for Windows, or with the Microsoft Visual Basic for Windows Professional Toolkit to create and play a linked Sound Recorder object from your Visual Basic for Windows program. The following is an example of this behavior. (Note that you must have the appropriate Windows sound drivers loaded in order to run this program):

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the OLECLIEN.VBX custom control file. The OLE Client tool appears in the Toolbox.

3. Double-click the OLE Client control on the tool bar to create an OLE Client control on your form.

4. Double-click the form to open the Code window, and enter the following code in the Form_Click event:

```
  OLEClient1.Class = "SoundRec"
  OLEClient1.Protocol = "StdFileEditing"
  OLEClient1.SourceDoc = "C:\windows\chimes.wav"   ' Name of .WAV file.
  OLEClient1.SourceItem = "LINK"
  OLEClient1.ServerType = 0      ' Linked object.

  OLEClient1.Action = 1      ' Create object from source file.
  OLEClient1.Action = 7      ' Activate Sound Recorder - plays sound.
  OLEClient1.Action = 10     ' Delete the object.
```

5. Press the F5 key to run the program.

The specified .WAV file should be played each time you click the form.

For more information on SPEAKER.DRV, query on the following words in the Microsoft Knowledge Base:

```
   SPEAKER.DRV and WDL AND windows AND 3.10
```

Additional reference words: 1.00 2.00 3.00 MCI.VBX
KBCategory:
KBSubcategory: PrgCtrlsCus

VB MCI Control Does Not Support Recording of MIDI Data

Article ID: Q84473

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Multimedia Device control called MCI (MCI.VBX), consists of a set of high level, device-independent commands that control audio and visual peripherals. However, the MCI control cannot record standard MIDI (Musical Instrument Data Interface) input. This is a limitation of the MCI control, not of Visual Basic for Windows.

Below is an example of using the MCI control to play back a MIDI file.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The MCI custom control can play back MIDI files if you have the necessary hardware and software installed. Typically, you need a sound board that supports MIDI and Windows, version 3.1 to use the MCI control to play back MIDI files. Windows 3.1 or (Windows 3.0 with Multimedia Extensions version 1.0) supplies MIDI drivers for several well-known hardware add-on boards that support MIDI.

The following is an example of using the MCI control to play back a MIDI file called TEST.MID.

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the MCI.VBX custom control file. The MCI tool appears in the Toolbox.

3. Add the following code to the Form_Load event procedure:

```
Sub Form_Load ()
  MMControl1.PlayVisible = -1
  MMControl1.StopVisible = -1
  MMControl1.FileName = "c:\midi\bach.mid"
  MMControl1.Wait = -1
  MMControl1.DeviceType = "sequencer"
  MMControl1.Command = "open"
End Sub
```

4. Add the following code to your Form_Unload event procedure:

```
Sub Form_Unload (Cancel As Integer)
  MMControl1.Command = "close"
End Sub
```

5. Press the F5 key to run the program. Click the play arrow of the MCI control to play the MIDI file.

Note: An MIDI file may play, but may not be audible due to MIDI configuration issues such as the MIDI channel and instrument.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

VB Grid Custom Control Refreshes on All Cell Change Events

Article ID: Q84584

The information in this article applies to:

- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

The Grid custom control (GRID.VBX) will refresh (update the control's contents) on all change events occurring to cells in the grid. So, for example, when you enter text into a cell in the grid, a refresh of the grid occurs after every letter of a word is entered. This behavior is by design.

MORE INFORMATION

It is normal behavior for the Grid control to refresh whenever a change occurs to a cell contained in it. This is desirable behavior, because it ensures that current information is always displayed in the grid.

However, slowdowns due to the refreshing time can be a problem. If a grid is large enough, it can take a significant amount of time to refresh it. If there is a large number of data items to enter, the wait is compounded. There is no way to toggle the refresh of the grid when text is entered into a cell; it always occurs. However, there are methods to minimize the number of change events that occur to the grid, thus minimizing the wait. Two of these methods are shown below.

Steps to Reproduce Behavior

1. Start Visual Basic for Windows or from the File menu, choose New Project (ALT F N) if Visual Basic for Windows is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the GRID.VBX custom control file. The Grid tool appears in the Toolbox.

3. Add a Grid control to default Form1 by double-clicking its icon in the Toolbox. Also add a text box control in the same manner.

4. Set the following properties for Grid1: Cols = 10, Rows = 20. Size the grid so that you can see all the columns and rows. Also, set the Text property of Text1 to "" (blank).

5. Add the following code:

   ```
   Sub Form_Load ()
   ```

```
   Form1.Show               ' This code fills the grid with ASCII values
   For columns = 0 To 9     ' to show the effect of refreshing Grid1.
      For rows = 0 To 9
         Grid1.Row = rows
         Grid1.Col = columns
         Grid1.Text = Chr$(63 + rows + columns)
      Next rows
   Next columns
   Text1.SetFocus
End Sub

Sub Text1_KeyPress (KeyAscii As Integer)
   Grid1.Text = Text1.Text  ' This sets the contents of Grid1.Text to
End Sub                     ' what is entered into Text1.
```

6. Press the F5 key to run the program.

   Enter some text into Text1. Notice how every entry on the keyboard causes the grid to update. You can tell this is occurring by the flickering of the contents of Grid1 on every key press.

   If direct entry of data into a cell is desired, a slight modification to the code above significantly reduces the number of times the grid refreshes. The code below allows entry of text into a text box, and the contents are transferred to a cell in the grid when the user presses the ENTER key. To demonstrate this behavior, change the code in the Text1_KeyPress event to the following:

```
Sub Text1_KeyPress (KeyAscii As Integer)
   If KeyAscii = 13 Then       ' Did the user press the ENTER key?
      Grid1.Text = Text1.Text  ' Yes - assign Text1.Text to Grid1.Text.
      KeyAscii = 0             ' Suppresses the default "beep" sound.
      Text1.Text = ""          ' Clear the text box for the next entry.
   End If
End Sub
```

   This change filters the input somewhat by only updating Grid1.Text when the user presses the ENTER key. (If you want to change it to some other value, use a different KeyAscii value.) The benefit of this method is that an update only occurs whenever the user presses the ENTER key, not on every key press event.

   Yet another alternative is to first store the data entered in the text box into an array. Then, when data entry is complete, transfer the contents of the array to the grid. This forces all changes to the grid to be done in one refresh, thus reducing the total waiting time required for the grid to refresh. To accomplish this, do the following:

7. Add a command button to Form1. Set the Caption property to "Place array items into grid".

8. Add the following code to the general Declarations section of Form1:

```
Dim Words$(100)
Dim GridNum As Integer
' (Add the following to the Command1 Click event procedure:)
```

```
Sub Command1_Click ()
   For y = 0 To (GridNum - 1)
      Grid1.Row = Int(19 * Rnd + 1) ' Sets the row & column to a random
      Grid1.Col = Int(9 * Rnd + 1)  ' place in the grid, and prints the
      Grid1.Text = Words$(y)        ' item there.
   Next y
   Erase Words$             ' Clears the array.
   GridNum = 0              ' Resets the array item counter.
   Text1.SetFocus           ' Sets the focus back to the text box.
End Sub
```

9. Replace the code in the Text1_KeyPress event of Form1 with the following:

```
Sub Text1_KeyPress (KeyAscii As Integer)
   If KeyAscii = 13 Then
      Words$(GridNum) = Text1.Text ' Transfers contents of Text1.Text
                                   ' to a string array.
      Text1.Text = ""
      GridNum = GridNum + 1  ' Increments the array item counter to
                             ' prepare for the next word to be
                             ' entered.
      Debug.Print GridNum    ' Prints the current record number in
                             ' the immediate window. (optional)
      KeyAscii = 0
   End If
End Sub
```

10. Press the F5 key to run the program. Enter a few words, pressing the ENTER key after each word. Notice that the grid does not refresh after the ENTER key is pressed. The items are being placed into an array with each press of the ENTER key. When you are finished, choose Command1 to place the new items in the grid. The grid will refresh only once now, as the new items are randomly placed in the grid.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

VB Graph: Use XPosData to Plot Fractional X-Axis Values

Article ID: Q85264

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can use the Graph custom control (GRAPH.VBX) XPosData property to give independent X-axis values for graphs. That is, the graph is not forced to plot on the X-axis, and has the ability to plot fractional data points between X-axis values. This property can be used with all graphs types except pie.

MORE INFORMATION

By default, graphs use a dependent X-axis scale. This means data points plotted conform to the whole-number increments shown on the X-axis. To plot fractional X-axis values, use the XPosData property. The XPosData values are set for each data point, allowing fractional X-axis plotted values to appear in the graph. XPosData sets the X coordinate, and GraphData sets the Y coordinate. The example below demonstrates this by plotting three different data points on a scatter graph:

Example

1. Run Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the GRAPH.VBX custom control file. The GRAPH tool appears in the Toolbox.

3. Place a Graph (Graph1) on Form1 by double-clicking the Graph tool in the Toolbox.

4. In the Properties box, set the following properties for Graph1:

   - AutoInc = 0 (Off)
   - GraphType = 9 (Scatter)
   - NumPoints = 3
   - SymbolData = 3 (Solid triangle - up)

5. Add the following code to Form1:

```
Sub Form_Load ()
```

```
      Graph1.ThisPoint = 1   'This indicates which datapoint to work on
      Graph1.GraphData = 10  'This sets the Y-axis value for this point
      Graph1.XPosData = .2   'This sets the X-axis value for this point

      Graph1.ThisPoint = 2
      Graph1.GraphData = 5
      Graph1.XPosData = 1.3

      Graph1.ThisPoint = 3
      Graph1.GraphData = 3
      Graph1.XPosData = 2.4

   End Sub
```

6. Press F5 to run the code.

This example, when run, plots three data points in (X,Y) format. In this case, XPosData is used to provide non-integer X-axis values. The three triangles are plotted using the following coordinates:

   (.2, 10), (1.3, 5), (2.4, 3)

XPosData works for other graph types too, except pie, for which X-axis data has no meaning. To try this example with another graph type, change the GraphType property of Grid1 to "4 - 3D Bar". Notice how the bars orient against the X-axis when you run the code.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

Toolkit 3-D Control (THREED.VBX) Default Property Values

Article ID: Q87766

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The 3-D Check Box, 3-D Command Button, 3-D Frame, 3-D Group Push Button, 3-D Option Button, and 3-D Panel custom controls retain custom properties from the control drawn before them. When the properties are customized for a particular 3-D control, they become the default properties for subsequent 3-D controls of the same type.

MORE INFORMATION

The example below demonstrates that the properties of THREED controls are retained from one control to the next.

Step-by-Step Example

1. Load the THREED.VBX file into Form1 in Visual Basic.

2. Add a 3-D Command Button.

3. Select the BevelWidth property from the property bar. The default value is 2.

4. Set the BevelWidth to a value of 4.

5. Add a second 3-D Command Button.

6. Select the BevelWidth property from the property bar. The default value will now be 4.

Additional reference words: 1.00 2.00 3.00 three dimension 3d
KBCategory:
KBSubcategory: PrgCtrlsCus

Using a Linked Sound Recorder Object with OLECLIEN.VBX

Article ID: Q87768

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The following program demonstrates the use of the Microsoft Visual Basic OLECLIEN.VBX custom control to create a linked Sound Recorder object.

The following OLEClient property settings are required to create a Sound OLE object:

Setting	Definition
Class	- "SoundRec"
SourceDoc	- The full path of the "wave" file to use (for example: C:\WINDOWS\CHIMES.WAV)
SourceItem	- The type of sound file object. "Wave" is the only sound format supported by the Windows operating system version 3.1 Sound Recorder.

Note: Sound Recorder does not come with Microsoft Windows version 3.0. You must have Windows version 3.1 to use this example. You must also have a computer capable of playing wave audio sounds (.WAV files). If you do not have a sound board, you can obtain a Windows sound driver for your PC Speaker. For information on obtaining this driver, query on the following words in the Microsoft Knowledge Base:

```
win31 and driver and speak.exe
```

MORE INFORMATION

The following program demonstrates how to create a linked Sound Recorder object in Microsoft Visual Basic for Windows by using the OLECLIEN.VBX custom control:

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the OLECLIEN.VBX custom control file. The OLE Client tool appears in the Toolbox.

3. Place a command button and an OLEClient control on Form1.

4. Enter the following code:

```
Sub Command1_Click ()
   OLEClient1.Class = "SoundRec"
   OLEClient1.Protocol = "StdFileEditing"
   OLEClient1.SourceDoc = "c:\windows\chimes.wav"

   '  Source Item for Sound Recorder is 'Wave', but
   '  Sound Recorder does not check this property so
   '  any value will do.
   OLEClient1.SourceItem = "Wave"

   OLEClient1.ServerType = 0  ' Linked.
   OLEClient1.Action = 1      ' CreateFromFile.
   Command1.Enabled = 0
End Sub

Sub OleClient1_DblClick ()
   OLEClient1.Action = 7  ' Activate (open for editing).
End Sub

Sub Form_Unload (Cancel As Integer)
   OLEClient1.Action = 9  ' Close (terminate connection).
End Sub
```

5. Press the F5 key to run the program. Choose the Command button to create the OLE object. Double clicking the OLEClient control starts Sound Recorder and plays the OLE sound.

Reference(s):

"Microsoft Professional Toolkit for Visual Basic: Custom Control Reference," pages 196-232

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: PrgCtrlsCus

PRB: THREED Check Box Is Not Grayed Out When Value = 2 in VB

Article ID: Q87771

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

If the Value property of a THREED Check Box is set to 2, the check box is not made unavailable (grayed out), as you might expect. Instead, an X is displayed in the THREED Check Box. If the Value property of a standard Visual Basic for Windows check box is set to 2, the check box is made unavailable.

CAUSE

There is no disabled state for a THREED check box. The value property of a THREED check box can only be true or false (0 or 1) whereas the standard check box can have a value of 0, 1, or 2.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Reproduce Behavior

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the THREED.VBX custom control file.

3. Place one THREED Check Box, one standard check box, and three command buttons on Form1.

4. Enter the following code in the appropriate event procedures:

```
Sub Form_Load ()
   Command1.Caption = "Value = 0"
   Command2.Caption = "Value = 1"
   Command3.Caption = "Value = 2"
End Sub
```

```
Sub Command1_Click ()
   Check1.Value = 0
   Check3D1.Value = 0
End Sub

Sub Command2_Click ()
   Check1.Value = 1
   Check3D1.Value = 1
End Sub

Sub Command3_Click ()
   Check1.Value = 2
   Check3D1.Value = 2
End Sub
```

5. Press F5 to run the program. First, click the Value = 0 button. Then click the Value = 1 button. Finally, click the Value = 2 button. When you click the Value = 2 button, the standard check box is disabled (grayed) but the THREED check box is not.

Additional reference words: 1.00 2.00 3.00 grey greyed 3d
KBCategory:
KBSubcategory: PrgCtrlsCus

How to Clear All or Part of Grid in Visual Basic

Article ID: Q88911

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.00
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can clear all or part of a grid by first selecting the region to clear using the SelStartCol, SelStartRow, SelEndCol, and SelEndRow properties, and then clearing the region by assigning a null string to the Clip property.

MORE INFORMATION

The example below demonstrates how to clear all the non-fixed cells of a grid.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File, and select GRID.VBX. The Grid tool appears in the Toolbox. (This step is automatic in version 2.0.)

3. Place a grid (Grid1) on Form1.

4. Set the Grid1 property Cols to 4, and Rows to 4. Size the grid so that you can see all the cells.

5. Enter the following code into the Form1 Load event procedure:

```
Sub Form_Load ()
   ' Load some data into the grid.
   For i% = Grid1.FixedCols To Grid1.Cols - 1
      For j% = Grid1.FixedRows To Grid1.Rows - 1
         Grid1.Col = i%
         Grid1.Row = j%
         Grid1.Text = Format$(i% + j%)
      Next
   Next
End Sub
```

6. Enter the following code into the Form1 Click event procedure:

```
Sub Form_Click ()
   ' Select all non-fixed grid cells.
```

```
      Grid1.SelStartCol = Grid1.FixedCols
      Grid1.SelStartRow = Grid1.FixedRows
      Grid1.SelEndCol = Grid1.Cols - 1
      Grid1.SelEndRow = Grid1.Rows - 1

      ' Clear the cells.
      Grid1.Clip = ""

      ' Clean up the grid.
      Grid1.Col = Grid1.FixedCols
      Grid1.Row = Grid1.FixedRows
      Grid1.SelEndCol = Grid1.SelStartCol
      Grid1.SelEndRow = Grid1.SelStartRow
   End Sub
```

7. Press F5 to run the program. The grid appears with numbers in the cells. Click Form1. The grid is cleared.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

How to Make a Spreadsheet-Style Grid that Allows Editing

Article ID: Q88912

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.00
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY
=======

The Grid custom control does not provide any text editing capability. However, you can create a spreadsheet-style grid that allows editing by using a picture box and a text box.

MORE INFORMATION
================

We do not recommend creating a spreadsheet-style grid with a large matrix of text box controls because doing so will slow down your program, and use excessive system resources.

An efficient way to create a grid is to draw vertical and horizontal lines to represent the cells of the grid. Use a single text box to allow editing of the active cell. Check for MouseDown events to move the text box to the currently active cell position, and use the Print method to draw text in a cell when the text box moves away from the cell. Then, store the grid cell values in a two dimensional array, indexed by the column and row.

Code can be added to allow for highlighting areas, using ARROW keys to move between cells, and so on.

Step-by-Step Example
----

1. Start Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Place a picture (Picture1) on Form1, and set its properties as follows:

Property	Value
AutoRedraw	True
ScaleMode	3 - Pixel
Height	2000
Width	3000

3. Place a text box (Text1) in Picture1 by clicking the text box tool. The mouse pointer turns to cross-hairs. Click and drag inside Picture1 to place a gray rectangle appears in Picture1.

4. Add the following code to the general Declarations section of Form1:

```
' Maximum grid size.
Const grid_col_max = 10
Const grid_row_max = 20

' Current grid size.
Dim grid_cols As Integer
Dim grid_rows As Integer

' Current cell position.
Dim grid_col As Integer
Dim grid_row As Integer

' Grid string contents.
Dim grid_text(grid_col_max, grid_row_max) As String

' Grid line positions.
Dim grid_line_col(grid_col_max) As Integer
Dim grid_line_row(grid_col_max) As Integer

' grid_edit_move.
'   Moves the grid edit text box to a new position.
'
Sub grid_edit_move (col As Integer, row As Integer)
   Dim x1 As Integer  ' Picture box positions.
   Dim y1 As Integer
   Dim x2 As Integer
   Dim y2 As Integer

   ' Save text box contents to grid array.
   grid_text(grid_col, grid_row) = Text1.Text

   ' Clear current cell.
   x1 = grid_line_col(grid_col) + 1
   y1 = grid_line_row(grid_row) + 1
   x2 = grid_line_col(grid_col + 1) - 1
   y2 = grid_line_row(grid_row + 1) - 1
   Picture1.Line (x1, y1)-(x2, y2), Picture1.BackColor, BF

   ' Print text box contents to current cell.
   Picture1.CurrentX = x1 + 3
   Picture1.CurrentY = y1 + 3
   Picture1.Print Text1.Text

   ' Set new grid current cell.
   grid_col = col
   grid_row = row

   ' Move text box to new cell.
   x1 = grid_line_col(grid_col)
   y1 = grid_line_row(grid_row)
   w! = grid_line_col(grid_col + 1) - x1
   h! = grid_line_row(grid_row + 1) - y1
   Text1.Move x1 + 1, y1 + 1, w! - 1, h! - 1

   ' Copy contents of new cell to text box.
   Text1.Text = grid_text(grid_col, grid_row)
```

```
End Sub
```

5. Add the following code to form Load event procedure:

```
Sub Form_Load ()
   ' Set grid size.
   grid_cols = 4
   grid_rows = 6

   ' Remove border.
   Picture1.BorderStyle = 0

   ' Set column widths and row heights.
   Dim i As Integer
   Dim d As Integer
   d = 0
   For i = 0 To UBound(grid_line_col)
      grid_line_col(i) = d
      d = d + 40
   Next
   d = 0
   For i = 0 To UBound(grid_line_row)
      grid_line_row(i) = d
      d = d + 20
   Next

   ' Draw grid lines.
   For i = 0 To grid_cols
      x2% = grid_line_col(i)
      y2% = grid_line_row(grid_rows)
      Picture1.Line (grid_line_col(i), 0)-(x2%, y2%)
   Next
   For i = 0 To grid_rows
      x2% = grid_line_col(grid_cols)
      y2% = grid_line_row(i)
      Picture1.Line (0, grid_line_row(i))-(x2%, y2%)
   Next

   Call grid_edit_move(0, 0)
End Sub
```

6. Add the following code to the Picture1 GotFocus event procedure:

```
Sub Picture1_GotFocus ()
   Text1.SetFocus
End Sub
```

7. Add the following code to the Picture1 MouseDown event procedure:

```
' The following line should appear on one line.
Sub Picture1_MouseDown (Button As Integer, shift As Integer,
      x As Single, y As Single)
   Dim col As Integer
   Dim row As Integer
   Dim i As Integer

   ' Find the cell clicked in.
```

```
    col = grid_col
    row = grid_row
    For i = 0 To grid_cols - 1
      If x>=grid_line_col(i) And x<grid_line_col(i+1) Then
        col = i
        Exit For
      End If
    Next
    For i = 0 To grid_rows - 1
      If y>=grid_line_row(i) And y<grid_line_row(i+1) Then
        row = i
        Exit For
      End If
    Next

    ' Move the text box there.
    Call grid_edit_move(col, row)
  End Sub
```

8. Press F5 to run the program. Click a cell and edit the text.

This example is very limited in functionality. Text can be edited in each cell but you must click a cell to move to that particular cell. This article shows a method of creating a grid without tying up a large amount of system resources. Feel free to add code to increase its functionality.

Additional reference words: 1.00 2.00 3.00 optimize
KBCategory:
KBSubcategory: PrgCtrlsStd PrgCtrlsCus

Masked Edit Control, Mask Property Clarification

Article ID: Q93129

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

This article clarifies the description of the Masked Edit Control's Mask property given in "Microsoft Visual Basic Professional Features" on pages 233-234, and in the help file CTRLREF.HLP.

MORE INFORMATION

Example Mask Settings

The description of the Mask property includes the following statement:

> ..., the following standard, predefined input masks are available at design time.

The statement is followed by a list of possible Mask settings such as:

```
##-???-##    Medium date (US). Example: 20-May-92
```

The settings listed are example settings. The Masked Edit control does not handle these particular settings specially, either at design time or run time. In this sense, they are neither standard nor predefined as the description states.

For instance, the setting ##-???-## does not restrict the user to valid dates. This setting only requires two digits, three letters, and two more digits. So, for example, an input of 99ZZZ99 is valid with this setting.

Mask Character Place Holder "&"

The description of the place holder "&" is given as:

> & Character placeholder. The valid value for the placeholder is any symbol or alphanumeric character.

This means that "&" is a place holder for any printable character.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

Name Property Cannot Be Set When Using Implicit Property

Article ID: Q93214

The information in this article applies to:

- Microsoft Visual Basic for Windows, version 2.0

SUMMARY

On Page 126 of the Visual Basic Programmer's Guide, it incorrectly states that all controls have an implicit property you can use for storing or retrieving values. Some controls supplied with the Professional Edition of Visual Basic for Windows use the Name property as their implicit property, which you cannot use at run-time.

MORE INFORMATION

The following controls from the Visual Basic Professional Edition use the Name property as their implicit property:

   Common dialog
   MAPI session
   MAPI message
   Spin button

Attempting to access the implicit property of these controls results in one of the following errors:

   'Name' property cannot be read at run time
   'Name' property cannot be set at run time

You access the implicit property of a control (also known as the "value of a control" or the "default value of a control") by writing the control name with no property. For example, with a text box named Text1, you can write the following statement to assign a value to the Text property:

```
Text1 = "hello world"
```

The following list shows the implicit properties for all the controls in both the Standard and Professional Editions:

Standard Control	Implicit Property
Check box	Value
Combo box	Text
Command button	Value
Directory list box	Path
Drive list box	Drive
File list box	FileName
Frame	Caption
Grid	Text
Image	Picture

Label	Caption
Line	Visible
List box	Text
Menu	Enabled
OLE client	Action
Option button	Value
Picture box	Picture
Scroll bar vertical	Value
Scroll bar horizontal	Value
Shape	Shape
Text box	Text
Timer	Enabled

Professional Control	Implicit Property
3D check box	Value
3D command button	Value
3D frame	Caption
3D group push button	Value
3D option button	Value
3D panel	Caption
Animated button	Value
Common dialog	Name (not usable)
Communications	Input
Gauge	Value
Graph	QuickData
Key status	Value
MAPI session	Name (not usable)
MAPI message	Name (not usable)
Masked edit	Text
Multimedia MCI	Command
Pen BEdit	Text
Pen HEdit	Text
Pen ink on bitmap	Picture
Pen on-screen keyboard	Visible
Picture clip	Picture
Spin button	Name (not usable)

Additional reference words: 2.00 docerr
KBCategory:
KBSubcategory: RefsDoc PrgCtrlsStd PrgCtrlsCus

New Features Added to Graph Control in Versions 2.0 and 3.0

Article ID: Q93322

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY
=======

In Visual Basic versions 2.0 and 3.0, the Graph Control now includes most of the major features that were requested with the Microsoft Visual Basic Professional Toolkit for Windows, version 1.0.

For example, in versions 2.0 and 3.0, the user has control over the labeling of the X axis and the minimum and maximum range of the Y axis. More information on the Graph Control is included in the "Professional Features" manual provided with the Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0.

New features of the Graph Control include the following:

New Properties

```
   FontSize         'Standard FontSize property.
   FontStyle        'Standard FontStyle property.
   FontUse          'Other font properties are applied against.
   HelpContextID    'Help file topic ID.
   hWnd             'Window handle.
   IndexStyle       'Enhanced usage of ThisSet and ThisPoint.
   LabelEvery       'Frequency of labels on the X axis.
   TickEvery        'Tick interval on X axis.
   Ticks            'Check if X or Y axis ticks are displayed.
   YAxisMax         'Maximum range of Y axis.
   YAxisMin         'Minimum range of Y axis.
   YAxisPos         'Position of Y axis, right or left.
   YAxisStyle       'Auto or manual.
   YAxisTicks       'Number of ticks on Y axis.
```

New Events

Events are the same as the Graph Control included with the Microsoft Visual Basic Professional Toolkit for Windows, version 1.0.

New Methods

```
   ZOrder          'Standard ZOrder method.
```


Additional reference words: 2.00 3.00 z-order
KBCategory:
KBSubcategory: PrgCtrlsCus

Create .MMM Movie Files with Macromedia Director for Macintosh

Article ID: Q94186

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Microsoft does not provide any product that can create .MMM movie files (multimedia animation files for use with the MCI.VBX control).

You can use Macromedia Director for Macintosh to create multimedia animation and use Macromedia Windows Player to convert the animation to a file in MMM movie file format.

MORE INFORMATION

For more information, contact the following company:

Macromedia (previously Macromind), Inc.
600 Townsend St
San Francisco CA 94103
(800)288-4797

Reference(s):

"Microsoft Multimedia Development Kit Programmer's Reference" and
"Microsoft Multimedia Development Kit Programmer's Workbook"

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

MaxFileSize Property Range in CMDDIALOG.VBX Can Be 1 to 2048

Article ID: Q95765

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 2.0

SUMMARY

The MaxFileSize property for the file open and file save common dialog boxes has a range of 1 to 2048 bytes not 1 to 32767 bytes.

MORE INFORMATION

The 1 to 2048 bytes is an internal limit of the Windows COMMDLG.DLL.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

Maximum Length of Name Property Depends on Events Supported

Article ID: Q96151

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The maximum length of the Name (CtlName in version 1.0) property for controls varies from control to control depending on the character length of its longest event name. Event procedures names are limited to a length of 40 characters including the control's Name property, the underscore, and the event name. Therefore, the longer the event name, the shorter the Name property can be.

In Visual Basic versions 2.0 and 3.0, the Label, Picture Box, and Text Box controls add support for the LinkNotify event, which is one character longer than any event supported in version 1.0 for these controls. The maximum length of the Name property for these controls is therefore one character fewer in versions 2.0 and 3.0. A table showing the maximum length of the Name property for all of the standard controls in Visual Basic versions 2.0 and 3.0 is listed below.

MORE INFORMATION

The Name property of forms are not dependent on the events supported because the property is not used in the name of form event procedures. Event procedures for forms all begin with Form and therefore can be up to the 40-character maximum in length.

Maximum length of Name properties for Version 2.0 and 3.0 controls

Control	Name Length Limit
Check Box	30
Combo Box	30
Command Button	30
Directory List Box	30
Drive List Box	30
File List Box	26
Frame	31
Grid	27
Image	30
Label	29
Line	39
List Box	30
Menu	34
OLE Client	30
Option Button	30

Picture Box	25
Scroll Bars	30
Shape	39
Text Box	29
Timer	34

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

Set DrawMode to 2 Or 3 to Update Changes to Graph at Run Time

Article ID: Q96450

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Changing properties of the graph control at run time does not update the control until DrawMode is set to 2 (Draw) or 3 (Blit).

MORE INFORMATION

The DrawMode property documentation states that at design time, when you change a property value, the graph is automatically redrawn to show the effect of the change. At run time, the graph is only redrawn when you set DrawMode to 2 or 3. This allows you to change as many property values as you want before displaying the graph. However, when the form containing a graph is first displayed, the graph is automatically displayed according to the current DrawMode value.

For more information, see the "Microsoft Visual Basic Programming System for Windows Professional Edition Professional Features," version 2.0, "Custom Control Reference," pages 149-150.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

How to Right Justify Standard Numbers in a Masked Edit Field

Article ID: Q97141

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

The Masked edit control does not provide a method to right justify numbers. Ordinary methods and the Format$ function do not work because the Masked edit control uses the underscore character to represent blanks in the text property. For example, if 300 is entered in a Masked edit field with a mask of #####, the text property would contain "__300" instead of " 300."

However, you can use the technique described in this article to right justify a Masked edit field using a standard number mask and format. This is done in three steps:

1. Create a string of underscore characters that matches the length of the mask in the Masked edit control.

2. Concatenate the text entered in the Masked edit control to the end of' the underscore string. This result is a string longer than the mask of the Masked edit control.

3. Use the Right$ function to remove the extra underscore characters from the beginning of the string.

MORE INFORMATION

The following example demonstrates this process:

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the MSMASKED.VBX control to the project.

3. Create the following controls on Form1, and assign the indicated properties:

Default Name	Caption	Mask	Format
MaskedEdit1	(Not applicable)	####	####
Command1	Right Justify		

4. Add the following code to the Command1_Click event:

```
' Ensure that the string is not already right-justified.
If InStr((Len(MaskedEdit1.Text)), MaskedEdit1.Text, "_") =
   Len(MaskedEdit1.Text) Then
```

```
      ' The first String$ function creates the underscore string. The
      ' Format$ trims the text property of the MaskedEdit control.
      ' Enter the following two lines as one, single line:
      MaskedEdit1.text = Right$(String$(Len(MaskedEdit1.Text), "_") &
         Format$(Val(MaskedEdit1.Text)), Len(MaskedEdit1.Text))

   End If
```

5. Press the F5 key to run the program.

6. Enter two numbers into the Masked edit field, and click the Right Justify button. Notice that the numbers are right-justified in the field.

Additional reference words: 2.00 3.00 alignment align right-align
KBCategory:
KBSubcategory: PrgCtrlsCus

Playing an .AVI File with the MCITEST Example

Article ID: Q98769

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.00 and 3.00

SUMMARY

This article shows how to modify the multimedia sample project MCITEST.MAK to load and play Microsoft Video for Windows files (.AVI files). To apply the information in this article, you must have Video for Windows drivers correctly installed and a valid .AVI file available.

More Information:

1. Open the project VB\SAMPLES\MCI\MCITEST.MAK.

2. Select MCITEST.FRM in the Project Window. From the View menu, choose Code. In the Object combo box, select AI_ANIMATION to display the AI_ANIMATION_Click event handler.

3. Modify the common dialog Filter to display .AVI files:

   ```
   change:  OpenDlg.CMDialog1.Filter = "Movie File (*.mmm)|*.mmm"
       to:  OpenDlg.CMDialog1.Filter = "Movie File (*.avi)|*.avi"
   ```

   Modify the DeviceType to access the AVI drivers:

   ```
   change:  Animate.MMControl1.DeviceType = "MMMovie"
       to:  Animate.MMControl1.DeviceType = "AVIVideo"
   ```

4. Select ANIMATE.FRM in the Project Window. From the View menu, choose Code. In the Object combo box, select AI_OPEN to display the AI_OPEN_Click event handler.

5. Modify the DeviceType to access the AVI drivers. Scroll down by pages to find the location for this change.

   ```
   change:  Animate.MMControl1.DeviceType = "MMMovie"
       to:  Animate.MMControl1.DeviceType = "AVIVideo"
   ```

6. Save the work, and run the application.

Additional reference words: 2.00 3.00 mci control multimedia multi media
KBCategory:
KBSubcategory: PrgCtrlsCus

PRB: Some ATI Video Drivers Hang When Using MSOUTLIN.VBX

Article ID: Q100194

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic programming system for Windows, version 3.0

SYMPTOMS

If you use an OutLine control in a Visual Basic project and you are using an ATI Mach 32 video driver this could cause your computer to hang (stop responding to input).

CAUSE

This is a problem with the ATI video driver not a problem with Visual Basic. The m32-86.drv and Mach32.drv drivers have been reported to cause this problem.

RESOLUTION

An updated driver may solve the problem. To contact ATI Technologies concerning an updated driver call the following number.

ATI Technologies Inc. (416) 756-0711 ATI technical support

Additional reference words: 3.00
KBCategory:
KBSubcategory: EnvtRun PrgCtrlsCus

International and U.S. Support for Crystal Reports

Article ID: Q100368

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

Microsoft supports setup and installation for the Crystal Reports product shipped with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows. For other Crystal Reports support, please contact Crystal Services, not Microsoft.

MORE INFORMATION

The following lists international and U.S. telephone numbers you can call to get technical support for Crystal Reports. Also listed is the CompuServe ID and mailing address for Crystal Reports support.

Canada/US

Crystal Services
Suite 2200 - 1050 West Pender Street
Vancouver, BC, Canada V6E 3S7

Phone: 604-669-8379 (8:00am - 5:00pm pacific time)
Fax: 604-681-7163
BBS: 604-681-9516

Product support via CompuServe:
Send CompuServe mail to : 71035,2430

England

Company: Contemporary Software
Phone: 273-483-979
Fax: 273-486-224

Netherlands

Company: Microscope
Phone 10-456-3799
Fax 10-456-5549

Australia

Company: Sourceware
Phone: 2-427-7999
Fax: 2-427-7255
"Ask for Tony Johnson"

For a complete list of Crystal Reports support offerings see the last three pages (PSS 1 - PSS 3) of the "Microsoft Visual Basic Professional Features Book 2" manual

Additional reference words: 3.00
KBCategory:
KBSubcategory: RefsProd PrgCtrlsCus

How to Fill (Populate) a Grid with Database Data -- 4 Methods

Article ID: Q103437

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

This article gives you four separate examples demonstrating how to use Visual Basic to fill a grid control with data coming from database tables.

- The first example uses a data control to fill the grid.
- The second example uses a Dynaset object to fill the grid.
- The third example uses a Snapshot object to fill the grid.
- The fourth example uses a Table object to fill the grid.

MORE INFORMATION

Example One

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add one Data1 control, one Grid control, one Command button and two Text boxes to Form1.

3. Using the following table as a guide, set the properties of the controls you added in step 2.

Control Name	Property	New Value	Comment
Data1	DatabaseName	BIBLIO.MDB	Provide the full path to this file, which should be in the Visual Basic directory -- C:\VB
Data1	RecordSource	Authors	
Data1	Visible	False	
Text1	DataSource	Data1	
Text1	DataField	AU_ID	
Text1	Visible	False	
Text2	DataSource	Data1	
Text2	DataField	Author	
Text2	Visible	False	
Grid1	Cols	3	
Grid1	Rows	50	
Command1	Caption	Press to Load Grid	

4. Place the following code in the Form1 Load event procedure:

```
Sub Form_Load ()
```

```
   'Initialize the colwidths for the grid and supply headers
   Show
   grid1.ColWidth(1) = 3000      'For Author name
   grid1.ColWidth(2) = 1000      'For Author ID
   grid1.Col = 1
   grid1.Row = 0
   grid1.Text = "Author Name"    'Header for Author Name
   grid1.Col = 2
   grid1.Row = 0
   grid1.Text = "Author ID"      'Header for Author ID
End Sub
```

5. Place the following code in the Command1 Click event procedure:

```
Sub Command1_Click ()
   ' The routine to load data into grid
   Dim counter%
   counter% = 1                          'Start counter at Row=1
   Do Until data1.Recordset.EOF
      grid1.Col = 1
      grid1.Row = counter%
      grid1.Text = data1.Recordset(1)    'Load the Author Name
      grid1.Col = 2
      grid1.Row = counter%
      grid1.Text = data1.Recordset(0)    'Load the Author ID
      counter% = counter% + 1
      data1.Recordset.MoveNext
   Loop
   data1.Recordset.Close
End Sub
```

6. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program. Click the Command1 button.

Example Two
----

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add one Grid control and one Command button to Form1.

3. Using the following table as a guide, set the properties of the controls you added in step 2.

Control Name	Property	New Value
Grid1	Cols	3
Grid1	Rows	50
Command1	Caption	Press to Load Grid

4. Place the following code in the Form1 Load event procedure:

```
Sub Form_Load ()
   'Initialize the colwidths for the grid and supply headers
   Show
   grid1.ColWidth(1) = 3000      'For Author name
```

```
      grid1.ColWidth(2) = 1000      'For Author ID
      grid1.Col = 1
      grid1.Row = 0
      grid1.Text = "Author Name"    'Header for Author Name
      grid1.Col = 2
      grid1.Row = 0
      grid1.Text = "Author ID"      'Header for Author ID
   End Sub
```

5. Place the following code in the Command1 Click event procedure:

```
   Sub Command1_Click ()
      ' The routine to load data into grid
      Dim db as Database
      Dim ds as Dynaset
      Dim counter%
      Set db = OpenDatabase("BIBLIO.MDB")
      Set ds = db.CreateDynaset("Authors")
      counter% = 1                  'Start counter at Row=1
      Do Until ds.EOF
         grid1.Col = 1
         grid1.Row = counter%
         grid1.Text = ds(1)         'Load the Author Name
         grid1.Col = 2
         grid1.Row = counter%
         grid1.Text = ds(0)         'Load the Author ID
         counter% = counter% + 1
         ds.MoveNext
      Loop
      ds.Close
      db.Close
   End Sub
```

6. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program. Click the Command1 button.

Example Three
---

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add one Grid control and one Command button to Form1.

3. Using the following table as a guide, set the properties of the controls you added in step 2.

Control Name	Property	New Value
Grid1	Cols	3
Grid1	Rows	50
Command1	Caption	Press to Load Grid

4. Place the following code in the Form1 Load event procedure:

```
   Sub Form_Load ()
      'Initialize the colwidths for the grid and supply headers
```

```
   Show
   grid1.ColWidth(1) = 3000      'For Author name
   grid1.ColWidth(2) = 1000      'For Author ID
   grid1.Col = 1
   grid1.Row = 0
   grid1.Text = "Author Name"    'Header for Author Name
   grid1.Col = 2
   grid1.Row = 0
   grid1.Text = "Author ID"      'Header for Author ID
End Sub
```

5. Place the following code in the Command1 Click event procedure:

```
Sub Command1_Click ()
   ' The routine to load data into grid
   Dim db as Database
   Dim Snap1 as Snapshot
   Dim counter%
   Set db = OpenDatabase("BIBLIO.MDB")
   Set Snap1 = db.CreateSnapshot("Authors")
   counter% = 1                  'Start counter at Row=1
   Do Until Snap1.EOF
      grid1.Col = 1
      grid1.Row = counter%
      grid1.Text = Snap1(1)      'Load the Author Name
      grid1.Col = 2
      grid1.Row = counter%
      grid1.Text = Snap1(0)      'Load the Author ID
      counter% = counter% + 1
      Snap1.MoveNext
   Loop
   Snap1.Close
   db.Close
End Sub
```

6. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program. Click the Command1 button.

Example Four
----

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add one Grid control and one Command button to Form1.

3. Using the following table as a guide, set the properties of the controls you added in step 2.

Control Name	Property	New Value
Grid1	Cols	3
Grid1	Rows	50
Command1	Caption	Press to Load Grid

4. Place the following code in the Form1 Load event procedure:

```
Sub Form_Load ()
   'Initialize the colwidths for the grid and supply headers
   Show
   grid1.ColWidth(1) = 3000      'For Author name
   grid1.ColWidth(2) = 1000      'For Author ID
   grid1.Col = 1
   grid1.Row = 0
   grid1.Text = "Author Name"    'Header for Author Name
   grid1.Col = 2
   grid1.Row = 0
   grid1.Text = "Author ID"      'Header for Author ID
End Sub
```

5. Place the following code in the Command1 Click event procedure:

```
Sub Command1_Click ()
   ' The routine to load data into grid
   Dim db as Database
   Dim t as Table
   Dim counter%
   Set db = OpenDatabase("BIBLIO.MDB")
   Set t = db.Opentable("Authors")
   counter% = 1         'Start counter at Row=1
   Do Until t.EOF
      grid1.Col = 1
      grid1.Row = counter%
      grid1.Text = t(1)          'Load the Author Name
      grid1.Col = 2
      grid1.Row = counter%
      grid1.Text = t(0)          'Load the Author ID
      counter% = counter% + 1
      t.MoveNext
   Loop
   t.Close
   db.Close
End Sub
```

6. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program. Click the Command1 button.

Additional reference words: 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

Error Listing for MCI.VBX Control

Article ID: Q103647

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

Below is a listing of the error codes and numbers that are related to the MCI.VBX control. This listing is presently missing from Microsoft Visual Basic version 3.0 for Windows, but it can be found in the MMSYSTEMS.H file from the Windows version 3.1 Software Development Kit and it can be found in the WINMMSYS.TXT file.

MORE INFORMATION

MCI Errors Defined	MCI Error Number
#define MCIERR_BASE	256
#define MCIERR_INVALID_DEVICE_ID	257
#define MCIERR_UNRECOGNIZED_KEYWORD	259
#define MCIERR_UNRECOGNIZED_COMMAND	261
#define MCIERR_HARDWARE	262
#define MCIERR_INVALID_DEVICE_NAME	263
#define MCIERR_OUT_OF_MEMORY	264
#define MCIERR_DEVICE_OPEN	265
#define MCIERR_CANNOT_LOAD_DRIVER	266
#define MCIERR_MISSING_COMMAND_STRING	267
#define MCIERR_PARAM_OVERFLOW	268
#define MCIERR_MISSING_STRING_ARGUMENT	269
#define MCIERR_BAD_INTEGER	270
#define MCIERR_PARSER_INTERNAL	271
#define MCIERR_DRIVER_INTERNAL	272
#define MCIERR_MISSING_PARAMETER	273
#define MCIERR_UNSUPPORTED_FUNCTION	274
#define MCIERR_FILE_NOT_FOUND	275
#define MCIERR_DEVICE_NOT_READY	276
#define MCIERR_INTERNAL	277
#define MCIERR_DRIVER	278
#define MCIERR_CANNOT_USE_ALL	279
#define MCIERR_MULTIPLE	280
#define MCIERR_EXTENSION_NOT_FOUND	281
#define MCIERR_OUTOFRANGE	282
#define MCIERR_FLAGS_NOT_COMPATIBLE	283
#define MCIERR_FILE_NOT_SAVED	286
#define MCIERR_DEVICE_TYPE_REQUIRED	287
#define MCIERR_DEVICE_LOCKED	288
#define MCIERR_DUPLICATE_ALIAS	289
#define MCIERR_BAD_CONSTANT	290

```
#define MCIERR_MUST_USE_SHAREABLE         291
#define MCIERR_MISSING_DEVICE_NAME        292
#define MCIERR_BAD_TIME_FORMAT            293
#define MCIERR_NO_CLOSING_QUOTE           294
#define MCIERR_DUPLICATE_FLAGS            295
#define MCIERR_INVALID_FILE               296
#define MCIERR_NULL_PARAMETER_BLOCK       297
#define MCIERR_UNNAMED_RESOURCE           298
#define MCIERR_NEW_REQUIRES_ALIAS         299
#define MCIERR_NOTIFY_ON_AUTO_OPEN        300
#define MCIERR_NO_ELEMENT_ALLOWED         301
#define MCIERR_NONAPPLICABLE_FUNCTION     302
#define MCIERR_ILLEGAL_FOR_AUTO_OPEN      303
#define MCIERR_FILENAME_REQUIRED          304
#define MCIERR_EXTRA_CHARACTERS           305
#define MCIERR_DEVICE_NOT_INSTALLED       306
#define MCIERR_GET_CD                     307
#define MCIERR_SET_CD                     308
#define MCIERR_SET_DRIVE                  309
#define MCIERR_DEVICE_LENGTH              310
#define MCIERR_DEVICE_ORD_LENGTH          311
#define MCIERR_NO_INTEGER                 312

#define MCIERR_WAVE_OUTPUTSINUSE          320
#define MCIERR_WAVE_SETOUTPUTINUSE        321
#define MCIERR_WAVE_INPUTSINUSE           322
#define MCIERR_WAVE_SETINPUTINUSE         323
#define MCIERR_WAVE_OUTPUTUNSPECIFIED     324
#define MCIERR_WAVE_INPUTUNSPECIFIED      325
#define MCIERR_WAVE_OUTPUTSUNSUITABLE     326
#define MCIERR_WAVE_SETOUTPUTUNSUITABLE   327
#define MCIERR_WAVE_INPUTSUNSUITABLE      328
#define MCIERR_WAVE_SETINPUTUNSUITABLE    329

#define MCIERR_SEQ_DIV_INCOMPATIBLE       336
#define MCIERR_SEQ_PORT_INUSE             337
#define MCIERR_SEQ_PORT_NONEXISTENT       338
#define MCIERR_SEQ_PORT_MAPNODEVICE       339
#define MCIERR_SEQ_PORT_MISCERROR         340
#define MCIERR_SEQ_TIMER                  341
#define MCIERR_SEQ_PORTUNSPECIFIED        342
#define MCIERR_SEQ_NOMIDIPRESENT          343

#define MCIERR_NO_WINDOW                  346
#define MCIERR_CREATEWINDOW               347
#define MCIERR_FILE_READ                  348
#define MCIERR_FILE_WRITE                 349

#define MCIERR_CUSTOM_DRIVER_BASE         512

Additional reference words: 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus
```

How to Include Return Receipt Functionality w/ MAPI Control

Article ID: Q104624

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

This article explains how to install return receipt functionality on a message sent in a Visual Basic application. When a message has return receipt functionality, it means that when the message you sent is opened by the recipient, a message is sent back to you to confirm that the message was opened by the recipient. The returned message typically contains the date, time, and original message subject.

MORE INFORMATION

In Visual Basic, you can send a message by calling the automatic dialog box or by manually programming the message properties.

Using the automatic dialog box, the sender can select the Return Receipt option in the Send Note dialog box.

To manually program Return Receipt functionality, use the following example as a guide:

```
   ' set up a session associated with the message:
   Const SESSION_SIGNON = 1
   mapisession1.Action = SESSION_SIGNON
   mapimessages1.SessionID = mapisession1.SessionID

   ' Send the message
   Const MESSAGE_SEND = 3
   mapimessages1.MsgIndex = -1                         ' The compose buffer
   mapimessages1.MsgNoteText = "How's it going?"       ' The message text
   mapimessages1.MsgOrigAddress = "FredBloggs"         ' Sender's alias
   mapimessages1.MsgSubject = "Hi"                     ' The message title
   mapimessages1.RecipDisplayName = "JoSmith"          ' Recipient's alias
   mapimessages1.MsgReceiptRequested = True            ' Request receipt
   mapimessages1.Action = MESSAGE_SEND                 ' Send message
```

Below is the example code that traps whether Request Receipt has been set. Place this code at the point where the user reads the message.

```
   If mapimessages1.MsgReceiptRequested Then           ' Check Return
receipt
      mapimessages1.MsgIndex = -1                      ' Compose buffer
      mapimessages1.RecipDisplayName = sender$         ' Set sender to
receiver
      mapimessages1.MsgSubject = "RECEIVED " + title$ ' Set message title
```

```
      mapimessages1.Action = MESSAGE_SEND             ' Send Return Receipt
   End If
```

The variables sender$ and title$ contain the alias and the message title of the original message.

Additional reference words: 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

PRB: Default Extension Ignores File Type in VB Common Dialog

Article ID: Q106682

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SYMPTOMS

The common dialog custom control (CMDIALOG.VBX) cannot determine which file type you choose in the Open or Save As dialog box under List Files of Type. Your chosen file type correctly displays existing files of that type and filters out other files. However, Visual Basic code cannot detect which file type you chose.

Also, the default file name extension set by the DefaultExt property is not affected by changes you make under List Files of Type. As a result, a file name that you enter without an extension will take the extension of DefaultExt instead of your choice under List Files of Type.

The above behavior of File Open and File Save As is different from many other Windows applications, such as Microsoft Excel. Excel determines which file-type filter you choose and automatically appends that extension to any file name that you may enter without an extension.

CAUSE

This behavior is by design in the common dialog control in Visual Basic.

WORKAROUND

Instead of using Visual Basic's common dialog custom control, you can write your own DLL routine in C to call the Windows common dialog routines located in COMMDLG.DLL. Then you can call that DLL from Visual Basic.

STATUS

This behavior is by design. A change in the design is under review and will be considered for inclusion in a future release.

MORE INFORMATION

Steps to Reproduce Behavior

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a common dialog control to Form1. This requires CMDIALOG.VBX to

be loaded in Visual Basic. You can load CMDIALOG.VBX automatically through your AUTOLOAD.MAK file or by choosing Add File from the File menu.

3. Add the following to the Form Load event code:

```
Sub Form_Load ()

    ' Enter the following two lines as one, single line:
    CMDialog1.Filter = "Text Files *.Txt|*.Txt|Basic Files *.Bas|*.Bas|
       All Files *.*|*.*"
    CMDialog1.FilterIndex = 1  'Sets default filter to *.txt.

    CMDialog1.DefaultExt = "TXT"  'Default extension if you enter none.
    ' In the dialog box, the default extension will be applied only if
you
    ' enter the filename with no period. If you type the file name with
    ' a period and no extension (such as FILEX.), then CMDialog1.Filename
    ' always returns a blank extension.

    ' Set the common dialog Action property to 1 to execute the File Open
    ' dialog or 2 to execute the File Save As dialog:
    CMDialog1.Action = 1  ' 1=Invokes the File Open common dialog box.

    ' Limitation: The value of FilterIndex doesn't change even if you
change
    ' the file type in the Open common dialog box:
    Debug.Print CMDialog1.FilterIndex

    ' The Filename property displays the filename and path that you
entered
    ' in the Open dialog:
    Debug.Print CMDialog1.Filename  'Prints filename with path prefix
    ' Debug.Print CMDialog1.Filetitle 'Prints filename without path

End Sub
```

4. Start the program or press the F5 key. The Open dialog now displays.

5. Under List Files of Type, select Basic Files *.Bas. Under File Name, enter a filename such as TESTFILE without an extension and without a period. Click OK.

   The debug window now shows the following limitations:

   a. The CMDialog1.FilterIndex property keeps its value of 1. CMDialog1.FilterIndex does not change in response to your changing the file type in the Open dialog box. This is by design in the common dialog custom control in Visual Basic.

   b. The CMDialog1.Filename property returns C:\VB\TESTFILE.TXT, which is the filename you entered plus the default extension .TXT. Notice that the program cannot detect the file type you chose in the Open dialog box. The default extension .TXT set by the DefaultExt property is independent of changes under List Files of Type. These are design limitations in the common dialog custom control in Visual Basic.

NOTE: File names that you enter with an extension keep that extension as desired.

Workaround for Windows API Programmers

Visual Basic's common dialog custom controls for Open and Save As pass their FilterIndex property to the Windows API function GetOpenFileName. GetOpenFileName is located in the Windows COMMDLG.DLL file. However, Visual Basic ignores the nFilterIndex value that the GetOpenFileName function returns. By design, your Visual Basic program cannot access the structure returned by the GetOpenFileName function, even by calling API routines.

You can write your own DLL routine in C to call the Windows common dialog routines located in COMMDLG.DLL. Then call this DLL from Visual Basic. The following documentation from the Windows Software Development Kit (SDK) explains how to use the nFilterIndex element of the structure passed to GetOpenFileName:

> nFilterIndex:
> Specifies an index into the buffer pointed to by the lpstrFiler member. The system uses the index value to obtain a pair of strings to use as the initial filter description and filter pattern for the dialog box. The first pair of strings has an index value of 1. When the user chooses the OK button to close the dialog box, the system copies the index of the selected filter strings into this location. If the nFilterIndex member is 0, the filter in the buffer pointed to by the lpstrCustomFilter member is used. If the nFilterIndex member is 0 and the lpstrCustomFilter member is NULL, the system uses the first filter in the buffer pointed to by the lpstrFilter member. If each of the three members is either 0 or NULL, the system does not use any filters and does not show any files in the File Name list box of the dialog box.

Additional reference words: 2.00 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsCus

PRB: Out of Memory Error Using VB Outline Control

Article ID: Q107769

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

MSOUTLIN.VBX is the outline custom control file in Visual Basic. An out of memory error can occur with an outline control for the reasons shown in the Cause section below.

CAUSE

An outline custom control can give an out of memory error if it contains more than 6537 items or uses too much string space. Each item cannot exceed 1K.

Each form is limited to a single 64K data segment for all properties of the form and all properties of controls on that form. All data that you add at run time to the outline controls on a form is stored in this same, shared 64K data segment. String space for outline controls will thus be limited by memory used by other controls on that form.

If the out of memory error occurs at design time instead of run time, you may be running into a code size limitation.

STATUS

This behavior is by design.

MORE INFORMATION

The size allowed for the List property of an outline control is smaller than for the following controls that store data in separate segments of memory:

- The List property of a combo box control or list box control.
- The Text property of a text box control that has the MultiLine property set to True.

Steps to Reproduce Behavior

1. If the MSOUTLIN.VBX custom control file is not already loaded in Visual Basic, choose Add File from the File menu to load it.

2. Add three outline controls to Form1.

3. Design the program to fill up the controls with 300 outline records of 60 bytes each at run time. Use the AddItem method to add records.

4. Start the program.

A customer reported an out of memory error while populating the third control with the 200th item. The point at which you receive an error will depend upon the memory usage of other controls on the form.

REFERENCES
==========

- "Visual Basic 3.0: Programmer's Guide," pages 641-642 of "Appendix D: Specifications and Limitations."

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsCus

VB CDK Cannot Access the Properties of the VB System Objects

Article ID: Q107850

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The Microsoft Visual Basic Control Development Kit (CDK) provides no mechanism to access the properties of the system objects. The Visual Basic system objects are Screen, Printer, App, Clipboard, and Debug.

This feature is under review and will be considered for inclusion in a future release.

MORE INFORMATION

The five Visual Basic system objects are as follows:

Object	Use
Screen	Supplies data on current form, control, and screen display.
Printer	Enables printing text and graphics to the printer.
App	Supplies information specific to the application.
Clipboard	Gives access to the Windows Clipboard.
Debug	Enables printing to the Debug window in Visual Basic.

Because the system objects are global, you can use them in code anywhere in your application. You cannot declare object variables for any of these system objects, and you cannot pass the system objects to a procedure.

The Control Development Kit is provided with the professional edition of Visual Basic for Windows. Support for the Microsoft Control Development Kit is currently provided only through CompuServe in the MSBASIC forum in section 16 or through service requests in Microsoft OnLine support services. For more information about CompuServe, please call CompuServe at (800)848-8990. For more information about the Microsoft OnLine support services, please call (800)443-4672.

REFERENCES

- "Visual Basic 3.0: Programmer's Guide," version 3.0, page 206, "System Objects."

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsCus

VB ver 3.0 CDK TN002.TXT: Custom Control Version Management

Article ID: Q107873

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY
=======

The following article contains the complete contents of the TN002.TXT file installed in the CDK directory of the Professional Edition of Visual Basic version 3.0 for Windows.

MORE INFORMATION
================

TN002.TXT

Microsoft Visual Basic version 3.00 for Windows
Microsoft Corporation Technical Notes

TN002.TXT: Custom Control Version Management

This note describes how to use the version management functionality for custom controls.

==========

Introduction

Visual Basic provides upward compatibility for custom controls. This means that a custom control created for Visual Basic version 1.0 will run in Visual Basic versions 2.0 and 3.0. If a custom control uses version-specific features, a custom control can always explicitly test for a specific version during initialization and thereby determine whether or not to load.

There are cases, however, when a Visual Basic application created with a new version of a custom control runs on a system with an older version of the custom control. Depending on the features and implementation of the custom control, the application may not work correctly -- or worse, may cause VB.EXE or VBRUNx00.DLL to crash.

The following sections describe Visual Basic version 3.0's custom control version management system, which can help avoid potential application failure.

MODEL Structure

This is the definition of the MODEL structure used in Visual Basic version 3.0. Note the addition of the last field (USHORT usCtlVersion).

```
typedef struct tagMODEL
    {
    USHORT      usVersion;           // VB version used by control
    FLONG       fl;                  // Bitfield structure
    PCTLPROC    pctlproc;            // The control procedure
    FSHORT      fsClassStyle;        // Window class style
    FLONG       flWndStyle;          // Default window style
    USHORT      cbCtlExtra;          // Number of bytes allocated
                                     // for control structure
    USHORT      idBmpPalette;        // BITMAP ID for tool palette
    PSTR        npszDefCtlName;      // Default control name prefix
    PSTR        npszClassName;       // Visual Basic class name
    PSTR        npszParentClassName; // Parent window class, if subclassed
    NPPROPLIST  npproplist;          // Property list
    NPEVENTLIST npeventlist;         // Event list
    BYTE        nDefProp;            // Index of default property
    BYTE        nDefEvent;           // Index of default event
    BYTE        nValueProp;          // Index of control value property
    USHORT      usCtlVersion;        // Identifies current version of the
                                     // custom control. The values 1 and
                                     // 2 are reserved for custom controls
                                     // created with VB1.0 and VB2.0
} MODEL;
```

Control Version

A custom control developer who wants to take advantage of the version management feature in Visual Basic version 3.0 needs to provide a valid version number in the usCtlVersion field. This value must be an unsigned integer (USHORT), and it should be changed any time the custom control changes its backward compatibility with previous versions.

Although any nonzero value is valid, the values 1 and 2 should not be used. Because Visual Basic versions 1.0 and 2.0 do not support version management, Visual Basic automatically assigns values 1 and 2 to any custom control that has the usVersion field in the control's MODEL structure set to VB100_VERSION and VB200_VERSION, respectively.

In order to take advantage of version management, you must set the usVersion field of the control's MODEL structure to VB300_VERSION or greater, or use VB_VERSION defined in a Visual Basic version 3.0 VBAPI.H file. This allows Visual Basic to determine whether the usCtlVersion field is defined in the MODEL structure of a given custom control.

If the custom control contains a value of 0 in the usCtlVersion field, no version control checks are made on this custom control.

How the System Works

When you create a Visual Basic executable (.EXE) file that uses a custom

control, Visual Basic retrieves the control version number (usCtlVersion) for that control and stores it in a special section of the .EXE file.

When you execute the application, the Visual Basic run-time support file (VBRUN300.DLL or greater) checks the control version number recorded in the .EXE file against the version number found in the custom control when it is loaded into the system. If the version found in the .EXE file is greater than the version of the control loaded into the system, Visual Basic displays a notification that the particular custom control (.VBX file) is out of date and will not load, consequently forcing the application to terminate.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsCus

Windows 3.1 VERSIONINFO - Version-Information Resource Example

Article ID: Q107992

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0
- Microsoft Windows version 3.1

SUMMARY

This article describes the version-information resource and the VERSIONINFO statement.

The VERSIONINFO statement is found in the file installation library found in Windows version 3.1. The VERSIONINFO statement creates a version-information resource in an application. The resource contains such information about the file as its version number, its intended operating system, and its original filename. The resource is intended to be used with the Windows file installation library functions.

MORE INFORMATION

For more information on the VERSIONINFO statement, double-click the "Win SDK Help" icon installed with the Professional Edition of Visual Basic version 3.0. This loads the VB3\WINAPI\WIN31WH.HLP Help file. Click the Search button, press V to scroll to topics that begin with V, then read the following topics:

- VERSIONINFO
- Version-Information Resource
- Version Functions (3.1)

Example of the VERSIONINFO Resource

An example of the VERSIONINFO resource can be found by decompiling a program's resources and looking for the word VERSIONINFO. For example, you could use Resource Workshop, from Borland International, to find or enter the VERSIONINFO resource. You can design an application's setup program to read the resource information to detect and handle existing old copies of your installed application.

```
1 VERSIONINFO LOADONCALL MOVEABLE FILEVERSION 1, 0, 0, 5 PRODUCTVERSION 1,
0, 0, 10 FILEOS VOS__WINDOWS16 FILETYPE VFT_APP BEGIN

BLOCK "StringFileInfo"
   BEGIN
      BLOCK "040904E4"
      BEGIN
         VALUE "CompanyName", "Some Company\000"
         VALUE "FileDescription", "What it is\000"
         VALUE "FileVersion", "03.00.0005\000"
```

```
         VALUE "InternalName", "XYZ.EXE\000"
         VALUE "LegalCopyright", "Copyright ) abcdefg"
         VALUE "LegalTrademarks", "Whatever you want\000"
         VALUE "ProductName", "asdfg\000"
         VALUE "ProductVersion", "see above"
         VALUE "Comments", "Some comments"
      END
   END
END
```

The File Installation Library in Windows Version 3.1

The file installation library in the Microsoft Windows version 3.1 operating system makes it easier for applications to install files properly and enables utility programs to analyze files that are currently installed.

The file installation library includes functions that determine where a file should be installed, identify conflicts with currently installed files, and perform the installation process. These functions enable installation programs to avoid the following problems:

- Installing older versions of components over newer versions
- Changing the language in a mixed-language system without notification
- Installing multiple copies of a library in different directories
- Copying files to network directories shared by multiple users

The file installation library also includes functions that enable applications to query a version resource for information about a file and present the information to the user in a clear format. This information includes the file's purpose, author, version number, and so on.

A version-information resource contains data that identifies the version, language, and distribution of the application, dynamic-link library, driver, or device containing the resource. Installation programs use the functions in the file installation library (VER.DLL) to retrieve the version-information resource from a file and to extract the version-information blocks from the resource.

For more information about the file installation library, see the "Microsoft Windows Version 3.1 Programmer's Reference," Volume 1.

Additional reference words: 3.00 3.10 setup set up install
KBCategory: Prg
KBSubcategory: PrgCtrlsCus

Category Keywords for All Visual Basic KB Articles

Article ID: Q108753

The information in this article applies to:

- Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Each article in the Visual Basic for Windows collection contains at least one keyword (called a KBSubcategory keyword) that places the article in an appropriate category. This article lists all the KBSubcategory keywords.

MORE INFORMATION

Category & Subcategory Description	KBSubcategory Keyword
Setup / Installation (Setins)	Setins
Environment-specific Issues (Envt)	
VB Design Environment	EnvtDes
Run-Time Environment	EnvtRun
Programming (Prg)	
Visual Basic Forms and Controls	
Standard Controls / Forms	PrgCtrlsStd
Custom Controls	PrgCtrlsCus
Third-Party Controls	PrgCtrlsThird
Optimization	
Memory Management	PrgOptMemMgt
General Optimization Tips	PrgOptTips
General VB Programming	PrgOther
Advanced programming (APrg)	
Network	APrgNet
Windows Programming (APIs / DLLs)	
Printing	APrgPrint
Graphics	APrgGrap
Windowing	APrgWindow
INI Files	APrgINI
Other API / DLL Programming	APrgOther
Data Access	
ODBC	APrgDataODBC
IISAM	APrgDataIISAM
Access	APrgDataAcc
General Database Programming	APrgDataOther
3rd Party DLL's	APrgThirdDLL

Inter-Application Programmability (IAP)

OLE	IAPOLE
DDE	IAPDDE
3rd Party Interoperability	IAPThird

Tools (Tls)

Setup Toolkit / Wizard	TlsSetWiz
Control Development Kit (CDK)	TlsCDK
Help Compiler (HC)	TlsHC

References (Refs)

Documentation / Help File Fixes	RefsDoc
Product Information	RefsProd
Third-Party Information	RefsThird
PSS-Only Information	RefsPSS

Using Keywords to Query the KB

At Microsoft, we use the subcategory keywords to organize the articles for Help files and for the FastTips Catalog. You can use them to query the Microsoft Knowledge Base for Visual Basic articles that apply to that category or subcategory. For example, you can find all the general database programming articles by querying on the following words in the Microsoft Knowledge Base:

```
visual and basic and APrgDataOther
```

Use the asterisk (*) wildcard to find articles that fall into the general categories or into an intermediate subcategory. The first element in each keyword is the category. For example, to find all the articles that apply to Visual Basic Forms and Controls regardless of whether they are standard, custom, or third-party controls, use the following words to query the Microsoft Knowledge Base:

```
visual and basic and PrgCtrls*
```

To find all advanced programming articles, query on these words:

```
visual and basic and APrg*
```

Add KBSubcategory Keyword to Each Article

When contributing an article to the Visual Basic Knowledge Base, add the appropriate KBSubcategory keyword to the bottom of the article on the KBSubcategory line. Each article in the Visual Basic for Windows collection contains the following section at the bottom of the article:

```
Additional reference words:
KBCategory:
KBSubcategory: <keyword>
```

An article usually has only one subcategory keyword, but it may have more.

If you are interested in contributing, please obtain the guidelines by

querying on the following words in the Microsoft Knowledge Base:

   visual and basic and kbguide and kbartwrite

Additional reference words: 3.00 dskbguide subcatkey
KBCategory:
KBSubcategory: RefsPSS

How to Display Multiple Foreground Text Colors in VB List Box

Article ID: Q108811

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

To set the foreground and background color of a list box control, set the ForeColor and BackColor properties at either design time or run time. All text in a list box uses the color set by the ForeColor property. The text is printed against a background color set by the BackColor property.

Visual Basic doesn't directly support the display of text of different colors simultaneously in the list box. This article describes how to display words of different colors simultaneously in a list box by using an indirect technique.

MORE INFORMATION

You can display lines or words of different colors simultaneously in a list box by using one of the following indirect techniques.

1. Simulate the list box with a picture box control. You can store the desired text strings in an array of strings, and use the Print method to write the array entries into the picture box with different ForeColor properties. For example:

   ```
   picture1.BackColor = QBColor(14)   ' 14=Light yellow
   picture1.ForeColor = QBColor(4)    '  4=Red
   picture1.Print "in living red"
   picture1.ForeColor = QBColor(2)    '  2=Green
   picture1.Print "in living green"
   ```

   You can also add a vertical scroll bar next to the picture box. When the scroll bar is scrolled, your code needs to redraw the picture box. The ForeColor property of the picture box controls the current color used by the Print method. The picture box will not let you highlight text. NOTE: The BackColor method erases any pre-existing text on the picture control.

2. For coloring list box entries with multiple foreground colors, the Desaware company provides two solutions:

   a. The MLIST2.VBX control file is included with the Custom Control Factory product from Desaware. MLIST2.VBX allows each line in a list box to be colored independently. All words on the same line must be the same color. MLIST2.VBX comes with full source code.

   b. A more flexible and advanced solution is to turn Visual Basic's list

box into an owner-draw list box. Desaware says that you can make true owner-draw list boxes with their SpyWorks-VB product. SpyWorks-VB allows you to color each entry of the list box with the full power of the Windows API drawing functions. SpyWorks-VB comes with sample source code for an owner-draw list box and command button, along with explanations of how to turn the standard controls into owner-draw controls. See the section on owner-draw controls further below.

How to Contact Desaware

NOTE: Desaware products are manufactured independent of Microsoft. Microsoft makes no warranty, implied or otherwise, regarding these products' performance or reliability.

Desaware
5 Town & Country Village #790
San Jose, CA 95128
Contact: Gabriel Appleman (213) 943-3305
          Dan Appleman (408) 377-4770
Fax: (408) 371-3530

The Desaware company offers the following products:

1. Custom Control Factory -- an interactive development tool for creating custom controls including Animated Pushbuttons, Multistate Buttons, enhanced buttons, check box, and option button controls for Windows applications.

2. CCF-Cursors -- provides you with complete control over cursors (mouse pointers) in Visual Basic applications. Create your own cursors or convert icons to cursors, and much more. Includes over 50 cursors.

3. SpyWorks-VB -- an advanced development tool for use with Visual Basic.

Owner-Draw Controls in Windows

The owner-draw list capability is appropriate for advanced programmers for Microsoft Windows. You will need a good reference for the Windows API to learn the required drawing functions.

Owner-draw controls were introduced in Windows version 3.0. Because your application does all the drawing of the contents of the controls, you can customize them any way you like. Owner-draw controls are similar to predefined controls in that Windows will handle the control's functionality and mouse and keyboard input processing. However, you are responsible for drawing the owner-draw control in its normal, selected, and focus states.

You can create owner-draw controls from the menu, button, and list-box classes. You can create owner-draw combo boxes, but they must have the CBS_DROPDOWNLIST style, which equates to a static text item and a list box. The elements of an owner-draw control can be composed of strings, bitmaps, lines, rectangles, and other drawing functions in any combination, in your choice of colors.

REFERENCES
==========

 - "Microsoft Windows Programmer's Reference"

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsCus PrgCtrlsStd PrgCtrlsThird

PRB: Using RecordCount with VB Dynasets, Snapshots, and Tables

Article ID: Q109053

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0.

SYMPTOMS

The RecordCount property, when used with a Dynaset or Snapshot, can sometimes return an incorrect number of records. This applies to the Microsoft Access database engine that is built into Visual Basic version 3.0.

CAUSE

For Dynasets and Snapshots, the RecordCount property does not automatically return the number of records that exist in the recordset. RecordCount returns the number of records accessed. If you don't do a MoveLast method immediately before checking the record count, you will get an incorrect, smaller count.

RESOLUTION

To determine the number of records in a Dynaset or Snapshot, use the MoveLast method before checking the RecordCount property. This requirement is by design.

A faster way to find the number of records in a whole table is to use the RecordCount property of a Table. See examples below.

NOTE: If you add or delete records to a table within a transaction, and then roll back the transaction, the value of the RecordCount property is not adjusted accordingly.

STATUS

This behavior is by design because otherwise, Visual Basic would have to do an implicit MoveLast. This would be very slow with large record sets and especially with remote databases, so the decision is left up to the programmer.

MORE INFORMATION

NOTE: If your data is displayed in a Grid control, the RecordCount will be one greater than the last line number in the grid because the grid starts at zero.

How to Count Records in Whole Table Quickly by Using RecordCount Property

1. Start a new project in Visual Basic. Form1 is created by default.

2. Double-click the form to open the code window. Add the following code to the Form Load event:

```
Sub Form_Load ()
   Dim MyDB As Database
   Dim tb As Table
   Set MyDB = OpenDatabase("C:\VB3\BIBLIO.MDB")
   Set tb = MyDB.OpenTable("authors")
   form1.Show  ' Must Show form1 in Load event before Print works.
   Print "Table record count=" & tb.RecordCount
End Sub
```

4. Start the program (or press F5). The BIBLIO.MDB file shipped with Visual Basic 3.0 contains 46 records in the Authors table. Close the form to end the program.

How to Count Records in Whole Table Quickly by Using ListTables Method

The following steps count the number of records in a table without opening the table:

1. Create a Snapshot of the Tabledefs collection using the ListTables method. The ListTables method creates a Snapshot with one record for each Table or QueryDef in a specified database.

2. Examine the RecordCount field of the record corresponding to your table in that Snapshot. That RecordCount field is not a property; it is a field in a record in a Snapshot that is returned by the ListTables method.

The following sample program performs the above two steps:

```
Sub Form_Load ()
Const DB_TABLE = 1  ' Constant taken from DATACONS.TXT file.
Dim db As Database
Dim snap As Snapshot
Set db = OpenDatabase("C:\VB3\BIBLIO.MDB")
Set snap = db.ListTables()  ' Copy Table information to Snapshot.
Do While Not snap.EOF
   If snap("TableType") = DB_TABLE Then
   'Enter the Table name for which you want a record count:
      If snap("Name") = "Authors" Then
         MyRecordCount = snap("RecordCount")
      End If
   End If
   snap.MoveNext  ' Move to next record.
Loop
snap.Close
form1.Show  ' Must Show form1 in Load event before Print works.
Print MyRecordCount
End Sub
```

How to Count Records in Snapshot, Dynaset, or Data Control

NOTE: A MoveLast will be slow on a large table or set. Only use the method in the section below for counting subsets of the table. To count the number of records that comprise the whole table, use one of the above table RecordCount techniques.

If you are using a Snapshot, Dynaset, or the data control, you can count the records in the current recordset by first doing a MoveLast. Then use the RecordCount property. This count is only accurate for that instant, because another user could be simultaneously adding or deleting records to the underlying table. By design, a data control is linked to a Dynaset.

A MoveLast on a recordset variable (a Dynaset or Snapshot) is faster than MoveLast on a data control. You can create a separate Snapshot variable of your data control's recordset and invoke a MoveLast on that Snapshot.

The following program shows how to use MoveLast and the RecordCount property to count the number of records in a Dynaset.

```
Sub Form_Load ()
   Dim MyDB As Database, MyDyna As Dynaset
   Set MyDB = OpenDatabase("C:\VB3\BIBLIO.MDB")
   Set MyDyna = MyDB.CreateDynaset("Authors")
   MyDyna.MoveLast
   MyRecordCount = MyDyna.RecordCount
   MyDyna.Close
   form1.Show
   Print MyRecordCount
End Sub
```

Records Must Be Properly Added Before They Are Counted

The Addnew method allocates space for a new record in your database. You then add data to the various table fields in the new record. You then do an Update method to write the new record to the table.

The Update method saves the contents of the copy buffer to a specified Table or Dynaset. Use Update to save any changes to a record after using Edit or AddNew. With a data control, if an Edit or AddNew operation is pending when you move to another record or close the recordset, Update is automatically invoked if not stopped during the Validate event.

NOTE: In the Professional Edition, if you are not using a data control and move to another record or close the recordset while an Edit or AddNew operation is pending, any existing changes will be lost and no error will occur.

Loops and RecordCount

Use EOF instead of RecordCount in loops. For example, don't use this:

```
   For i = 1 to ds.RecordCount ' Bad code
   ...
   Next
```

Use the following instead:

```
   Do Until ds.EOF              ' Good code
   ...
   Loop
```

REFERENCES
==========

- "Microsoft Visual Basic Version 3.0: Professional Features Book 2: Data Access Guide," page 24 (RecordCount field) and pages 61-63 (RecordCount property).

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsCus

Using Table Objects Versus Dynaset/Snapshot Objects in VB

Article ID: Q109218

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY
=======

This article discusses the advantages and disadvantages of using Table objects versus Dynaset and Snapshot objects for finding and updating data in a database table. This applies to the Microsoft Access database engine used in Visual Basic version 3.0.

MORE INFORMATION
================

The three types of recordsets are Tables, Dynasets, and Snapshots. All recordsets have records (rows) and fields (columns). The Professional Edition of Visual Basic lets you create object variables of type Dynaset, Snapshot, and Table. The Standard Edition supports Dynaset object variables but not Snapshot or Table object variables.

A table is a fundamental part of a Database and contains data about a particular subject. A Table object is a logical representation of a physical table.

To make a Snapshot or Dynaset, use the CreateSnapshot or CreateDynaset method on a Database or any recordset. A Snapshot is a static, read-only picture of a set of records that you can use to find data or generate reports. The records in a Snapshot cannot be updated (or modified), whereas records in a Dynaset can be updated.

The move methods (MoveFirst, MoveLast, MoveNext, and MoveLast) apply to all three types of recordsets (Dynasets, Snapshots, and Tables).

The find methods (FindFirst, FindLast, FindNext, and FindPrevious) apply to Dynaset objects and Snapshot objects, but not to Table objects. The Seek method applies only to Table objects.

For intensive searches, you may want to use both Table and Dynaset objects on the same base table. You can use the Seek method on the Table objects and the find methods on any open Dynasets.

Visual Basic data controls always use Dynasets. Data controls don't use Snapshot objects or Table objects.

Dynaset objects are a set of record pointers to those records which existed in the base table in the Database at the time the Dynaset was created. Your Dynaset also adds pointers to any new records which you add to the Dynaset, and deletes pointers of deleted records.

If you add a record to a base table, the record does not immediately appear

in any currently existing Dynaset based on that table. You would need to recreate the Dynaset to see a new record that was added to the base table after the Dynaset was created. However, if you add a new record to a Dynaset, the record appears immediately in both the Dynaset and the base table. Deleting a record is reflected in a similar way.

Dynasets Versus Tables in Multiuser and Single-User Environments

Table objects connect directly to base tables that are globally accessible to all users on a multiuser system. All users using Table object variables can see all records in the base table at all times. In contrast, Dynasets are local to each program. Your local additions and deletions are reflected in the Dynaset. Dynasets don't reflect records that other users added or deleted after the local Dynaset was created.

In a multiuser environment (computer network), Dynasets may not be suitable for updating shared tables. Data controls, because they use Dynasets, are unsuitable for such applications as a multiuser order entry system. NOTE: Two programs simultaneously using the same table on a single computer act as a multiuser environment.

If another user on a multiuser system updates a record for which you have a pointer in your Dynaset, you will see the changes whenever you request that record. If another user adds a record to the table, you cannot see that record because the current Dynaset doesn't contain a pointer to that record. If another user of the base table deletes a record that is in your Dynaset, your Dynaset keeps a pointer to that non-existent record. Your subsequent attempts to access that non-existent record will give an error.

Data controls are suitable for most types of data browsing (read-only access) and many types of simple data entry.

In a single-user environment, Table Objects and Dynaset Objects both update the base table in a similar fashion when records are added or deleted. Data controls are thus quite suitable for updating databases in single-user environments.

Dynaset Objects:

- Dynasets are set-oriented. You can create any arbitrary set of records from a single Table, or set of records joined from multiple Tables using an SQL SELECT statement. If you need to join tables or use subsets, a Dynaset is required. The only way to join more than one table is with a Dynaset object.

- When Visual Basic creates a Dynaset, the Dynaset's records are ordered using indexes for greater speed. After the Dynaset is created, find and move methods within a Dynaset are non-indexed, sequential, and relatively slow. Using the Dynaset will be faster if you limit its size to a small subset of the records in the base table. Recreating the Dynaset with a different subset of records is faster than creating a huge Dynaset and navigating it using find and move methods.

- You can sort a Dynaset on any arbitrary field, including expressions, such as mid([myfield],2,3), whether the field is indexed or not.

- Using a Dynaset, you can attach external database tables to a Microsoft Access format database, which is the format native to Visual Basic. An attached table is a table from an external database linked at run time to a Microsoft Access format database. You cannot create a Table object on an attached table.

Table Objects:

- Table objects are record-oriented rather than set-oriented. The methods for Table objects let you only retrieve one row at a time, and only from one Table at a time. Table objects don't support SQL queries or subsets, unless you create a Dynaset or Snapshot from the Table.

- The Seek method finds a given record very quickly because it uses the Table's indexes. The Seek method is significantly faster than the find methods. For speed and flexibility, you can change the Index property of the Table object to change the order of the Seek. The Seek method can find values that are in indexed fields, but not in non-indexed fields.

- You can only order the data in Table objects based on existing indexes.

Example Showing Speed of Seek in a Table Versus SQL SELECT in a Dynaset
---

The fastest way to find a specific record in a recordset is usually a Seek method on a Table object. The equivalent SQL SELECT statement on a Dynaset object is usually very close in performance, as long as the SELECT finds just one record. A SQL SELECT that finds more than one record may be slower.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Double-click the form to open the code window. Add the following code to the Form Load event:

```
Sub Form_Load ()
   form1.Show  ' In form Load event, must show form before Print works.
   Dim t As Table
   Dim ds As Dynaset
   Dim db As database
   Set db = OpenDatabase("C:\ACCESS\NWIND.MDB")
   Set t = db.OpenTable("Customers")
   t.Index = "PrimaryKey"

   ' The following Seek is about as fast as the SQL SELECT below:
   Print Time$
   t.Seek "=", "WOLVH"
   Print Time$
   Print t("Customer ID") 'Print Customer ID value of current record

   Print Time$
   ' Enter the following two lines as one, single line:
   Set ds = db.CreateDynaset(
      "SELECT * FROM Customers WHERE 'Customer ID' = 'WOLVH' ")
   Print Time$
End Sub
```

3. Start the program (or press the F5 key). Close the form to end the program.

REFERENCES
==========

- Visual Basic online Help for the Table, Dynaset, and Snapshot objects, and their methods and properties.

- Microsoft Visual Basic version 3.0 "Professional Features Book 2," Data Access Guide section, Chapter 3.

- The VISDATA.MAK file installed in the VB3\SAMPLES\VISDATA directory loads extensive examples of data access. The VISDATA sample program uses every data access function in Visual Basic. You can refer to the VISDATA source code for examples of how to use each data access function.

Additional reference words: 3.00 pros and cons multi-user
KBCategory: Prg
KBSubcategory: PrgCtrlsCus

PRB: Common Dialog Open: Err=20476 Buffer lpstrFile Too Small

Article ID: Q110185

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

The following behavior occurs with the Open file dialog box of the Common Dialog custom control when you set the Flags property to &H200& or OFN_ALLOWMULTISELECT, to allow selection of more than one file at once.

If you highlight more than about 20 files at once in the Open file dialog box, the following error message displays when you click the OK button:

   The buffer at which the member lpstrFile points is too small.

Or, if error trapping is on, the Err function reports error number 20476.

In the Visual Basic environment, if you press F5 or choose Continue from the Run menu after getting this error, you get the following error message:

   Invalid filename.

CAUSE

This behavior occurs when the length of the Open dialog string containing the file names exceeds 256, the default MaxFileSize property value.

RESOLUTION

To resolve the problem, increase the MaxFileSize property to 2048:

```
CMDialog1.MaxFileSize = 2048
```

2048 is the maximum string size for the Common Dialog File-Open box. 256 is the default.

STATUS

This behavior is by design.

MORE INFORMATION

You can set the Flags property of the Common Dialog to &H200&, or OFN_ALLOWMULTISELECT, to allow multiple selections in the File Name list box. You can select more than one file at run time by pressing the SHIFT

key and using the UP and DOWN ARROW keys to select the desired files. The FileName property returns a string containing the names of all selected files. The names in the string are delimited by spaces.

Steps to Reproduce Behavior
---

1. Start a new project in Visual Basic. Form1 is created by default.

   If the common dialog CMDIALOG.VBX file is not automatically loaded by AUTOLOAD.MAK in Visual Basic, load CMDIALOG.VBX as follows: choose Add File from the File menu, and select CMDIALOG.VBX from your WINDOWS\SYSTEM directory.

2. Add a Common Dialog custom control to Form1.

3. Add the following to the Form Load event code:

```
Sub Form_Load ()
   CMDialog1.Flags = &H200&   ' Allows selection of more than 1 file.
   CMDialog1.Filter = "ALL|*.*"   ' File types to list in file box.
   ' CMDialog1.MaxFileSize = 2048  ' Add this to correct the behavior.
   CMDialog1.Action = 1      ' Action 1 = standard Open file dialog box
End Sub
```

4. Start the program (or press F5). Select 20 or files at once by selecting the first article then pressing SHIFT+END. That highlights all files down to the last article. Choose the OK button. The following error message displays if the number of characters in all the selected file names, plus one character for each file name, exceeds 256:

   The buffer at which the member lpstrFile points is too small.

   If error trapping (On Error Goto) is on, the Err function reports error number 20476. In the Visual Basic environment, if you press F5 or choose Continue from the Run menu after getting this error, you get the following error message:

   Invalid filename.

To improve this behavior, add the CMDialog1.MaxFileSize=2048 statement before setting the CMDialog1 Action property to 1. This increases the maximum allowed returned string size from 256 to 2048.

REFERENCES

The following corrections apply to the "Language Reference":

- On page 363 for the MaxFileSize property, change the "Applies to:" section to read "Common dialog (file dialogs)." Also say that the maximum MaxFileSize property value is 2048.

- On page 89, add MaxFileSize and FilterIndex to the Common Dialog Control "Properties (File Dialogs)" section.

Additional reference words: 3.00

KBCategory: Prg
KBSubcategory: PrgCtrlsCus

VB Crystal Reports Files to Distribute with Your .EXE Program

Article ID: Q110721

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

You cannot distribute the CRW.EXE file to other users. CRW.EXE is the Crystal report-designer environment for making custom reports.

To distribute custom reports that you made using CRW.EXE, add a Crystal Custom Control to your program. You can attach your custom report to the Crystal Custom Control by using the ReportFileName property. Visual Basic can compile that program into an executable .EXE file. If you want to distribute that program to other people who do not have Visual Basic version 3.0 installed, you need to include additional files listed in the More Information section below.

For an example showing how to use the Crystal Custom Control, run the CRVBXSAM.MAK sample project located in the Report directory in Visual Basic.

MORE INFORMATION

You can run the Crystal Reports design environment in any of the following ways:

- From the Window menu in Visual Basic, choose Report Designer.
- In the Visual Basic 3.0 program group in Windows Program Manager, double-click the "Crystal Reports for Visual Basic" icon.
- In File Manager in Windows, double-click the CRW.EXE file located in the Report directory in your Visual Basic directory.
- You can use the Shell function to invoke the Crystal Reports design environment from an executable program written in Visual Basic. For example:

```
x = Shell("CRW.EXE")
```

Microsoft does not allow you to distribute the Crystal Reports design environment (CRW.EXE) with your application. To access Crystal Reports from executable program written in Visual Basic, add a Crystal Custom Control to your program. You can make executable .EXE program files from programs that contain a Crystal Custom Control. If you want to distribute, sell, or test that .EXE file on another computer that does not have Visual Basic version 3.0 for Windows installed, you need to distribute the following files with that .EXE file:

```
   CRYSTAL.VBX       Crystal Custom Control file
   CRPE.DLL          Interface to the print engine.
   COMMDLG.DLL       Printer and file selection dialog box routines.
   CRXLATE.DLL       Only needed if your program uses ToWords() function.

 * PDIRJET.DLL       Crystal engine DLL.
 * PDBJET.DLL        Crystal engine DLL.
 * MSAJT110.DLL      Microsoft Access engine DLL.
 * MSAES110.DLL      Microsoft Access engine DLL.
** MSABC110.DLL      Microsoft Access engine DLL.
** CTL3D.DLL         Microsoft Access engine DLL.
   VB.INI            Visual Basic initialization file.
```

NOTES:

 * Files marked with one or two asterisks should be added to the list on

   page 582 in the "Visual Basic Version 3.0: Programmer's Guide."

** Files marked with two asterisks need to be added to the list in
   Appendix A of the "Crystal Reports for Visual Basic User's Manual."

Visual Basic's run-time dynamic link library must also be distributed with any executable Visual Basic program:

   VBRUN300.DLL

If you are making reports with ODBC, ODBC SQL Server, ODBC Oracle, Paradox, dBASE, or Btrieve, you will need to distribute additional database-specific files. See the REFERENCES section below.

REFERENCES
==========

 - See the "Runtime file requirements" topic in the Index of the Help menu topic for Crystal Reports. This is more up-to-date than the printed User's Manual.

 - Microsoft Visual Basic Version 3.0 "Professional Features Book 2", Crystal Reports for Visual Basic User's Manual.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgCtrlsCus

How to Use SizeMode Property of OLE Control to Size Display

Article ID: Q112043

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article shows by example how to use the SizeMode property of the OLE Control for Visual Basic version 3.0 to obtain a proportional display in a limited screen space.

MORE INFORMATION

The SizeMode property of the OLE Control determines how the OLE control is sized or how its image is displayed when it contains an object.

Here are the valid settings for the SizeMode property:

0 (Default) Clip -- The object is displayed in actual size. If the object is larger than the OLE control, its image is clipped by the control's borders, showing the upper-left portion of the image.

1 Stretch -- The object's image is sized to fill the OLE control. The height of the image is stretched (or shrunk) to fit the OLE control. The same thing happens to the width. As a result, you may get a distorted image. Height and width are independent, not proportional.

2 Autosize -- The OLE control is resized to display the entire object. The object is displayed in actual size. By using the resize event of the OLE control, you can adjust the HeightNew and WidthNew parameters to maintain size limits of the OLE control. However, the display behaves as if you had used the clip setting.

Step-by-Step Example

The goal of this example is to obtain a proportional display that is restricted to a limited screen space, that of the size of the form the OLE control is on.

The example uses AutoSize and Stretch settings to find a proportional best fit. The height and width of the container form supply the maximum height and width of the image. The Autosize property supplies best-fit information.

Then the Autosize property is set to Stretch mode for display.

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add an MSOLE2 control to the form. When the Insert Object dialog box asks what type of object to insert, select the Create From File option and a Microsoft Word .DOC file.

3. Set the SizeMode property of the OLE control to 1 (Stretch).

4. Add the following code to the Form_Load event:

```
Sub Form_Load()
   OLE1.Top = 0
   OLE1.Left = 0
End Sub
```

5. Add the following code to the Resize event of the OLE control:

```
Sub OLE1_Resize (heightNew As Single, WidthNew As Single)
   Dim wRatio As Single, hRatio As Single, cRatio As Single
   Dim MaxHeight As Single, MaxWidth As Single

   If OLE1.SizeMode = 2 Then
      MaxHeight = Me.Height
      MaxWidth = Me.Width
      'Calculate hRatio
      If heightNew > MaxHeight Then
         hRatio = 1 - (heightNew - MaxHeight) / heightNew
      Else
         hRatio = 1 + (MaxHeight - heightNew) / heightNew
      End If
      'Calculate wRatio
      If WidthNew > MaxWidth Then
         wRatio = 1 - (WidthNew - MaxWidth) / WidthNew
      Else
         wRatio = 1 + (MaxWidth - WidthNew) / WidthNew
      End If
      'Pick best ratio for cRatio
      If hRatio < wRatio Then
         cRatio = hRatio
      Else
         cRatio = wRatio
      End If
      'Apply changes
      OLE1.Height = CInt(cRatio * heightNew)
      OLE1.Width = CInt(cRatio * WidthNew)
      OLE1.SizeMode = 1
   ElseIf OLE1.SizeMode = 1 Then
      'Use AutoSize to recalculate ratio.
      OLE1.SizeMode = 2
   End If
End Sub
```

6. Run the program.

When you double-click the OLE control and make changes to your document, you should see the control itself change size to fit proportionally on your form.

Additional reference words: 3.00 OLE Automation OA OLE2
KBCategory:
KBSubcategory: PrgCtrlsCus

How to View Microsoft Word Toolbars Using OLE Control

Article ID: Q112044

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0
- Microsoft Word for Windows, version 6.0

SUMMARY

This article shows by example how to make a Visual Basic program display Microsoft Word version 6.0 toolbars. It is not possible to display the toolbars using the OLE control alone, so the following example uses an OLE object in combination with the OLE control.

MORE INFORMATION

When editing a Microsoft Word document that is in an OLE control, you may find it helpful to use the Microsoft Word toolbars. However, if you close all your toolbars, there is no way to use the OLE control alone to get the toolbars back. The following example shows how to access the Word toolbars after closing them.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add an MSOLE2 control (OLE1) to Form1. When the Insert Object dialog box asks what type of object to insert, choose the Create From File option. and select a Microsoft Word .DOC file.

3. Set the SizeMode property of the OLE1 control to 1 (Stretch).

4. Add a command button (Command1) to Form1.

5. Add the following code to the click event of the Command1 button:

```
Sub Command1_Click()
   Dim wbObject As Object
   ole1.Action = 7
   Set wbObject = CreateObject("Word.basic")
   wbObject.ViewToolbars "Standard", , , , , , , 1
End Sub
```

6. Run the program.

7. Double-click the OLE1 control. If you see the toolbars, close them and press the ESC key to return control to Visual Basic. Double-click the OLE1 control again. This time the toolbars are gone. Now click the

Command1 button to bring up the Standard toolbar. To bring up any of the other toolbars from this point, move the mouse pointer over the Standard toolbar. Then click the right mouse button, and choose the toolbar you want.

Additional reference words: 3.00
KBCategory:
KBSubcategory: IAPOLE PrgCtrlsCus

How to Print an Embedded Word Document in Visual Basic

Article ID: Q112196

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0
- Microsoft Word for Windows, version 6.0

SUMMARY

Microsoft Word for Windows version 6.0 disables the ability to use the FilePrint and FilePrintDefault methods while an object is being edited in an OLE container. While the menu options may not be enabled, it is still possible to get around this in code. This article explains how.

MORE INFORMATION

Commands that are part of the workspace are the responsibility of the top container (the Visual Basic application). That is, the application is responsible for the organization of windows, file level operations, and how edits are ultimately saved. The top container must supply a single File menu that contains file level commands such as Open, Close, Save, and Print. If the object is an opened object server application, the commands in its File menu are modified to show containership (Close & Return to <container doc>, Exit & Return to <container doc>).

A well-behaved OLE server will not allow workspace commands to be executed. This is why they are disabled. To work around the problem, edit the object in the server application instead of using in-place editing. In the server workspace, commands are enabled. Therefore, you can edit the object in the server workspace and use OLE Automation to control the server to execute the Workspace commands.

Example Program Using OLE Automation

The following example activates the Word object in the server, and uses OLE Automation to execute the FilePrintDefault method.

NOTE: By default, Word sets background printing On. If Word quits before printing is completed, the print job is aborted. There are two ways to work around this:

- Define the Word Objects globally. The objects will remain in memory until the container application (Visual Basic) quits. This is the easiest way to do it.

  -or-

- Disable background printing in Word. You can do this by using OLE automation. The command is not available during in-place editing. The

following example shows how to do this in code.

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a command button (Command1) to Form1.

3. Add an MSOLE2.VBX control (OLE1) to Form1. When the Insert Object dialog comes up, choose the Create From File option button, and select a Word for Windows document.

4. Add the following code to the Command1_Click event:

```
Sub Command_Click()
    ' Open application in separate application Window:
    ole1.Verb = -2
    ' Activate Object:
    ole1.Action = 7
    Dim WB As object
    ' Alias WordBasic Object:
    Set WB = ole1.Object.application.wordbasic
    ' Disable background printing:
    WB.ToolsOptionsPrint , , , , , , , , , , , 0
    WB.FilePrintDefault  'Print the Word Object.
    ' Hint: it may be necessary to check page layout parameters before
    ' printing. If parameters are outside of the printable region, Word
    ' will display an error message.
End Sub
```

5. Run the program, and click the Command1 button.

Additional reference words: 3.00
KBCategory:
KBSubcategory: IAPOLE PrgCtrlsCus

PRB: Serial Port Driver for WFW 3.11 Sends Extra Byte

Article ID: Q112418

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SYMPTOMS

The MSCOMM.VBX custom control may appear to send an unexpected byte when the port is closed by setting the PortOpen property to false.

CAUSE

There is a known problem with the miniport driver SERIAL.386 that was released with Windows for Workgroups 3.11. This is not a problem with the MSCOMM control since it can be reproduced by calling the CloseComm Windows API function directly.

STATUS

Microsoft has confirmed this to be a problem in Windows for Workgroups version 3.11. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

Steps to Reproduce Problem

1. Connect two machines using a null modem cable. You must run Windows for Workgroups 3.11 on the machine running Visual Basic.

2. Start a new project in Visual Basic, Form1 is created by default.

3. Add an MSCOMM (Comm1) control and a command button (Command1) to Form1.

4. Add the following code to the Command1 Click event procedure:

```
Sub Command1_Click ()
   comm1.PortOpen = True
   comm1.PortOpen = False
End Sub
```

5. Start the Terminal application on the second machine. In Terminal choose Settings Communications (ALT, S, C) and change the Baud Rate to 9600.

6. From the Run menu in Visual Basic on the first machine, choose Start (ALT, R, S) or press the F5 key to run the program. Click the

Command1 button and the machine running Terminal will indicate that a byte has been transmitted from closing the port.

The problem can also be reproduced with the following method.

1. Connect two machines using a null modem cable. You must run Windows for Workgroups 3.11 on the machine running Visual Basic.

2. Start a new project in Visual Basic. Form1 is created by default.

3. Add a command button (Command1) to Form1.

4. Add the following Declare statements to the General declarations section of Form1:

```
' Enter each of the following Declare statements on one, single line:
Declare Function OpenComm Lib "User" (ByVal lpComName As String, ByVal
   wInQueue As Integer, ByVal wOutQueue As Integer) As Integer
Declare Function CloseComm Lib "User" (ByVal nCid As Integer)
   As Integer
```

5. Add the following code to the Command1 Click event procedure:

```
Sub Command1_Click ()
  Dim id As Integer, success As Integer
  id = OpenComm("COM1", 1024, 128)
  success = CloseComm(id)
End Sub
```

6. Start the Terminal application on the second machine. In Terminal choose Settings Communications (Alt, S, C) and change the Baud Rate to 9600.

7. From the Run menu in Visual Basic on the first machine, choose Start (ALT, R, S) or press the F5 key to run the program. Click the Command1 button and the machine running Terminal will indicate that a byte has been transmitted from closing the port.

Additional reference words: serial comm port
KBCategory:
KBSubcategory: PrgCtrlsCus

How to Save an Embedded Word Document in Visual Basic

Article ID: Q112440

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Word for Windows, version 6.0

SUMMARY

Word version 6.0 for Windows disables the ability to do a FileSaveAs while an object is being edited in an OLE container. While these methods may not be enabled, it is possible to work around this limitation in code. This article explains how.

MORE INFORMATION

Commands that are part of the workspace are the responsibility of the top container (the Visual Basic application). That is, the application is responsible for the organization of windows, file level operations, and how edits are ultimately saved. The top container must supply a single File menu that contains file level commands such as Open, Close, Save, and Print.

If the object is an opened object server application, the commands in its File menu are modified to show containership (Close & Return to <container doc>, Exit & Return to <container doc>).

A well-behaved OLE server will not allow workspace commands to be executed. This is why they are disabled. To work around the problem, edit the object in the server application -- without using in-place editing. In the server, you'll find that the workspace commands are enabled. Therefore edit the object in the server and use OLE Automation to control the server to execute

the Workspace commands.

Step-by-Step Example

The following example uses an OLE2 control called OLE1, which contains an embedded Word version 6.0 document and a CommonDialog control called CMDialog1. To make the code generic, the OLE1 control is passed to the WordFileSave subroutine.

1. Start a new project in Visual Basic, Form1 is created by default.

2. Add a command button (Command1), MSOLE2.VBX (OLE1) control, and a CMDIALOG.VBX (CMDialog1) control to Form1.

3. Add the following code to the Command1_Click event:

```
Sub Command1_Click ()
   ' Pass the name of the Control to WordFileSave subroutine.
   WordFileSave OLE1
End Sub
```

4. Add the following code to the general declarations section of Form1:

```
Sub WordFileSave (OLECtrl As Control)
   'Purpose: Example of how to save an embedded Word object from
   'Visual Basic as a Word Document.

   'Overview of technique:      '
   'Activate Object. Select its contents. Copy contents to clipboard.
   'Launch a hidden instance of Word. Create a new file.
   'Paste clipboard into document. Save document.

   Dim Word As Object  'Alias to Hidden instance of Word.
                       'Only if Word is not already running.
   Dim WB As Object    'alias to WordBasic object.

   OLECtrl.Action = 7  'Activate OLE control. This must be done in order
                       'to have the Word Basic alias act on the correct
                       'instance of Word.
   Set WB = CreateObject("Word.Basic") 'Set the object variable.

   WB.editselectall    'Select the contents of the embedded object.
   WB.EditCopy         'Copy the selection to the clipboard.
   OLECtrl.Action = 9  'Deactivate the OLE control. This must be
                       'done before the following set statements to
                       'reference the correct instances of Word.

   'Use the Common dialog control to display a SaveAs dialog.
   CMDialog1.Filter = "Word Document (*.Doc)|*.doc"  'Set the filter
   CMDialog1.DefaultExt = "*.doc"            'Set the default extension
   CMDialog1.FileName = OLECtrl.SourceDoc 'Set default filename
   CMDialog1.Action = 2                      'Display the dialog.


   Set WB = Nothing    'Free the WB object reference.
   Set Word = GetObject("", "Word.Document.6")  'Create a hidden inst.
   Set WB = Word.application.Wordbasic  'Set WB to the WordBasic object
                                        'of the new instance of Word.

   WB.filenew        'Create a New file in hidden instance of Word.
   WB.editpaste      'Paste contents of clipboard into new document.
   WB.filesaveas CMDialog1.Filename  'Save file as selected by user.
   WB.fileclose       'Close document.

   Set WB = Nothing   'Free WordBasic object
   Set Word = Nothing 'Free Word Document object, if Word wasn't
                      'running previously, Word will shut itself down
                      'from memory; otherwise, it is up to the user to
                      'shut Word down.
End Sub
```

5. Run the program. The program will ask you to input a name and then save

the document to the name that you input.

Additional reference words: 3.00
KBCategory:
KBSubcategory: IAPOLE PrgCtrlsCus

How to Create a Gantt Chart in VB Using a Graph Custom Control

Article ID: Q112650

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

A Gantt chart is a horizontal bar chart used for project planning and reporting. The Graph custom control that comes with the Professional Edition of Visual Basic for Windows can be used to create a Gantt chart. This article shows by examnple how to create a Gantt chart.

MORE INFORMATION

To create a Gantt chart using the Graph custom control, you need to know how many bars your graph will require. Typically this includes one project bar and several activity bars. You will also need to know the number of sections required in each bar. Typically each bar is divided into two sections -- one to show actual progress on the project, the other to show plans for the time remaining.

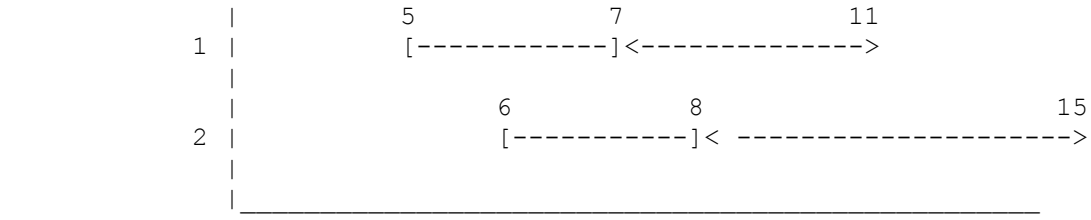
In the graph custom control, NumPoints represents the number of horizontal bars needed, so the following is true:

```
NumSets = (Number of Bars) * (Sections on a Bar)

   or

NumSets = NumPoints * (typically 2)
```

To make the graph more readable, you can set the range of the horizontal axis using YAxisStylealong with YAxisMax. To place tick marks, use YAxisTicks. To clear previous data assigned to the graph, use DataReset.

To plot the bars on the graph, you need to assign values to the GraphData property of Graph1. GraphData values that follow ThisSet = 1 assignment indicate the beginning of the section on a bar, and the corresponding GraphData values following the ThisSet = 2 assignment, indicate the end of the section. For example:

To plot the following Gantt chart:



Make the following assignments:

```
Graph1.ThisSet = 1     ' Beginning of intervals
Graph1.GraphData = 5
Graph1.GraphData =  6
Graph1.GraphData =  7
Graph1.GraphData =  8
Graph1.ThisSet = 2      ' End of intervals
Graph1.GraphData =  7
Graph1.GraphData = 8
Graph1.GraphData =  11
Graph1.GraphData = 15
```

After assigning values to the graph, set the GraphType to 5 to indicate a Gantt chart, and set GraphStyle to 1 if you want spaced bars. Set DrawMode to 2 and DrawStyle to 1 if you want color.

Step-by-Step Example

This example creates the Gantt chart.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Place a graph control (Graph1) on Form1.

3. Add the following code to the Form_Load event:

```
Sub Form_Load ()
   Graph1.NumSets = 4
   Graph1.YAxisStyle = 2
   Graph1.YAxisMax = 20
   Graph1.YAxisTicks = 10
   Graph1.DataReset = 1
End Sub
```

4. Add the following code to the Graph1_Click event:

```
Sub Graph1_Click ()
   Graph1.ThisSet = 1     ' Set starting values for bar sections
   Graph1.GraphData = 5   ' Left half
   Graph1.GraphData = 6
   Graph1.GraphData =  7  ' Right half
   Graph1.GraphData =  8

   Graph1.ThisSet = 2     ' Set ending values for bar sections
   Graph1.GraphData = 7   ' Left half
   Graph1.GraphData = 8
   Graph1.GraphData = 11  ' Right half
   Graph1.GraphData = 15

   Graph1.GraphType = 5
   Graph1.GraphStyle = 1

   Graph1.DrawMode = 2
   Graph1.DrawStyle = 1
```

```
   End Sub
```

5. Run the program, and click the graph.

Additional reference words: 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

How to Size the Rows and Columns of a Grid to Fit Exactly

Article ID: Q112861

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic Programming System for Windows, version 2.0
- Standard and Professional Editions of Microsoft Visual Basic Programming System for Windows, version 3.0

SUMMARY

This article explains how to size the rows and columns of a Visual Basic grid control to allow for the display of all cells in the grid within the current bounds of the grid. This technique is useful when the number of rows and columns in the grid changes dynamically and you need to see all the cells at once.

MORE INFORMATION

To allow all cells to be proportional and sized so that all can be seen within the current grid, you need to set the ScaleMode property of the grid's parent window to pixels. If the grid is on a form, the grid's parent window is the form. However, if the grid is nested within a picture box, then the picture box is the grid's parent window. Changing the parent's ScaleMode property to pixels changes the grid control's Height and Width properties to pixels. Once the grid's height and width are in pixels, it is possible to calculate the row height and column width in pixels. The example routine given below changes the form's ScaleMode to pixels temporarily, and then changes it back after it is done.

To calculate the height of a row in pixels, the example code uses this formula:

```
PixelRowHeight = (Grd.Height - 2) \ nRows
```

The formula subtracts two pixels from the grid height to take into account the top and bottom border. Then it divides the result by the number of rows to get the height in pixels.

To make the rows fit exactly into the grid, the example code calculates the remaining pixels. The number of remaining pixels is the remainder of the division used to calculate the row height. The remaining pixels are calculated by using this formula:

```
PixelsRemaining = (Grd.Height - 2) Mod nRows
```

The example distributes one pixel of the remaining pixels to the height of each row until it runs out of pixels.

Once you determine the height of a row in pixels, you can set the row height. The RowHeight property expects the height in twips. So it is

necessary to multiply the pixel height by Screen.TwipsPerPixelY to convert to twips. The row height does not include the border, so one pixel must be subtracted from the row height prior to converting the height to twips and assigning it.

Similar methods are used to calculate the width of each row in pixels and assign it to the grid.

Step-by-Step Example
----

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the following controls and set their properties using the following table as a guide:

Name	Properties	Value
Text1	Text	2
Text2	Text	2
Command1	Caption	Resize
Grid1	ScrollBars	0-None

3. Put the following code in the General Declarations section of the form:

```
' Enter the following two lines as one, single line:
Sub SizeGrid (Frm As Form, Grd As Grid, nRows As Integer,
   nCols As Integer)

   Dim FormScaleMode As Integer  'Used to save ScaleMode

   'Save ScaleMode of form and change to pixels:
   FormScaleMode = Frm.ScaleMode
   Frm.ScaleMode = 3 'Pixels

   'Determine the height of a row in pixels and the remaining pixels:
   PixelRowHeight = (Grd.Height - 2) \ nRows
   PixelsRemaining = (Grd.Height - 2) Mod nRows

   'Set the height of each column:
   Grd.Rows = nRows
   For i = 0 To nRows - 1
      If i < PixelsRemaining Then
         'Set the height of a row:
         'One pixel is added to eat up remainder and get a perfect fit.
         Grd.RowHeight(i) = PixelRowHeight * Screen.TwipsPerPixelY
      Else
         'Set the height of a row
         Grd.RowHeight(i) = (PixelRowHeight - 1) * Screen.TwipsPerPixelY
      End If
   Next

   'Determine the width of a column and the remaining pixels:
   PixelColWidth = (Grd.Width - 2) \ nCols
   PixelsRemaining = (Grd.Width - 2) Mod nCols

   'Set the width of each column:
```

```
      Grd.Cols = nCols
      For i = 0 To nCols - 1
         If i < PixelsRemaining Then
            'Set the width of a column:
            'One pixel is added to eat up remainder and get a perfect fit
            Grd.ColWidth(i) = PixelColWidth * Screen.TwipsPerPixelX
         Else
            'Set the width of a column:
            Grd.ColWidth(i) = (PixelColWidth - 1) * Screen.TwipsPerPixelX
         End If
      Next

      'Return form to original ScaleMode:
      Frm.ScaleMode = FormScaleMode

   End Sub
```

4. Add the following code to the command button's click event:

```
   Sub Command1_Click ()
      Call SizeGrid(Form1, Grid1, CInt(Text1), CInt(Text2))
   End Sub
```

5. Save the project.

6. Run the example.

Enter values for the number of rows and columns in the two text boxes and click the command button. The grid should have the exact number of rows and columns that you specified. Also, the rows and columns should fit exactly into the grid.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

How to Send a Mail Message Using Visual Basic MAPI Controls

Article ID: Q113033

The information in this article applies to:

- Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

This article demonstrates how to create a Microsoft Mail message programmatically and send it by using the Visual Basic MAPI controls. You can use this technique to automate the process of sending messages.

MORE INFORMATION

The professional editions of Visual Basic versions 2.0 and 3.0 come with the custom control MSMAPI.VBX, which contains two controls (MAPI session control and MAPI message control) for creating Microsoft Mail enabled applications. The MAPI session control is used to manipulate a Microsoft Mail session, and the MAPI message control is used to create and manipulate mail messages. It is possible to use these two controls to automate the process of sending mail messages.

The following example illustrates the use of the MAPI controls to send messages. The example creates a mail message with an attachment and sends it to a recipient.

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add MSMAPI.VBX to the project.

3. Add a MAPI session control (MAPISession1) and a MAPI message control (MAPIMessages1) to the form.

4. Add a command button (Command1) to the form.

5. Put the following code in the command button click event.

```
Sub Command1_Click()
   'MAPI constants from CONSTANT.TXT file:
   Const SESSION_SIGNON = 1
   Const MESSAGE_COMPOSE = 6
   Const ATTACHTYPE_DATA = 0
   Const RECIPTYPE_TO = 1
   Const MESSAGE_RESOLVENAME = 13
   Const MESSAGE_SEND = 3
   Const SESSION_SIGNOFF = 2
```

```
    'Open up a MAPI session:
    MapiSession1.Action = SESSION_SIGNON
    'Point the MAPI messages control to the open MAPI session:
    MapiMessages1.SessionID = form1.MapiSession1.SessionID

    MapiMessages1.Action = MESSAGE_COMPOSE   'Start a new message

    'Set the subject of the message:
    MapiMessages1.MsgSubject = "This is the subject."
    'Set the message content:
    MapiMessages1.MsgNoteText = " This is the mail message."

    'The following four lines of code add an attachment to the message,
    'and set the character position within the MsgNoteText where the
    'attachment icon will appear. A value of 0 means the attachment will
    'replace the first character in the MsgNoteText. You must have at
    'least one character in the MsgNoteText to be able to attach a file.
    MapiMessages1.AttachmentPosition = 0
    'Set the type of attachment:
    MapiMessages1.AttachmentType = ATTACHTYPE_DATA
    'Set the icon title of attachment:
    MapiMessages1.AttachmentName = "System Configuration File"
    'Set the path and file name of the attachment:
    MapiMessages1.AttachmentPathName = "C:\CONFIG.SYS"

    'Set the recipient type. RECIPTYPE_TO sends message to recipient:
    MapiMessages1.RecipType = RECIPTYPE_TO
    'Set the recipient's E-Mail name.
    'You can have multiple recipients separated by semicolons
    '*Change to a valid e-mail name*
    MapiMessages1.RecipDisplayName = "EddieSpaghetti"
    'MESSAGE_RESOLVENAME checks to ensure the recipient is valid and puts
    'the recipient address in MapiMessages1.RecipAddress
    'If the E-Mail name is not valid, a trappable error will occur.
    MapiMessages1.Action = MESSAGE_RESOLVENAME
    'Send the message:
    MapiMessages1.Action = MESSAGE_SEND

    'Close MAPI mail session:
    MapiSession1.Action = SESSION_SIGNOFF
End Sub
```

6. Save the project.

7. Run the code, and click the command button.

The program should start a MAPI session, create a message, send the message, and then close the session.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus

VB Out of Stack Space Error w/ LoadPicture in Form_Paint Event

Article ID: Q72675

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

An "Out of stack space" error can occur when you use a LoadPicture method within a Form_Paint event.

MORE INFORMATION

The Visual Basic stack can be exhausted when the LoadPicture method is executed within a [control/form]_Paint event. The LoadPicture method generates a [control/form]_Paint event itself, and when performed within a _Paint event, the program will repeat the cycle until the stack is exhausted.

The following code example demonstrates that the Form_Paint event is a recursive procedure when a LoadPicture method is included in the _Paint event code.

After you add the code to your program, run the program and notice how many times the message "Form_Paint Count :" is displayed within the Immediate Window before you receive the "Out of stack space" error message.

```
Sub Form_Paint ()
      Static Count
      Count = Count + 1
      Debug.Print "Form_Paint Count : "; Count
      Form1.picture = LoadPicture("c:\windows\chess.bmp")
End Sub
```

To remedy the situation, move the LoadPicture to another event handler, such as the Form_Load event. Since these bitmaps are automatically refreshed when needed, you don't have to maintain the picture within a Paint event.

The Visual Basic stack is limited to 16K bytes, and cannot be changed.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: APrgGrap PrgOptMemMgt

Comments and Blank Lines Increase Size of VB 1.0 .EXE File

Article ID: Q73697

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Each line containing blank space or a comment in any code window of a Visual Basic for Windows, version 1.0 application adds 2 bytes to the size of the compiled executable file (.EXE).

This behavior does not occur in Microsoft Visual Basic programming system for Windows, versions 2.0 and 3.0.

MORE INFORMATION

The 2 byte overhead for each line containing blank space or a comment is generated as part of the pseudo-code for the application in the VB.EXE development environment. The program is run in "interpreted mode" based on this pseudo-code. Because an .EXE program is generated based on this pseudo-code (in other words, Visual Basic for Windows does not use a compiler and linker), the 2 byte overhead is copied to the .EXE program. The only workaround for this behavior is to remove comments and blank lines before compiling the Visual Basic for Windows project.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgOptTips

How to Optimize Size and Speed of Visual Basic Applications

Article ID: Q73798

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article describes how to optimize Visual Basic applications for size and speed.

MORE INFORMATION

Below are guidelines to help increase speed, available resources, available RAM, and available disk space in Visual Basic:

Increase Speed

- Preload forms.
- Store graphics as bitmaps.
- Place debug routines in a separate module.
- Use Dynamic Link Library (DLL) routines.

Increase Available Resources

- Create simulated controls using a graphic object.
- Draw graphics images during run time.

Increase Available RAM

- Use Integer variables whenever possible.
- Create dynamic arrays to free arrays when not needed.
- Drop/unload controls and forms when not needed.
- Use local variables.

Increase Disk Space

- Build controls at load time.
- Minimize header size.
- Delete unnecessary functions and subroutines.
- Delete unused objects and associated methods.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgOptTips

How to Determine Display State of a VB Form, Modal or Modeless

Article ID: Q77316

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Show method in the Visual Basic for Windows language can display a form either as modal or modeless. No direct support exists in the language to determine the display state of the form without maintaining global variables that contain the display state of the form. However, the Windows API function GetWindowLong can be used to check the display state of the form.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

When Visual Basic for Windows displays a modal form (.Show 1), all other forms will be modified to contain the Window Style WS_DISABLED. The Windows API function GetWindowLong can be used to return the Window Style of another form to check for the WS_DISABLED style.

The following code demonstrates this process:

Add the following to the General Declarations section of Form1 and Form2:

```
DefInt A-Z
Global Const GWL_STYLE = (-16)
Global Const WS_DISABLED = &H8000000
Declare Function GetWindowLong& Lib "user" (ByVal hWnd, ByVal nIndex)
```

Form1.Frm

```
Sub Form_Click ()
  ' Flip between "Modeless" and "Modal" display states.
  Static ShowStyle
  Unload form2
  form2.Show ShowStyle
  ShowStyle = (ShowStyle + 1) Mod 2
End Sub
```

Form2.Frm

```
Sub Form_Paint ()
  ' Get the Window Style for Form1.
```

```
   WinStyle& = GetWindowLong(Form1.hWnd, GWL_STYLE)
   If WinStyle& And WS_DISABLED Then
      ' The WS_DISABLED style is set on "FORM1" when "FORM2"
      ' is displayed with the Modal flag (Show 1).
      Print "Modal    - Show 1"
   Else
      ' The WS_DISABLED style is not set on "FORM1" when "FORM2"
      ' is displayed with the Modeless flag (Show or Show 0).
      Print "Modeless - Show"
   End If
End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgWindow PrgOptTips

Example of Sharing a Form Between Projects in VB for Windows

Article ID: Q81222

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Microsoft Visual Basic for Windows allows you to share forms between projects. When you make a change to a shared form in one project, that change will be automatically updated in the other projects that share the form.

A workaround is also available if you want to change a shared form but do not want to update the form in other projects.

Further below is an example of how to use this shared form feature in Visual Basic for Windows, and an example of how to change a shared form without updating it in shared projects.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Below are two examples: the first shows how to update shared forms, and the second demonstrates how to change a shared form without having those changes affect the same form in other projects.

Example 1

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Add a couple text boxes and command buttons to Form1 by double-clicking the appropriate tools in the toolbox and placing the controls at certain locations on the form. From the Properties Bar, change the FormName property of Form1 to Test1.

3. From the File menu, choose Save Project As. Save Test1 as TEST1.FRM and save the project as TEST1.MAK.

4. Start a new project by choosing New Project from the File menu.

5. From the File menu, choose Add File, and select TEST1.FRM.

6. Once TEST1.FRM is loaded into the project, delete the command buttons, and replace them with picture boxes.

7. From the File menu, choose Save Project As. Save the project as TEST2.MAK, and save TEST1.FRM with the same name.

8. From the File menu, choose Open Project. In the Files box, select TEST1.MAK.

Notice that the form has been updated to include picture boxes and the command buttons were deleted.

Example 2

(Note that the following steps are very similar to the example above, but with a change in step 5.)

This example demonstrates how to share forms between projects, but with the forms being designed differently.

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Add a couple text boxes and command buttons to Form1 by double-clicking the appropriate tools in the toolbox and placing the controls at certain locations on the form. From the Properties Bar, change the FormName property of Form1 to Test3.

3. From the File menu, choose Save Project As. Save Test3 as TEST3.FRM and save the project as TEST3.MAK.

4. From the File menu, choose New Project.

5. From the File menu, choose Add File. In the Files box, select TEST3.FRM. Once the file is loaded, delete the command buttons and replace them with picture boxes.

6. From the File menu, choose Save File As, and save the form as TEST4.FRM.

7. From the File menu, choose Save Project As, and save the project as TEST4.MAK.

8. From the File menu, choose Open Project. In the Files box, select TEST3.MAK.

Notice that the form's controls have NOT been updated with picture boxes.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgOptTips

Limit of 15 or 31 Timer Controls in Visual Basic for Windows

Article ID: Q81455

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Windows versions 3.0 and 3.1

SUMMARY
=======

The Visual Basic timer control is very useful for initiating specific code at certain time intervals. Microsoft Windows version 3.0 allows up 16 timers, and Microsoft Windows version 3.1 allows up to 32 timers. Windows requires the use of one of the timers for itself, so you can have up to 15 timers in a Visual Basic version 1.0 or 2.0 application in Windows version 3.0 and up to 31 timers in Microsoft Windows version 3.1.

Additional reference words: 1.00 2.00 3.00 3.10
KBCategory:
KBSubcategory: PrgOptTips

Redim: Array Already Dimensioned Msg After Dim w/ Subscripts

Article ID: Q83238

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can use the ReDim statement to redimension a dynamic array only if the array has been previously dimensioned with empty parentheses (no subscripts), or if the array has been previously redimensioned with ReDim. If you specified subscripts to originally dimension the array in a Global or Dim statement, or if you previously dimensioned the array using the Static statement in a Sub or Function, redimensioning the array will cause an "Array already dimensioned" error.

MORE INFORMATION

You can use the ReDim statement to dimension an array that you have already declared with empty parentheses either in the Global module or in the general Declarations section. You can also use ReDim to redimension arrays that you have dimensioned with ReDim previously from any Sub or Function procedure.

Therefore, if you need to redimension an array in your program after using the array, first dimension the array in the Global module using the Global statement, or in the general Declarations section using Dim with no subscripts. Then use ReDim with the original dimensions. Later on, you can redimension this array again with different subscripts.

This will enable you to change the number of subscripts in each dimension of an array [for example, from x(15, 15) to x(32, 24)]. However, you cannot use ReDim to change the number of dimensions in an array. For example, you cannot redimension an array from two dimensions, such as x(15, 15), to three dimensions, such as x(64, 1, 5).

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgOptTips

LONG: List of VB Version 1.0 for Windows Trappable Errors

Article ID: Q87003

The information in this article applies to:

- Microsoft Visual Basic Programming System for Windows, version 1.0

SUMMARY

This article lists error codes, messages, and explanations of the errors that you can trap at run-time using the On Error statement and the Err function in Microsoft Visual Basic.

MORE INFORMATION

The following information was taken from the help file VB.HLP.

Trappable Errors

3 Return without GoSub

A Return statement doesn't have a corresponding GoSub statement.

Unlike For...Next, While...Wend, and Sub...End Sub, which are matched at compile time, GoSub and Return are matched at run time.

5 Illegal function call

Possible causes:

- A statement or function has an improper or out-of-range argument. For example:
  - A negative or unreasonably large subscript.
  - A negative number raised to a non-integer power.
  - A negative record number in a Get or Put statement.
  - An I/O function or statement (Loc or LOF, for example) performed on a device that does not support it.
- Strings are concatenated to create a string greater than 65,535 characters in length.
- Invalid method call, such as an inappropriate method argument. See Help on the method for the valid arguments.

6 Overflow

Possible causes:

- The result of an assignment, calculation, or data type conversion is too large to be represented within the range allowed for that type of variable.
- An assignment to a property exceeds the maximum value the

property can accept. The assignment may not necessarily be one that you have made in your code. Some methods make automatic assignments to properties. For example, in the course of execution, the Print method changes CurrentX and CurrentY properties. Any attempt to print at an invalid CurrentX or CurrentY results in an Overflow error.

7 Out of memory

More memory was required than is available. To prevent this error you should try the following:

- Close any unneeded applications, documents, or source files that are in memory.
- If you have extremely large modules or procedures, consider breaking them into smaller ones. This doesn't save memory but it can prevent hitting 64K segment boundaries.
- If you are running Windows in standard mode on a 386 or 486 computer, try enhanced mode.
- If you are running Windows in enhanced mode, free up some disk space, or least ensure that some space is available.
- Eliminate terminate-and-stay-resident programs.
- Eliminate unneeded device drivers.
- Reduce the size of your MS-DOS buffers. To do this, reset the "Buffers =" setting in your CONFIG.SYS file and reboot.

9 Subscript out of range

Possible causes:

- Reference to a non-existent array element. The subscript may be larger or smaller than the range of possible subscripts, or the array may not have dimensions assigned at this point in the program.
- You are trying to Dim or ReDim an array to a size greater than 64K (65,535) bytes.

10 Duplicate definition

The specified name is already used at this level of scope. You can use the Find command on the Code menu to help you locate the duplicate name. Before you do this, remove the type declaration character, if there is one, since a conflict occurs if the names are the same and the type declaration characters are different.

Possible causes:

- A new variable or procedure has the same name as an existing variable or procedure.
- A Const statement uses the same name as an existing variable or procedure.
- You declared a fixed array more than once.
- You tried to use Dim or ReDim to declare the dimensions of an already-dimensioned dynamic array without first using Erase to deallocate the array.

11 Division by zero

This error is caused by dividing by zero in an expression.

13 Type mismatch

Possible causes:

- The variable or property is not of the correct type. For example, a variable that requires an integer value cannot accept a string value.
- An If TypeOf statement was used with something other than a control.
- An object that is not a form has been passed to a procedure that is expecting a form as an argument.
- An object that is not a control has been passed to a procedure that is expecting a control as an argument.

14 Out of string space

Possible causes:

- You tried to exceed the 64K string space allowed for all your global and module-level string variables.
- You tried to exceed the 64K string space allowed for all your local (procedure-level) string variables.
- You tried to create a single string array larger than 64K.
- You may have run out of memory, which has prevented a 64K string space from being allocated.

16 String formula too complex

A string expression is too complicated. Strings not assigned to variables (such as those returned by functions) are assigned to temporary locations during string expression evaluation. Having a large number of such strings can cause this error. Try assigning these strings to variables and use the variables in the expression instead.

17 Can't continue

You have made a change to the code that prevents execution from continuing. Choose Restart or End from the Run menu.

19 No Resume

The program encountered the end of the program while the program was executing an error-handling routine. Add a Resume statement to the error-handling routine. If you want to exit the error-handling routine without resuming, use an Exit statement.

20 Resume without error

A Resume statement has been encountered but there is no active error-handling routine because there is no On Error statement.

28 Out of stack space

Possible Causes:

- Too many active Function or Sub calls. Check that both event and general recursive procedures are not nested too deeply and that they terminate properly.
- Any cascading events.
- Local variables require more local variable space than is available. Try declaring some variables at the module level instead. You can also use the Static keyword with Sub or Function to declare the entire procedure, in which case all local variables will be static. Or you can use the Static statement to declare static variables within individual procedures.
- Fixed-length strings use more stack space than variable-length strings. Try redefining some of your fixed-length strings as variable-length strings.
- Too many nested DoEvents.

35 Sub or Function not defined

A Sub or Function procedure is called but is not defined.

Possible causes:

- The specified procedure is not visible to the calling procedure. Procedures in forms cannot be called from procedures outside the form. Use Find on the Code menu to locate the procedure.
- You have declared a dynamic-link library routine, but the routine is not in the specified library.
- You have misspelled the name of your procedure.

48 Error in loading DLL

The specified DLL cannot be loaded. This is usually because the file specified with the Lib clause in the Declare statement is not a valid DLL.

Possible causes:

- The file is not DLL executable.
- The file is not a Windows DLL.
- The file is an old Windows DLL incompatible with Windows protect mode.

51 Internal error

An internal malfunction occurred in Visual Basic. Use the Product Assistance Request form included with your documentation to report to Microsoft the conditions under which the message appeared.

52 Bad file name or number

A statement or command refers to a file with a file number or filename that is not specified in the Open statement or is out of the range of file numbers specified earlier in the program.

53 File not found

Possible causes:

- A Kill, Name, or Open statement refers to a file that does not exist.
- An attempt has been made to call a procedure in a DLL and the library filename specified in the Lib clause of the Declare statement cannot be found.

54 Bad file mode

Possible causes:

- A Put or Get statement specified a sequential file.
- A Print # statement specified a sequential file opened for input.
- An Input # statement specified a file opened for output or appending.

55 File already open

Possible causes:

- A sequential-output-mode Open statement was executed for a file that is already open.
- A Kill statement refers to an open file.

57 Device I/O error

An input or output error occurred while your program was using a device such as the printer or disk drive.

58 File already exists

The filename specified in a Name statement is identical to a filename that already exists.

59 Bad record length

The length of a record variable for a Get or Put statement does not match the length specified in the corresponding Open statement. Because a 2-byte descriptor is always added to a variable-length string Put to a random access file, the variable-length string must at least two characters shorter than the record length specified in the Len clause of the Open statement.

61 Disk full

Possible causes:

- There isn't enough room on the disk for the completion of a Print #, Write #, or Close operation.
- There isn't enough room on the disk for Visual Basic to create files it requires for successful operation.

Move some files to another disk, or delete some files.

62 Input past end of file

An Input # statement is reading from a null (empty) file, or from a file in which all data has already been read. To avoid this error, use the EOF function to detect the end-of-file just before the Input # statement.

63 Bad record number

The record number in a Put or Get statement is less than or equal to zero.

64 Bad file name

A filename does not follow MS-DOS naming conventions.

67 Too many files

Possible causes:

- There is a limit to the number of disk files that can be open at one time. This limit is a function of the "Files=" setting in your CONFIG.SYS file. Increase that number and reboot.
- The operating system has a limit to the number of files in the root directory (usually 512). If your program is opening, closing, and/or saving files in the root directory, change your program so it uses a subdirectory.

68 Device unavailable

The device you are trying to access is not online or does not exist.

70 Permission denied

An attempt was made to write to a write-protected disk, or to access a locked file. For example, this error will occur if an Open For Output statement is performed on a write-protected file.

71 Disk not ready

There is no disk in the drive specified, or the drive door is open. Insert a disk in the drive, close the door, and retry the operation.

74 Can't rename with different drive

You cannot use the Name statement to rename a file with a new drive designation. Write the file to another drive and delete the old file with the Kill statement.

75 Path/File access error

During an Open, MkDir, ChDir, or RmDir operation, the operating system could not make a connection between the path and the

filename.

Make sure the file specification is formatted correctly. A filename can contain a fully qualified or relative path. A fully qualified path starts with the drive name (if the path is on another drive) and lists the explicit path from the root to the file. Any path that is not fully qualified is relative to the current drive and directory.

This error can also occur while attempting to save a file that would replace an existing read-only file.

76 Path not found

During an Open, MkDir, ChDir, or RmDir operation, the operating system was unable to find the specified path. Make sure the path is typed correctly.

260 No timer available

Possible causes:

- There are too many timer controls active. There is a limit of 16 timer controls in the environment.
- There is not enough memory to load a timer control. Try to free some memory by closing some applications.

280 DDE channel not fully closed; awaiting response from foreign application

Your Visual Basic application has terminated one DDE conversation with another application and has attempted to start another, but the other application has not finished handling the termination of the first conversation.

Possible causes:

- The foreign application is waiting for a response from the user. Switch to that application and take an action appropriate to the message it is displaying.
- Your code is not yielding to allow other applications to handle events. Call the DoEvents function and try establishing the link again.

281 No More DDE channels

Too many DDE conversations active at the same time. Terminate some existing DDE conversations by setting LinkMode to 0 (None) before attempting to establish new conversations.

282 No foreign application responded to a DDE initiate

Visual Basic could not find an application and topic corresponding to the application name and topic in the LinkTopic property.

Possible causes:

- The application specified in LinkTopic is not running.
- The application is running, but doesn't recognize the topic of the link.

283 Multiple applications responded to a DDE initiate

At least two running applications responded to the application name and topic you specified in the LinkTopic property. This can happen if several instances of the same application are running and you attempt to initiate a DDE conversation on a topic more than one of them recognize. To establish a DDE conversation, the combination of application and topic must be unique.

284 DDE channel locked

You have attempted to initiate a new DDE conversation or perform a DDE method on a DDE link that another application has not freed. Try calling the DoEvents function before setting LinkTopic to Hot (1) or Cold (2) or before performing a LinkExecute, LinkPoke, LinkRequest, or LinkSend method.

285 Foreign application won't perform DDE method or operation

An application refused to perform the DDE method or operation you attempted.

Possible causes:

- You supplied data or commands that the other application did not recognize. Check the application's documentation to see what data or commands it recognizes.
- The LinkItem property is not set to an item that the other application recognizes as valid for the topic of the conversation. Check the application's documentation to see what items it recognizes.

286 Timeout while waiting for DDE response

The other application in a DDE conversation did not respond in the time specified by the LinkTimeout property.

Possible causes:

- The other application is not responding because it is waiting for a response from the user. Switch to that application and close the dialog box or take an action appropriate to the message it is displaying.
- The LinkTimeout property is set to a value that is too low. Try increasing the value.
- The other application is too busy to respond to DDE messages. Try calling the DoEvents function before performing this DDE operation.

287 User pressed Alt key during DDE operation

You pressed the Alt key while waiting for a DDE operation to be completed. If DDE operations are taking too long, try setting the

LinkTimeout property to a lower value.

288 Destination is busy

The other application in the DDE conversation is busy and cannot perform a DDE operation. Try calling DoEvents and attempt the DDE operation again.

289 Data not provided in DDE operation

Visual Basic has encountered an unexpected error while attempting to perform a DDE operation. The other application informed Visual Basic it was supplying data in a DDE conversation but did not provide it when requested. The other application may not be performing DDE correctly.

290 Data in wrong format

An application in a DDE conversation supplied data in an unexpected format. It may not be performing DDE correctly.

Possible causes:

- The application is supplying data in a format that Visual Basic does not recognize. Try initiating the conversation with a different topic.
- The application is supplying text data to a picture box or picture data to a text box. Try initiating the conversation with a different control.

291 Foreign application quit

The other application in a DDE conversation quit unexpectedly. A DDE operation can't take place unless the other application is running.

292 DDE conversation closed or changed

The other application has closed or changed the DDE conversation unexpectedly. Terminate this conversation and attempt to establish a new conversation with the application.

293 DDE method invoked with no channel open

A DDE method (LinkExecute, LinkPoke, LinkRequest, or LinkSend) was performed on a control that is not involved in a valid DDE conversation.

Possible causes:

- Changing the LinkTopic property terminates an existing DDE conversation but does not automatically establish a new conversation. After changing the LinkTopic property for a control, you must set the LinkMode property to 1 (Hot) or 2 (Cold) before executing a DDE method on this control.
- You executed a DDE method on a control with LinkMode set to 0 (None). Set the LinkMode to 1 (Hot) or 2 (Cold) and try again.

294 Invalid DDE Link format

The other application in a DDE conversation passed data in CF_LINK format but it is not valid link data.

295 Message queue filled; DDE message lost

Visual Basic cannot keep up with the number of DDE operations attempted.

Possible causes:

- Too many DDE conversations. Try terminating some DDE conversations.
- Too much code in event procedures is executing because of incoming DDE data. Reduce the amount of code being called as a result of DDE changes, or try calling the DoEvents function.

296 PasteLink already performed on this control

You have already performed a Paste Link on this control. To paste a new link, first set the LinkMode property of this control to 0 (None), then use the Paste Link command.

297 Can't set LinkMode; invalid LinkTopic

You've tried to set the LinkMode property but can't because no valid LinkTopic property has been specified.

320 Can't use character device names in filenames: 'item'

From within Visual Basic, you cannot give a file the same name as a character device driver, such as AUX, CON, COM1, COM2, LPT1, LPT2, LPT3, LPT4, or NUL.

321 Invalid file format

The form file is damaged. Try replacing it with an undamaged copy.

340 Control array element 'item' does not exist

You used an index value that does not correspond to an existing element in this control array. Adjust the value or Load a control into the array with an index equal to the value.

341 Invalid object array index

The index for an object array element cannot be greater than 32,767 or less than 0.

342 Not enough room to allocate control array 'item'

There isn't enough memory to create all the elements of a control array. If a control array has discontiguous indexes, such as 0, 2, and 4, Visual Basic will use more memory than if the indexes were contiguous (0, 1, 2). Check either how you've assigned indexes at

design time or how your program assigns indexes as it creates new control array elements, and make them contiguous.

343 Object not an array

A control that is not part of any array was referred to as if it had an index, for example, Command1(3).Caption and Command1.Text. You can refer to an object as an array element only if it is defined to be part of an control array. See the online Help topic titled "Creating a Control Array."

344 Must specify index for object array

The control referred to is part of a control array. Refer to it using CtlName(index). If you created the control at design time, you can determine the index of the control by selecting it and viewing its Index property on the Properties bar.

345 Reached limit: cannot create any more controls on this form

No more than 255 controls are allowed on each form. The total of all menu items and controls on your form would exceed 255 if any more were added.

360 Object already loaded

The control in the control array has already been loaded. If it was loaded during run time, it can be removed with the Unload statement.

361 Can't load or unload this object

Possible causes:

- You've attempted to Load or Unload a system object - Screen, Printer, or Clipboard.
- You've attempted to Load or Unload a control that is not an element of an existing control array. For example, Load CtlArray will produce this error, while Load CtlArray(3) will not. If a control is already loaded, you can make it visible with the Visible property.

362 Can't unload controls created at design time

Only control array elements loaded at run time can be unloaded. If a control is created at design time, it cannot later be unloaded, even if it is part of a control array. However, you can hide any control by setting the Visible property to True.

363 Custom control 'item' not found

The form being loaded contains a custom control that is not part of the current project. Use the Add File command on the File menu to add the custom control to the project.

364 Object was unloaded

A form was unloaded from its own Form_Load procedure. The form that was unloaded may have been implicitly loaded. For example, the following will implicitly load Form2:

```
Form2.BackColor = Form1.BackColor
```

365 Unable to unload within this context

In some situations you are not allowed to unload a form or a control on a form. Some examples of when this error occurs include trying to unload a form or control on the form:

- During any Paint event for the form or any control on the form.
- Whenever a Combo Box is pulled down on the form being unloaded or containing the control being unloaded.

380 Invalid property value

An inappropriate value is assigned to a property. To find out find out what values are valid for a property, see Remarks in the property's Help topic.

381 Invalid property array index

You tried to use an inappropriate property array index value. List Property and Fonts Property index values must be greater than 0 and less than 32,767. For example, List1.List(3) is valid.

382 'item' property can't be set at run time

The following properties can't be set at run time:

ActiveControl
ActiveForm
BorderStyle (for form and text box only)
ControlBox
CtlName
FontCount
Fonts
FormName
hDC
hWnd
Image
Index
List
ListCount
MaxButton
MinButton
MultiLine
Parent
ScrollBars
Sorted
Style
Text (for list box and combo box only)
Width (for Screen only)

383 'item' property is read-only

The following properties are read-only at both design and run time:

Property	Object
hDC	Form, picture box, Printer
Height	Combo box (with Style =1), drive, Printer
hWnd	Form
Image	Form, picture box
List	Dir, drive, file
ListCount	Combo box, dir, drive, file, list box
Page	Printer
Parent	Any control
Text	Combo box (with Style = 2), list box
Width	Printer

384 'item' property can't be modified when form is minimized or maximized

The Left, Top, and Height, Width properties cannot be changed on a minimized or maximized form. You can set or return the state of a form with the WindowState property. You can prevent a user from maximizing or minimizing a form by setting the MaxButton or MinButton form properties to False (0).

385 Must specify index when using property array

You must specify an index when using the List property or the Fonts property. Index values must be greater than 0 and less than 32,767. For example, List1.List is invalid because no index is specified. However, List1.List(3) is valid.

386 'item' property not available at run time

The CtlName and FormName properties are not available at runtime.

387 'item' property can't be set on this control

Possible causes:

- The Checked box for a Menu control can't be selected when that control is a parent or top-level menu.
- The separator bar on a menu control can't be set when the control is a parent or top-level menu item.
- The Visible property can't be set to False (0) for the last visible submenu on a parent menu. You can't have a parent menu with no visible submenu items.

388 Can't set Visible property from a parent menu

The Visible property of a submenu item cannot be set from its parent's menu code.

400 Form already displayed; can't show modally

You cannot use the Show method to display a modal form if the form is already visible. Either Unload or Hide the form before

attempting to show it as a modal form. Don't try to display it as a modal form.

401 Can't show non-modal form when a modal form is displayed

You cannot show a non-modal form when a modal form is displayed. Unload or Hide the modal form before attempting to use the Show method on another form.

402 Must close or hide topmost modal form first

The modal form you are trying to close or hide is not on top; you need to Unload or Hide all modal forms that are on top of this one before you can continue. A modal form is a form displayed by the Show method with the style% argument equal to 1.

420 Invalid object reference

The object that is referred to is not loaded.

421 Method not applicable for this object

Object.Method is referred to, but the Method is not appropriate for the object. For example, Command1.AddItem produces this error because the AddItem method is used only with a list box or combo box.

422 Property 'item' not found

Control.property or Control(index).property is referred to, but property is not defined for this control or you may have misspelled the name of the property. To see what properties are defined for this control, see Properties, Events and Methods in Help for complete information on a specific control.

423 Property or control 'item' not found

Form.control or Form.property is referred to, but control or property is not defined for this form or you may have misspelled the name of the control or property. To see what properties are defined for a form, see the Form topic in Help for complete information. To see what controls are on this form, look at the list in the Code window's Object box.

424 Object required

Property.property is referred to; you need to specify object.property, where object is either a form or control.

425 Invalid object use

You've attempted an invalid assignment using a form or control. If you want to assign a value to a property or a property value to a variable, remember to include the property name in the object specification. For example Form1.Command1.Caption = "OK" instead of Form1.Command1 = "OK".

430 No currently active control

Because no control has the focus, the reference to ActiveControl has no effect. You can use the SetFocus method to set the focus to a specified object.

431 No currently active form

Because no form has the focus, the reference to ActiveForm has no effect. You can use the SetFocus method to set the focus to a specified object.

460 Invalid Clipboard format

The specified Clipboard format is incompatible with the method being executed. GetText and SetText can be used only with CF_TEXT or CF_LINK formats. GetData and SetData can be used only with CF_BITMAP, CF_METAFILEPICT (Windows metafile picture), or CF_DIB (device independent bitmap) formats.

Note: The Clipboard formats, such as CF_TEXT, and CF_BITMAP, are global constants that must be set to numerical values found in CONSTANT.TXT. If you do not initialize these in your code, Visual Basic treats them as variables and initializes them to zero.

See Also
GetFormat

461 Specified format does not match format of data

In a GetData or SetData method, the Clipboard format you specified does not match the actual data. For example, you might have specified CF_BITMAP for the format, but the data is in the Windows metafile format (CF_METAFILEPICT). Make sure you specify the correct format for the data.

Note: The Clipboard formats, such as CF_TEXT, and CF_BITMAP, are global constants that must be set to numerical values found in CONSTANT.TXT. If you do not initialize these in your code, Visual Basic treats them as variables and initializes them to zero.

See also
GetFormat

480 Can't create AutoRedraw image

There isn't enough available memory to create a persistent bitmap for automatic redraw of the form or picture. Make the picture box control or form smaller, or reset the AutoRedraw property and perform your own redraw in the Paint event procedure.

481 Invalid picture

You attempted to assign something to the Picture property of a form or picture box that Visual Basic doesn't recognize as an icon, bitmap, or Windows metafile.

482 Printer error

There is some problem that prevents printing. Some possible causes are:

- You don't have a printer selected from the Windows Control Panel.
- Your printer is not online.
- Your printer is jammed or out of paper.
- You tried to print a form to a printer than can accept only text.

520 Can't empty Clipboard

Another application is using the Clipboard and will not release it to your application.

521 Can't open Clipboard

Another application is using the Clipboard and will not release it to your application.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: PrgOptTips

Differences Between QuickBasic and Visual Basic Statements

Article ID: Q87004

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The following is a list of statements and functions not shared between Microsoft QuickBasic version 4.5 for MS-DOS, Microsoft Visual Basic programming system version 1.0 for Windows, and the Standard and Professional Editions of Microsoft Visual Basic version 2.0 for Windows.

MORE INFORMATION

The following table compares reserved words in QuickBasic for MS-DOS to those in Visual Basic for Windows.

Note that in many cases a statement or function is not supported in one language or the other due to a different approach to achieve the same result. A good example of this is the LPRINT statement in QuickBasic. Visual Basic has a Printer object that handles the output to the system printer.

In some cases, a statement or function is not supported because it is a low level MS-DOS operation that conflicts with Windows, or it is a function that is only supported in Windows.

Keyword	QuickBasic 4.5 for MS-DOS	Visual Basic 1.0 for Windows	Visual Basic 2.0 for Windows
BLOAD	Yes	No	No
BSAVE	Yes	No	No
CALL ABSOLUTE	Yes	No	No
CALLS	Yes	No	No
CCur	No	Yes	Yes
CDbl	No	Yes	Yes
CHAIN	Yes	No	No
ChDrive	No	Yes	Yes
CInt	No	Yes	Yes
CLEAR	Yes	No	No
CLng	No	Yes	Yes
COLOR	Yes	No	No
COM	Yes	No	No
CSng	No	Yes	Yes
CSRLIN	Yes	No	No
CVD	Yes	No	No
CVDMBF	Yes	No	No

CVI	Yes	No	No
CVS	Yes	No	No
CVSMBF	Yes	No	No
DATA	Yes	No	Yes
DateSerial	No	Yes	Yes
DateValue	No	Yes	Yes
Day	No	Yes	Yes
DEF FN	Yes	No	No
DEF SEG	Yes	No	No
Dir$	No	Yes	Yes
DoEvents	No	Yes	Yes
DRAW	Yes	No	No
ERDEV	Yes	No	No
ERDEV$	Yes	No	No
Error$	No	Yes	Yes
FIELD	Yes	No	No
FILES	Yes	No	No
FRE	Yes	No	No
Global	No	Yes	Yes
Hour	No	Yes	Yes
INKEY$	Yes	No	No
INP	Yes	No	No
InputBox$	No	Yes	Yes
IOCTL	Yes	No	No
IOCTL$	Yes	No	No
KEY	Yes	No	No
Load	No	Yes	Yes
LoadPicture	No	Yes	Yes
LOCATE	Yes	No	No
LPOS	Yes	No	No
LPRINT	Yes	No	No
Minute	No	Yes	Yes
MKD$	Yes	No	No
MKI$	Yes	No	No
MKL$	Yes	No	No
MKS$	Yes	No	No
MKSMBF$	Yes	No	No
Month	No	Yes	Yes
MsgBox	No	Yes	Yes
Now	No	Yes	Yes
ON COM	Yes	No	No
ON KEY	Yes	No	No
ON PLAY	Yes	No	No
ON STRIG	Yes	No	No
ON TIMER	Yes	No	No
OUT	Yes	No	No

PAINT	Yes	No	No
PALETTE	Yes	No	No
PCOPY	Yes	No	No
PEEK	Yes	No	No
PEN	Yes	No	No
PLAY	Yes	No	No
PMAP	Yes	No	No
POKE	Yes	No	No
POS	Yes	No	No
PRESET	Yes	No	No
QBColor	No	Yes	Yes
RESTORE	Yes	No	No
RGB	No	Yes	Yes
RUN	Yes	No	No
SADD	Yes	No	No
SavePicture	No	Yes	Yes
SCREEN	Yes	No	No
Second	No	Yes	Yes
SendKeys	No	Yes	Yes
SETMEM	Yes	No	No
SLEEP	Yes	No	No
SOUND	Yes	No	No
STICK	Yes	No	No
STRIG	Yes	No	No
SWAP	Yes	No	No
TimeSerial	No	Yes	Yes
TimeValue	No	Yes	Yes
TROFF	Yes	No	No
TRON	Yes	No	No
Unload	No	Yes	Yes
USING$	Yes	No	No
VARPTR	Yes	No	No
VARPTR$	Yes	No	No
VARSEG	Yes	No	No
VIEW	Yes	No	No
WAIT	Yes	No	No
WeekDay	No	Yes	Yes
WIDTH	Yes	No	No
WINDOW	Yes	No	No
Year	No	Yes	Yes

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgOptMemMgt

PRB: For Loop w/ Integer Counter & Increment <=.5 Causes Hang

Article ID: Q87769

This information applies to the following Microsoft Basic products:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft QuickBasic for MS-DOS, versions 4.0, 4.0b, and 4.5
- Microsoft Basic Professional Development System (PDS) for MS-DOS, versions 7.0 and 7.1

SYMPTOMS

If you write a FOR loop with an INTEGER or LONG variable as the FOR loop counter and use a floating point value less than or equal to 0.5 as the FOR loop increment, the loop never terminates. This causes the computer to hang (stop responding to input).

CAUSE

All Basic programs convert floating point values less than 0.5 to the integer value 0.

RESOLUTION

To stop a program that is executing in this type of an endless loop, press CTRL+BREAK.

STATUS

This behavior is by design. In other words, this is not a problem with the FOR statement; this is the way Basic is designed to operate.

MORE INFORMATION

Steps to Reproduce Behavior in Visual Basic for Windows

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add the following code to the Form_Click event procedure for Form1:

```
Sub Form_Click ()
   For j& = .005 To .0062 Step .0001
      total! = total! + j&
   Next j&
   Print total!
```

```
   End Sub
```

3. Press F5 to run the example.

No value appears on the form. The program is in an endless loop. You cannot access any menus. Press CTRL+BREAK to stop the program.

To change this example program so that the loop terminates, change the type of the counter variable from LONG to SINGLE (change j& to j!).

Additional reference words: 1.00 2.00 4.00 4.00b 4.50 7.00 7.10 b_quickbas b_basiccom
KBCategory:
KBSubcategory: PrgOptMemMgt

How to Emulate MKI$ and CVI in VB Using Windows HMemCpy

Article ID: Q87970

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Visual Basic for Windows does not support the MKx$ and CVx family of conversion functions found in earlier versions of Microsoft QuickBasic and Basic Professional Development System (PDS) for MS-DOS. However, you can write functions that provide this support using the hmemcpy API routine provided by Windows version 3.1.

This article provides example routines that simulate the MKI$, MKL$, MKS$, MKD$, CVI, CVL, CVS, and CVD functions.

MORE INFORMATION

The MKx$ functions convert numeric values to strings by placing the ASCII value of each byte that represents the numeric value into a string.

Function	Description
MKI$	Converts an integer to a 2-byte string
MKL$	Converts a long-integer to a 4-byte string
MKS$	Converts a single precision variable to a 4-byte string
MKD$	Converts a double-precision variable to an 8-byte string

The CVx functions convert strings created with the MKx$ functions back into numeric values.

Function	Description
CVI	Converts a 2-byte string created with MKI$ to an integer
CVL	Converts a 4-byte string created with MKL$ to a long integer
CVS	Converts a 4-byte string created with MKS$ to a single-precision number
CVD	Converts an 8-byte string created with MKD$ to a double-precision number

The hmemcpy API function can be used to emulate these functions as demonstrated in the example below. Note that the hmemcpy API function is not provided with Windows version 3.0, so the example below requires Windows version 3.1.

The hmemcpy routine copies bytes from a source buffer to a destination buffer. You can use this routine to copy the value of each byte in a numeric value to a corresponding byte in a string to emulate the MKx$

functions. Similarly, you can use the same technique to copy the bytes from a string to a numeric value, to emulate the CVx functions.

Note that the hmemcpy routine requires the addresses pointing to the actual location of the data to be copied from and written to. Therefore, it is necessary to pass strings by value (ByVal) in order to pass the location of the string data, as opposed to passing the location of the string descriptor. Similarly, it is necessary to initialize the string size by assigning the string to an appropriate number of characters.

To use the following routines in your Visual Basic for Windows application, you must Declare the hmemcpy routine. Add the following code to the general declarations section of the form:

```
' Enter the following Declare statement on one, single line.
Declare Sub hmemcpy Lib "kernel" (hpvDest As Any, hpvSource As Any,
   ByVal cbCopy As Long)

Function MKI$ (x As Integer)
   temp$ = Space$(2)
   hmemcpy ByVal temp$, x%, 2
   MKI$ = temp$
End Function

Function CVI (x As String) As Integer
   If Len(x) <> 2 Then
      MsgBox "Illegal Function Call"
      Stop
   End If
   hmemcpy temp%, ByVal x, 2
   CVI = temp%
End Function

Function MKL$ (x As Long)
   temp$ = Space$(4)
   hmemcpy ByVal temp$, x&, 4
   MKL$ = temp$
End Function

Function CVL (x As String) As Long
   If Len(x) <> 4 Then
      MsgBox "Illegal Function Call"
      Stop
   End If
   hmemcpy temp&, ByVal x, 4
   CVL = temp&
End Function

Function MKS$ (x As Single)
   temp$ = Space$(4)
   hmemcpy ByVal temp$, x!, 4
   MKS$ = temp$
End Function

Function CVS (x As String) As Single
   If Len(x) <> 4 Then
      MsgBox "Illegal Function Call"
```

```
         Stop
      End If
      hmemcpy temp!, ByVal x, 4
      CVS = temp!
   End Function

   Function MKD$ (x As Double)
      temp$ = Space$(8)
      hmemcpy ByVal temp$, x, 8
      MKD$ = temp$
   End Function

   Function CVD (x As String) As Double
      If Len(x) <> 8 Then
         MsgBox "Illegal Function Call"
         Stop
      End If
      hmemcpy temp#, ByVal x, 8
      CVD = temp#
   End Function
```

Reference(s):

"Microsoft Windows SDK: Programmer's Reference," Volume 2: Functions," version 3.1

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgOptMemMgt

Diagnosing General Protection Fault / UAE in VB for Windows

Article ID: Q90871

This information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Windows, versions 3.0 and 3.1

SUMMARY

This article describes steps you can take to determine the cause of and possibly fix a General Protection Fault (GPF) under Windows version 3.1, or an Unrecoverable Application Error (UAE) under Windows version 3.0. The problems listed below can cause a GPF/UAE.

- Calling a dynamic-link library (DLL) or Windows API routine with incorrect parameters
- Using a faulty DLL routine or custom control
- Loading corrupted Visual Basic forms or modules

MORE INFORMATION

To determine if a GPF/UAE is caused by a call to a Windows API routine, a DLL routine, or by a custom control, temporarily remove references to the DLL routine or custom control and re-run the program to see if the GPF/UAE still occurs. You may need to replace such references with statements that simulate return values.

To determine if a GPF/UAE is caused by corrupted code, save the code in your forms and modules as text, then load the code as text. This process cleans the internal representation of the code (P-code).

Steps to Clean Code

1. In the Project window, select the form or module to clean.
2. From the Code menu, choose Save Text... and select OK.
3. From the Code menu, choose View Code.
4. From the Code menu, choose Load Text..., select the same file, and click Replace.

To determine if a GPF/UAE is caused by a corrupted form, recreate the form. To recreate a form, add a new form to your project and create new controls and menus to match the old form. Copy the code by saving code as text from the old form and loading as text into the new form.

Finally, remove the old form, and rename the new form.

Steps to Recreate Form
----

1. From the File menu, choose Add Form. Create the same controls and menus on this new form as are on the old form.

2. In the Project window, select the old form or module.

3. From the Code menu, choose Save Text... and select OK.

4. From the File menu, choose Remove File.

5. In the Project window, select the new form.

6. From the Code menu, choose View Code.

7. From the Code menu, choose Load Text..., select the same code text file, and click Replace.

8. Set the new form CtlName property to the old form's value.

You can also clean a project file (.MAK) by starting a new project, then adding all the forms and modules to the new project.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgOptMemMgt

Visual Basic 3.0 General Information Questions & Answers

Article ID: Q92545

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, version 3.0

---

1. Q. What are the new features in Microsoft Visual Basic version 3.0 for Windows?

   A. The following is a list of some of the main new features:

   - Microsoft Access version 1.1 database engine provides direct connectivity to Access, FoxPro, dBASE, Paradox, and Btrieve databases.

   - Two new controls:

     - Data control provides a visual and semi-automated method to connect to databases.
     - Outline control provides an easy way to create hierarchical list boxes.

   - Full ODBC support for SQL, Sybase, and Oracle.

   - Three new tools:

     - Crystal Reports report generator.
     - Data Manager for easily generating a database file.
     - Setup Wizard for automating the creation of setup and distribution disks.

   - Pop-up menus.

   - OLE 2.0 Automation.

   For additional information on these and other product features, please call Microsoft Visual Basic startup and installation support at (206)646-5105.

2. Q. What are the system requirements for Microsoft Visual Basic version 3.0 for Windows?

   A. To use Microsoft Visual Basic version 3.0 for Windows, you need:

   1. Microsoft Windows operating system version 3.0 or higher running in standard or 386 enhanced mode.

   2. An IBM PC or compatible computer, or an IBM PS/2 with an 80286 or better microprocessor.

   3. 2 Megabytes (MB) of available memory (4 MB or higher recommended) for the design environment

4. Hard Drive with 32 MB available.

5. A 5.25-inch or 3.5-inch high-density disk drive.

6. A Microsoft mouse or compatible pointing device.

7. EGA or higher resolution monitor.

3. Q. Does Microsoft Visual Basic version 3.0 for Windows work with the new Microsoft Windows NT operating system?

   A. Yes. However, Visual Basic version 3.0 for Windows will not take advantage of the 32 bit features of Microsoft Windows NT. Visual Basic runs in the 16 bit emulation layer in Windows NT.

4. Q. Where can I get information on available 3rd-party custom controls or 3rd-party books for use with Microsoft Visual Basic?

   A. Included with Microsoft Visual Basic version 3.0 for Windows is a catalog called "Custom Controls and Other Companion Products and Services for Visual Basic for Windows." In addition, you can find this information in an article in the Microsoft Knowledge Base titled "List of Visual Basic Companion Products and Services Available." The item identification number for this article is Q78962. You can have the article faxed to you by calling Microsoft FastTips (800)936-4300.

Additional reference words: 3.00 ivrfax fasttips
KBCategory: Prg
KBSubcategory: PrgOptTips

How to Break Long Statements into Multiple Lines

Article ID: Q94696

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- The Standard and Professional Editions of Microsoft Visual Basic for MS-DOS, version 1.0
- Microsoft Basic Professional Development System (PDS) for MS-DOS, version 7.1
- Microsoft QuickBASIC for MS-DOS, version 4.5

SUMMARY

This article describes how to break lengthy control-flow statements such as IF/THEN statements or WHILE loops into multiple shorter statements while retaining their functionality. There is no line continuation character in Basic or Visual Basic. It is useful to break up lines of code so they are easy to view in the edit window without scrolling and are within the compiler's (BC.EXE) line limit of 255 characters.

MORE INFORMATION

The following examples show how to use temporary variables to break up an IF/THEN statement and a WHILE loop into multiple shorter lines:

The IF/THEN statement is a control-flow statement that branches if a condition is true. A long IF/THEN statement such as:

```
MAX = 3
VALUE = 2
CURRENTVALUE = 1

IF ((VALUE > CURRENTVALUE) OR (VALUE < CURRENTVALUE)) AND (VALUE < MAX)
   THEN 'Combine with previous line -- Should all be on a single line
   PRINT "VALUE is not equal to CURRENTVALUE and less than MAX"
END IF
```

Can be broken down using temporary variables to:

```
MAX = 3
VALUE = 2
CURRENTVALUE = 1

TEMPVAL = (VALUE > CURRENTVALUE) OR (VALUE < CURRENTVALUE)
TEMPVAL = TEMPVAL AND (VALUE < MAX)
IF TEMPVAL THEN
   PRINT "VALUE is not equal to CURRENTVALUE and less than MAX"
END IF
```

These two code fragments are equivalent. They evaluate and execute the

PRINT statement under the same conditions.

The following demonstrates the same technique with a WHILE loop:

```
   MAX = 10
   VALUE = 5
   CURRENTVALUE = 1

   WHILE ((VALUE > CURRENTVALUE) OR (VALUE < CURRENTVALUE)) AND (
   VALUE < MAX) ' This should all be on one line
      MAX = MAX - 1
   WEND
   PRINT "Out of WHILE Loop"
```

This is the revised version using temporary values:

```
   MAX = 10
   VALUE = 5
   CURRENTVALUE = 1

   TEMPVAL = (VALUE > CURRENTVALUE) OR (VALUE < CURRENTVALUE)
   TEMPVAL = TEMPVAL AND (VALUE < MAX)
   WHILE TEMPVAL
      MAX = MAX - 1
      TEMPVAL = (VALUE > CURRENTVALUE) OR (VALUE < CURRENTVALUE)
      TEMPVAL = TEMPVAL AND (VALUE < MAX)
   WEND
   PRINT "Out of WHILE Loop"
```

In both code examples, the TEMPVAL variable contains a value of 0 or -1 to signify a logical TRUE or FALSE.

Additional reference words: 1.00 2.00 VBMSDOS QUICKBAS 4.50 BASICCOM 7.10
KBCategory:
KBSubcategory: PrgOther

Basic Products Can Create and Use Non-Standard File Names

Article ID: Q94783

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- The Standard and Professional Editions of Microsoft Visual Basic for MS-DOS, version 1.0
- Microsoft Basic Professional Development System (PDS) for MS-DOS, version 7.1
- Microsoft QuickBASIC for MS-DOS, version 4.5

SUMMARY

Microsoft Visual Basic and other Basic products can create and use non-standard MS-DOS file names. For example, a file name with an embedded space is a non-standard file name. However, Microsoft doesn't recommend that you use non-standard file names because they can cause problems with other software.

MORE INFORMATION

According to the MS-DOS documentation, file names must:

- Have no more than eight characters.

- Contain only the letters A through Z, the numbers 0 through 9, and the following special characters: underscore (_), caret (^), dollar sign ($), tilde (~), exclamation point (!), number sign (#), percent sign (%), ampersand (&), hyphen (-), braces ({}), parentheses (), at sign (@), apostrophe ('), and the accent grave (`). No other special characters are acceptable.

- Not contain spaces, commas, backslashes, or periods except for the period that separates the name from the extension.

- Not be the following reserved filenames: CLOCK$, CON, AUX, COMn (where n=1-4), LPTn (where n=1-3), NUL, and PRN.

The Basic OPEN command allows you to open a file name that breaks some of these rules. For example, you can open a file that has a space embedded in its name.

The following example creates a file giving it a name that contains a space. Then it writes data to the file, reopens it, and prints the data on the screen:

```
OPEN "A B" FOR OUTPUT AS #1 'There is a space between A and B
PRINT #1, "HELLO THERE"
CLOSE #1
```

```
  OPEN "A B" FOR INPUT AS #1
  INPUT #1, A$
  PRINT A$
  CLOSE #1
```

Additional reference words: 1.00 2.00 3.00 4.50 7.10
KBCategory:
KBSubcategory: PrgOther

Obtaining Date or Serial Result from DateSerial or DateValue

Article ID: Q95510

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

In Visual Basic version 2.0 and 3.0, the DateSerial and DateValue functions return a variant data type of VarType 7 (Date) instead of a date serial number. The date is still stored internally in the serial number format returned by the DateSerial and DateValue functions in Visual Basic version 1.0 and can be obtained by using the CDbl function in versions 2.0 and 3.0.

This is not a bug. This is a special feature of versions 2.0 and 3.0.

MORE INFORMATION

The return value for DateSerial and DateValue is a formatted date string in the format MM/DD/YY. If you pass it as an argument to the Print Method, Print # and Write # statements are printed as such. The line of code below prints the date as a formatted date string:

```
Form1.Print DateSerial(1992,1,1)
```

To use the underlying serial number, use the Visual Basic CDbl statement to convert the variant return value to a double precision number. This can be useful for storing dates in a random access file because a double precision variable uses eight 8 bytes and a variant uses 16. The line of code below prints the date in serial number format:

```
Form1.Print CDbl(DateSerial(1992,1,1))
```

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgOptTips

FileDatetime Doesn't Include Time If File Time Is Midnight

Article ID: Q96098

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Passing a file name with a time stamp of midnight to the function FileDateTime, returns a string containing only the date, not the time. This is consistent with the Format/Format$ function's General Date format, which when passed a DateTime string with a time of midnight returns a string containing only the date.

If your program needs to display a DateTime string with midnight represented by 12:00 AM or 00:00 (in 24-hour format), use the Format($) functions to perform the necessary conversion. By using Format$ or Format with the time format symbols h, m, and s, you can cause the Format($) functions to include a time format for midnight.

In the example below, a message box showing both the date and time of VB.EXE, which is midnight for version 2.0, is displayed with a time stamp.

```
MsgBox Format$(FileDateTime("VB.EXE"), "mm/dd/yy hh:mm AMPM")
```

MORE INFORMATION

The internal structure of a serial number is a double precision number. The integral portion represents the number of days since December 30, 1899 and the fractional portion represents the time as a fraction of a day. Midnight is the beginning of a day and therefore it's represented by the fraction zero.

For example the serial number for 10/21/92 6:00 AM is represented by:

```
33898.25
```

The date is 33898 days since 12/30/1899. The time is represented as one-fourth of the 24-hour day passed since midnight. One-fourth of 24 is exactly 6, so the time is 6 hours, 0 minutes, 0 seconds.

Using the General Date format for a DateTime string without using a time in Format($), automatically returns a formatted string without a time portion. This is by design. Because both midnight and DateTime strings without a time are represented internally as the same number, the General Date format processes the strings identically.

Steps to Reproduce Problem

1. Run Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add the following code to the Form_Click procedure for Form1:

```
Sub Form_Click
   MsgBox FileDateTime("VB.EXE")
End Sub
```

A message box appears with the date 10/21/92, but the time stamp is not displayed.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgOther

PRB: File Not Found Error When Running .EXE on Other Computer

Article ID: Q101580

The information in this article applies to:

- Professional and Standard Editions of Microsoft Visual Basic Programming System for Windows, version 3.0

SYMPTOMS

When you try to use the IIf function in Visual Basic version 3.0 for Windows, you receive a "File Not Found" error when you try to run your executable program on a separate computer that does not contain the file MSAFINX.DLL.

CAUSE

The "File Not Found" error occurs because the IIf function is not included in VBRUN300.DLL file but is located in the MSAFINX.DLL file.

RESOLUTION

To prevent the error, install the MSAFINX.DLL file on the customer's computer in the \WINDOWS\SYSTEM subdirectory.

MORE INFORMATION

Here is a list of all the financial functions in the MSAFINX.DLL file:

DATEPART	DATEDIFF	DATEADD	DDB	FV
IIF	IPMT	IRR	MIRR	NPER
NPV	PARTITION	PMT	PPMT	PV
RATE	SLN	SYD		

A "File Not Found" error occurs if you use any of the these functions in your program and then use the program on a computer that does not contain the MSAFINX.DLL file.

MORE INFORMATION

Steps to Reproduce Behavior

The following steps cause the "File Not Found" error in Visual Basic version 3.00 for Windows.

1. Start Visual Basic (VB.EXE).

2. Add a text box (Text1) and a label (Label1) to Form1.

3. Enter the line of code on page 274 in the Language Reference manual:

```
Sub Label1_Click ( )
   Label1.Caption = IIf(Val(text1.text) > 1000, "Large", "Small")
   '*** note you may want to add the Val statement for numbers
End Sub
```

4. Run the example and enter a number in the Text1 text box. Then click Label1 to see if the example works in the environment.

5. From the File menu, choose Make Exe File... Name the executable IIFTEST.EXE. Save the project as IIFTEST.MAK, and save the form as IIFTEST.FRM.

6. Copy the IIFTEST.EXE and VBRUN300.DLL files to a floppy disk.

7. Take the floppy disk to a computer that does not have Visual Basic version 3.0 installed. Try and run the IIFTEST.EXE file from File Manager on that computer. You should get the "File Not Found" error.

8. If you add the file MSAFINX.DLL to the floppy disk, and then run the IIFTEST.EXE file, no error will occur.

Additional reference words: 3.00
KBCategory:
KBSubcategory: PrgOther

Sum Of VB Strings Can Exceed 64K in Certain Circumstances

Article ID: Q104554

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

In all cases, individual variable length strings have a maximum size of 64K worth of characters. However, the sum of the lengths of multiple strings can exceed 64K in the circumstances described in this article.

MORE INFORMATION

Visual Basic for Windows goes beyond any previous Microsoft Basic product in its flexibility when dealing with string variables. As documented in the "Microsoft Visual Basic Programmer's Guide," Appendix D: Individual strings always have a maximum size of 64K characters. However, this is not an absolute limit when dealing with multiple strings. The sum of the lengths of multiple strings can exceed 64K in the circumstances described below:

1. Global strings declared at the module level and assigned values elsewhere can each have a value of up to 64K and their total can exceed that. For example, if you have the following module-level declarations:

   ```
   Global a as string
   Global b as string
   Global c as string
   ```

   you could have the following code in a Sub procedure:

   ```
   a = Space(64000)
   b = Space(64000)
   c = Space(64000)
   ```

2. The sum of all module level variable length strings can exceed 64K. For example, if you have the following module-level declarations:

   ```
   Dim a as string
   Dim b as string
   Dim c as string
   ```

   you could have the following code in a Sub procedure in the same module:

   ```
   a = Space(64000)
   b = Space(64000)
   c = Space(64000)
   ```

3. The sum of all local variable-length string variables can exceed 64K, but only across different Sub procedures. The limit within a single Sub procedure is 64K for all local variable-length strings. For example, the following code would work correctly:

```
Sub MySub1()
   Dim a As String
   Dim b As String
   a = Space(32000)
   b = Space(32000)
End Sub

Sub MySub2()
   Dim a As String
   Dim b As String
   a = Space(32000)
   b = Space(32000)
End Sub
```

   This is true even when more than one of the Sub procedures are currently active such as when MySub1 is called and it calls MySub2. Both are in memory and each has a 64K segment available for local variable-length strings.

   The following code would not work. It would respond correctly with an "Out of String Space" error message because it tries to exceed 64K of local variable-length strings.

```
Sub MySub3()
   Dim a As String
   Dim b As String
   Dim c As String
   a = Space(32000)
   b = Space(32000)
   c = Space(32000)
End Sub
```

4. The variable-length string elements of a user defined type are individually limited to 64K each, but their sum may exceed 64K. For example, if you have the following module-level declarations:

```
Type Test
   a As String
   b As String
   c As String
End Type
Dim x as Test
```

   you can have the following code in a Sub procedure:

```
x.a = Space(64000)
x.b = Space(64000)
x.c = Space(64000)
```

5. Assigning more than 64K to an array of variable-length strings causes an "Out of String Space" error.

For example, if you have the following module-level declaration:

```
Dim MyArray(12) as String
```

The following code in a Sub procedure would cause an error:

```
MyArray(1) = Space(64000)
MyArray(2) = Space(64000)
```

To solve the problem, dimension the array as type Variant:

```
Dim MyArray(12) as Variant
```

Then the following Sub procedure code will correctly create two 64K variants tagged as strings.

```
MyArray(1) = Space(64000)
MyArray(2) = Space(64000)
```

Reference(s):

"Microsoft Visual Basic for Windows Programmer's Guide," version 3.0, Appendix D, pages 644-647.

Additional reference words: 2.00 3.00
KBCategory: Prg
KBSubCategory: PrgOther

How to Retrieve Hidden/System Files Using Dir[$]() Function

Article ID: Q104685

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 2.0 and 3.0

SUMMARY

This article shows by example how to use the Dir[$]() function in conjunction with the GetAttr() function to retrieve read-only, hidden, or system files.

MORE INFORMATION

The Dir[$] functions take a filespec and an attrmask as optional arguments.

If the attrmask argument specifies the volume label, the Dir[$] functions ignores all other attributes. If the attrmask argument is ATTR_HIDDEN, ATTR_SYSTEM, or ATTR_DIRECTORY, the functions also return the files that do not have any special attributes.

If the filespec argument is used, the functions return files that do not have any hidden, system, or directory attributes and meet the filespec requirements.

To retrieve only read-only, hidden, or system files, use the Dir[$]() functions in conjunction with the GetAttr() function. The following shows by example how to retrieve only hidden files (files that have the HIDDEN or ATTR_HIDDEN+ATTR_ARCHIVE attributes) by using the Dir() function in conjunction with the GetAttr() function.

Step-by-Step Example

1. Start Visual Basic or begin a new project if Visual Basic is already running. Form1 is created by default.

2. Place the following code in the general declarations area for Form1:

```
Const ATTR_NORMAL = 0
Const ATTR_READONLY = 1
Const ATTR_HIDDEN = 2
Const ATTR_SYSTEM = 4
Const ATTR_VOLUME = 8
Const ATTR_DIRECTORY = 16
Const ATTR_ARCHIVE = 32
```

3. Add a List box and a command button to Form1.

4. Add the following code to the command button's click event procedure:

```
Sub Command1_Click ()
   Dim filename As String
   Dim attr As Integer
   ' retrieve hidden and normal files
   filename = Dir$("c:\", ATTR_HIDDEN)
   Do Until filename = ""
      attr = GetAttr("c:\" & filename)
      ' if the file has the hidden attribute
      If (attr And ATTR_HIDDEN) Then
         ' select it
         List1.AddItem filename
      End If
      filename = Dir$
   Loop

End Sub
```

5. Run the program and click the command button to see any existing hidden files in the root directory.

Additional reference words: 2.00 3.00 Dir Dir$ GetAttr
KBCategory: Prg
KBSubcategory: PrgOther

PRB: Can't Set Formal Parameter When Setting Object Vars

Article ID: Q105230

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SYMPTOMS

Trying to use a Set statement on an object variable that is a formal parameter of a procedure results in this error:

```
Can't Set Formal Parameter.
```

CAUSE

Object variables can be parameters of a Sub or Function procedure, but if an object variable is a parameter, its value cannot be changed inside the called procedure.

RESOLUTION

If you make the object variable Global instead of passing it as a parameter,
you can use Set statements inside procedures.

MORE INFORMATION

Objects as parameters can be thought of as a copy of the structure that defines the object. If Set statements were allowed on these objects, this would change the value inside the routine, but upon returning from the routine the changes would be lost and the object variable would revert back to its original value.

Steps to Reproduce Behavior

1. Start a new project in Visual Basic and add the following procedure to the application:

   ```
   Sub s (tb As Table)
      Set tb = Nothing
   End Sub
   ```

2. Press the F5 key to run the application. The error "Can't Set Formal Parameter" should occur immediately.

Trying to force these objects to be passed by value by setting the ByVal keyword results in this error:

Expected: Integer or Long or Single or Double or Currency or
String or Variant.

ByVal is allowed with the variable types listed in the error message,
but it is not allowed with any other variable type.

Additional reference words: 3.00
KBCategory:
KBSubCategory: PrgOther

Expected Expression Error: Dynamic Array Not OK in User-Type

Article ID: Q108709

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SYMPTOMS

An "Expected: expression" error occurs within a Type statement in Visual Basic when you follow an array name with empty parentheses ().

You can use the Type ... End Type statement block to define your own data type using Basic's predefined data types as components. You can dimension variables or arrays using that user-defined type.

CAUSE

Visual Basic interprets the empty parentheses () that follow a variable name in the Type statement as a declaration of a dynamic array. Visual Basic does not allow dynamic arrays to be declared within a user-defined Type statement block.

RESOLUTION

Within the Type ... End Type statement block, parentheses that follow a variable name must contain a number to indicate the number of elements in a static array. Visual Basic allows Type statements to contain static arrays but not dynamic arrays.

NOTE: Once you correctly define a user-defined type, you can dimension a dynamic array of that type. See further below for an example.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Reproduce Behavior

1. Start a new project in Visual Basic. Form1 is created by default.

2. From the File menu, choose New Module.

3. Add the following code to the new module, MODULE1.BAS:

```
Type newinfo
   tarray() as string  'Gives "Expected: Expression" error on ()
   numstores As Long
End Type
```

4. Edit the line containing tarray(). Then select any other line. The automatic syntax checker in Visual Basic correctly highlights the () and gives the following error:

   Expected: Expression

   Running the program by pressing the F5 key also correctly reports this syntax error.

5. To correct this programming error, add a number of array elements in the empty parentheses. For example, change tarray() to tarray(10). This changes the array from dynamic to static.

Visual Basic interprets the empty parentheses in tarray() in the Type statement as a declaration of a dynamic array. Visual Basic does not allow dynamic arrays to be declared within a user-defined Type ... End Type statement block. The parentheses () must contain a number to indicate the number of elements in a static array.

How to Make a Dynamic Array of User-Defined Type

1. Start a new project in Visual Basic. Form1 is created by default.

2. From the File menu, choose New Module.

3. Add the following code to the new module, MODULE1.BAS:

```
Type newinfo
   tarray(20) As String  'Static array declared in user-defined type
   numstores As Long
End Type
```

4. Double-click Form1 to display the form's code window. Add the following to the form load event:

```
Sub Form_Load ()
   ' Use ReDim to declare or redimension a dynamic array:
   ReDim arrayx(20) As newinfo  'Make dynamic array of user-defined type
   arrayx(18).tarray(12) = "Ruby slippers"       ' Assign value.
   arrayx(18).numstores = 999                    ' Assign value.
   form1.Show  ' In load event, must Show form before Print can work.
   Print arrayx(18).tarray(12)   ' Print value.
   Print arrayx(18).numstores    ' Print value.
End Sub
```

NOTE: You cannot change the number of elements in static arrays at run time, but you can use the ReDim statement to change the number of elements in dynamic arrays.

REFERENCES

- Visual Basic version 3.0 for Windows, "Programmer's Guide," Chapter 7, "User-Defined Types (Structures)", pages 176-178. A user-defined type can contain an ordinary (fixed-size) array, but not a dynamic array.

Additional reference words: 2.00 3.00
KBCategory: Prg
KBSubcategory: PrgOther

Category Keywords for All Visual Basic KB Articles

Article ID: Q108753

The information in this article applies to:

- Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Each article in the Visual Basic for Windows collection contains at least one keyword (called a KBSubcategory keyword) that places the article in an appropriate category. This article lists all the KBSubcategory keywords.

MORE INFORMATION

Category & Subcategory Description	KBSubcategory Keyword
Setup / Installation (Setins)	Setins
Environment-specific Issues (Envt)	
VB Design Environment	EnvtDes
Run-Time Environment	EnvtRun
Programming (Prg)	
Visual Basic Forms and Controls	
Standard Controls / Forms	PrgCtrlsStd
Custom Controls	PrgCtrlsCus
Third-Party Controls	PrgCtrlsThird
Optimization	
Memory Management	PrgOptMemMgt
General Optimization Tips	PrgOptTips
General VB Programming	PrgOther
Advanced programming (APrg)	
Network	APrgNet
Windows Programming (APIs / DLLs)	
Printing	APrgPrint
Graphics	APrgGrap
Windowing	APrgWindow
INI Files	APrgINI
Other API / DLL Programming	APrgOther
Data Access	
ODBC	APrgDataODBC
IISAM	APrgDataIISAM
Access	APrgDataAcc
General Database Programming	APrgDataOther
3rd Party DLL's	APrgThirdDLL

Inter-Application Programmability (IAP)

OLE	IAPOLE
DDE	IAPDDE
3rd Party Interoperability	IAPThird

Tools (Tls)

Setup Toolkit / Wizard	TlsSetWiz
Control Development Kit (CDK)	TlsCDK
Help Compiler (HC)	TlsHC

References (Refs)

Documentation / Help File Fixes	RefsDoc
Product Information	RefsProd
Third-Party Information	RefsThird
PSS-Only Information	RefsPSS

Using Keywords to Query the KB

At Microsoft, we use the subcategory keywords to organize the articles for Help files and for the FastTips Catalog. You can use them to query the Microsoft Knowledge Base for Visual Basic articles that apply to that category or subcategory. For example, you can find all the general database programming articles by querying on the following words in the Microsoft Knowledge Base:

```
visual and basic and APrgDataOther
```

Use the asterisk (*) wildcard to find articles that fall into the general categories or into an intermediate subcategory. The first element in each keyword is the category. For example, to find all the articles that apply to Visual Basic Forms and Controls regardless of whether they are standard, custom, or third-party controls, use the following words to query the Microsoft Knowledge Base:

```
visual and basic and PrgCtrls*
```

To find all advanced programming articles, query on these words:

```
visual and basic and APrg*
```

Add KBSubcategory Keyword to Each Article

When contributing an article to the Visual Basic Knowledge Base, add the appropriate KBSubcategory keyword to the bottom of the article on the KBSubcategory line. Each article in the Visual Basic for Windows collection contains the following section at the bottom of the article:

```
Additional reference words:
KBCategory:
KBSubcategory: <keyword>
```

An article usually has only one subcategory keyword, but it may have more.

If you are interested in contributing, please obtain the guidelines by

querying on the following words in the Microsoft Knowledge Base:

   visual and basic and kbguide and kbartwrite

Additional reference words: 3.00 dskbguide subcatkey
KBCategory:
KBSubcategory: RefsPSS

How to Capitalize the First Letter of Each Word in a String

Article ID: Q109220

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

This article shows by example how to capitalize the first letter of each word in a string.

MORE INFORMATION

Step-by-Step Example

The following example capitalizes the first word, and any word preceded by a space or carriage-return-plus-linefeed sequence.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a text box and a command button to Form1.

3. Double-click the text box to open the code window. Add the following code to the LostFocus event for the Text box:

```
Sub Text1_LostFocus ()

   Dim t As String
   t = Text1.Text  ' Put contents of text box into a string variable.
   If t <> "" Then
      Mid$(t, 1, 1) = UCase$(Mid$(t, 1, 1))
      For i = 1 To Len(t) - 1
         If Mid$(t, i, 2) = Chr$(13) + Chr$(10) Then
            ' Capitalize words preceded by carriage return plus
            ' linefeed combination. This only applies when the
            ' text box's MultiLine property is set to True:
            Mid$(t, i + 2, 1) = UCase$(Mid$(t, i + 2, 1))
         End If
         If Mid$(t, i, 1) = " " Then
            ' Capitalize words preceded by a space:
            Mid$(t, i + 1, 1) = UCase$(Mid$(t, i + 1, 1))
         End If
      Next
      Text1.Text = t
   End If

End Sub
```

4. Start the program or press the F5 key.

5. Enter lowercase words in the text box. Click the command button or press TAB to cause the text box to lose the focus. The first letter of each word in the text box will be capitalized. You may continue to enter more text and change the focus as often as you want. Close the form to end the program.

Additional reference words: 2.00 3.00
KBCategory: Prg
KBSubcategory: PrgOther

How to Convert a Decimal Number to a Binary Number in a String

Article ID: Q109260

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

The following sample program shows how to convert a decimal number into its equivalent binary representation stored in a string.

This program accepts a nine-digit positive decimal number and returns a 32-character string that represents the number in binary notation. Negative numbers are converted into the 32-digit, twos-complement binary format used by long integers in Basic.

MORE INFORMATION

In decimal numbers (base-ten numbers), every decimal place is a power of 10. Decimal digits can have values from zero to nine. In binary numbers (base-two numbers), every decimal place is a power of two. Binary digits can only have values of 0 or 1.

Sample Program

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add two text boxes to Form1. Make each text box more than 32 characters wide.

3. Double-click the Text1 text box to open its code window. Choose the Change event from the Proc box. Add the following code to the Text1 Change event:

```
Sub Text1_Change ()

   Dim i As Long, x As Long, bin As String
   Const maxpower = 30   ' Maximum number of binary digits supported.
   text1.MaxLength = 9   ' Maximum number of decimal digits allowed.
   text2.Enabled = False ' Prevent typing in second text box.
   bin = ""  'Build the desired binary number in this string, bin.
   x = Val(text1.Text) 'Convert decimal string in text1 to long integer

   If x > 2 ^ maxpower Then
      MsgBox "Number must be no larger than " & Str$(2 ^ maxpower)
      text2.Text = ""
      Exit Sub
   End If
```

```
      ' Here is the heart of the conversion from decimal to binary:

      ' Negative numbers have "1" in the 32nd left-most digit:
      If x < 0 Then bin = bin + "1" Else bin = bin + "0"

      For i = maxpower To 0 Step -1
         If x And (2 ^ i) Then   ' Use the logical "AND" operator.
            bin = bin + "1"
         Else
            bin = bin + "0"
         End If
      Next
      text2.Text = bin  ' The bin string contains the binary number.

   End Sub
```

4. Start the program, or press the F5 key. Enter decimal numbers into the first text box. The binary equivalent number displays in the second text box.

NOTE: This program converts negative decimal numbers into the internal twos-complement binary format used by Basic. In that format, the left-most binary digit (the thirty-second digit in a long integer) will always be 1 for a negative number and 0 for a positive number.

Decimal Value	Binary Value
0	00000000000000000000000000000000
21	00000000000000000000000000010101
1024	00000000000000000000010000000000
32767	00000000000000000111111111111111
32768	00000000000000001000000000000000
65536	00000000000000010000000000000000
16777216	00000001000000000000000000000000
999999999	00111011100110101100100111111111
-1	11111111111111111111111111111111
-3	11111111111111111111111111111101

Additional reference words: 2.00 3.00
KBCategory: Prg
KBSubcategory: PrgOther

How to Use TABs in a VB Text Box Without Changing the Focus

Article ID: Q109261

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article shows by example how to use the TAB keypress within a control, such as a text box. Normally, the TAB key causes the focus to move away from that control. The sample program in this article shows you how to change this behavior so you can use the TABs within a text box.

The sample program does this by setting the TabStop property of all controls on the form to False when the text box has the focus. Tabbing changes focus between any controls which have a TabStop property equal to True, which is the default. When the TabStop property is true for one or more controls on a form, Visual Basic does not allow tabs to be entered directly into a control.

MORE INFORMATION

Step-by-Step Example

In the example below, the Text2 box will accept and hold TAB keystrokes, keeping them in the Text property along with the other entered characters. The Text1 and Text3 boxes will not accept TAB keystrokes. When Text1 and Text3 have the focus, pressing the TAB key changes the focus to the next control in the tab order.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add three text boxes (Text1, Text2, and Text3) to Form1. Select the Text2 box and press the F4 key to display the Properties window. Set the MultiLine property of Text2 to True.

   NOTE: When you press the TAB key, single-line text boxes beep and do not accept the TAB keystroke, but multiLine text boxes do accept TAB keystrokes.

3. Double-click the Text2 box to open the code window. Choose the GotFocus event from the Proc box. Add the following code to the Text2 GotFocus event:

```
Sub Text2_GotFocus ()
    ' When Text2 gets the focus, clear all TabStop properties on all
    ' controls on the form. Ignore all errors, in case a control does
    ' not have the TabStop property.
    On Error Resume Next
```

```
   For i = 0 To Controls.Count - 1   ' Use the Controls collection
      Controls(i).TabStop = False
   Next
End Sub
```

NOTE: See the "Controls Collection" section below for an explanation of the Controls collection.

4. Choose the LostFocus event from the Proc box. Add the following code to the Text2 LostFocus event:

```
Sub Text2_LostFocus ()
   ' When Text2 loses the focus, make the TabStop property True for all
   ' controls on the form. That restores the ability to tab between
   ' controls. Ignore all errors, in case a control does not have the
   ' TabStop property.
   On Error Resume Next
   For i = 0 To Controls.Count - 1   ' Use the Controls collection
      Controls(i).TabStop = True
   Next
End Sub
```

5. Start the program, or press the F5 key. Press the TAB key to give focus to Text2. Enter text into the Text2 box, pressing the TAB key as needed. Whenever Text1 or Text3 has the focus, pressing the TAB key moves the focus to the next control. Whenever Text2 has the focus, TAB keystrokes remain with the text in the text box. Close the form to end the program.

Tab Order

By default, Visual Basic assigns tab order to controls in the order you draw them on a form. Each new control is placed last in the tab order. You can control the order that controls gain focus in your application by changing the tab order at design time through the Properties window, or at run time through code.

To change tab order at design time:

1. Click a control to select it.

2. From the Properties window, select TabIndex. Visual Basic displays the current tab position in the Settings box.

3. Type the number for the tab order position you want the control to have.

4. Click the Enter button. You can test the tab order at design time by pressing Tab.

To enable or disable a tab stop at design time:

1. Click a control to select it.

2. From the Properties window, select TabStop. Visual Basic displays the current Boolean value in the Settings box.

3. Select True to designate the control as a tab stop, or select False to

bypass the control in the tab order.

4. Click the Enter button.

When you change a control's tab order position, Visual Basic automatically renumbers the tab order positions of other controls to reflect insertions and deletions.

A control whose TabStop property has been set to False maintains its position in the actual tab order as set by the TabIndex property, even though the control is skipped when you cycle through the controls by using the TAB key. If the TabStop property is False for all controls on the form, you can enter TAB keystrokes into MultiLine text boxes.

Controls Collection

The Controls collection is a collection whose elements represent each control on a form, including elements of control arrays. The Controls collection has a single property (Count) that specifies the number of elements in an array.

The Controls collection enumerates loaded controls on a form and is useful for iterating through them. The index in the syntax is between 0 and Controls.Count-1.

NOTE: Controls is a keyword but not a reserved word. It identifies an intrinsic form-level variable named Controls. If you omit the optional form reference, you must include the Controls keyword. If you include a form reference, you can omit the Controls keyword. For example, the following two lines have the same effect:

```
MyForm.Controls(6).Top = MyForm.Controls(5).Top + increment
MyForm(6).Top = MyForm(5).Top + increment
```

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgOther

PRB: VB 3.0 AppActivate Fails on 32-Bit Windows NT Application

Article ID: Q109262

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

Visual Basic's AppActivate statement fails to make a 32-bit application the active window under Windows NT.

To reproduce this behavior under Microsoft Windows NT, run Notepad (NOTEPAD.EXE). Then run the following code in Visual Basic:

```
Sub Form_Load ()
   AppActivate "Notepad - (Untitled)"
End Sub
```

Visual Basic fails to give focus to the Notepad session.

CAUSE

Under the 32-bit Windows NT system, 16-bit Windows-subsystem applications may not be fully available to other 16-bit programs. Visual Basic version 3.0 is a 16-bit program originally designed for the 16-bit Windows operating environment, so this behavior is by design.

WORKAROUND

To work around this behavior, use the FindWindow and SetWindowPos Windows API functions as follows:

1. Start a new project in Visual Basic. Form1 is created by default.

2. Double-click the form to open the code window. Select (general) from the Object box. Enter the following in the (general) (declarations) window:

   ```
   Declare Function FindWindow% Lib "USER" (ByVal Class&, ByVal Caption$)
   ' The following Declare statement must be on one, single line:
   Declare Sub SetWindowPos Lib "user" (ByVal hwnd%, ByVal hwndAfter%,
      ByVal x%, ByVal y%, ByVal cx%, ByVal cy%, ByVal swp%)
   ```

3. Select form from the Object box. Add the following code to the Form Click event:

   ```
   Sub Form_Click ()

      Const SWP_NOSIZE% = &H1
   ```

```
    Const SWP_NOMOVE% = &H2
    AppActivate "Notepad - (Untitled)"
    x = FindWindow(0, "Notepad - (Untitled)")
    SetWindowPos x, 0, 0, 0, 0, 0, SWP_NOSIZE Or SWP_NOMOVE
    Debug.Print Hex$(x) ' Print return code from FindWindow API function.

End Sub
```

4. Start Notepad in Windows NT version 3.1.

5. Start the Visual Basic program, or press the F5 key. Click the form to activate Notepad. When finished, close the form to end the Visual Basic program.

STATUS
======

This behavior is by design. It is under review and will be considered for enhancement in a future release of Visual Basic.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgOther

How to Find Num of Days Between Dates Outside of Normal Range

Article ID: Q109451

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

To find the number of days between any two dates, you can take the difference between the values returned by the DateSerial function for two dates. However, the DateSerial function only supports dates from January 1, 100 through December 31, 9999.

To support a much wider range of dates, use the AstroDay function as in this example:

```
Function AstroDay(inyear, inmonth, inday)
' The AstroDay function returns the Astronomical Day for any given date.
y = inyear + (inmonth - 2.85) / 12
AstroDay=Int(Int(Int(367*y)-1.75*Int(y)+inday)-.75*Int(.01*y))+1721119
' NOTE: Basic's Int function returns the integer part of a number.
End Function
```

For example, the number of days between February 28, 12000 and March 1, 12000 is 2 because the year 12000 is a leap year:

```
Print AstroDay(12000, 3, 1) - AstroDay(12000, 2, 28)   'Prints 2
```

In addition, the AstroWeekDay function defined farther below returns the day of the week, Sunday through Monday, for any given AstroDay. AstroWeekDay supports dates outside the range (January 1, 100 through December 31, 9999) of Visual Basic's WeekDay function.

MORE INFORMATION

The AstroDay function defined in this article is a modified version of the Julian date formula used by astronomers.

Visual Basic's DateSerial function returns a Variant of VarType 7 (Date) containing a date that is stored internally as a double-precision number. This number represents a date from January 1, 100 through December 31, 9999, where January 1, 1900 is 2. Negative numbers represent dates prior to December 30, 1899.

Leap years are accurately handled by both Visual Basic's DateSerial function and the Astronomical Day function (AstroDay) defined in this article.

A leap year is defined as all years divisible by 4, except for years divisible by 100 that are not also divisible by 400. Years divisible by 400

are leap years. 2000 is a leap year. 1900 is not a leap year.

Step-by-Step Example
---

1. Start a new project in Visual Basic. Form1 is created by default.

2. Double-click the form to open the code window. Add the following code to the Form Load event:

```
Sub Form_Load ()

   form1.Show  ' Must first Show form in Load event for Print to work.
   Print AstroDay(12000, 3, 1) - AstroDay(12000, 2, 28)  'Prints 2
   Print AstroDay(-12400, 3, 1) - AstroDay(-12400, 2, 28)  'Prints 2
   Print AstroDay(12000, 3, 1) - AstroDay(-12000, 2, 28) 'Prints 8765822
   Print AstroDay(1902, 2, 28) - AstroDay(1898, 3, 1)  'Prints 1459 days
   Print AstroWeekDay(AstroDay(1993, 12, 1))  ' Prints Wednesday
   Print AstroWeekDay(AstroDay(12000, 3, 2))  ' Prints Thursday

   ' You can also use Visual Basic's DateSerial function as follows to
   ' find the number of days between two dates:
   Print DateSerial(1902, 2, 28) - DateSerial(1898, 3, 1)

   ' Visual Basic's WeekDay function returns an integer betw 1 (Sunday)
   ' and 7 (Saturday), which represents the day of the week for a date
   ' argument:
   Print "Day of week = " & Weekday(DateSerial(1898, 3, 1))

End Sub
```

3. In the Object box on the Form1.Frm code window, select (general). Add the following functions:

```
Function AstroDay(inyear, inmonth, inday)
' The AstroDay function returns the Astronomical Day for any given date.
y = inyear + (inmonth - 2.85) / 12
AstroDay=Int(Int(Int(367*y)-1.75*Int(y)+inday)-.75*Int(.01*y))+1721119
' NOTE: Basic's Int function returns the integer part of an number.
End Function

Function AstroWeekDay (aday)
' The AstroWeekDay function returns the day of the week, Sunday through
' Monday, for any given AstroDay. The aday parameter must be a day
' number returned by the AstroDay function.
weekdayx = (aday - 3) Mod 7
Select Case weekdayx
Case 0
   AstroWeekDay = "Sunday"
Case 1
   AstroWeekDay = "Monday"
Case 2
   AstroWeekDay = "Tuesday"
Case 3
   AstroWeekDay = "Wednesday"
Case 4
   AstroWeekDay = "Thursday"
```

```
   Case 5
      AstroWeekDay = "Friday"
   Case 6
      AstroWeekDay = "Saturday"
   End Select
   End Function
```

4. Start the program (or press the F5 key). Close the form to end the program.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgOther

How to Scroll a Form When VB Forms Are Limited to Screen Size

Article ID: Q109741

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

A Visual Basic form cannot be sized larger than the screen. This article explains how to scroll the contents of a form to enlarge the usable area of a form.

The sample program below works by scrolling a picture box control which is larger than the form and contains attached controls. When the picture box scrolls, all the attached controls scroll together.

MORE INFORMATION

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a horizontal scroll bar control and a vertical scroll bar control to Form1. (The size doesn't matter because the program automatically sizes the scroll bars in the Form Resize event code.)

3. Add a picture box control to Form1. Draw a text box control inside the upper left corner of the picture box such that when the picture box moves, the text box moves with it.

   Select the Text1 box and press the F4 key to display the Properties window. Set the Text1 Index property to 0, which is required at design time to make an array of text controls.

   You can also enhance this sample program by placing more controls into the picture box. When the picture box scrolls, all the controls scroll.

4. Add the following code to the Form Load event:

```
Sub Form_Load ()
    ' Make the picture box bigger than the form:
    Picture1.Move 0, 0, 1.4 * ScaleWidth, 1.2 * ScaleHeight
    ' Place some sample controls in the picture box:
    Dim i As Integer
    For i = 1 To 20
        Load Text1(i)
        Text1(i).Visible = True
        Text1(i).Left = i * Picture1.Height / 20
        Text1(i).Top = Text1(i).Left
```

```
      Next
   End Sub
```

5. Add the following code to the Form Resize event:

```
   Sub Form_Resize ()
      ' Position the scroll bars:
      hscroll1.Left = 0
      vscroll1.Top = 0
      If Picture1.Width > scalewidth Then
         hscroll1.Top = ScaleHeight - hscroll1.Height
      Else
         hscroll1.Top = ScaleHeight
      End If
      If Picture1.Height > hscroll1.Top Then
         vscroll1.Left = scalewidth - vscroll1.Width
         If Picture1.Width > vscroll1.Left Then
            hscroll1.Top = ScaleHeight - hscroll1.Height
         End If
      Else
         vscroll1.Left = scalewidth
      End If
      hscroll1.Width = scalewidth
      vscroll1.Height = hscroll1.Top

      ' Set the scroll bar ranges
      hscroll1.Max = Picture1.Width - vscroll1.Left
      vscroll1.Max = Picture1.Height - hscroll1.Top
      hscroll1.SmallChange = Abs(hscroll1.Max \ 16) + 1
      hscroll1.LargeChange = Abs(hscroll1.Max \ 4) + 1
      vscroll1.SmallChange = Abs(vscroll1.Max \ 16) + 1
      vscroll1.LargeChange = Abs(vscroll1.Max \ 4) + 1
      hscroll1.ZOrder 0
      vscroll1.ZOrder 0

   End Sub
```

6. Add the following code to the HScroll1 Change event:

```
   Sub HScroll1_Change ()
      Picture1.Left = -HScroll1.Value
   End Sub
```

7. Add the following code to the VScroll1 Change event:

```
   Sub VScroll1_Change ()
      Picture1.Top = -VScroll1.Value
   End Sub
```

8. Start the program (or press the F5 key). Click the scroll bars to scroll the form. Close the form to end the program.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgOther

How to Speed Up Data Access by Using BeginTrans & CommitTrans

Article ID: Q109830

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

You can speed up database operations by many times in a Microsoft Access database by using transactions. A transaction starts with a BeginTrans statement and ends with a CommitTrans or Rollback statement.

The sample program below is more than 17 times faster when using BeginTrans/CommitTrans. Performance may vary on different computers.

MORE INFORMATION

You can tune the performance of Visual Basic by using transactions for operations that update data. A transaction is a series of operations that must execute as a whole or not at all. You mark the beginning of a transaction with the BeginTrans statement. You use the Rollback or CommitTrans statement to end a transaction.

You can usually increase the record updates per second (throughput) of an application by placing operations that update data within an Access Basic transaction.

Because Visual Basic locks data pages used in a transaction until the transaction ends, using transactions will prevent access to those data pages by other users while the transaction is pending. If you use transactions in a multi-user environment, try to find a balance between data throughput and data access.

If database operations are not within a transaction, every Update method causes a disk write.

Transactions are very fast because they are written to a buffer in memory instead of to disk. CommitTrans writes the changes in the transaction buffer to disk. The size of the transaction buffer can be set in your MSACCESS.INI file, found in your Windows directory. See the PERFORM.TXT file in your Visual Basic directory for more information. Robust error trapping is important when using transactions to avoid losing writes if the program gets an error in the middle of a transaction.

For more performance tuning tips for data access in Microsoft Visual Basic version 3.0, see the PERFORM.TXT file.

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the following to the Form Load event code:

```
Sub Form_Load ()
   Dim Starttime, Endtime
   Dim db As Database
   Dim t As Table
   Dim i As Integer
   Dim tempName As String
   Dim temphone As String
   Set db = OpenDatabase("c:\BIBLIO.MDB")  ' Uses a copy of BIBLIO.MDB
   Set t = db.OpenTable("Publishers")
   Starttime = Now
   'BeginTrans   ' Add this and CommitTrans (below) for greater speed.
   For i = 1 To 100
      tempName = "testname" & Str$(i)  'Make an arbitrary unique string.
      tempPhone = Str$(i)              'Make arbitrary number.
      t.AddNew  'AddNew clears copy buffer to prepare for new record.
      t!PubID = 30 + i  ' Set primary key to unique value.
      t!Name = tempName  ' Set Name field to unique value.
      t!Telephone = tempPhone  ' Set Telephone field to unique value.
      t.Update   ' Write the record to disk or to transaction buffer.
   Next i
   'CommitTrans   ' Add this and BeginTrans (above) for greater speed.
   Endtime = Now
   MsgBox "Time required= " & Format(Endtime - Starttime, "hh:mm:ss")
   t.Close
   db.Close
   End
End Sub
```

   The above code adds 100 new records to the BIBLIO.MDB database file. Add the records to a copy of BIBLIO.MDB instead of to the original.

3. Start the program (or press the F5 key). A message box reports the time required to add 100 new records. Close the form to end the program.

If you don't use the BeginTrans and CommitTrans statements, this program reports 17 seconds to add 100 records on a 486/66 PC. When you add BeginTrans and CommitTrans as shown in the program comments above, the program takes less than 1 second on that computer. Performance may vary on different computers.

REFERENCES
==========

- "Microsoft Developer Network News" newspaper, January 1994, Volume 3, Number 1, published by Microsoft Corporation.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataAcc PrgOptTips

LONG: Microsoft Consulting Services Naming Conventions for VB

Article ID: Q110264

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

It is a good idea to establish naming conventions for your Visual Basic code. This article gives you the naming conventions used by Microsoft Consulting Services (MCS).

This document is a superset of the Visual Basic coding conventions found in the Visual Basic "Programmer's Guide."

NOTE: The third-party controls mentioned in this article are manufactured by vendors independent of Microsoft. Microsoft makes no warranty, implied or otherwise, regarding these controls' performance or reliability.

MORE INFORMATION

Naming conventions help Visual Basic programmers:

- Standardize the structure, coding style and logic of an application.
- Create precise, readable, and unambiguous source code.
- Be consistent with other language conventions (most importantly, the Visual Basic Programmers Guide and standard Windows C Hungarian notation).
- Be efficient from a string size and labor standpoint, thus allowing a greater opportunity for longer and fuller object names.
- Define the minimal requirements necessary to do the above.

Setting Environment Options

Use Option Explicit. Declare all variables to save programming time by reducing the number of bugs caused by typos (for example, aUserNameTmp vs. sUserNameTmp vs. sUserNameTemp). In the Environment Options dialog, set Require Variable Declaration to Yes. The Option Explicit statement requires you to declare all the variables in your Visual Basic program.

Save Files as ASCII Text. Save form (.FRM) and module (.BAS) files as ASCII text to facilitate the use of version control systems and minimize the damage that can be caused by disk corruption. In addition, you can:

- Use your own editor
- Use automated tools, such as grep
- Create code generation or CASE tools for Visual Basic
- Perform external analysis of your Visual Basic code

To have Visual Basic always save files as ASCII text, from the Environment Options dialog, set the Default Save As Format option to Text.

Object Naming Conventions for Standard Objects

The following tables define the MCS standard object name prefixes. These prefixes are consistent with those documented in the Visual Basic version 3.0 Programmers Guide.

Prefix	Object Type	Example
ani	Animation button	aniMailBox
bed	Pen Bedit	bedFirstName
cbo	Combo box and drop down list box	cboEnglish
chk	Checkbox	chkReadOnly
clp	Picture clip	clpToolbar
cmd (3d)	Command button (3D)	cmdOk (cmd3dOk)
com	Communications	comFax
ctr	Control (when specific type unknown)	ctrCurrent
dat	Data control	datBiblio
dir	Directory list box	dirSource
dlg	Common dialog control	dlgFileOpen
drv	Drive list box	drvTarget
fil	File list box	filSource
frm	Form	frmEntry
fra (3d)	Frame (3d)	fraStyle (fra3dStyle)
gau	Gauge	gauStatus
gpb	Group push button	gpbChannel
gra	Graph	graRevenue
grd	Grid	grdPrices
hed	Pen Hedit	hedSignature
hsb	Horizontal scroll bar	hsbVolume
img	Image	imgIcon
ink	Pen Ink	inkMap
key	Keyboard key status	keyCaps
lbl	Label	lblHelpMessage
lin	Line	linVertical
lst	List box	lstPolicyCodes
mdi	MDI child form	mdiNote
mpm	MAPI message	mpmSentMessage
mps	MAPI session	mpsSession
mci	MCI	mciVideo
mnu	Menu	mnuFileOpen
opt (3d)	Option Button (3d)	optRed (opt3dRed)
ole	OLE control	oleWorksheet
out	Outline control	outOrgChart
pic	Picture	picVGA
pnl3d	3d Panel	pnl3d
rpt	Report control	rptQtr1Earnings
shp	Shape controls	shpCircle
spn	Spin control	spnPages
txt	Text Box	txtLastName
tmr	Timer	tmrAlarm
vsb	Vertical scroll bar	vsbRate

Object Naming Convention for Database Objects

Prefix	Object Type	Example
db	ODBC Database	dbAccounts
ds	ODBC Dynaset object	dsSalesByRegion
fdc	Field collection	fdcCustomer
fd	Field object	fdAddress
ix	Index object	ixAge
ixc	Index collection	ixcNewAge
qd	QueryDef object	qdSalesByRegion
qry (suffix)	Query (see NOTE)	SalesByRegionQry
ss	Snapshot object	ssForecast
tb	Table object	tbCustomer
td	TableDef object	tdCustomers

NOTE: Using a suffix for queries allows each query to be sorted with its associated table in Microsoft Access dialogs (Add Table, List Tables Snapshot).

Menu Naming Conventions

Applications frequently use an abundance of menu controls. As a result, you need a different set of naming conventions for these controls. Menu control prefixes should be extended beyond the initial mnu label by adding an additional prefix for each level of nesting, with the final menu caption at the end of the name string. For example:

Menu Caption Sequence	Menu Handler Name
Help.Contents	mnuHelpContents
File.Open	mnuFileOpen
Format.Character	mnuFormatCharacter
File.Send.Fax	mnuFileSendFax
File.Send.Email	mnuFileSendEmail

When this convention is used, all members of a particular menu group are listed next to each other in the object drop-down list boxes (in the code window and property window). In addition, the menu control names clearly document the menu items to which they are attached.

Naming Conventions for Other Controls

For new controls not listed above, try to come up with a unique three character prefix. However, it is more important to be clear than to stick to three characters.

For derivative controls, such as an enhanced list box, extend the prefixes above so that there is no confusion over which control is really being used. A lower-case abbreviation for the manufacturer would also typically be added to the prefix. For example, a control instance created from the Visual Basic Professional 3D frame could uses a prefix of fra3d to avoid confusion over which control is really being used. A command button from MicroHelp could use cmdm to differentiate it from the standard command button (cmd).

Third Party Controls

Each third party control used in an application should be listed in the application's overview comment section, providing the prefix used for the control, the full name of the control, and the name of the software vendor:

Prefix	Control Type	Vendor
cmdm	Command Button	MicroHelp

Variable and Routine Naming

Variable and function names have the following structure:

<prefix><body><qualifier><suffix>

Part	Description	Example
<prefix>	Describes the use and scope of the variable.	iGetRecordNext
<body>	Describes the variable.	iGetNameFirst
<qualifier>	Denotes a derivative of the variable.	iGetNameLast
<suffix>	The optional Visual Basic type character.	iGetRecordNext%

Prefixes:

The following tables define variable and function name prefixes that are based on Hungarian C notation for Windows. These prefixes should be used with all variables and function names. Use of old Basic suffixes (such as %, &, #, etc.) are discouraged.

Variable and Function Name Prefixes:

Prefix	Converged	Variable Use	Data Type	Suffix
b	bln	Boolean	Integer	%
c	cur	Currency - 64 bits	Currency	@
d	dbl	Double - 64 bit signed quantity	Double	#
dt	dat	Date and Time	Variant	
e	err	Error		
f	sng	Float/Single - 32 bit signed floating point	Single	!
h		Handle	Integer	%
i		Index	Integer	%
l	lng	Long - 32 bit signed quantity	Long	&
n	int	Number/Counter	Integer	%
s	str	String	String	$
u		Unsigned - 16 bit unsigned quantity	Long	&
	udt	User-defined type		
vnt	vnt	Variant	Variant	
a		Array		

NOTE: the values in the Converged column represent efforts to pull together the naming standards for Visual Basic, Visual Basic for Applications, and Access Basic. It is likely that these prefixes will become Microsoft standards at some point in the near future.

Scope and Usage Prefixes:

Prefix	Description
g	Global
m	Local to module or form
st	Static variable
(no prefix)	Non-static variable, prefix local to procedure
v	Variable passed by value (local to a routine)
r	Variable passed by reference (local to a routine)

Hungarian notation is as valuable in Visual Basic as it is in C. Although the Visual Basic type suffixes do indicate a variable's data type, they do not explain what a variable or function is used for, or how it can be accessed. Here are some examples:

iSend - Represents a count of the number of messages sent
bSend - A Boolean flag defining the success of the last Send operation
hSend - A Handle to the Comm interface

Each of these variable names tell a programmer something very different. This information is lost when the variable name is reduced to Send%. Scope prefixes such as g and m also help reduce the problem of name contention especially in multi-developer projects.

Hungarian notation is also widely used by Windows C programmers and constantly referenced in Microsoft product documentation and in industry programming books. Additionally, the bond between C programmers and programmers who use Visual Basic will become much stronger as the Visual C++ development system gains momentum. This transition will result in many Visual Basic programmers moving to C for the first time and many programmers moving frequently back and forth between both environments.

The Body of Variable and Routine Names

The body of a variable or routine name should use mixed case and should be as long as necessary to describe its purpose. In addition, function names should begin with a verb, such as InitNameArray or CloseDialog.

For frequently used or long terms, standard abbreviations are recommended to help keep name lengths reasonable. In general, variable names greater than 32 characters can be difficult to read on VGA displays.

When using abbreviations, make sure they are consistent throughout the entire application. Randomly switching between Cnt and Count within a project will lead to unnecessary confusion.

Qualifiers on Variable and Routine Names

Related variables and routines are often used to manage and manipulate a common object. In these cases, use standard qualifiers to label the derivative variables and routines. Although putting the qualifier after the body of the name might seem a little awkward (as in sGetNameFirst, sGetNameLast instead of sGetFirstName, sGetLastName), this practice will help order these names together in the Visual Basic editor routine lists, making the logic and structure of the application easier to understand.

The following table defines common qualifiers and their standard meaning:

Qualifier	Description (follows Body)
First	First element of a set.
Last	Last element of a set.
Next	Next element in a set.
Prev	Previous element in a set.
Cur	Current element in a set.
Min	Minimum value in a set.
Max	Maximum value in a set.
Save	Used to preserve another variable that must be reset later.
Tmp	A "scratch" variable whose scope is highly localized within the code. The value of a Tmp variable is usually only valid across a set of contiguous statements within a single procedure.
Src	Source. Frequently used in comparison and transfer routines.
Dst	Destination. Often used in conjunction with Source.

User Defined Types

Declare user defined types in all caps with _TYPE appended to the end of the symbol name. For example:

```
Type CUSTOMER_TYPE
   sName As String
   sState As String * 2
   lID as Long
End Type
```

When declaring an instance variable of a user defined type, add a prefix to the variable name to reference the type. For example:

```
Dim custNew as CUSTOMER_TYPE
```

Naming Constants

The body of constant names should be UPPER_CASE with underscores (_) between words. Although standard Visual Basic constants do not include Hungarian information, prefixes like i, s, g, and m can be very useful in understanding the value and scope of a constant. For constant names, follow the same rules as variables. For Example:

```
mnUSER_LIST_MAX   ' Max entry limit for User list (integer value,
                  ' local to module)
gsNEW_LINE        ' New Line character string (global to entire
                  ' application)
```

Variant Data Type

If you know that a variable will always store data of a particular type, Visual Basic can handle that data more efficiently if you declare a variable of that type.

However, the variant data type can be extremely useful when working with databases, messages, DDE, or OLE. Many databases allow NULL as a valid value for a field. Your code needs to distinguish between NULL, 0 (zero), and "" (empty string). Many times, these types of operations can use a generic service routine that does not need to know the type of data it receives to process or pass on the data.

For example:

```
Sub ConvertNulls(rvntOrg As Variant, rvntSub As Variant)
   ' If rvntOrg = Null, replace the Null with rvntSub
   If IsNull(rvntOrg) Then rvntOrg = rvntSub
End Sub
```

The are some drawbacks, however, to using variants. Code statements that use variants can sometimes be ambiguous to the programmer.

For example:

```
vnt1 = "10.01" : vnt2 = 11 : vnt3 = "11" : vnt4 = "x4"
vntResult = vnt1 + vnt2  ' Does vntResult = 21.01 or 10.0111?
vntResult = vnt2 + vnt1  ' Does vntResult = 21.01 or 1110.01?
vntResult = vnt1 + vnt3  ' Does vntResult = 21.01 or 10.0111?
vntResult = vnt3 + vnt1  ' Does vntResult = 21.01 or 1110.01?
vntResult = vnt2 + vnt4  ' Does vntResult = 11x4 or ERROR?
vntResult = vnt3 + vnt4  ' Does vntResult = 11x4 or ERROR?
```

The above examples would be much less ambiguous and easier to read, debug, and maintain if the Visual Basic type conversion routines were used instead.

For Example:

```
iVar1 = 5 + val(sVar2)   ' use this (explicit conversion)
vntVar1 = 5 + vntVar2    ' not this (implicit conversion)
```

Commenting Your Code

All procedures and functions should begin with a brief comment describing the functional characteristics of the routine (what it does). This description should not describe the implementation details (how it does it) because these often change over time, resulting in unnecessary comment maintenance work, or worse yet, erroneous comments. The code itself and any necessary in-line or local comments will describe the implementation.

Parameters passed to a routine should be described when their functions are not obvious and when the routine expects the parameters to be in a specific range. Function return values and global variables that are changed by the

routine (especially through reference parameters) must also be described at the beginning of each routine.

Routine header comment blocks should look like this (see the next section "Formatting Your Code" for an example):

Section	Comment Description
Purpose	What the routine does (not how).
Inputs	Each non-obvious parameter on a separate line with in-line comments
Assumes	List of each non-obvious external variable, control, open file, and so on.
Returns	Explanation of value returned for functions.
Effects	List of each effected external variable, control, file, and so on and the affect it has (only if this is not obvious)

Every non-trivial variable declaration should include an in-line comment describing the use of the variable being declared.

Variables, controls, and routines should be named clearly enough that in-line commenting is only needed for complex or non-intuitive implementation details.

An overview description of the application, enumerating primary data objects, routines, algorithms, dialogs, database and file system dependencies, and so on should be included at the start of the .BAS module that contains the project's Visual Basic generic constant declarations.

NOTE: The Project window inherently describes the list of files in a project, so this overview section only needs to provide information on the most important files and modules, or the files the Project window doesn't list, such as initialization (.INI) or database files.

Formatting Your Code

Because many programmers still use VGA displays, screen real estate must be conserved as much as possible, while still allowing code formatting to reflect logic structure and nesting.

Standard, tab-based, block nesting indentations should be two to four spaces. More than four spaces is unnecessary and can cause statements to be hidden or accidentally truncated. Less than two spaces does not sufficiently show logic nesting. In the Microsoft Knowledge Base, we use a three-space indent. Use the Environment Options dialog to set the default tab width.

The functional overview comment of a routine should be indented one space. The highest level statements that follow the overview comment should be indented one tab, with each nested block indented an additional tab.

For example:

```
'*********************************************************************
'Purpose:   Locate first occurrence of a specified user in UserList array.
```

```
'Inputs:    rasUserList():  the list of users to be searched
'           rsTargetUser:   the name of the user to search for
'Returns:   the index of the first occurrence of the rsTargetUser
'           in the rasUserList array. If target user not found, return -1.
'**************************************************************************
'Enter the next two lines as one, single line:
Function iFindUser (rasUserList() As String, rsTargetUser as String)
   As Integer
   Dim i As Integer           ' loop counter
   Dim bFound As Integer      ' target found flag
   iFindUser = -1
   i = 0
   While i <= Ubound(rasUserList) and Not bFound
      If rasUserList(i) = rsTargetUser Then
         bFound = True
         iFindUser = i
      End If
   Wend
End Function
```

Variables and non-generic constants should be grouped by function rather than by being split off into isolated areas or special files. Visual Basic generic constants such as HOURGLASS should be grouped in a single module (VB_STD.BAS) to keep them separate from application-specific declarations.

Operators

Always use an ampersand (&) when concatenating strings, and use the plus sign (+) when working with numerical values. Using a plus sign (+) with nonnumerical values, may cause problems when operating on two variants.

For example:

```
   vntVar1 = "10.01"
   vntVar2 = 11
   vntResult = vntVar1 + vntVar2         ' vntResult =  21.01
   vntResult = vntVar1 & vntVar2         ' vntResult = 10.0111
```

Scope

Variables should always be defined with the smallest scope possible. Global variables can create enormously complex state machines and make the logic of an application extremely difficult to understand. Global variables also make the reuse and maintenance of your code much more difficult.

Variables in Visual Basic can have the following scope:

Scope	Variable Declared In:	Visibility
Procedure-level	Event procedure, sub, or function	Visible in the procedure in which it is declared
Form-level,	Declarations section of a form	Visible in every

Module-level	or code module (.FRM, .BAS)	procedure in the form or code module
Global	Declarations section of a code module (.BAS, using Global keyword)	Always visible

In a Visual Basic application, only use global variables when there is no other convenient way to share data between forms. You may want to consider storing information in a control's Tag property, which can be accessed globally using the form.object.property syntax.

If you must use global variables, it is good practice to declare all of them in a single module and group them by function. Give the module a meaningful name that indicates its purpose, such as GLOBAL.BAS.

With the exception of global variables (which should not be passed), procedures and functions should only operate on objects that are passed to them. Global variables that are used in routines should be identified in the general comment area at the beginning of the routine. In addition, pass arguments to subs and functions using ByVal, unless you explicitly want to change the value of the passed argument.

Write modular code whenever possible. For example, if your application displays a dialog box, put all the controls and code required to perform the dialog's task in a single form. This helps to keep the application's code organized into useful components and minimizes its runtime overhead.

Third Party Controls

NOTE: The products discussed below are manufactured by vendors independent of Microsoft. Microsoft makes no warranty, implied or otherwise, regarding these products' performance or reliability.

The following table lists standard third party vendor name prefix characters to be used with control prefixes:

Vendor	Abbv
MicroHelp (VBTools)	m
Pioneer Software	p
Crescent Software	c
Sheridan Software	s
Other (Misc)	o

The following table lists standard third party control prefixes:

Control Type	Control Name	Abbr	Vendor	Example	VBX File Name
Alarm	Alarm	almm	MicroHelp	almmAlarm	MHTI200.VBX
Animate	Animate	anim	MicroHelp	animAnimate	MHTI200.VBX
Callback	Callback	calm	MicroHelp	calmCallback	MHAD200.VBX
Combo Box	DB_Combo	cbop	Pioneer	cbopComboBox	QEVBDBF.VBX
Combo Box	SSCombo	cbos	Sheridan	cbosComboBox	SS3D2.VBX
Check Box	DB_Check	chkp	Pioneer	chkpCheckBox	QEVBDBF.VBX

Chart	Chart	chtm	MicroHelp	chtmChart	MHGR200.VBX
Clock	Clock	clkm	MicroHelp	clkmClock	MHTI200.VBX
Button	Command Button	cmdm	MicroHelp	cmdmCommandButton	MHEN200.VBX
Button	DB_Command	cmdp	Pioneer	cmdpCommandButton	QEVBDBF.VBX
Button (Group)	Command Button (multiple)	cmgm	MicroHelp	cmgmBtton	MHGR200.VBX
Button	Command Button (icon)	cmim	MicroHelp	cmimCommandButton	MHEN200.VBX
CardDeck	CardDeck	crdm	MicroHelp	crdmCard	MHGR200.VBX
Dice	Dice	dicm	MicroHelp	dicmDice	MHGR200.VBX
List Box (Dir)	SSDir	dirs	Sheridan	dirsDirList	SS3D2.VBX
List Box (Drv)	SSDrive	drvs	Sheridan	drvsDriveList	SS3D2.VBX
List Box (File)	File List	film	MicroHelp	filmFileList	MHEN200.VBX
List Box (File)	SSFile	fils	Sheridan	filsFileList	SS3D2.VBX
Flip	Flip	flpm	MicroHelp	flpmButton	MHEN200.VBX
Scroll Bar	Form Scroll	fsrm	MicroHelp	fsrmFormScroll	???
Gauge	Gauge	gagm	MicroHelp	gagmGauge	MHGR200.VBX
Graph	Graph	gpho	Other	gphoGraph	XYGRAPH.VBX
Grid	Q_Grid	grdp	Pioneer	grdpGrid	QEVBDBF.VBX
Scroll Bar	Horizontal Scroll Bar	hsbm	MicroHelp	hsbmScroll	MHEN200.VBX
Scroll Bar	DB_HScroll	hsbp	Pioneer	hsbpScroll	QEVBDBF.VBX
Graph	Histo	hstm	MicroHelp	hstmHistograph	MHGR200.VBX
Invisible	Invisible	invm	MicroHelp	invmInvisible	MHGR200.VBX
List Box	Icon Tag	itgm	MicroHelp	itgmListBox	MHAD200.VBX
Key State	Key State	kstm	MicroHelp	kstmKeyState	MHTI200.VBX
Label	Label (3d)	lblm	MicroHelp	lblmLabel	MHEN200.VBX
Line	Line	linm	MicroHelp	linmLine	MHGR200.VBX
List Box	DB_List	lstp	Pioneer	lstpListBox	QEVBDBF.VBX
List Box	SSList	lsts	Sheridan	lstsListBox	SS3D2.VBX
MDI Child	MDI Control	mdcm	MicroHelp	mdcmMDIChild	???
Menu	SSMenu	mnus	Sheridan	mnusMenu	SS3D3.VBX
Marque	Marque	mrqm	MicroHelp	mrqmMarque	MHTI200.VB
Picture	OddPic	odpm	MicroHelp	odpmPicture	MHGR200.VBX
Picture	Picture	picm	MicroHelp	picmPicture	MHGR200.VBX
Picture	DB_Picture	picp	Pioneer	picpPicture	QEVBDBF.VBX
Property Vwr	Property Viewer	pvrm	MicroHelp	pvrmPropertyViewer	MHPR200.VBX
Option (Group)	DB_RadioGroup	radp	Pioneer	radqRadioGroup	QEVBDBF.VBX
Slider	Slider	sldm	MicroHelp	sldmSlider	MHGR200.VBX
Button (Spin)	Spinner	spnm	MicroHelp	spnmSpinner	MHEN200.VBX
Spreadsheet	Spreadsheet	sprm	MicroHelp	sprmSpreadsheet	MHAD200.VBX
Picture	Stretcher	strm	MicroHelp	strmStretcher	MHAD200.VBX
Screen Saver	Screen Saver	svrm	MicroHelp	svrmSaver	MHTI200.VBX
Switcher	Switcher	swtm	MicroHelp	swtmSwitcher	???
List Box	Tag	tagm	MicroHelp	tagmListBox	MHEN200.VBX
Timer	Timer	tmrm	MicroHelp	tmrmTimer	MHTI200.VBX
ToolBar	ToolBar	tolm	MicroHelp	tolmToolBar	MHAD200.VBX
List Box	Tree	trem	MicroHelp	tremTree	MHEN200.VBX
Input Box	Input (Text)	txtm	MicroHelp	inpmText	MHEN200.VBX
Input Box	DB_Text	txtp	Pioneer	txtpText	QEVBDBF.VBX
Scroll Bar	Vertical Scroll Bar	vsbm	MicroHelp	vsbmScroll	MHEN200.VBX
Scroll Bar	DB_VScroll	vsbp	Pioneer	vsbpScroll	QEVBDBF.VBX

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgOther RefsDoc

How to Encrypt a String with Password Security in VB

Article ID: Q110308

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

To restrict access by computer hackers, you can encrypt strings, such as strings stored in your compiled executable .EXE program. Encryption helps protect your program from unauthorized modifications.

The sample Visual Basic code below encrypts a string with the XOR operator using password security. You can also adapt this technique to other dialects of Basic or other languages.

MORE INFORMATION

Software tools for debugging and viewing binary code can easily find ASCII strings stored in compiled executable .EXE programs. If you want to hide or protect strings in .EXE programs, you can use techniques such as these:

- Append a series of Chr$() functions in your Basic code. For example, the following string concatenates ASCII values 65 through 68, which represent the capital letters A through D:

  ```
  A$ = Chr$(65) + Chr$(66) + Chr$(67) + Chr$(68)
  Print A$
  ```

  This code prints ABCD. Since the ABCD is stored in code instead of in a string constant, it is not directly visible when viewed in a debugger. This method is okay for small amounts of data, but is inefficient for larger strings.

- Add or subtract a constant or a letter's position value to the ASCII/ANSI value of each character in the string.

  NOTE: Don't exceed the byte value range of 0 to 255. For example, if you add 50 to the extended-ASCII character CHR$(230) and assign it to a string in Basic, you get an "Illegal function call" error.

- Use the Xor function in a formula using a key or password string. Byte values changed with Xor always stay within the byte value range 0 to 255. This technique is flexible and elegant. See the sample program in the next section below.

  NOTE: The Xor function is also known as the exclusive-OR function. An exclusive-OR means A or B, but not both. For example, if A is true, and B is false, then A Xor B is true, but if both A and B are true, then A Xor B is false.

- Use third-party compression and encryption software, such as LZEXE or PKLITE, on the compiled executable EXE file. For secure encryption routines, see the QuickPak Professional Library from Crescent Software, Inc., shown in the Reference section below.

- Keep a checksum to protect against viruses or hackers. You can keep a checksum of various important values in the program, and the program can refuse to run if any checksums have changed.

  NOTE: A checksum is an error-detection scheme that involves creating a sum of the bits in a set of bytes of data, then using that sum to later check for a change in the data.

Example of String Encryption Using Xor Function

Calling the Encrypt routine below encrypts a string using a password. Calling the routine again decrypts the encrypted string.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the following code to the Form Load event:

```
Sub Form_Load ()
   form1.Show  ' Must Show form in Load event before Print is visible.
   secret$ = "This is the string that will be encrypted."
   PassWord$ = "password"

   Call Encrypt(secret$, PassWord$)     'Encrypt the string.
   Print " After encrypting it once: "  'Print the result.
   Print secret$
   Print

   Call Encrypt(secret$, PassWord$)  'A second encryption decrypts it.
   Print "After a second encryption: "
   Print secret$
End Sub
```

3. Add the following Encrypt procedure to the general declarations section:

```
Sub Encrypt (secret$, PassWord$)
   ' secret$ = the string you wish to encrypt or decrypt.
   ' PassWord$ = the password with which to encrypt the string.
   L = Len(PassWord$)
   For X = 1 To Len(secret$)
      Char = Asc(Mid$(PassWord$, (X Mod L) - L * ((X Mod L) = 0), 1))
      Mid$(secret$, X, 1) = Chr$(Asc(Mid$(secret$, X, 1)) Xor Char)
   Next
End Sub
```

4. Start the program, or press the F5 key. Close the form to end the program.

Xor: The Exclusive-OR Operator

The exclusive-OR operator (Xor in the Basic language) performs a logical exclusion on two expressions. For example:

```
Result = expr1 Xor expr2
```

A useful behavior of Xor is that the first expression expr1 is returned without losing any bits when you perform Result Xor expr2. This ability to restore the first expression from the Result combined with the second expression is why the Xor function is useful for encryption.

The Xor operator performs a bit-wise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in the result according to the following truth table:

If bit in expr1 is:	And bit in expr2 is:	The result is:
0	0	0
0	1	1
1	0	1
1	1	0

ASCII and ANSI Character Sets

For a listing of the ASCII and ANSI character sets, see the Help menu in Visual Basic.

American Standard Code for Information Interchange (ASCII) is the 7-bit character set widely used to represent letters and symbols found on a standard United States keyboard. The ASCII character set is the same as the first 128 characters (0 to 127) in the American National Standards Institute (ANSI) character set. The ANSI character set uses all 8 bits in a byte, and includes 256 characters (0 to 255). ANSI characters 128 to 255 are sometimes referred to as the extended-ASCII characters.

REFERENCES

The following company offers encryption software and other products for Basic:

```
Crescent Software, Inc.
11 Bailey Ave
Ridgefield, CT  06877  USA
Contact:  Don Malin  (203) 438-5300
Fax: (203) 431-4626
Ask about the QuickPak Professional Library for Windows.
```

Products from Crescent Software are manufactured independent of Microsoft. Microsoft makes no warranty, implied or otherwise, regarding these products' performance or reliability.

Additional reference words: 2.00 3.00
KBCategory: Prg
KBSubcategory: PrgOther

Searchable Electronic VB Info: VB Help Files & MSDN CD-ROM

Article ID: Q110392

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

The Professional Edition of Visual Basic, versions 2.0 and 3.0, contains the VBKNOWLG.HLP Help file, "Microsoft Visual Basic Knowledge Base Articles." VBKNOWLG.HLP contains a subset of the Visual Basic portion of the Microsoft Knowledge Base that was available at the time that the product shipped.

You can get the complete and most current Microsoft Visual Basic for Windows Knowledge Base in two help files (VB_BUGS.HLP and VB_TIPS.HLP) with or without full-text search. These help files are updated regularly. For information about these Help files and how to obtain them, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID:Q105541
TITLE :How to Get Entire VB KB in 2 Help Files with Full-Text Search

Another source of recent information on Visual Basic is available in the Microsoft Developer Network (MSDN) Compact Disk (CD), and in other sources listed farther below. The MSDN CD is updated quarterly. You can perform full-text searches across all information in the MSDN CD.

MORE INFORMATION

The Microsoft Knowledge Base

The Microsoft Knowledge Base is a primary Microsoft product information source for Microsoft support engineers and customers. This comprehensive article collection contains over 40,000 detailed how-to articles, bug lists, fix lists, documentation errors, and answers to technical support questions. Over 600 Visual Basic articles are available.

The Microsoft Knowledge Base is available and updated weekly through CompuServe, the Internet, GEnie, and Microsoft OnLine. The Microsoft Knowledge Base is also available in the bimonthly Microsoft TechNet CD-ROM disk, and in the quarterly Microsoft Developer Network (MSDN) CD-ROM disk.

VBKNOWLG.HLP Shipped with Visual Basic Version 3.0

The VBKNOWLG.HLP online-Help file ships with the Professional Edition of Visual Basic versions 2.0 and 3.0 for Windows.

You can load VBKNOWLG.HLP by running the "Knowledge Base" icon in your "Visual Basic 3.0" group in the Program Manager for Windows. The Help file title bar says "Microsoft Visual Basic Knowledge Base Articles."

VB_TIPS.HLP and VB_BUGS.HLP Are Available for Download

There are nearly 600 categorized articles in the Visual Basic for Windows collection. The two help files (VB_TIPS.HLP and VB_BUGS.HLP) that hold these articles have been placed in a self-extracting file that you can download from CompuServe, the Internet, or the Microsoft Download Service (MSDL). Choose to download either VBKB_FT.EXE or VBKB.EXE, not both.

The Help files in VBKB_FT.EXE have an additional Find button that allows full-text search. The Help files in VBKB.EXE do not have the Find button and do not allow full-text search. The technical content in VBKB.EXE is identical to that in VBKB_FT.EXE. VBKB.EXE is less than a megabyte in size while VBKB_FT.EXE is approximately 2.5 megabytes. VBKB_FT.EXE is larger because it includes index and .DLL files needed for full-text search.

Download VBKB.EXE or VBKB_FT.EXE (both are self-extracting files) from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL and download VBKB.EXE (without full-text search).

  -or-

  GO MSL and download VBKB_FT.EXE (with full-text search).

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download VBKB.EXE or VBKB_FT.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get VBKB.EXE

  -or-

  Get VBKB_FT.EXE

Microsoft Developer Network (MSDN) CD-ROM Disk

Here's what the MSDN CD contains specifically for Visual Basic:

- Complete electronic copy of the printed documentation that ships with the Professional Edition of Visual Basic.
- Articles from "BasicPro" magazine.
- Articles from the Cobb Group's "Inside Visual Basic" monthly magazine.
- Articles written by the Developer Network Technology Group.

The MSDN CD contains the entire Windows SDK documentation as well as Charles Petzold's "Programming Windows 3.1." This is very useful if you are calling Windows API functions from Visual Basic.

How to Join the Microsoft Developer Network (MSDN)

The Microsoft Developer Network (MSDN) provides technical information and development toolkits for all developers who write applications for Microsoft operating systems. MSDN members receive a quarterly CD-ROM disk and a bimonthly newsletter. The CD contains code samples, technical articles, development tools, and the Microsoft Knowledge Base.

To join the Microsoft Developer Network (MSDN):

- In the U.S. and Canada, call (800) 759-5474; this number is available 24 hours a day, 7 days a week.
- In France, call 05 90 59 04 (toll-free).
- In Germany, call 0130 81 02 1.
- In the Netherlands, call 06 022 24 80 (toll-free).
- In the United Kingdom, call 0800 96 02 79 (toll-free).
- In Japan, call 03 5461-2617.
- For any other country in Europe, call +31 10 258 88 64.
- Outside Europe, the U.S., Canada, or Japan, call (402) 691-0173.

This information was taken from the "Microsoft Developer Network News" newsletter dated January 1994. This information is subject to change.

Additional reference words: 2.00 3.00
KBCategory: Prg
KBSubcategory: PrgOther

How to Remove Menu Items from a Form's Control-Menu Box

Article ID: Q110393

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The Control-menu box is located in the upper-left corner of a Visual Basic form. You can remove certain menu items from the Control-menu box by using the using form's MaxButton, MinButton, and BorderStyle properties. You can also remove Control-menu items by using Windows API functions, as shown in a sample program in the More Information section below.

To completely remove the Control-menu box, set the form's ControlBox property to False.

MORE INFORMATION

The default Control-menu box in the upper left-hand corner of a Visual Basic form contains the following nine entries, including separators:

```
Restore
Move
Size
Minimize
Maximize
----------------------
Close            Alt+F4
----------------------
Switch to...   Ctrl+Esc
```

You can remove certain menu items from the Control-menu box by using a form's MaxButton, MinButton, and BorderStyle properties:

MaxButton property:

Setting the MaxButton property to False at design time removes the Maximize item in the Control-menu box and also removes the Maximize arrow in the upper right corner of the form. Setting MaxButton to be False also prevents a double-click on the title bar from maximizing the form.

MinButton property:

Setting the MinButton property to False at design time removes the Minimize item in the Control-menu box and also removes the minimize arrow in the upper right corner of the form.

BorderStyle property:

Setting	Description
0	No border and no related border elements.
1	Fixed Single. Can include Control-menu box, title bar, Maximize button, and Minimize button. Resizable only using Maximize and Minimize buttons.
2	(Default) Sizable. Resizable using any of the optional border elements listed for setting 1.
3	Fixed Double. Can include Control-menu box and title bar; cannot include Maximize or Minimize buttons. Not resizable.

Example Uses API Functions to Remove Control-Menu Items

The following program invokes Windows API functions to remove all items in the Control-menu box except for Restore and Minimize.

1. Start a new project in Visual Basic. Form1 is created by default.

   NOTE: In this program, the following Form properties should be left with their design-time defaults: ControlBox = True, MaxButton = True, MinButton = True. The API functions will make the necessary changes to the form's properties.

2. Add the following to the Form Load event code:

```
Sub Form_Load ()

   Dim hSysMenu%, r%, j%, dw&, rr&
   Const MF_BYPOSITION = &H400
   ' Me refers to the form where code is currently executing:
   hSysMenu = GetSystemMenu(Me.hWnd, 0)
   For j = 8 To 4 Step -1
      r = RemoveMenu(hSysMenu, j, MF_BYPOSITION)
   Next j
   For j = 2 To 1 Step -1
      r = RemoveMenu(hSysMenu, j, MF_BYPOSITION)
   Next j
   ' Leave the Restore and Minimize items.
   dw& = GetWindowLong(Me.hWnd, -16)   'Window style
   dw& = dw& And &HFFFEFFFF    'Turn off bits for Maximize arrow button
   rr& = SetWindowLong(Me.hWnd, -16, dw&)

End Sub
```

   The default Control-menu items are numbered 0 through 8 from the top down. You may remove any or all items using Windows API functions. Be sure to remove items in reverse sequence, from 8 to 0, or else the numbering will become confused.

3. Add a command button to the form. Double-click the command button and add the following code to the Command1 click event:

```
Sub Command1_Click ()
```

```
      End
   End Sub
```

   This button lets you end the program, since Close is removed from the Control-menu box.

4. Add the following Declare statements to the general declarations section:

```
   ' Enter each of the following Declare statements as one, single line:
   Declare Function RemoveMenu% Lib "User" (ByVal hMenu%, ByVal nPosition%,
      ByVal wFlags%)
   Declare Function GetSystemMenu% Lib "User" (ByVal hWnd%, ByVal revert%)
   Declare Function GetWindowLong Lib "User" (ByVal hWnd As Integer,
      ByVal nIndex As Integer) As Long
   Declare Function SetWindowLong Lib "User" (ByVal hWnd As Integer,
      ByVal nIndex As Integer, ByVal dwNewLong As Long) As Long
```

5. Start the program, or press the F5 key.

The form's Control menu shows Restore (grayed) and Minimize. Double-clicking the title-bar doesn't maximize the form, as desired.

Clicking the Minimize arrow or choosing the Minimize menu item minimizes the form to an icon. A single-click on that icon does not open a control-menu, unlike normal Visual Basic application icons. A double-click is required to restore the form to its full-screen state.

Creating a Form with No Title Bar

To create a Microsoft Visual Basic for Windows form with a border but with no title bar, the Caption property of a form must be set to a zero-length string; the BorderStyle property must be set to Fixed Single (1), Sizable (2) or Fixed Double; and the ControlBox, MaxButton and MinButton properties must be set to False (0).

If any text (including spaces) exists for the Caption property or if the ControlBox, MaxButton, or MinButton property is set to True, a title bar will appear on the form. Note that setting the BorderStyle property to None (0) will always make a form with no title bar.

REFERENCES

- "Microsoft Visual Basic Version 3.0: Programmer's Guide" pages 97-98.

- "PC Magazine's Visual Basic Programmer's Guide to the Windows API" by Daniel Appleman (of Desaware), published by Ziff-Davis Press, pages 414 and 418. This reference describes most Windows API functions that can be used from within Visual Basic.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgOther

PRB: Week Starts Sunday and Ends Saturday for Format Function

Article ID: Q110667

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

For the date of Sunday January 2, 1994, the Format("01/02/94","ww") function returns week number 2 instead of week number 1.

NOTE: Visual Basic handles dates according to the Country settings in the International option of the Windows Control Panel. When the Country is set to United States, dates such as "01/05/94" are by default interpreted with the month followed by the day, as in mm/dd/yy. For most other Country settings, the day precedes the month, and "05/01/94" is interpreted as dd/mm/yy.

CAUSE

In the Format function, weeks start on a Sunday and go through the following Saturday. In 1994, January 1 is a Saturday, which is the only day in week 1. Week 2 of 1994 starts on Sunday January 2.

Therefore, the Format("01/01/94","ww") function returns 1, and Format("01/02/94","ww") returns 2.

STATUS

This behavior is by design.

MORE INFORMATION

Below are some of the format expressions for the day, week, month, and quarter, as supported by the Format function:

dd	Displays the day as a number with a leading zero (01-31).
ddd	Displays the day as an abbreviation (Sun-Sat).
dddd	Displays the day as a full name (Sunday-Saturday).
w	Displays the day of the week as a number (1 for Sunday through 7 for Saturday.)
ww	Displays the week of the year as a number (1-53).
m	Displays the month as a number without a leading zero (1-12). If m

immediately follows h or hh, the minute rather than the month is displayed.

mm Displays the month as a number with a leading zero (01-12). If m immediately follows h or hh, the minute rather than the month is displayed.

mmm Displays the month as an abbreviation (Jan-Dec).

mmmm Displays the month as a full month name (January-December).

q Displays the quarter of the year as a number (1-4).

For more information, see the Format and Format$ function topics in the Visual Basic Help menu.

Steps to Reproduce Behavior
---

1. Start a new project in Visual Basic. Form1 is created by default.

2. Double-click the form. Add the following to the Form Load event code:

```
Sub Form_Load ()
   form1.Show
   Print Format("01/01/94", "ww dddd")
   Print Format("01/05/94", "ww dddd")
End Sub
```

3. Start the program, or press the F5 key to see the following print:

```
   1 Saturday
   2 Wednesday
```

   The Format function correctly shows Wednesday January 5 as being in week 2 of 1994. Close the form to end the program.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgOther

How to Get a Handle to MS-DOS Application and Change Title

Article ID: Q110701

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article shows by example how to get the handle to a MS-DOS application, and then use that handle to change the MS-DOS Window title or automatically unload the MS-DOS Window from Visual Basic.

MORE INFORMATION

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add two command buttons (Command1 and Command2) to Form1.

3. Enter the following code in the General Declarations of a form or module:

```
' Enter each of the following Declare statements on one, single line:
Declare Function PostMessage Lib "User"
   (ByVal hWnd As Integer, ByVal wMsg As Integer,
   ByVal wParam As Integer, lParam As Any) As Integer
Declare Sub SetWindowText Lib "User"
   (ByVal hWnd As Integer, ByVal lpString As String)
Declare Function GetActiveWindow Lib "User" () As Integer
Dim MhWnd as Integer
```

4. Enter the following code in the Click Event of Command1:

```
Sub Command1_Click()
   Dim X as Integer
   X = Shell("c:\windows\dosprmpt.pif", 1) ' Open an MS-DOS Window
   For X = 0 To 100
      DoEvents
      ' a bunch of DOEVENTS to wait for the MS-DOS Window to open.
   Next X

   ' Get the handle when the MS-DOS Window has the focus:
   Mhwnd = GetActiveWindow()

   ' Now pass the handle to the window with a new title:
   Call SetWindowText(Mhwnd, "My Application!")
End Sub
```

5. Place the following code in the Click Event of Command2:

```
Sub Command2_Click()
   Dim X as Integer
   ' Note: In order for this to work on a MS-DOS Window, you have
   ' to have a PIF setup that will allow an MS-DOS Window to be closed.
   ' In the PIF editor, select "Advanced", then click "Close When
   ' Active". This allows MS-DOS applications to be closed
   ' programatically
   X = PostMessage(MhWnd, WM_CLOSE, 0, 0) ' Return greater than zero
                                          ' if successful.
   If X > 0 Then
      MsgBox "Application did not close!"
   End If
End Sub
```

6. Start the program, or press the F5 key.

7. Click the Command1 button to see the title change.

Additional reference words: 2.00 3.00
KBCategory: Prg
KBSubcategory: PrgOther

Example of NPV and IRR Financial Functions in VB for Windows

Article ID: Q110888

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

This article explains the NPV and IRR financial functions and gives a sample program.

MORE INFORMATION

The NPV, IRR, and MIRR functions are used for investments that are a series of nonconstant cash payments made at equal intervals. You pass the series of nonconstant payments in an array.

The nonconstant-payment functions (NPV, IRR, and MIRR) are in a different category than the financial functions for annuity investments (FV, IPmt, Rate, NPer, PV, Pmt, and PPmt). In an annuity, each cash payment is the same constant amount, made at equal intervals.

The present value (PV) of a future cash receipt is the amount of money that, if received today, would be considered equivalent to the future receipt, at a given interest rate. The present value is less than the future receipt because you can earn interest on money received today. NPV (Net Present Value) compares (subtracts) the current value of a series of future cash flows with an amount invested today.

NPV is useful to compare investment opportunities at a given discount (interest) rate. The discount rate (found with the Rate function) can be viewed as the rate of return you want out of your investment. If NPV is greater than or equal to 0, the investment equals or exceeds your interest (discount) rate requirement; if NPV is less than 0, the investment does not meet your interest rate requirement.

The IRR function returns Internal Rate of Return. IRR returns the discount rate at which NPV would return 0 (zero). For a given array of cash flow values, IRR can be thought of as an average interest rate (which compounds at each period). If IRR is lower than the interest rate you desire for this investment, then it is not a good investment.

The first element of the input cash-flow array should usually be negative, indicating your initial investment. A high (positive) income early in the value array will make IRR higher than if the same high income instead occurred later in the array. This is an example of the time value of money.

Please refer to an Accounting textbook for more information about these standard Accounting functions.

Step-by-Step Example with Code
---

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the following code to the General Declarations section of Form1:

```
DefDbl A-Z
' Enter each Declare statement on one, single line:
Declare Function NPVC Lib "MSAFINx.DLL" (ByVal Rate1#, values#,
   ByVal cvalues%) As Double
Declare Function IRRC Lib "MSAFINx.DLL" (values#, ByVal cvalues%,
   ByVal Guess#) As Double

Function IRR (values() As Double, ByVal Guess As Double) As Double
   On Error GoTo IrrErr
   iArgMin% = LBound(values)
   cArg% = UBound(values) - iArgMin%
   IRR = IRRC#(values(iArgMin%), cArg%, Guess)
   Exit Function
IrrErr:
   MsgBox "Error " & (Str$(Err))
   Exit Function
End Function

Function NPV (ByVal Rate1 As Double, values() As Double) As Double
   On Error GoTo NpvErr
   iArgMin% = LBound(values)
   cArg% = UBound(values) - iArgMin%
   NPV = NPVC#(Rate1, values(iArgMin%), cArg%)
   Exit Function
NpvErr:
   MsgBox "Error " & (Str$(Err))
   Exit Function
End Function
```

3. Add the following code to the form Load event:

```
Sub Form_Load ()

   form1.Show  ' Must Show form in Load event for Print to work.
   ' Array holds cash flow values, one value per period (such as year):
   ReDim valuearray(5) As Double
   guess = .05        ' Guess the IRR (use .1 if in doubt).

   valuearray(0) = -70000 ' 0. First value negative initial investment
   valuearray(1) = 22000  ' 1. Return on investment after 1 period.
   valuearray(2) = 25000  ' 2. (Pos value is return on investment.)
   valuearray(3) = 28000  ' 3. (Neg value is additional investment.)
   valuearray(4) = 31000  ' 4. Return on investment after 4 periods.
   ' For the above values, IRR returns 17.7% return per period

   irreturn = IRR(valuearray(), guess)

   discountrate = 0  ' If discountrate = 0,
                     ' NPV returns sum of valuearray().
   netpresval = NPV(discountrate, valuearray())
```

```
      ' Notes for NPV() function:
      ' If discountrate = value returned by IRR(), then NPV returns zero.
      ' If discountrate = zero, NPV returns sum of values in valuearray().
      ' If discountrate > zero, NPV returns an amount smaller than sum of
      ' values in valuearray() due to the discount effect at each period.
      ' If discountrate < zero, NPV returns an amount larger than the sum
      ' of the values in valuearray().

      Print "IRR (fractional return on investment per period) = "; irreturn
      Print "NPV = "; Format$(netpresval, "Standard")

   End Sub
```

4. Press the F5 key to run the program.

For the above values, IRR returns .177 (17.7% return per period).

NOTE: By definition, IRR returns the discount rate at which NPV returns 0.

NPV, IRR, MIRR: Reference to Undefined Function or Array

For more information, see the following article in the Microsoft Knowledge Base:

   ARTICLE-ID: Q101245
   TITLE   : BUG: Ref to NPV / IRR / MIRR Gives Undefined Functions Error

If you try to run an application that contains a reference to the NPV, IRR, or MIRR financial function, Visual Basic for Windows generates this error:

   Reference to undefined Function or Array

Visual Basic does not recognized these as Visual Basic functions because they were incorrectly referenced in the financial DLL file (MSAFINX.DLL) that ships with Visual Basic version 3.0.

To work around the problem, you can declare the NPVC, IRRC, and MIRRC functions located in MSAFINX.DLL and alias them as NPV, IRR, and MIRR respectively, as shown farther above.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgOther

How to Mimic HIWORD, LOWORD, HIBYTE, LOBYTE C Macros in VB

Article ID: Q112651

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, versions 2.0 and 3.0

SUMMARY

Visual Basic does not provide any bitwise functions for pulling apart numeric values. In C, there are macros (HIBYTE, LOBYTE, HIWORD, and LOWORD) to separate parts of long integers into two parts, or separate integers into two parts. This article shows by example how to do the same thing in a Visual Basic program.

MORE INFORMATION

The HIBYTE, LOBYTE, HIWORD, and LOWORD macros are defined in C in the WINDOWS.H file. The HIBYTE and LOBYTE macros are used to retrieve the high-order or low-order byte of the integer passed to them. The HIWORD and LOWORD macros retrieve the high-order or low-order word from a long integer value passed to them.

Step-by-Step Example

This example uses Visual Basic to mimic the HIBYTE, LOBYTE, HIWORD, and LOWORD macros

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add two command buttons (Command1 and Command2) to Form1.

3. Add the following code to the Comamnd1_Click event:

```
Sub Command1_Click ()
   Dim wParam As Integer
   Dim LOBYTE As Integer
   Dim HIBYTE As Integer
   ' Set wParam to a value:
   wParam = &H77FF
   ' Make call to function:
   ret = gethilobyte(wParam, LOBYTE, HIBYTE)
   ' Print out return values:
   Print LOBYTE, HIBYTE
End Sub
```

4. Add the following code to the Comamnd2_Click event:

```
Sub Command2_Click ()
   Dim lParam As Long
```

```
   Dim LOWORD As Long
   Dim HIWORD As Long
   ' Set lParam to a value:
   lParam = &H7777FFFF
   ' Make call to function:
   ret = gethiloword(lParam, LOWORD, HIWORD)
   ' Print out return values:
   Print LOWORD, HIWORD
End Sub
```

5. Add the following code to the general declarations section of Form1:

```
' Enter the following Function statement as one, single line:
Function gethilobyte(wparam as integer, LOWORD as integer,
   HIWORD as integer)
   ' This is the LOBYTE of the wParam:
   LOBYTE = wParam And  &HFF&
   ' This is the HIBYTE of the wParam:
   HIBYTE = lParam  \  &H100  And  &HFF&
   gethilobyte = 1
End Function
```

6. Add the following code to the general declarations section of Form1:

```
Function gethiloword(lparam as long, LOWORD as long, HIWORD as long)
   ' This is the LOWORD of the lParam:
   LOWORD = lParam And  &HFFFF&
   ' LOWORD now equals 65,535 or &HFFFF
   ' This is the HIWORD of the lParam:
   HIWORD = lParam  \  &H10000  And  &HFFFF&
   ' HIWORD now equals 30,583 or &H7777
   gethiloword = 1
End Function
```

7. Run the program. Click the Command1 button to pull apart an integer value into its high-order and low-order bytes. Click the Command2 button to pull apart a long integer into its high-order and low-order words.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgOther

How to Determine If a File Exists by Using DIR$

Article ID: Q112674

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

Visual Basic does not have any built-in functions that tell if a file exists or not. This article demonstrates how to find out if a file exists or not by using a Visual Basic program.

MORE INFORMATION

There are two different methods you can use to determine if a file exists:

- Use the DIR/DIR$ command to determine if a file exists. This method is shown in the example below. The example performs a DIR on the filename and checks the return value. If the return value is "" (nothing), then the file doesn't exist.

- Use the OPEN statement to open a file for input and an error trap. This method can run into problems when dealing with SHARE in Windows. If the file does not exist, a trappable error occurs. The only problem with this method is if the file that is being opened with the OPEN statement is in use, you will generate a sharing violation, which is a system level error that is not trappable from Visual Basic.

Step-by-Step Example

This example show how to check for the existence of a file by using the DIR$ function.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a command button (Command1) to Form1.

3. Place the following code in the Command1_Click Event:

```
Sub Command1_Click()
   Dim TheFile as String
   Dim Results as String

   TheFile = "C:\AUTOEXEC.BAT"
   Results = Dir$(TheFile)

   If Results  = "" Then
      MsgBox "File Doesn't Exist!"
   Else
```

```
         MsgBox "File does Exist!"
      End If
   End Sub
```

4. Run the program, and click the Command1 button.

5. Stop the program and make the TheFile variable point to a file that doesn't exist.

6. Run the program again, and click the Command1 button.

REFERENCES
==========

For more information, see the Programmer's Reference, File Manipulation and the DIR/DIR$ commands.

Additional reference words: 1.00 2.00 3.00
KBCategory: Prg
KBSubcategory: PrgOther

How To Seek a CD Track by Using the MCI Control

Article ID: Q112732

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

The MCI Control supports seeking a certain Track by using the Commmand and To properties. In order to seek to a specific track, specify which track in the TO property of the MCI Control. Then, using the Command property, perform the seek. This article gives an example.

MORE INFORMATION

The To property of the MCI Control will allow the seek to move to another position based on the current time format. If you set the TimeFormat property to MCI_FORMAT_TMSF (10), it will allow you to move to the beginning of specific tracks.

Step-by-Step Example

The following example shows how to seek through all tracks on a CD and play them for five seconds with the MCI Control.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add an MCI Control (MMControl1) to Form1.

3. Place the following code in the Form_Load event of Form1:

```
Sub Form_Load()
   MMControl1.Device = "CDAudio"
   MMControl1.Command = "Open"
End Sub
```

4. Add the following code to the Command1_Click event:

```
Sub Command1_Click()
   ' Set the timeformat to allow seek to move between tracks:
   mmcontrol1.TimeFormat = 10
   ' Loop through all tracks and play each for five seconds:
   For i = 1 To MMControl1.Tracks
      MMControl1.To = Str$(i)
      MMControl1.Command = "Seek"
      MMControl1.Command = "Play"
      x = Timer
      While Timer < x + 5
         DoEvents
```

```
         Wend
      Next
      ' Stop the CD:
      MMControl1.Command = "pause"
   End Sub
```

5. Run the program. The MCI Control should start playing Track 2

NOTE: Before starting this program, a CD must be inserted and ready to play. Therefore, you should add code to detect when a CD has been inserted and then run the above code.

REFERENCES
==========

For more information, see the Multimedia Programmer's Reference, CDAudio.

Additional reference words: 2.00 3.00
KBCategory: Prg
KBSubcategory: PrgOther

How To Get the Total Playing Time of an Audio CD

Article ID: Q112766

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The MultiMedia Application Programming Interface (API) and the MCI Control have built-in methods that retrieve the total playing time of a CD. The Length property of the MCI Control will retrieve total playing time of the current media.

MORE INFORMATION

After setting the MCI Control's TimeFormat property to MCI_FORMAT_TMSF(10) and summing the values returned by the DateAdd function to add the total time values for minutes and seconds together, you can calculate a CD's total playing time.

DateAdd starts adding from 12:00:00 AM, so you need to check to see if the total time is less than one hour. If it is, strip off the first three characters (12:) and the AM. If the time is greater than one hour, strip off the AM so that the return value looks like this:

1:09:43

 - or -

57:45

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a text box (Text1) and an MCI Control (MMControl1) to Form1.

3. Place the following code in the Form_Load event of Form1:

```
Sub Form_Load ()
    ' Initialize the CD:
    mmcontrol1.TimeFormat = 10
    mmcontrol1.DeviceType = "CDAudio"
    mmcontrol1.Command = "Open"
    ' Dimension variables:
    Dim Length As Variant
    Dim CDSeconds As Integer, CDMinutes As Integer
    ' Calculate minutes and seconds:
    CDSeconds = CDSeconds + (mmcontrol1.Length And &HFF00&) / &H100
```

```
      CDMinutes = CDMinutes + (mmcontrol1.Length And &HFF)
      ' Sum minutes and seconds:
      Length = DateAdd("s", CDSeconds, Length) ' Add Seconds
      Length = DateAdd("n", CDMinutes, Length) ' Add Minutes
      ' Determine if total running time is less than one hour:
      If (Left$(Length, 2)) = "12" Then ' Less than 1 hour
         Text1.Text = Mid(Length, 4, (Len(Length) - 4))
      Else ' Greater than 1 hour:
         Text1.Text = Left(Length, (Len(Length) - 3))
      End If
   End Sub
```

4. Run the program. You should see the total playing displayed in the Text1 box.

NOTE: Before starting this program, a CD must be inserted and ready to play. Therefore, you should add code to detect when a CD has been inserted and then run the above code.

REFERENCES
==========

For more information, see the Multimedia Programmer's Reference, CDAudio

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgOther

How to Get or Create a Unique Audio CD Volume Label

Article ID: Q112768

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

Part of the Multimedia standard calls for all Audio CDs to have a unique volume identifier following a specific (suggested) format of:

```
XXXX-#####
```

XXXX is the alpha vendor code, and ##### is a five-digit unique number for the CD. This information is stored on the inner track of the CD and is also usually etched on the inner edge of the inside ring of the CD. Not all manufacturers have a unique volume identifier, nor do all follow this standard volume label format. Although this is a standard, at this time the MultiMedia Application Programming Interface (API) does not have a built-in function that will retrieve this information. This article shows you how to retrieve or create this information programmatically.

MORE INFORMATION

In order to retrieve the unique volume identifier from the CD, you need to call MSCDEX directly. The information on how to do this is contained within the MSCDEX 2.20 specification.

Because some manufacturers aren't including this unique identifier on their CDs, Microsoft recommends that you create a unique volume identifying number based on the track information already included on the CD. The following example shows you how.

Step-by-Step Example

This example shows a method for creating a unique number to identify an audio CD based on track information. The purpose of this number is to make it possible for Visual Basic programs to recognize a loaded CD and retrieve information from it.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a text box (Text1) and an MCI control (MMControl1) to Form1.

3. Place the following code in the Form_Load event of Form1:

```
Sub Form_Load ()
   Dim DiskID As Long
   Dim Track As Integer
```

```
      ' Initialize CD:
      mmcontrol1.DeviceType = "CDAudio"
      mmcontrol1.Command = "open"
      ' Make unique number based on tracks and tracklength:
      DiskID = mmcontrol1.Tracks
      For Track = 1 To mmcontrol1.Tracks
         mmcontrol1.Track = Track
         DiskID = DiskID + mmcontrol1.TrackLength  ' Add 4-byte TrackLength
         DiskID = DiskID + mmcontrol1.Length       ' Add 4-byte CD Length
      Next Track
      ' Set text to unique value:
      Text1.Text = DiskID
   End Sub
```

4. Load a CD.

5. Start the program, or press the F5 key.

6. The Text1 box should have a unique identifying number for the CD.

NOTE: Before starting the program, a CD must be inserted and ready to play. Therefore, you should add code to detect when a CD has been inserted and then run the above code.

REFERENCES
==========

For more information, see the MultiMedia Programmer's Reference, CDAudio

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgOther

General Memory Management in Visual Basic Vers 3.0 for Windows

Article ID: Q112860

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article outlines how various memory areas in Visual Basic are managed. It covers the following areas:

- General Memory Management:
  - Windows Limits
  - Application Limits
  - Form Limits
  - Data Limits
  - Global Name Table
  - Global Symbol Table
  - Global Data Segment
  - Module Name Table
  - Module Symbol Table
  - Module Data Segment
  - Module Code Segment
  - System Resources
  - Stack Space
  - When Code and Resources are Loaded/Unloaded
- "Out of Memory" and Related Error Messages:
  - Common Ways to Avoid "Out of Memory"
  - Common Ways to Diagnose "Out of Memory"
  - Verifying a Memory Leak

MORE INFORMATION

GENERAL MEMORY MANAGEMENT

Windows Limits

- Windows version 3.1 imposes a limit of 600 windows in existence at one time. All windows in all applications running in Windows count toward this limit. In Visual Basic, each form and non-graphical control (all controls except shape, line, image, and label) counts as one window.

Application Limits

- Maximum of 256 distinct objects in a project

- Maximum of approximately 230 forms in a project, with up to 80 loaded at one time

- The total count of all procedures, form and code modules, and DLL declarations must be less than 5,200 -- not counting event procedures that are empty of code.

Form Limits

- Maximum of 470 controls on a single form depending on control type.

- Maximum of 254 different control names per form. A control array counts as one name.

- The data in all form properties and all control properties on a given form is stored in a single data segment limited to 64K, except for the List property of combo and list box controls and the Text property of multiline text box controls.

Data Limits

- Specific data limits for Visual Basic version 3.0 are well documented in Appendix D of the Microsoft Visual Basic "Programmer's Guide." Please look there for information on data limits.

Global Name Table

Each application uses a single Global name table (up to 64K in size) that contains all global names. Global names include:

- The actual text of the Form or Code module name.

- The actual text of each event procedure name goes in the global symbol table only once. For example, two different forms each have a Form_Load event procedure, but only one entry for the event procedure is made in the global symbol table.

- The actual text of each non-event Sub or Function procedure name in a form module (.FRM file). Even though these Sub or Function procedures are private to the form, within the application, their names are listed globally but with a flag set to indicate that they are private to that form.

- The actual text of each non-event Sub or Function procedure name in a code module (.BAS file). As with Sub or Function procedure names in a form module, the code module Sub and Function procedure names are listed globally but with a flag set to indicate that they are global to the application -- unless the Private key word is used, in which case the flag is set to indicate that they are private to that code module.

- The actual text for the name of each Global constant.

- The actual text for the name of each Global variable.

- The actual text for the name of each user-defined type definition.

- The actual text for the name of each Global DLL Sub or Function procedure declaration.

- Four bytes of overhead for each of the above listed global names in the global name table as well as approximately 100 bytes of overhead for the hash table for these Global names.

NOTE: When your project is made into an .EXE file, the Global name table is not needed. It is needed only within the Visual Basic environment to create the .EXE file. However, in some special cases, some global symbols (such as DLL function names) may be required in the .EXE file.

Global Symbol Table

Each application has a single Global symbol table that is up to 64K in size. This table contains descriptive information about each of the items named in the Global name table. Specifically, this table contains:

- Type definitions: 4 bytes for the type + 4 bytes for each element.

- Type variables: 12 bytes

- Fixed String Variables: 12 bytes

- Object variables: 8 bytes (approximately)

- Everything else: 10 bytes (approximately)

- In addition, there is overhead (approximately 100 bytes) for the hash table within the Global symbol table.

NOTE: As with the Global name table, when your project is made into an .EXE file, the Global symbol table is not needed. It is only needed within the Visual Basic environment to create the .EXE file itself. However in some special cases (such as names of DLL functions), some global symbol information may be required in the .EXE itself.

Global Data Segment

Each application has a single 64K Global data segment that contains all global data for all global variables and global constants.

Module Name Table

Each form and code module has a single Module name table that is up to 64K in size. The Module name table includes:

- The actual text for the name of each module-level and local variable name.

- The actual text for the name of each module-level DLL Sub or Function procedure declaration.

- The actual text used for line numbers and line labels.

- As with the global name table, there is an additional four-byte overhead for each of the above listed module-level names in the Module name table as well as about 100 bytes overhead for the hash table for these module-level names.

NOTE: As with the Global name table, when your project is made into an .EXE file, most of the Module name table is not needed.

Module Symbol Table

Each form and code module has a single Module symbol table that is up to 64K in size. The module symbol table contains much of the same information as the Global symbol table, except for Type definitions.

NOTE: As with the Module symbol table, when your project is made into an .EXE file most of the Module name table is not needed.

Module Data Segment

Each form and code module has their own 64K module data segment. The contents of the module data segment includes:

- Local variables declared with the Static key word.

- Local variables declared within a Static Sub or Function procedure.

- Module-level fixed-length string variables.

- Module-level variables other than arrays and variable-length strings.

- Module-level constants.

- Tracking data for arrays and variable-length strings (two-byte pointer each).

- Tracking data for controls referenced in code for a form (two-byte pointer each).

- Tracking data for non-static local variables (two-byte pointer each).

Module Code Segment

Each form and code module has one 64K code segment. This code segment contains the p-code (the internal representation of your code) for all procedure code and module level declarations in a given form or code module. The amount of p-code in a procedure or the declarations section of a module is roughly equivalent to the number of ASCII text characters in the code. The Visual Basic documentation recommends that you save your forms and modules as ASCII text and that you keep the size of each of the

form and code modules in your application to under 64K. However, practical experience by Visual Basic programmers has shown it is best to keep the size of these files to under about 45K to 50K in size.

System Resources

The data space (heap) attached to the Windows libraries USER.EXE and GDI.EXE is the space allocated for system resources. The data space used by the GDI library contains graphics information regarding brushes, fonts, icons, and so on. The data space used by the USER library contains style information pertaining to windows (forms) and controls. The data space for each of these libraries is limited to 64K. The "About Program Manager" menu option displays, as a percentage, the lower of these two areas of memory. If you run out of either of these two heaps you will receive an "Out of Memory" error message.

Visual Basic uses Windows resources shared with other Windows applications, so it's important to make efficient use of these resources to avoid "Out of memory" messages. For larger applications, you should reduce the number of forms and controls loaded at any one time. Forms should be loaded only when needed. Here are some ways to reduce the number of controls:

- Use the Visual Basic Load and Unload statements to load forms only when needed at run time.

- Use so called light-weight controls (labels, images, lines, and shapes in Visual Basic version 3.0) when possible. These controls are similar to other controls in that they do use system resources when they are being drawn or updated in the screen. But unlike other controls, the light-weight controls release resources back to the system when finished.

- Replace controls such as command buttons that react to mouse events, with an image control that has a picture of a control in it. This provides your application with similar functionality but uses less memory (system resources).

- Try to replace several related controls with a single, larger control. For example, if several text boxes were placed on a form to represent the cells of a spreadsheet, you could replace these controls with a single picture control containing an image of the cells -- lines drawn vertically and horizontally. Hit testing could be performed at run time from within the MouseDown event of the picture control. In other words, you could use the x and y coordinates passed to the MouseDown event to determine which "cell" was clicked (hit). The cell image on the picture control could be temporarily replaced by a real text box -- positioned by using the Move method -- to allow for user input. In this manner, only two controls would be required to simulate input on a spreadsheet, a picture control and a text box.

Stack Space

Each Visual Basic application uses a single stack limited to 20K in size. The 20K size cannot be changed, so an "Out of Stack Space" error can easily occur if your program performs uncontrolled recursion, such as a cascading

event.

Arguments and local variables in Sub and Function procedures take up stack space at runtime. However, global, module-level, and static variables don't take up stack space because they're allocated in the Module Data Segment for form and code modules.

When Code and Resources Are Loaded or Unloaded

When an application is executed, memory is allocated for the Global Data Segment and the appropriate data is loaded into it. This stays persistent until the application terminates.

When a form is loaded, system resources are allocated for the Form and all controls on that form. In addition Memory is allocated for the Module data segment and the appropriate data is loaded into it.

When an event, Sub, or Function procedure is called, the code for that Form or code module is loaded. Note that once code for a code module is loaded into memory, the code remains in memory until the application terminates.

When a form is unloaded, the code for that form is unloaded, as well as all resources used by that form and the controls on that form. However, when a Form is unloaded the Module Data Segment is not unloaded. It remains in memory until the application terminates. This means that if you load Form2, change the values of some module level variables, unload Form2, and then load Form2 again, the value you last assigned to the module level variable in Form2 will still be present. In order to deallocate memory used by the Module data segment in the form, use the following line of code after you unload the form:

```
Unload Form2
Set Form2 = Nothing
```

This will clear all data in the Module code segment for the form.

"OUT OF MEMORY" AND RELATED ERROR MESSAGES

Common Ways to Avoid "Out of Memory"

Here are some tips to help you avoid the "Out of Memory" error:

- Read Appendix D (Specifications and Limits) of the "Programmers Guide." The discussion of the Global Symbol Table in the manual is enhanced by details presented below.

- Read Chapter 11, "Optimizing Your Applications for Size and Speed," in the "Programmer's Guide." This offers a good starting place in its discussion of optimizing your applications memory requirements.

- Keep the number of forms loaded at any given time to as few as possible. While the specifications indicate that 80 forms can be loaded into memory, in practice you will want to avoid pushing this upper limit.

- If possible, take advantage of non-graphical controls (shape, line, image, and label). These controls use less system resources as well as memory.

- Use control arrays if possible rather than a large number of static controls of the same type.

- Minimize the number of controls on a form. The upper limit of controls on a form is 470, however they can use only 254 control names (making control arrays necessary). In practice, a form this heavily laden with controls would be slow in performance. Minimize the number and type of custom controls on a given form.

- Minimize the size of the names used in event, private, and global Sub and Function procedures. See the "Global Name Table" discussion above for details.

- Save your forms and modules as text. See the "Module Code Segment" discussion above for details.

- Ensure that the declaration for any API or other DLL calls you make are correct. Ensure that memory allocated by routines inside a DLL is actually deallocated when the process using it is finished.

- Explicitly turn off timer controls when your application exits. If you do not, it is possible the parent form will not be unloaded, and the timer will continue firing.

- Explicitly unload forms -- especially when the application terminates. The End statement does not trigger the Unload event of a form. The only way to ensure that a form is actually unloaded is to issue the Unload statement. The Forms Collection is useful for being certain that all forms, loaded or not, are unloaded.

Tips for Diagnosing "Out of Memory" Problems

When you receive an "Out of Memory" error, there may be few limits to the lengths you will have to go to diagnose and locate the actual source of the problem. Make a backup of your project and place it in a temporary directory. Take any drastic measures on that backup copy. The following are some tips to help you narrow the focus in finding the cause of the "Out of Memory" error:

- Remove any and all unnecessary device drivers from the CONFIG.SYS file. Is there any particular problem that seems to induce the error?

- Remove any TSR (terminate and stay resident) programs from the AUTOEXEC.BAT file. Again, is there any particular TSR that induces the error?

- Use Program Manager as the Windows desktop.

- Do not run anything else in Windows that is not necessary. Check this by bringing up the task list (CTRL-ESC). Are Visual Basic and the Program Manager the only tasks running?

- Change the Windows Video driver to standard VGA. It may be necessary to check if your display's vendor has released an update to the display driver. There may an incompatibility with your video driver and Visual Basic causing video memory to be incorrectly allocated or referenced.

- Change any API or DLL calls you are making into comments by adding a single quotation mark as the first character on the line. This may not be conclusive, as the error can be generated as a result of the values returned by the calls, and not the calls themselves. In that case, hard code the return values, or provide some mechanism to simulate their action so you can test the code that responds to them.

- Remove any custom controls you are using from the form.

Verifying a Memory Leak

Memory leaks can occur when memory is allocated to perform a particular process, but is not deallocated when the process concludes.

You can use a tool such as HeapWalker from the Windows Software Development kit or Visual C/C++. HeapWalker allows you to analyze your application's memory requirements and memory profile by seeing how many code segments from your application are in memory at any given time.

The Windows API GetFreeSpace() function scans the global heap and returns the number of bytes of memory currently available. However, use of this function in a large application can be misleading. The best way to use this function is to create a demonstration program that tracks memory before and after a given process. It is important to minimize the scope of that process as much as possible or else the results will be inconclusive.

REFERENCES

- Visual Basic version 3.0 for Windows "Programmer's Guide," Appendix D, "Specifications and Limitations," pages 641-646.

- Visual Basic version 3.0 for Windows "Programmer's Guide," Chapter 11, "Optimizing your Application for Size and Speed," pages 255-263.

- Microsoft Windows Software Development Kit "Programmer's Reference," Volume 2, "Functions."

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgOptMemMgt

PRB: Error When Assign DB Value to Var: Invalid Use of Null

Article ID: Q113032

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic Programming System for Windows, version 3.0

SYMPTOMS

It is possible to receive the error message "Invalid Use of Null" when assigning data from an empty field in a database to a variable or control. This article explains why this error occurs and how to work around it.

CAUSE

The variant data type can hold several types of data. It can also be Null or Empty. It is important to distinguish between Null and Empty. A Null variant contains no valid data, while an Empty variant has not been initialized.

When an Empty variant is assigned to a string it is converted to "". When an Empty variant is assigned to a numeric it is converted to 0. The Null variant on the other hand has no valid data, so it cannot be assigned to a string or a numeric. Trying to assign a Null variant generates the "Invalid Use of Null" error. The following example demonstrates this behavior.

```
Dim a as Variant
Dim b as Integer
a = Null
b = a
```

Some properties, functions, and methods also return Null. An obvious example is the Null function. To avoid the "Invalid Use of Null" error, don't assign a function or method that returns Null to a string or numeric. The following example demonstrates this behavior:

```
Dim b as Integer
b = Null
```

The "Invalid Use of Null" error can also occur when assigning a value to a string or numeric property of a control. The text property of a text box is a string property. The following example shows how the "Invalid Use of Null" error can occur with a text box.

```
Text1.Text = Null
```

In database programming, you may also receive the "Invalid Use of Null" error when assigning the value of a field to a text box. This happens because the Value property returns Null when the field contains no valid data. Here's an example that demonstrates this:

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a data control (Data1) and a text box (Text1) to the form.

3. Add the following code to the Form_Load event:

```
   Sub Form_Load()
      Data1.DatabaseName = "C:\VB\BIBLIO.MDB"
      Data1.RecordSource = "Authors"
      Data1.Refresh
      Data1.Recordset.AddNew
      Data1.Recordset.Update
      Data1.Recordset.Bookmark = Data1.Recordset.LastModified
      Text1.Text = Data1.Recordset("Author")
   End Sub
```

4. Run the program.

WORKAROUND
==========

Visual Basic provides two mechanisms for working around the error.

 - The IsNull() function method.
 - The ampersand (&) concatenation method.

The IsNull() Function Method

The IsNull() function allows you to detect Null. Here's how you could use IsNull() in a database program:

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a data control (Data1) and a text box (Text1) to the form.

3. Add the following code to the Form_Load event:

```
   Sub Form_Load()
      Data1.DatabaseName = "C:\VB\BIBLIO.MDB"
      Data1.RecordSource = "Authors"
      Data1.Refresh
      Data1.Recordset.AddNew
      Data1.Recordset.Update
      Data1.Recordset.Bookmark = Data1.Recordset.LastModified
      If IsNull(Data1.RecordSet("Author")) Then
         Text1.Text=""
      Else
         Text1.Text = Data1.Recordset("Author")
      End If
   End Sub
```

4. Run the program.

The Ampersand (&) Concatenation Method

The other method is to take advantage of Visual Basic's string concatenation operator -- the ampersand (&). If one of the arguments in a concatenation is valid and the other is Null, a concatenation will convert the null value to "". You can take advantage of this behavior when assigning values that might return Null. When concatenating a valid string with a value that could return null, the result will always be a valid string. Here's an example that uses string concatenation:

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a data control (Data1) and a text box (Text1) to the form.

3. Add the following code to the Form_Load event:

```
Sub Form_Load()
   Data1.DatabaseName = "C:\VB\BIBLIO.MDB"
   Data1.RecordSource = "Authors"
   Data1.Refresh
   Data1.Recordset.AddNew
   Data1.Recordset.Update
   Data1.Recordset.Bookmark = Data1.Recordset.LastModified
   Text1.Text = "" & Data1.Recordset("Author")
End Sub
```

4. Run the program.

Additional reference words: 3.00
KBCategory: Prg
KBSubcategory: PrgOther

How to Clear a VB List Box with a Windows API Function

Article ID: Q71069

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Customers often ask how to quickly clear the contents of a list box without clearing one item at a time. The following article shows how to instantly clear the contents of a list box in Visual Basic version 2.0 by sending the list box a LB_RESETCONTENT message.

The Clear method was introduced in Visual Basic version 3.0. You can use the Clear method to clear the contents of a list box or combo box, or to clear the contents of the Clipboard.

MORE INFORMATION

Clearing a List Box in Visual Basic Version 3.0

In version 3.0, you can use the Clear method to clear all items from a list box. The following example shows you how.

Version 3.0 Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a List box (List1) to Form1.

3. Add the following code to the Declarations section of Form1.

```
Sub Form_Click ()

   Dim Entry, I, Msg    ' Declare variables
   Msg = "Choose OK to add 100 items to your list box."
   MsgBox Msg  ' Display message.
   For I = 1 To 100  ' Count from 1 to 100.
      EEntry = "Entry " & I   ' Create entry.
      LList1.AddItem Entry  ' Add the entry.
   Next I
   Msg = "Choose OK to remove every other entry."
   MsgBox Msg  ' Display message.
   For I = 1 To 50   ' Determine how to remove every other item.
      LList1.RemoveItem I
   Next I
   Msg = "Choose OK to remove all items from the list box."
   MsgBox Msg    ' Display message.
```

```
      List1.Clear   ' Clear list box.

   End Sub
```

4. Press the F5 key to run the program and click anywhere in the form to clear the contents of List1.

Clearing a List Box in Visual Basic Version 2.0

No single command in Visual Basic version 2.0 will clear out the entries of a list box, but a simple While Loop will, as follows:

```
   Do While List1.ListCount > 0
      List1.RemoveItem 0
   Loop
```

If you want a single command to clear all list box entries at once in version 2.0, you can use the SendMessage Windows API function. The arguments to SendMessage with the LB_RESETCONTENT parameter are

```
   SendMessage(hWnd%, wMsg%, wParam%, lParam&)
```

where:

```
   hWnd%    Identifies the window that is to receive the message
   wMsg%    The message to be sent  (&H405)
   wParam%  Is not used (NULL)
   lParam&  Is not used (NULL)
```

Setting wMsg% equal to &H405 removes all strings from the list box and frees any memory allocated for those strings.

To get hWnd%, you must call the Windows API function GetFocus(). This method will return the handle to the control that currently has focus, in this case the list box that you want to clear.

Version 2.0 Example

The following shows by example how to delete entries from a list box in version 2.0.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a list box (List1) to Form1.

3. Declare the following Windows API functions at the module level or in the Global section of your code as follows:

```
   ' Enter the following Declare statement as one, single line:
   Declare Function SendMessage% Lib "user" (ByVal hWnd%,
      ByVal wMsg%, ByVal wParam%, ByVal lParam&)
   Declare Function GetFocus% Lib "user" ()
   Declare Function PutFocus% Lib "user" Alias "SetFocus" (ByVal hWnd%)
```

4. Declare the following constants in the same section:

```
   Const WM_USER = &H400
   Const LB_RESETCONTENT = WM_USER + 5
```

5. Create the following Sub procedure in the (Declarations) section of the Code window:

```
   Sub ClearListBox (Ctrl As Control)
      hWndOld% = GetFocus()
      Ctrl.SetFocus
      x = SendMessage%(GetFocus(), LB_RESETCONTENT, 0, 0)
      Suc% = PutFocus(hWndOld%)
   End Sub
```

6. Within an event procedure, call ClearListBox with the name of the list box as a parameter:

```
   Sub Form_Click ()
      ClearListBox List1
   End Sub
```

7. Place some entries into the list box:

```
   Sub Form_Load ()
      For i = 1 To 10
         List1.AddItem Format$(i)  'Put something into list box.
      Next
   End Sub
```

8. Run the program and click anywhere on Form1 to clear the list box.

REFERENCES
==========

"Programming Windows: the Microsoft Guide to Writing Applications for Windows 3," by Charles Petzold, Microsoft Press, 1990

"Microsoft Windows Software Development Kit: Reference Volume 1" version 3.0

WINSDK.HLP file shipped with Microsoft Windows 3.0 Software Development Kit

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Emulate QuickBasic's SOUND Statement in Visual Basic

Article ID: Q71102

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The SOUND statement found in Microsoft QuickBasic is not implemented within Microsoft Visual Basic. You can perform sound through a Windows 3.00 API call that is equivalent to the QuickBasic SOUND statement.

MORE INFORMATION

The QuickBasic version of the SOUND statement can be executed by calling several Windows 3.0 API function calls. Within Windows, you must open up a VoiceQueue with the OpenSound call routine. Using the function SetVoiceSound, place all of the values corresponding to the desired frequencies and durations. Once the VoiceQueue has the desired frequencies and durations, you start the process by calling StartSound. After the sounds have been played, you must free up the VoiceQueue by calling CloseSound. If you plan on placing a large amount of information into the VoiceQueue, you may need to resize the VoiceQueue buffer by calling the SetVoiceQueueSize function.

After executing the StartSound function, you cannot place any more sound into the VoiceQueue until the VoiceQueue is depleted. Placing more sound into the queue will overwrite any information that was previously in the VoiceQueue. If you are going to place sound into the VoiceQueue after a StartSound statement, you will need to call WaitSoundState with an argument of one. When WaitSoundState returns NULL, the VoiceQueue is empty and processing can continue.

Below is an example of using the Windows API function calls, which will imitate the QuickBasic SOUND statement:

In the general section place the following:

```
Declare Function OpenSound Lib "sound.drv" () As Integer
Declare Function VoiceQueueSize Lib "sound.drv"
            (ByVal nVoice%, ByVal nBytes%) As Integer
Declare Function SetVoiceSound Lib "sound.drv"
            (ByVal nSource%, ByVal Freq&, ByVal nDuration%) As Integer
Declare Function StartSound Lib "sound.drv" () As Integer
Declare Function CloseSound Lib "sound.drv" () As Integer
Declare Function WaitSoundState Lib "sound.drv" (ByVal State%) As Integer
```

Note: All Declare statements above each must be placed on one line.

The SetVoiceSound takes two arguments. The first variable, Freq, is a two WORD parameter. The HIGH WORD will hold the actual frequency in hertz. The LOW WORD will hold the fractional frequency. The formula, X * 2 ^ 16, will shift the variable "X" into the HIGH WORD location. The second variable, Duration%, is the duration in clock ticks. There are 18.2 tick clicks per second on all Intel computers.

The following simplistic example shows how you can place several frequencies and durations into the VoiceQueue before starting the sound by calling the StartSound function:

```
Sub Form_Click ()
   Suc% = OpenSound()
   S% = SetVoiceSound(1, 100 * 2 ^ 16, 100)    ' Frequency = 100 hz
   S% = SetVoiceSound(1, 90 * 2 ^ 16, 90)      ' Frequency = 90 hz
   S% = SetVoiceSound(1, 80 * 2 ^ 16, 90)      ' Frequency = 80 hz
   S% = StartSound()
   While (WaitSoundState(1) <> 0): Wend        ' Wait for sound to play.
   Succ% = CloseSound()
End Sub
```

The following is another simple example, which creates a siren sound:

1. Within the general section, place the following Sound procedure:

```
Sub Sound (ByVal Freq as Long, ByVal Duration%)
   Freq = Freq * 2 ^ 16                  ' Shift frequency to high byte.
   S% = SetVoiceSound(1, Freq, Duration%)
   S% = StartSound()
   While (WaitSoundState(1) <> 0): Wend
End Sub
```

2. Place the code below into any event procedure. The example below uses the Form_Click event procedure. Clicking any position on the form will create a police siren.

```
Sub Form_Click ()
   Suc% = OpenSound()
   For j& = 440 To 1000 Step 5
      Call Sound(j&, j& / 100)
   Next j&
   For j& = 1000 To 440 Step -5
      Call Sound(j&, j& / 100)
   Next j&
   Succ% = CloseSound()
End Sub
```

Reference(s):

"Programming Windows: the Microsoft Guide to Writing Applications for Windows 3," Charles Petzold, Microsoft Press, 1990

"Microsoft Windows Software Development Kit: Reference Volume 1," version 3.0

WINSDK.HLP file shipped with Microsoft Windows 3.0 Software Development Kit

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Flood Fill (Paint) in VB using ExtFloodFill Windows API

Article ID: Q71103

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

You can fill an area on a window in Visual Basic through a Windows API function call. Depending on the type of fill to be performed, you can use the ExtFloodFill function to achieve the desired effect. This feature is similar to the paint feature found in painting programs.

This information applies to Microsoft Visual Basic programming system version 1.0 for Windows.

MORE INFORMATION

The Windows API ExtFloodFill function call fills an area of the display surface with the current brush, as shown in the example below.

Code Example

From the VB.EXE Code menu, choose View Code, and enter the following code (on just one line) for Form1 (using [general] from the Object box and [declarations] from the Procedure box):

```
Declare Function ExtFloodFill Lib "GDI" (ByVal hdc%, ByVal i%,
   ByVal i%, ByVal w&, ByVal i%) As Integer
```

To demonstrate several fill examples, create a picture box called Picture1. Set the following properties:

```
AutoSize = TRUE           ' Scale picture to size of imported picture.
ScaleMode = 3             ' Set the scale mode to pixels, not twips
FillColor = &HFF00FF      ' This will be the selected fill color.
FillStyle = Solid         ' Necessary to create a fill pattern.
Picture = Chess.bmp       ' This should be in your Windows directory.
```

Create a push button in a location that will not be overlapped by Picture1. Within the Click event, create the following code:

```
Sub Command1_Click ()
 ' Make sure that the FillStyle is not transparent.
 ' crColor& specifies the color for the boundary.
  Const FLOODFILLBORDER = 0  ' Fill until crColor& color encountered.
  Const FLOODFILLSURFACE = 1 ' Fill surface until crColor& color not
                             ' encountered.
  X% = 1
```

```
  Y% = 1
  crColor& = RGB(0, 0, 0)
  wFillType% = FLOODFILLSURFACE
  Suc% = ExtFloodFill(picture1.hDC, X%, Y%, crColor&, wFillType%)
End Sub
```

When you click the push button, the black background will change to the FillColor. The fill area is defined by the color specified by crColor&. Filling continues outward from (X%,Y%) as long as the color is encountered.

Now change the related code to represent the following:

```
   crColor& = RGB(255, 0, 0)  'Color to look for.
   wFillType% = FLOODFILLBORDER
   Suc% =  ExtFloodFill(picture1.hDC, X%, Y%, crColor&, wFillType%)
```

Executing the push button will now fill the area until crColor& is encountered. In the first example, the fill was performed while the color was encountered; in the second example, the fill was performed while the color was not encountered. In the last example, everything is changed except the "floating pawn".

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap

How to Use Windows BitBlt Function in Visual Basic Application

Article ID: Q71104

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Windows GDI.EXE has a function called BitBlt that will move the source device given by the hSrcDC parameter to the destination device given by the hDestDC parameter. This article explains in detail the arguments of the Windows BitBlt function call.

MORE INFORMATION

This information is included with the Help file provided with Microsoft Professional Toolkit for Visual Basic version 1.0, Microsoft Visual Basic version 2.0, and Microsoft Visual Basic version 3.0.

To use BitBlt within a Visual Basic application, you must Declare the BitBlt function in one of these places:

- Global module if using Visual Basic version 1.0.
- Declaration section of any code module if using Visual Basic version 2.0 or higher.
- Declaration section of a code window for the form.

Use the following Declare statement to declare the Function. Enter the entire Declare statement on one, single line:

```
Declare Function BitBlt Lib "GDI" (ByVal hDestDC%, ByVal X%, ByVal Y%,
   ByVal nWidth%, ByVal nHeight%, ByVal hSrcDC%, ByVal XSrc%, ByVal YSrc%,
   ByVal dwRop&) As Integer
```

The following defines each of the formal parameters used in the Declare:

Parameter	Definition
hDestDC	Specifies the device context that is to receive the bitmap.
X,Y	Specifies the logical x-coordinate and y-coordinate of the upper-left corner of the destination rectangle.
nWidth	Specifies the width (in logical units) of the destination rectangle and the source bitmap.
nHeight	Specifies the height (in logical units) of the destination rectangle and the source bitmap.

hSrcDC	Identifies the device context from which the bitmap will be copied. It must be NULL(zero) if the dwRop& parameter specifies a raster operation that does not include a source.
XSrc	Specifies the logical x-coordinate and the y-coordinate of the upper-left corner of the source bitmap.
dwRop	Specifies the raster operation to be performed as defined below.

The following Raster operations are defined using the predefined constants found in the WINDOWS.H file supplied with the Microsoft Windows Software Development Kit (SDK). The value in the parentheses () is the value to assign to the dwRop& variable.

Code/Value (hex)	Description
BLACKNESS (42)	Turn output black.
DSINVERT(550009)	Inverts the destination bitmap.
MERGECOPY(C000CA)	Combines the pattern and the source bitmap using the Boolean AND operation.
MERGEPAINT(BB0226)	Combines the inverted source bitmap with the destination bitmap using the Boolean OR operator.
NOTSRCCOPY(330008)	Copies the inverted source bitmap to the destination.
NOTSRCERASE(1100A6)	Inverts the result of combining the destination and source bitmap using the Boolean OR operator.
PATCOPY(F00021)	Copies the pattern to the destination bitmap.
PATINVERT(5A0049)	Combines the destination bitmap with the pattern using the Boolean XOR operator.
PATPAINT(FB0A09)	Combines the inverted source bitmap with the pattern using the Boolean OR operator. Combines the result of this operation with the destination bitmap using the Boolean OR operator.
SRCAND(8800C6)	Combines pixels of the destination and source bitmap using the Boolean AND operator.
SRCCOPY(CC0020)	Copies the source bitmap to destination bitmap.
SRCERASE(4400328)	Inverts the destination bitmap and combines the results with the source bitmap using the Boolean AND operator.
SRCINVERT(660046)	Combines pixels of the destination and source bitmap using the Boolean XOR operator.
SRCPAINT(EE0086)	Combines pixels of the destination and source bitmap using the Boolean OR operator.

WHITENESS(FF0062) Turns all output white.

Step-by-Step Example

Here is an example showing how to copy the contents of a picture control to the contents of another picture control.

1. Define a form (Form1) and place two picture controls (Picture1 and Picture2) on Form1.

2. Display some graphics on Picture1 by loading from a picture file or by pasting from the clipboard at design time. You can load a picture from a file as follows:

   - Select Picture from the Properties list box and click the arrow at the right of the Settings box.
   - Then select the desired picture file such as a .BMP or .ICO file supplied with Microsoft Windows from the dialog box.

3. Add the following code to the Form_Click procedure:

```
Sub Form_Click ()

   ' Assign information of the destination bitmap. Note that Bitblt
   ' requires coordinates in pixels.
   Const PIXEL = 3
   Picture1.ScaleMode = PIXEL
   Picture2.ScaleMode = PIXEL
   hDestDC% = Picture2.hDC
   X% = 0: Y% = 0
   nWidth% = Picture2.ScaleWidth
   nHeight% = Picture2.ScaleHeight

   ' Assign information of the source bitmap.
   hSrcDC% = Picture1.hDC
   XSrc% = 0: YSrc% = 0

   ' Assign the SRCCOPY constant to the Raster operation.
   dwRop& = &HCC0020

   Suc% = BitBlt(hDestDC%, X%, Y%, nWidth%, nHeight%,_
   hSrcDC%, XSrc%, YSrc%, dwRop&)
End Sub
```

4. Run the program.

Click the form. The contents of the first picture will be displayed on the the second picture.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap

How to Pass One-Byte Parameters from VB to DLL Routines

Article ID: Q71106

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Calling some routines in dynamic link libraries (DLLs) requires BYTE parameters in the argument list. Visual Basic for Windows possesses no BYTE data type as defined in other languages such as C, which can create DLLs. To pass a BYTE value correctly to an external FUNCTION (in a DLL), which requires a BYTE data type, you must pass an integer data type for the BYTE parameter.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Visual BASIC for Windows has the ability to call external code in the form of dynamic link libraries (DLLs). Some of these libraries require BYTE parameters in the argument list. An example of this is located in the KEYBOARD.DRV FUNCTION as defined below:

```
    FUNCTION GetTempFileName (BYTE  cDrive,
                              LPSTR lpPrefix,
                              WORD  wUnique,
                              LPSTR lpTempFileName)
```

GetTempFileName is documented on page 4-217 of the "Microsoft Windows 3.0 Software Development Kit, Reference - Volume 1." In Visual Basic for Windows, declare the FUNCTION on one line in the main module of your code:

```
   DECLARE FUNCTION GetTempFileName LIB "keyboard.drv"
                   (BYVAL A%,  BYVAL B$, BYVAL C%, BYVAL D$)
```

Because the architecture of the 80x86 stack is segmented into word boundaries, the smallest type pushed onto the stack will be a word. Therefore, both the BYTE and the integer will be pushed onto the stack in the same manner, and require the same amount of memory. This is the reason you can use an integer data type for a BYTE data type in these types of procedure calls.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Send an HBITMAP to Windows API Function Calls from VB

Article ID: Q71260

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Several Windows API functions require the HBITMAP data type. Visual Basic for Windows does not have a HBITMAP data type. This article explains how to send the equivalent Visual Basic for Windows HBITMAP handle of a picture control to a Windows API function call.

MORE INFORMATION

This information is also included with the Microsoft Knowledge Base Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.
The HBITMAP data type represents a 16-bit index to GDIs physical drawing object. Several Windows API routines need the HBITMAP data type as an argument. Sending the [picture-control].Picture as an argument is the equivalent in Visual Basic for Windows.

The code sample below demonstrates how to send HBITMAP to the Windows API function ModifyMenu:

```
' Enter the following Declare statement as one, single line:
Declare Function SetMenuItemBitMaps% Lib "user" (ByVal hMenu%, ByVal nPos%,
   ByVal wFlag%, ByVal BitmapUnChecked%, ByVal hBitmapChecked%)
```

The SetMenuItemBitMap takes five arguments. The fourth and fifth arguments are HBITMAP data types.

The following code segment will associate the specified bitmap Picture1.Picture in place of the default check mark:

```
X% = SetMenuItemBitMap(hMenu%, menuID%,0,0, Picture1.Picture)
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap

How to Create a Flashing Title Bar on a Visual Basic Form

Article ID: Q71280

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

When calling a Windows API function call, you can create a flashing window title bar on the present form or any other form for which you know the handle.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Visual Basic for Windows has the ability to flash the title bar on any other form if you can get the handle to that form. The function FlashWindow flashes the specified window once. Flashing a window means changing the appearance of its caption bar, as if the window were changing from inactive to active status, or vice versa. (An inactive caption bar changes to an active caption bar; an active caption bar changes to an inactive caption bar.)

Typically, a window is flashed to inform the user that the window requires attention when that window does not currently have the input focus.

The function FlashWindow is defined as

```
FlashWindow(hWnd%, bInvert%)
```

where:

hWnd%	- Identifies the window to be flashed. The window can be either open or iconic.
bInvert%	- Specifies whether the window is to be flashed or returned to its original state. The window is flashed from one state to the other if the bInvert parameter is nonzero. If the bInvert parameter is zero, the window is returned to its original state (either active or inactive).

FlashWindow returns a value that specifies the window's state before the call to the FlashWindow function. It is nonzero if the window was active before the call; otherwise, it is zero.

The following section describes how to flash a form while that form

does not have the focus:

1. Create two forms called Form1 and Form2.

2. On Form1, create a timer control and set the Interval Property to 1000. Also set the Enabled Property to FALSE.

3. Within the general-declarations section of Form1, declare the FlashWindow function as follows:

```
' The following Declare statement must appear on one line.
Declare Function FlashWindow% Lib "user" (ByVal hWnd%,
                                          ByVal bInvert%)
```

4. In Visual Basic version 1.0 for Windows, define the following constants in the declarations section:

```
Const TRUE = -1
Const FALSE = 0
```

5. In the Form_Load event procedure, add the following code:

```
Sub Form_Load ()
   Form2.Show
End Sub
```

6. In the Sub Timer1_Timer () procedure of Form1, add the following code:

```
Sub Timer1_Timer ()
   Succ% = FlashWindow(Form1.hWnd, 1)
End Sub
```

7. In the GotFocus event procedure of Form1, create the following code:

```
Sub Form_GotFocus ()
  Timer1.Enabled =  False
End Sub
```

8. In the Click event for Form2, add the following code:

```
Sub Form_Click ()
   Form1.Timer1.Enabled = True
End Sub
```

9. Run the program. Form1 will be in the foreground with Form2 in the background. Click anywhere in Form2; Form1's Caption Bar will flash until you click Form1.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgWindow APrgOther

How to Implement a Bitmap Within a Visual Basic Menu

Article ID: Q71281

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

No command provided by the Visual Basic language can add a bitmap to the menu system. However, you can call several Windows API functions to place a bitmap within the menu system of a Visual Basic program. You may also change the default check mark displayed.

MORE INFORMATION

There are several Windows API functions you can call that will display a bitmap instead of text in the menu system.

Below is a list of the required Windows API functions:

- GetMenu% (hwnd%)

  hwnd% - Identifies the window whose menu is to be examined
  Returns: Handle to the menu

- GetSubMenu% (hMenu%, nPos%)

  hMenu% - Identifies the menu
  nPos% - Specifies the position (zero-based) in the given menu of the pop-up menu
  Returns: Handle to the given pop-up menu

- GetMenuItemID% (hMenu%, nPos%)

  hMenu% - Identifies the handle to the pop-up menu that contains the item whose ID is being retrieved
  nPos% - Specifies the position (zero-based) of the menu whose ID is being retrieved
  Returns: The item ID for the specified item in the pop-up menu

- ModifyMenu% (hMenu%, nPos%, wFlags%, wIDNewItem%, lpNewItem&)

  hMenu% - Identifies the handle to the pop-up menu that contains the item whose ID is being retrieved
  nPos% - Specifies the menu item to be changed. The interpretation of the nPos parameter depends on the wFlags parameter.
  wFlags% - BF_BITMAP = &H4

```
    wIDNewItem% - Specifies the command ID of the modified menu item
    lpNewItem&  - 32-bit handle to the bitmap
    Returns:  TRUE (-1) if successful, FALSE (0) if unsuccessful

 - SetMenuItemBitmaps% (hMenu%, nPos%, Flags%, hBitmapUnchecked%,
                        hBitmapChecked%)

    hMenu%  - Identifies menu to be changed
    nPos%   - Command ID of the menu item
    wFlags% - &H0
    hBitmapUnchecked% - Handle to "unchecked" bitmap.
    hBitmapChecked%)  - Handle to the "check" bitmap.
    Returns: TRUE (-1) if successful, FALSE (0) if unsuccessful.
```

There are two different ways to implement bitmaps within Visual Basic: the first method is to use static bitmaps; the other method is to use dynamic bitmaps.

A static bitmap is fixed and does not change during the execution of the program (such as when it is taken from an unchanging .BMP file). A dynamic bitmap changes during execution of your program. You may change dynamic bitmap attributes such as color, size, and text. The sample code below describes how to create both types of menus.

Define a menu system using the Menu Design window. Create a menu system such as the following:

Caption	Control Name	Indented	Index
BitMenu	TopMenu	No	
Sub Menu0	SubMenu	Once	0
Sub Menu1	SubMenu	Once	1
Sub Menu2	SubMenu	Once	2

Create a picture control array with three bitmaps by creating three picture controls with the same control Name using the Properties list box.

Control Name	Caption	Index	FontSize
Picture1		0	N/A
Picture1		1	N/A
Picture1		2	N/A
Picture2		N/A	N/A 'check BMP
Picture3		0	'set Picture3 FontSize all different
Picture3		1	9.75
Picture3		2	18
Command1	Static		
Command2	Dynamic		

For each control index of Picture1, add a valid bitmap to the Picture property. Because these bitmaps will be displayed in the menu, you should use smaller bitmaps. Add a bitmap to the Picture2 Picture property that you want to be your check mark when you select a menu option.

Both types of bitmap implementations will need to have the following

declarations in the declaration or global section of your code:

```
' Enter each Declare statement on one, single line:
Declare Function GetMenu% Lib "user" (ByVal hwnd%)
Declare Function GetSubMenu% Lib "user" (ByVal hMenu%, ByVal nPos%)
Declare Function GetMenuItemID% Lib "user" (ByVal hMenu%, ByVal nPos%)
Declare Function ModifyMenu% Lib "user" (ByVal hMenu%, ByVal nPosition%,
   ByVal wFlags%, ByVal wIDNewItem%, ByVal lpNewItem&)
Declare Function SetMenuItemBitmaps% Lib "user" (ByVal hMenu%,
   ByVal nPosition%, ByVal wFlags%, ByVal hBitmapUnchecked%,
   ByVal BitmapChecked%)
Const MF_BITMAP = &H4
Const CLR_MENUBAR = &H80000004 ' Defined for dynamic bitmaps only.
Const TRUE = -1, FALSE = 0
Const Number_of_Menu_Selections = 3
```

The following Sub will also need to be defined to handle the actual redefinition of the "check" bitmap:

```
Sub SubMenu_Click (Index As Integer)
' Uncheck presently checked item, check new item, store
' index
   Static LastSelection%
   SubMenu(LastSelection%).Checked = FALSE
   SubMenu(Index).Checked = TRUE
   LastSelection% = Index
End Sub

Sub Command1_Click ()
  '* example to create a static bitmap menu
  hMenu% = GetMenu(hWnd)
  hSubMenu% = GetSubMenu(hMenu%, 0)
  For i% = 0 To Number_of_Menu_Selections - 1
    menuId% = GetMenuItemID(hSubMenu%, i%)
    x% = ModifyMenu(hMenu%, menuId%, MF_BITMAP, menuId%,
                    CLng(picture1(i%).Picture))
    x% = SetMenuItemBitmaps(hMenu%, menuId%, 0, 0,
                    CLng(picture2.Picture))
  Next i%
End Sub

'This code sample will change the actual menu bitmaps size,
'font size, color, and caption. Run the application and
'select the BitMenu and view the selections. Then click
'the form and revisit the BitMenu.
'-------------------------------------------------------
Sub Command2_Click ()
   '* Example to create a dynamic menu system
   hMenu% = GetMenu(hWnd)
   hSubMenu% = GetSubMenu(hMenu%, 0)
   For i% = 0 To Number_of_Menu_Selections - 1
   '* Place some text into the menu.

      SubMenu(i%).Caption = Picture3(i%).FontName +
                Str$(Picture3(i%).FontSize) + " Pnt"

   '* 1. Must be AutoRedraw for Image().
```

```
   '* 2. Set Backcolor of Picture control to that of the
   '*    current system Menu Bar color, so Dynamic bitmaps
   '*    will appear as normal menu items when menu bar
   '*    color is changed via the control panel
   '* 3. See the bitmaps on screen, this could all be done
   '*    at design time.

     Picture3(i%).AutoRedraw = TRUE
     Picture3(i%).BackColor = CLR_MENUBAR
   '* You can uncomment this
   '* Picture3(i%).Visible = FALSE

   '* Set the width and height of the Picture controls
   '* based on their corresponding Menu items caption,
   '* and the Picture controls Font and FontSize.
   '* DoEvents() is necessary to make new dimension
   '* values to take affect prior to exiting this Sub.

    Picture3(i%).Width = Picture3(i%).TextWidth(SubMenu(i%).Caption)
    Picture3(i%).Height = Picture3(i%).TextHeight(SubMenu(i%).Caption)
    Picture3(i%).Print SubMenu(i%).Caption

   '* - Set picture controls backgroup picture (Bitmap) to
   '*   its Image.
     Picture3(i%).Picture = Picture3(i%).Image
     x% = DoEvents()
   Next i%

    '* Get handle to forms menu.
   hMenu% = GetMenu(Form1.hWnd)

   '* Get handle to the specific menu in top level menu.
   hSubMenu% = GetSubMenu(hMenu%, 0)

   For i% = 0 To Number_of_Menu_Selections - 1

   '* Get ID of sub menu
     menuId% = GetMenuItemID(hSubMenu%, i%)

   '* Replace menu text w/bitmap from corresponding picture
   '* control
     x% = ModifyMenu(hMenu%, menuId%, MF_BITMAP, menuId%,
              CLng(Picture3(i%).Picture))  'append this to previous line

   '* Replace bitmap for menu check mark with custom check
   '* bitmap
     x% = SetMenuItemBitmaps(hMenu%, menuId%, 0, 0, CLng(picture2.Picture))
   Next i%
End Sub
```

Reference(s):

"Programming Windows: the Microsoft Guide to Writing Applications for Windows 3," Charles Petzold, Microsoft Press, 1990

"Microsoft Windows Software Development Kit: Reference Volume 1," version 3.0

WINSDK.HLP file shipped with Microsoft Windows 3.0 SDK

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap

How to Create Rubber-Band Lines/Boxes in Visual Basic

Article ID: Q71488

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Creating rubber bands within Visual Basic can be done using the DrawMode property. Rubber bands are lines that stretch as you move the mouse cursor from a specified point to a new location. This can be very useful in graphics programs and when defining sections of the screen for clipping routines.

MORE INFORMATION

The theory of drawing a rubber-band box is as follows:

1. Draw a line from the initial point to the location of the mouse cursor using:

   ```
   [form].DrawMode = 6. {INVERT}
   ```

2. Move the mouse cursor.

3. Save the DrawMode.

4. Set the [form].DrawMode to 6. {INVERT}

5. Draw the same line that was drawn in step 1. This will restore the image underneath the line.

6. Set the [form].DrawMode back to the initial DrawMode saved in step 3.

7. Repeat the cycle again.

DrawMode equal to INVERT allows the line to be created using the inverse of the background color. This allows the line to be always displayed on all colors.

The sample below will demonstrate the rubber-band line and the rubber-band box. Clicking the command buttons will allow the user to select between rubber-band line or a rubber-band box. The user will also be able to select a solid line or a dashed line.

Create and set the following controls and properties:

Control	Name	Caption	Picture

```
   Form1          Form1        c:\windows\chess.bmp
   Command1       RubberBand
   Command2       RubberBox
   Command3       Dotted
   Command4       Solid
```

In the general section of your code, define the following constants:

```
Const INVERSE = 6       '*Characteristic of DrawMode property(XOR).
Const SOLID = 0         '*Characteristic of DrawStyle property.
Const DOT = 2           '*Characteristic of DrawStyle property.
Const TRUE = -1
Const FALSE = 0
Dim DrawBox As Integer  '*Boolean-whether drawing Box or Line
Dim OldX, OldY, StartX, StartY As Single  '* Mouse locations
```

In the appropriate procedures, add the following code:

```
Sub Form_MouseDown (Button As Integer, Shift As Integer, X As
                                       Single, Y As Single)
   '* Store the initial start of the line to draw.
   StartX = X
   StartY = Y

   '* Make the last location equal the starting location
   OldX = StartX
   OldY = StartY
End Sub

Sub Form_MouseMove (Button As Integer, Shift As Integer, X As
                     Single, Y As Single)
   '* If the button is depressed then...
   If Button Then
      '* Erase the previous line.
      Call DrawLine(StartX, StartY, OldX, OldY)

      '* Draw the new line.
      Call DrawLine(StartX, StartY, X, Y)

      '* Save the coordinates for the next call.
      OldX = X
      OldY = Y
   End If
End Sub

Sub DrawLine (X1, Y1, X2, Y2 As Single)
   '* Save the current mode so that you can reset it on
   '* exit from this sub routine. Not needed in the sample
   '* but would need it if you are not sure what the
   '* DrawMode was on entry to this procedure.
   SavedMode% = DrawMode

   '* Set to XOR
   DrawMode = INVERSE

   '*Draw a box or line
   If DrawBox Then
```

```
      Line (X1, Y1)-(X2, Y2), , B
   Else
      Line (X1, Y1)-(X2, Y2)
   End If

   '* Reset the DrawMode
   DrawMode = SavedMode%
End Sub

Sub Form_MouseUp (Button As Integer, Shift As Integer, X As Single,
                  Y As Single)
   '* Stop drawing lines/boxes.
   StartEvent = FALSE
End Sub

Sub Command2_Click ()
   '* Boolean value to determine whether to draw a line or box.
   DrawBox = TRUE
End Sub

Sub Command1_Click ()
   '* Boolean value to determine whether to draw a line or box.
   DrawBox = FALSE
End Sub

Sub Command3_Click ()
   '* Create a dotted line
   Form1.DrawStyle = DOT
End Sub

Sub Command4_Click ()
   '* Create a solid line.
   Form1.DrawStyle = SOLID
End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap PrgCtrlsStd

How to Create Flashing/Rotating Rubber-Band Box in VB

Article ID: Q71489

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Several programs, such as Excel, create a flashing border (which appears to rotate) when selecting items of the windows when using the Edit Copy selection of the menu system. You can create a flashing, rotating border with the DrawMode and DrawStyle properties of a Visual Basic form.

MORE INFORMATION

By drawing a dashed line on the form and then within a timer event creating a solid line on the dashed line with DrawMode set to INVERSE, you can create a special effect of a flashing border that appears to rotate.

You can draw a rotating rubber-band box as follows:

1. Draw a line using:

   ```
   DrawStyle = 2 {Dot}
   ```

2. Save the [form].DrawMode and the [form].DrawStyle.

3. Set the [form].DrawMode = 6 {Inverse}.

4. Set [form].DrawStyle = 0 {Solid}.

5. Draw the same line as in step 1.

6. Reset the properties saved in step 2.

7. Delay some time interval.

8. Repeat starting at step 2.

The following code demonstrates the rotating (flashing) border. Pressing the mouse button and then dragging the cursor some distance will create a dotted line. Releasing the button will display a rotating rubber-band box.

In VB.EXE, create a form called Form1. On Form1, create a timer control with the name Timer1 and with an interval of 100.

Duplicate the following code within the general declaration section of your code window:

```
Const INVERSE = 6     'Characteristic of DrawStyle property(Inverse).
Const SOLID = 0       'Characteristic of DrawMode property.
Const DOT = 2         'Characteristic of DrawMode property.
Const TRUE = -1
Const FALSE = 0
Dim OldX, OldY, StartX, StartY As Single
```

Add the following code in the appropriate event procedures for Form1:

```
Sub Form_Load ()
   '* Must draw a dotted line to create effect.  Load a bitmap. Not
      required but shows full extent of line drawing.
   DrawStyle = DOT
End Sub

Sub Timer1_Timer ()
   SavedDrawStyle% = DrawStyle

  '* Solid is need to create the inverse of the dashed line.
   DrawStyle = SOLID

 '* Invert the dashed line.
   Call DrawLine(StartX, StartY, OldX, OldY)

  '* Restore the DrawStyle back to what it was previously.
   DrawStyle = SavedDrawStyle%
End Sub

Sub Form_MouseDown (Button As Integer, Shift As Integer, X As
                                       Single, Y As Single)
' The above Sub statement must be on just one line.
   '* Don't add effect as you draw box.
   Timer1.Enabled = FALSE
   '* Save the start locations.
   StartX = X
   StartY = Y
   '* Set the last coord. to start locations.
   OldX = StartX
   OldY = StartY
End Sub

Sub Form_MouseMove (Button As Integer, Shift As Integer, X As
                                       Single, Y As Single)
' (The above Sub statement must be on just one line.)
'* If button is depress then...
If Button Then
      '* Restore previous lines background.
      Call DrawLine(StartX, StartY, OldX, OldY)
      '* Draw new line.
      Call DrawLine(StartX, StartY, X, Y)
      '* Save coordinates for next call.
      OldX = X : OldY = Y
   End If
End Sub
```

```
Sub DrawLine (X1, Y1, X2, Y2 As Single)
   '* Save the current mode so that you can reset it on
   '* exit from this sub routine. Not needed in the sample
   '* but would need it if you are not sure what the
   '* DrawMode was on entry to this procedure.
   SavedMode% = DrawMode

   '* Set to XOR
   DrawMode = INVERSE

   '*Draw a box
   Line (X1, Y1)-(X2, Y2), , B

   '* Reset the DrawMode
   DrawMode = SavedMode%
End Sub

Sub Form_MouseUp (Button As Integer, Shift As Integer, X As Single,
                                              Y As Single)
' (The above Sub statement must be on just one line.)
   StartEvent = FALSE
   Timer1.Enabled = TRUE
End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap

Declare Currency Type to Be Double When Returning from DLL

Article ID: Q72274

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

When using Microsoft Visual Basic for Windows, if you want to pass a parameter to a dynamic link library (DLL) routine, or receive a function return value of type Currency from a DLL routine written in Microsoft C, the parameter or function returned should be declared as a "double" in the C routine.

Note that C does not support the Basic Currency data type, and although specifying the parameter as type "double" in C will allow it to be passed correctly, you will have to write your own C routines to manipulate the data in the Currency variable. For information on the internal format of the Currency data type, query in the Microsoft Knowledge Base using the following words:

Basic and Currency and internal and format

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

When creating a DLL function that either receives or returns a Currency data type, it may be useful to include the following declaration:

```
typedef double currency;
```

Based on this typedef, a sample DLL routine to return a currency value might be declared as follows:

```
currency FAR pascal test(...);
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Create a System-Modal Program/Window in Visual Basic

Article ID: Q72674

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

From a Microsoft Visual Basic for Windows program, you can disable the ability to switch to other Windows programs by calling the Windows API function SetSysModalWindow.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Microsoft Windows is designed so that the user can switch between applications without terminating one program to run another program. There may be times when the program needs to take control of the entire environment and run from only one window, restricting the user from switching to any other application. An example of this is a simple security system, or a time-critical application that may need to go uninterrupted for long periods of time.

Passing the handle to the window through the argument of SetSysModalWindow will limit the user to that particular window. This will not allow the user to move to any other applications with the mouse or use ALT+ESC or CTRL+ESC to bring up the Task Manager. You can even remove the system menu if you do not want the user to exit through the ALT+F4 (Close) combination.

All child windows that are created by the system-modal window become system-modal windows. When the original window becomes active again, it is system-modal. To end the system-modal state, destroy the original system-modal window.

Care must be taken when using the SetSysModalWindow API from within the Visual Basic for Windows programming environment. Pressing CTRL+BREAK to get to the [break] mode leaves your modal form with no way to exit unless you restart your system. When using the SetSysModalWindow within the environment, be sure to exit your application by destroying the window with either the ALT+F4 in the system menu, or by some other means from within your running program.

To use the SetSysModalWindow API function, declare the API call in your global section, as follows:

```
Declare Function SetSysModalWindow Lib "User" (ByVal hwnd%) As Integer
```

At an appropriate place in your code, add the following:

```
   Success% = SetSysModalWindow(hwnd)
```

Once this line is executed, your window will be the only window that can get focus until that window is destroyed.

Note: Because Visual Basic for Windows was not designed with system modal capabilities in mind, using a MsgBox, InputBox, or Form.Show of another form from a system modal window will not work correctly. If you want to show another window from a system modal form, use another Visual Basic for Windows form and call SetSysModalWindow for this second form also, so that it becomes the system modal window. When the second form is unloaded, the original system modal form will again become the system modal window. Note that because the window(s) shown from a system modal window must also call SetSysModalWindow, and since MsgBox/InputBox windows cannot have associated code, you should not call the MsgBox or InputBox functions from a system modal window.

Additional reference words: 1.00 2.00 3.00 dialog
KBCategory:
KBSubcategory: APrgOther

VB Out of Stack Space Error w/ LoadPicture in Form_Paint Event

Article ID: Q72675

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

An "Out of stack space" error can occur when you use a LoadPicture method within a Form_Paint event.

MORE INFORMATION

The Visual Basic stack can be exhausted when the LoadPicture method is executed within a [control/form]_Paint event. The LoadPicture method generates a [control/form]_Paint event itself, and when performed within a _Paint event, the program will repeat the cycle until the stack is exhausted.

The following code example demonstrates that the Form_Paint event is a recursive procedure when a LoadPicture method is included in the _Paint event code.

After you add the code to your program, run the program and notice how many times the message "Form_Paint Count :" is displayed within the Immediate Window before you receive the "Out of stack space" error message.

```
Sub Form_Paint ()
      Static Count
      Count = Count + 1
      Debug.Print "Form_Paint Count : "; Count
      Form1.picture = LoadPicture("c:\windows\chess.bmp")
End Sub
```

To remedy the situation, move the LoadPicture to another event handler, such as the Form_Load event. Since these bitmaps are automatically refreshed when needed, you don't have to maintain the picture within a Paint event.

The Visual Basic stack is limited to 16K bytes, and cannot be changed.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: APrgGrap PrgOptMemMgt

How to Limit User Input in VB Combo Box with SendMessage API

Article ID: Q72677

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can specify a limit to the amount of text that can be entered into a combo box by calling SendMessage (a Windows API function) with the EM_LIMITTEXT constant.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The following method can be used to limit the length of a string entered into a combo box. Check the length of a string inside a KeyPress event for the control, if the length is over a specified amount, then the formal argument parameter KeyAscii will be set to zero.

Or, the preferred method of performing this type of functionality is to use the SendMessage API function call. After you set the focus to the desired edit control, you must send a message to the window's message queue that will reset the text limit for the control. The argument EM_LIMITTEXT, as the second parameter to SendMessage, will set the desired text limit based on the value specified by the third arguments. The SendMessage function requires the following parameters for setting the text limit:

```
SendMessage (hWnd%,EM_LIMITTEXT, wParam%, lParam)
```

wParam%	Specifies the maximum number of bytes that can be entered. If the user attempts to enter more characters, the edit control beeps and does not accept the characters. If the wParam parameter is zero, no limit is imposed on the size of the text (until no more memory is available).
lParam	Is not used.

The following steps can be used to implement this method:

1. Create a form called Form1.

2. Add a combo box called Combo1 to Form1.

3. Add the following code to the general declarations section of Form1:

```
    '*** Note: Each Declare statement must be on just one line:

    Declare Function GetFocus% Lib "user" ()
    Declare Function SendMessage& Lib "user" (ByVal hWnd%,
                                              ByVal wMsg%,
                                              ByVal wParam%,
                                              lp As Any)
    Const WM_USER = &H400
    Const EM_LIMITTEXT = WM_USER + 21
```

4. Add the following code to the Form_Load event procedure:

```
    Sub Form_Load ()
       Form1.Show            ' Must show form to work on it.
       Combo1.SetFocus       ' Set the focus to the list box.
       cbhWnd% = GetFocus()  ' Get the handle to the list box.
       TextLimit% = 5        ' Specify the largest string.
       retVal& = SendMessage(cbhWnd%, EM_LIMITTEXT, TextLimit%, 0)
    End Sub
```

5. Run the program and enter some text into the combo box. You will notice that you will only be able to enter a string of five characters into the combo box.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

Determining Number of Lines in VB Text Box; SendMessage API

Article ID: Q72719

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

To determine the number of lines of text within a text box control, call the Windows API function SendMessage with EM_GETLINECOUNT(&H40A) as the wMsg argument.

Calling SendMessage with the following parameters will return the amount of lines of text within a text box:

```
hWd%   - Handle to the text box.
wMsg%   - EM_GETLINECOUNT(&H40A)
wParam% - 0
lParam% - 0
```

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

For example, to determine the amount of lines within a text box, perform the following steps:

1. Create a form with a text box and a command button. Change the MultiLine property of the text box to TRUE.

2. Declare the API SendMessage function in the global-declarations section of your code window (the Declare statement must be on just one line):

```
Declare Function SendMessage% Lib "user" (ByVal hWd%,
                                          ByVal wMsg%,
                                          ByVal wParam%,
                                          ByVal lParam&)
```

3. In Visual Basic version 1.0 for Windows, you will need to declare another API routine to get the handle of the text box. Declare this routine also in your global declarations section of your code window. The returned value will become the hWd% argument to the SendMessage function. For example:

```
Declare Function GetFocus% Lib "user" ()
```

4. Within the click event of your button, add the following code:

```
Sub Command1_Click ()
   Const EM_GETLINECOUNT = &H40A  ' Defined within Windows SDK
                                  ' file, WINDOWS.H.

   ' Command button has focus, give focus to text box.
   Text1.SetFocus

   ' For Visual Basic 1.0 for Windows get the handle of the text box.
   ' hWd% = GetFocus()

   ' Print the amount of lines to the immediate window.
   Debug.Print SendMessage(Text1.hWnd, EM_GETLINECOUNT, 0, 0)
   ' For Visual Basic 1.0 for Windows use hWd% instead of Text1.hWnd.
End Sub
```

5. Run the program. Add several lines of text to the text box. Click the command button to see the number of lines printed out to the immediate window.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther PrgCtrlsStd

How VB Can Determine if a Specific Windows Program Is Running

Article ID: Q72918

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

To determine if a specific program is running, call the Windows API function FindWindow.

FindWindow returns the handle of the window whose class is given by the lpClassname parameter and whose window name (caption), is given by the lpCaption parameter. If the returned value is zero, the application is not running.

MORE INFORMATION

This information is included with the Help file provided with Microsoft Professional Toolkit for Visual Basic version 1.0, Microsoft Visual Basic version 2.0, and Microsoft Visual Basic version 3.0.

By calling FindWindow with a combination of a specific program's class name and/or the title bar caption, your program can determine whether that specific program is running.

When an application is started from the Program Manager, it registers the class name of the form. The window class provides information about the name, attributes, and resources required by your form. MDI forms in Visual Basic have ThunderMDIForm as their class name, and all other Visual Basic forms have ThunderForm as their class name.

You can determine the class name of an application by using SPY.EXE that comes with the Microsoft Windows Software Development Kit (SDK) version 3.0 or 3.1.

If the window has a caption bar title, you can also use the title to locate the instance of the running application. This caption text is valid even when the application is minimized to an icon.

Because another instance of your Visual Basic program will have the same class name and may have the same title bar caption, you must use dynamic data exchange (DDE) to determine if another instance of your Visual Basic program is running. (This DDE technique is not shown in this article).

Step-by-Step Example

The following example shows three ways to determine if the Windows

Calculator is running. To create the program, do the following:

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Declare the Windows API function FindWindow in the Global declarations section of Form1. The variables are declared as "Any" because you can pass either a pointer to a string, or a NULL (or 0&) value. You are responsible for passing the correct variable type.

   ```
   ' Enter the following Declare statement on one, single line:
   Declare Function FindWindow% Lib "user" (ByVal lpClassName As Any,
      ByVal lpCaption As Any)
   ```

3. Add the following code to the form's Click event. This example demonstrates how you can find the instance of the application with a combination of the class name and/or the window's caption. In this example, the application will find an instance of the Windows calculator (CALC.EXE).

   ```
   Sub Form_Click ()
      lpClassName$ = "SciCalc"
      lpCaption$ = "Calculator"

      Print "Handle = ";FindWindow(lpClassName$, 0&)
      Print "Handle = ";FindWindow(0&, lpCaption$)
      Print "Handle = ";FindWindow(lpClassName$,lpCaption$)
   End Sub
   ```

4. Run this program with CALC.EXE running and without CALC.EXE running. If CALC.EXE is running, your application will print an arbitrary handle. If CALC.EXE is not running, your application will print zero as the handle.

Below are some class names of applications that are shipped with Windows:

Class Name	Application	
SciCalc	CALC.EXE	
CalWndMain	CALENDAR.EXE	
Cardfile	CARDFILE.EXE	
Clipboard	CLIPBOARD.EXE	
Clock	CLOCK.EXE	
CtlPanelClass	CONTROL.EXE	
XLMain	EXCEL.EXE	
Session	MS-DOS.EXE	
Notepad	NOTEPAD.EXE	
pbParent	PBRUSH.EXE	
Pif	PIFEDIT.EXE	
PrintManager	PRINTMAN.EXE	
Progman	PROGMAN.EXE	(Windows Program manager)
Recorder	RECORDER.EXE	
Reversi	REVERSI.EXE	
#32770	SETUP.EXE	
Solitaire	SOL.EXE	
Terminal	TERMINAL.EXE	
WFS_Frame	WINFILE.EXE	

```
MW_WINHELP          WINHELP.EXE
#32770              WINVER.EXE
OpusApp             WINWORD.EXE
MSWRITE_MENU        WRITE.EXE
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Scroll VB Text Box Programmatically and Specify Lines

Article ID: Q73371

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

By making a call to the Windows API function SendMessage, you can scroll text a specified number of lines or columns within a Microsoft Visual Basic for Windows text box. By using SendMessage, you can also scroll text programmatically, without user interaction. This technique extends Visual Basic for Windows' scrolling functionality beyond the built-in statements and methods. The sample program below shows how to scroll text vertically and horizontally a specified number of lines.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Note that Visual Basic for Windows itself does not offer a statement for scrolling text a specified number of lines vertically or horizontally within a text box. You can scroll text vertically or horizontally by actively clicking the vertical and horizontal scroll bars for the text box at run time; however, you do not have any control over how many lines or columns are scrolled for each click of the scroll bar. Text always scrolls one line or one column per click the scroll bar. Furthermore, no built-in Visual Basic for Windows method can scroll text without user interaction. To work around these limitations, you can call the Windows API function SendMessage, as explained below.

Example

To scroll the text a specified number of lines within a text box requires a call to the Windows API function SendMessage using the constant EM_LINESCROLL. You can invoke the SendMessage function from Visual Basic for Windows as follows:

```
r& = SendMessage& (hWd%, EM_LINESCROLL, wParam%, lParam&)
```

hWd%	The window handle of the text box.
wParam%	Parameter not used.
lParam&	The low-order 2 bytes specify the number of vertical lines to scroll. The high-order 2 bytes specify the number of horizontal columns to scroll. A positive value for lParam& causes text to scroll upward or to the left. A negative value causes text to scroll downward or to the right.

r& Indicates the number of lines actually scrolled.

The SendMessage API function requires the window handle (hWd% above) of the text box. To get the window handle of the text box, you must first set the focus on the text box using the SetFocus method from Visual Basic. Once the focus has been set, call the GetFocus API function to get the window handle for the text box. Below is an example of how to get the window handle of a text box.

```
' The following appears in the general declarations section of
' the form:
Declare Function GetFocus% Lib "USER" ()

' Assume the following appears in the click event procedure of a
' command button called Scroll.
Sub Command_Scroll_Click ()
     OldhWnd% =  Screen.ActiveControl.Hwnd
     ' Store the window handle of the control that currently
     ' has the focus.

     ' For Visual Basic 1.0 for Windows use the following line:
     ' OldhWnd% =  GetFocus ()

     Text1.SetFocus
     hWd% = GetFocus()
End Sub
```

To scroll text horizontally, the text box must have a horizontal scroll bar, and the width of the text must be wider than the text box width. Calling SendMessage to scroll text vertically does not require a vertical scroll bar, but the length of text within the text box should exceed the text box height.

Below are the steps necessary to create a text box that will scroll five vertical lines or five horizontal columns each time you click the command buttons labeled "Vertical" and "Horizontal":

1. From the File menu, choose New Project (press ALT, F, N).

2. Double-click Form1 to bring up the code window.

3. Add the following API declaration to the General Declarations section of Form1. Note that you must put all Declare statements on a separate and single line. Also note that SetFocus is aliased as PutFocus because there already exists a SetFocus method within Visual Basic for Windows.

   ```
   Declare Function GetFocus% Lib "user" () ' For Visual Basic 1.0 only.
   Declare Function PutFocus% Lib "user" Alias "SetFocus" (ByVal
                                                          hWd%)
   Declare Function SendMessage& Lib "user" (ByVal hWd%,
                                             ByVal wMsg%,
                                             ByVal wParam%,
                                             ByVal lParam&)
   ```

4. Create a text box called Text1 on Form1. Set the MultiLine property to True and the ScrollBars property to Horizontal (1).

5. Create a command button called Command1 and change the Caption to "Vertical".

6. Create a another command button called Command2 and change the Caption to "Horizontal".

7. From the General Declarations section of Form1, create a procedure to initialize some text in the text box as follows:

```
Sub InitializeTextBox ()
   Text1.Text = ""
   For i% = 1 To 50
      Text1.Text = Text1.Text + "This is line " + Str$(i%)

      ' Add 15 words to a line of text.
      For j% = 1 to 10
         Text1.Text = Text1.Text + " Word "+ Str$(j%)
      Next j%

      ' Force a carriage return (CR) and linefeed (LF).
      Text1.Text = Text1.Text + Chr$(13) + Chr$(10)

      x% = DoEvents()
   Next i%
End Sub
```

8. Add the following code to the load event procedure of Form1:

```
Sub Form_Load ()
   Call InitializeTextBox
End Sub
```

9. Create the actual scroll procedure within the General Declarations section of Form1 as follows:

```
' The following two lines must appear on a single line:
Function ScrollText& (TextBox As Control, vLines As Integer, hLines
                      As Integer)
   Const EM_LINESCROLL = &H406

   ' Place the number of horizontal columns to scroll in the high-
   ' order 2 bytes of Lines&. The vertical lines to scroll is
   ' placed in the low-order 2 bytes.
   Lines& = Clng(&H10000 * hLines) + vLines

   ' Get the window handle of the control that currently has the
   ' focus, Command1 or Command2.
   SavedWnd%  = Screen.ActiveControl.Hwnd
   ' For Visual Basic 1.0 use the following line instead of the one
   ' used above.
   ' SavedWnd% = GetFocus%()

   ' Set the focus to the passed control (text control).
   TextBox.SetFocus

   ' For Visual Basic 1.0, get the handle to current focus (text
```

```
        ' control).
        ' TextWnd% = GetFocus%()

        ' Scroll the lines.
        Success& = SendMessage(TextBox.HWnd, EM_LINESCROLL, 0, Lines&)
        ' For Visual Basic 1.0 use the following line instead of the one
        ' used above.
        ' Success& = SendMessage(TextWnd%, EM_LINESCROLL, 0, Lines&)

        ' Restore the focus to the original control, Command1 or
        ' Command2.
        r% = PutFocus% (SavedWnd%)

        ' Return the number of lines actually scrolled.
        ScrollText& = Success&

    End Function
```

10. Add the following code to the click event procedure of Command1 labeled "Vertical":

```
    Sub Command1_Click ()
       ' Scroll text 5 vertical lines upward.
       Num& = ScrollText&(Text1, 5, 0)
    End Sub
```

11. Add the following code to the click event procedure of Command2 labeled "Horizontal":

```
    Sub Command2_Click ()
        ' Scroll text 5 horizontal columns to the left.
        Num& = ScrollText&(Text1, 0, 5)
    End Sub
```

12. Run the program. Click the command buttons to scroll the text five lines or columns at a time.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd APrgWindow

WINAPI.TXT: Windows API Declarations and Constants for VB

Article ID: Q73694

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The file WINAPI.TXT supplies declarations for Microsoft Visual Basic programmers who want to call Windows API routines.

WINAPI.TXT can be found in the Software/Data Library by searching on the word BV0447, the Q number of this article, or S13104. BV0447 was archived using the PKware file-compression utility. After you decompress BV0447, you will obtain the following two files:

```
WINAPI.TXT
README.NOW
```

WINAPI.TXT is provided here in the assumption that you already have a reference for Windows API calls, such as the documentation provided with the Microsoft Windows Software Development Kit (SDK).

If you don't have a reference manual for Windows API calls, you can obtain the Visual Basic add-on kit number 1-55615-413-5, "Microsoft Windows Programmer's Reference" and Online Resource (which includes WINAPI.TXT on disk), available at a charge from Microsoft.

MORE INFORMATION

WINAPI.TXT can be found on CompuServe in the MSLANG forum (GO MSLANG), as well as in the Microsoft Software Library on CompuServe.

Contents of README.NOW

WINAPI.TXT is an ASCII text file containing the functions and constants in the Microsoft Windows 3.0 API, declared in the format used by Microsoft Visual Basic.

To use WINAPI.TXT, you must have the book "Microsoft Windows Programmer's Reference" for Windows version 3.0 (published by Microsoft Press, 1990), or you must have the reference manuals provided with the Microsoft Windows SDK.

WINAPI.TXT includes the following:

- External procedure declarations for all the Microsoft Windows API functions that can be called from Visual Basic.

- Global constant declarations for all the constants used by the Microsoft Windows API.
- Type declarations for the user-defined types (structures) used by the Microsoft Windows API.

WINAPI.TXT is too large to be loaded directly into a Visual Basic module. Attempting to load it directly into Visual Basic will cause an "Out of Memory" error message.

WINAPI.TXT is also too large for the Notepad editor supplied with Microsoft Windows, but it can be loaded by Microsoft Write. To use WINAPI.TXT, load it into an editor (such as Microsoft Write) that can handle large files. Copy the declarations you want and paste them into the global module in your Visual Basic application.

Note: Some of the Windows API declarations are very long. Some editors will wrap these onto a second line, and will copy them as multiple lines rather than a single line. Declarations in Visual Basic cannot span lines, so if you paste these as multiple lines, Visual Basic will report an error. If an error occurs, you can either adjust the margins in the editor before copying or remove the line break after pasting.

The global module is the recommended place for the declarations that you copy from the WINAPI.TXT file; however, you can place the external procedure declarations in the Declarations section of any form or module. You can also place the constant declarations anywhere in any module or form code if you remove the Global keyword. Type declarations must be placed in the global module.

Once you have pasted the declaration for a Windows API routine (as well as any associated constant and type declarations) into your application, you can call that routine as you would call any Visual Basic procedure.

For more information about declaring and calling external procedures, see Chapter 23, "Extending Visual Basic," in "Microsoft Visual Basic: Programmer's Guide."

Warning

Visual Basic cannot verify the data you pass to Microsoft Windows API routines. Calling a Microsoft Windows API routine with an invalid argument can result in unpredictable behavior: your application, Visual Basic, or Windows may crash or hang. When experimenting with Windows API routines, save your work often.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

PRB: Duplicate PostScript Font Names in VB Printer.Fonts List

Article ID: Q75092

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

When a PostScript printer driver is active in Microsoft Windows version 3.0, the Fonts(index%) property of Visual Basic's Printer object may return one or more duplicate font names at run time. This will not occur in either Visual Basic version 1.0 or 2.0 if you are using Microsoft Windows version 3.1.

CAUSE

This problem is caused by Microsoft Windows version 3.0 itself, not by Microsoft Visual Basic.

STATUS

Microsoft has confirmed this to be a problem with Microsoft Windows version 3.0. The problem was corrected in Microsoft Windows version 3.1.

MORE INFORMATION

The following program displays the list of font names available for the PostScript printer currently selected in the Windows Control Panel:

```
Sub Form_Click ()
   For J% = 0 to Printer.FontCount - 1
   Print Printer.Fonts(J%)
   Next J%
End Sub
```

In some cases, when a PostScript printer is active in Windows, one or more fonts are listed twice.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: APrgPrint

Determining Whether TAB or Mouse Gave a VB Control the Focus

Article ID: Q75411

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can determine whether a Microsoft Visual Basic for Windows control received the focus from a mouse click or a TAB keystroke by calling the Microsoft Windows API function GetKeyState in the control's GotFocus event procedure. By using GetKeyState to check if the TAB key is down, you can determine if the user pressed the TAB key to get to the control. If the TAB key was not used and the control does not have an access key, the user must have used the mouse to click the control to set the focus.

MORE INFORMATION

The GetKeyState Windows API function takes an integer parameter containing the virtual key code for the desired key states. GetKeyState returns an integer. If the return value is negative, the key has been pressed.

The following is a code example. To use this example, start with a new project in Visual Basic for Windows. Add a text box and a command button to Form1. Enter the following code in the project's GLOBAL.BAS module:

```
' Global Module.
Declare Function GetKeyState% Lib "User" (ByVal nVirtKey%)
Global Const VK_TAB = 9
```

Add the following code to the GotFocus event procedure for the Text1 text box control:

```
Sub Text1_GotFocus()
   If GetKeyState(VK_TAB) < 0 Then
     Text1.SelStart = 0
     Text1.SelLength = Len(Text1.Text)
   Else
     Text1.SelLength = 0
   End If
End Sub
```

Run the program. If you use the TAB key to move the focus from the command button to the text box, you should see the text in the text box selected. If you change the focus to the text box by clicking it with the mouse, the text will not be selected.

An access key is assigned by using an ampersand (&) in the control's caption property. If the control has an access key, you may also want to check the state of the virtual ALT key by using GetKeyState to see if the user used the access key to change the focus. The virtual key code for ALT, actually known as VK_MENU, is 12H (&H12).

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtRun APrgOther

How to Access Windows Initialization Files Within Visual Basic

Article ID: Q75639

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

There are several Microsoft Windows API functions that can manipulate information within a Windows initialization file. GetProfileInt, GetPrivateProfileInt, GetProfileString, and GetPrivateProfileString allow a Microsoft Visual Basic for Windows program to retrieve information from a Windows initialization file based on an application name and key name. WritePrivateProfileString and WriteProfileString are used to create/update items within Windows initialization files.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Windows initialization files contain information that defines your Windows environment. Examples of Windows initialization files are WIN.INI and SYSTEM.INI, which are commonly found in the C:\WINDOWS subdirectory. Microsoft Windows and applications for Microsoft Windows can use the information stored in these files to configure themselves to meet your needs and preferences. For a description of initialization files, review the WIN.INI file that comes with Microsoft Windows.

An initialization file is composed of at least an application name and a key name. The contents of Windows initialization files have the following format:

```
[Application name]
keyname=value
```

There are four API function calls (GetProfileInt, GetPrivateProfileInt, GetProfileString, and GetPrivateProfileString) that you can use to retrieve information from these files. The particular function to call depends on whether you want to obtain string or numerical data.

The GetProfile family of API functions is used when you want to get information from the standard WIN.INI file that is used by Windows. The WIN.INI file should be part of your Windows subdirectory (C:\WINDOWS). The GetPrivateProfile family of API functions is used to retrieve information from any initialization file that you specify. The formal arguments accepted by these API functions are described farther below.

The WriteProfileString and WritePrivateProfileString functions write information to Windows initialization files. WriteProfileString is used to modify the Windows initialization file, WIN.INI. WritePrivateProfileString is used to modify any initialization file that you specify. These functions search the initialization file for the key name under the application name. If there is no match, the function adds to the user profile a new string entry containing the key name and the key value specified. If the key name is found, it will replace the key value with the new value specified.

To declare these API functions within your program, include the following Declare statements in the global module or the General Declarations section of a Visual Basic for Windows form:

```
Declare Function GetProfileInt% Lib "Kernel"(ByVal lpAppName$,
        ByVal lpKeyName$, ByVal nDefault%)

Declare Function GetProfileString% Lib "Kernel" (ByVal lpAppName$,
        ByVal lpKeyName$, ByVal lpDefault$, ByVal lpReturnedString$,
        ByVal nSize%)

Declare Function WriteProfileString% Lib "Kernel"(ByVal lpAppName$,
        ByVal lpKeyName$, ByVal lpString$)

Declare Function GetPrivateProfileInt% Lib "Kernel"
        (ByVal lpAppName$, ByVal lpKeyName$, ByVal nDefault%,
        ByVal lpFileName$)

Declare Function GetPrivateProfileString% Lib "Kernel"
        (ByVal lpAppName$, ByVal lpKeyName$, ByVal lpDefault$,
        ByVal lpReturnedString$, ByVal nSize%, ByVal lpFileName$)

Declare Function WritePrivateProfileString% Lib "Kernel"
        (ByVal lpAppName$, ByVal lpKeyName$, ByVal lpString$,
        ByVal lpFileName$)
```

Note: Each Declare statement must be on a single line.

The formal arguments to these functions are described as follows:

Argument	Description
lpAppName$	Name of a Windows application that appears in the initialization file.
lpKeyName$	Key name that appears in the initialization file.
nDefault$	Specifies the default value for the given key if the key cannot be found in the initialization file.
lpFileName$	Points to a string that names the initialization file. If lpFileName does not contain a path to the file, Windows searches for the file in the Windows directory.
lpDefault$	Specifies the default value for the given key if the key cannot be found in the initialization file.

lpReturnedString$	Specifies the buffer that receives the character string.
nSize%	Specifies the maximum number of characters (including the last null character) to be copied to the buffer.
lpString$	Specifies the string that contains the new key value.

Below are the steps necessary to create a Visual Basic for Windows sample program that uses GetPrivateProfileString to read from an initialization file that you create. The program, based on information in the initialization file you created, shells out to the Calculator program (CALC.EXE) that comes with Windows. The sample program demonstrates how to use GetPrivateProfileString to get information from any initialization file.

1. Create an initialization file from a text editor (for example, you can use the Notepad program supplied with Windows) and save the file under the name of "NET.INI". Type in the following as the contents of the initialization file (NET.INI):

   ```
   [NetPaths]
   WordProcessor=C:\WINWORD\WINWORD.EXE
   Calculator=C:\WINDOWS\CALC.EXE
   ```

   Note: If CALC.EXE is not in the C:\WINDOWS subdirectory (as indicated after "Calculator=" above), replace C:\WINDOWS\CALC.EXE with the correct path.

2. Save the initialization file (NET.INI) to the root directory of your hard drive (such as C:\) and exit the text editor.

3. Start Visual Basic for Windows.

4. Create a form called Form1.

5. Create a push button called Command1.

6. Within the Global Declaration section of Form1, add the following Windows API function declarations. Note that the Declare statement below must appear on a single line.

   ```
   Declare Function GetPrivateProfileString% Lib "kernel"
     (ByVal lpAppName$, ByVal lpKeyName$,ByVal lpDefault$,
     ByVal lpReturnString$,ByVal nSize%, ByVal lpFileName$)
   ```

7. Within the (Command1) push button's click event add the following code:

```
Sub Command1_Click ()
   '* If an error occurs during SHELL statement then handle the error.
   On Error GoTo FileError

   '* Compare these to the NET.INI file that you created in step 1
   '* above.
   lpAppName$ = "NetPaths"
```

```
    lpKeyName$ = "Calculator"
    lpDefault$ = ""
    lpReturnString$ = Space$(128)
    Size% = Len(lpReturnString$)

    '* This is the path and name the NET.INI file.
    lpFileName$ = "c:\net.ini"

    '* This call will cause the path to CALC.EXE (that is,
    '* C:\WINDOWS\CALC.EXE) to be placed into lpReturnString$. The
    '* return value (assigned to Valid%) represents the number of
    '* characters read into lpReturnString$. Note that the
    '* following assignment must be placed on one line.
    Valid% = GetPrivateProfileString(lpAppName$, lpKeyName$,
                                     lpDefault$, lpReturnString$,
                                     Size%, lpFileName$)

    '* Discard the trailing spaces and null character.
    Path$ = Left$(lpReturnString$, Valid%)

    '* Try to run CALC.EXE. If unable to run, FileError is called.
    Succ% = Shell(Path$, 1)
    Exit Sub

FileError:
    MsgBox "Can't find file", 16, "Error lpReturnString"
    Resume Next

End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgINI

How to Print the ASCII Character Set in Visual Basic

Article ID: Q75857

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The default font used by Visual Basic is the standard ANSI character set. To display the ASCII character set, which is more commonly used in MS-DOS-based applications, you must call the two Windows API functions GetStockObject and SelectObject. In addition, to display the unprintable characters such as TAB, linefeed, and carriage return characters, you need to use the TextOut Windows API function because the standard Visual Basic printer object does not display the unprintable characters. By using the Windows API TextOut function, you circumvent the Visual Basic printer object and therefore allow all the characters to be displayed.

MORE INFORMATION

Windows supports a second character set, referred to as the OEM character set. This is generally the character set used internally by MS-DOS for screen display at the MS-DOS prompt. The character codes 32 to 127 are usually identical for the OEM, ASCII, and ANSI character sets. The ANSI characters represented by the remaining character codes (codes 0 to 31 and 128 to 255) are generally different from characters represented by the OEM and ASCII character sets. However, the OEM and ASCII character sets are identical for these ranges. Under the ASCII and OEM character sets, the character codes 128 to 255 correspond to the extended ASCII character set, which includes line drawing characters, graphics characters, and special symbols. The characters represented by this range of character codes generally differ between the ASCII (or OEM) and ANSI character sets.

To change the selected font from ANSI to the OEM ASCII font, you must get a handle to the OEM character set by calling GetStockObject. When this handle is passed as an argument to SelectObject, the ANSI font will be replaced by the OEM ASCII font. This API function also returns the handle to the font object previously used. Once you finish displaying the desired characters, you should call SelectObject again to reselect the original font object.

Note that there is also an API function called DeleteObject. This function need not be called to delete a stock object. The purpose of this API function is to delete objects loaded with the API function GetObject.

Here is the syntax for the functions:

GetStockObject% (nIndex%)

nIndex%

   Specifies the type of stock object desired. Use the constant
   OEM_FIXED_FONT to retrieve the handle to the OEM character set.
   The value of this constant is 10.

Return Value

   The return value identifies the desired logical object if the
   function is successful. Otherwise, it is NULL.

SelectObject% (hDC%, hObject%)
---
hDC%

   Identifies the device context.

hObject%

   Identifies the object to be selected. Use the return value from
   GetStockObject% (above) to select the OEM character set.

Return Value

   The return value identifies the handle to the object previously
   used. This value should be saved in a variable such that
   SelectObject can be called again to restore the original object
   used. It is NULL if there is an error.

Step-by-Step Example

The following example steps demonstrate how to create a program that prints
ASCII characters.

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N)
   if Visual Basic is already running. Form1 is created by default.

3. Add a command button (Command1) to Form1.

4. Add the following code to the General Declarations section of Form1:

```
   ' Enter each Declare statement on one, single line.
   Declare Function GetStockObject% Lib "GDI" (ByVal nIndex%)
   Declare Function SelectObject% Lib "GDI" (ByVal hDC%, ByVal hObject%)
   Declare Function TextOut Lib "GDI" (ByVal hDC As Integer,
      ByVal X As Integer, ByVal Y As Integer, ByVal lpString As String,
      ByVal nCount As Integer) As Integer
```

5. Place the following code in the Command1 click event procedure:

```
   Sub Command1_Click ()

      Const OEM_FIXED_FONT = 10
      Const PIXEL = 3

      Dim hOEM As Integer   '*handle the OEM Font Object
      Dim Y, H As Single
```

```
'*save the scale mode so that you can reset later
Saved% = Form1.ScaleMode

'*alter the current scale mode
Form1.ScaleMode = PIXEL

'* get the character height and subtract the external leading
H = Form1.TextHeight(Chr$(200)) - 1

'* get the handle to the desired font
hOEM = GetStockObject(OEM_FIXED_FONT)

'* select the object relating to the font handle
PreviousObject% = SelectObject%(Form1.hDC, hOEM)

'* if successful then print the desired characters.
If PreviousObject% Then

   '* establish border
   Edge$ = "0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 "

   '* initialize output location
   xMark = 10
   yMark = 10

   '* position cursor & print top border
   Form1.CurrentX = xMark
   Form1.CurrentY = yMark
   '* print top ruler edge
   T$ = "  " + Edge$ + "  "
   ret% = TextOut(Form1.hDC, yMark, xMark, T$, Len(T$))

   '* Cycle through 256 characters beginning at character 0
   For Row% = 0 To 15

      '* prep left border
      T$ = Mid$(Edge$, (Row% * 2) + 1, 2)

      '* assemble string of characters
      For Col% = 0 To 15
          Ch = (Row% * 16) + Col%
          T$ = T$ + Chr$(Ch) + " "
      Next

      '* prep right border
      T$ = T$ + Mid$(Edge$, (Row% * 2) + 1, 2)

      '* prepare for display at next row
      xMark = xMark + H

      '* print the assembled string of characters
      ret% = TextOut(Form1.hDC, yMark, xMark, T$, Len(T$))

   Next

   '* prepare for display at next row
```

```
         xMark = xMark + H

         '* print bottom border
         T$ = "  " + Edge$ + "  "
         ret% = TextOut(Form1.hDC, yMark, xMark, T$, Len(T$))

         '* reinstate the previous font
         hOEM = SelectObject(Form1.hDC, PreviousObject%)

      Else

         '* SelectObject was unsuccessful
         MsgBox "Couldn't Find OEM Fonts", 48

      End If

      '* reset the scale mode
      Form1.ScaleMode = Saved%

   End Sub
```

6. From the Run menu, choose Start.

7. Click the Command1 button.

When the Command1 button is clicked or selected, a small box with a double border will be drawn in the upper-left corner of the screen. The box is drawn using characters associated with the extended ASCII character set.

ASCII and ANSI Character Sets

For a listing of the ASCII and ANSI character sets, see the Visual Basic Help menu.

American Standard Code for Information Interchange (ASCII) is the 7-bit character set widely used to represent letters and symbols found on a standard United States keyboard. The ASCII character set is the same as the first 128 characters (0 to 127) in the American National Standards Institute (ANSI) character set. The ANSI character set uses all 8 bits in a byte, and includes 256 characters (0 to 255). Characters 128 to 255 are sometimes called the extended-ASCII characters.

REFERENCES
==========

1. "Programming Windows: the Microsoft Guide to Writing Applications for Windows 3," by Charles Petzold (published by Microsoft Press, 1990)

2. "Peter Norton's Windows 3.0 Power Programming Techniques," by Peter Norton & Paul Yao (published by Bantam Computer Books, 1990)

3. "Microsoft Windows 3.0 Software Development Kit: Reference Volume 1"

4. The WINSDK.HLP file shipped with Microsoft Windows 3.0 Software

Development Kit.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgPrint

How to Clear a VB Combo Box with a Windows API Function

Article ID: Q76513

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article explains how to instantly clear the contents of a Visual Basic combo box by sending the combo box a CB_RESETCONTENT message.

MORE INFORMATION

No single command within Visual Basic will clear out the entries of a combo box. However, you can clear all entries at once with a simple While loop, as follows:

```
Do While Combo1.ListCount > 0
    Combo1.RemoveItem 0
Loop
```

If you want a single command to clear all combo box entries at once, you can use the SendMessage Windows API function. The arguments to SendMessage with the CB_RESETCONTENT parameter are

```
SendMessage(hWnd%, wMsg%, wParam%, lParam&)
```

where

```
hWnd%    Identifies the window that is to receive the message
wMsg%    The message to be sent  (CB_RESETCONTENT = &H411)
wParam%  Is not used (NULL)
lParam&  Is not used (NULL)
```

Specifying wMsg% equal to &H411 sends a CB_RESETCONTENT message to the combo box. This removes all strings from the combo box and frees any memory allocated for those strings.

To get hWnd%, the handle to the target window, you must call the Windows API function GetFocus. This method will return the handle to the control that currently has focus. For a combo box with a Style property of 2 (Dropdown List), this will return the handle to the combo box that you want to send the message to. For other styles of combo boxes, the focus is set to a child edit control that is part of the combo box, and you must use the GetParent() Windows API function to get the handle to the combo box itself.

The following steps demonstrate how to delete entries from a combo box:

1. Create a combo box called Combo1 on Form1.

2. Declare the following Windows API functions at the module level or in the Global section of your project:

```
Declare Function SendMessage% Lib "user" (ByVal hWnd%,_
                          ByVal wMsg%,_
                          ByVal wParam%,_
                          ByVal lParam&)
Declare Function GetFocus% Lib "user" ()
Declare Function PutFocus% Lib "user" Alias "SetFocus" (ByVal hWnd%)
Declare Function GetParent% Lib "user" (ByVal hWnd%)
```

(Note: Each Declare statement must be written on one line, leaving out the underscore (_) line-continuation symbol shown above.)

3. Declare the following constants in the same section:

```
Global Const WM_USER = &H400
Global Const CB_RESETCONTENT = WM_USER + 11
Global Const DROP_DOWN_LIST = 2
```

4. Place some entries into the combo box in the Form_Load procedure:

```
Sub Form_Load ()
    For i = 1 To 10
     Combo1.AddItem Format$(i)  'Put something into combo box.
    Next
End Sub
```

5. Create a Sub within the (Declarations) section of the Form1 Code window with the following code:

```
Sub ClearComboBox (Combo As Control)
    hWndOld% = GetFocus()
    Combo.SetFocus
    If Combo.Style = DROP_DOWN_LIST then
    x = SendMessage(GetFocus(), CB_RESETCONTENT, 0, 0)
    Else
    x = SendMessage(GetParent(GetFocus()), CB_RESETCONTENT, 0, 0)
    End If
    Suc% = PutFocus(hWndOld%)
End Sub
```

6. Within an event procedure, call ClearComboBox with the name of the Combo box as a parameter:

```
Sub Form_Click ()
   ClearComboBox Combo1
End Sub
```

7. Run the program and click anywhere on Form1. This will clear the combo box.

Reference(s):

"Programming Windows: the Microsoft Guide to Writing Applications for Windows 3," Charles Petzold. Microsoft Press, 1990

"Microsoft Windows Software Development Kit Reference Volume 1," version 3.0

WINSDK.HLP file shipped with Microsoft Windows 3.0 Software Development Kit

Additional reference words: 1.00 3.00
KBCategory:
KBSubcategory: APrgOther

BUG: Bad Text in Long Right-Aligned Labels in Windows ver 3.0

Article ID: Q76515

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Windows versions 3.0 and 3.1

SYMPTOMS

When you use Visual Basic with Windows version 3.0, the caption of a right-aligned label that is set to a length exceeding 255 characters displays unusual (incorrect) characters. A left-aligned or centered caption displays correctly, and all captions display correctly when using Visual Basic with Windows version 3.1.

STATUS

Microsoft has confirmed this to be a problem in Windows version 3.0. This problem was corrected in Windows version 3.1.

MORE INFORMATION

Steps to Reproduce Problem

1. In the Visual Basic environment (VB.EXE), place a label on a blank form.

2. Add the following code to the form's Form_click event procedure:

```
Label1.alignment = 1  'right justified
Label1.caption = string$ (277, "k")
Label1.refresh
```

3. From the Run menu, choose Start or press the F5 key.

4. Click anywhere inside the form except on the label to see unexpected characters appear in the rightmost portion of the caption.

Additional reference words: 1.00 2.00 3.00 garbage corrupted
KBCategory:
KBSubcategory: APrgOther

Using Windows API Functions to Better Manipulate Text Boxes

Article ID: Q76518

The information in this article applies to:

- Microsoft Visual Basic for Windows, versions 1.0, 2.0, and 3.0

SUMMARY
=======

By calling Windows API functions from Microsoft Visual Basic for Windows, you can retrieve text box (or edit control) information that you cannot obtain using only Visual Basic for Windows' built-in features. Note that in Visual Basic versions 2.0 and 3.0 for Windows, you can use the new HWND property of a text box instead of calling the GetFocus() function.

This article supplies a sample program that performs the following useful features (making use of the Windows message constants shown in parentheses, obtained by calling Windows API routines):

- Copy a specific line of text from the text box (EM_GETLINE).
- Retrieve the number of lines within the text box (EM_GETLINECOUNT).
- Position the cursor at a specific character location (EM_GETSEL) in the text box.
- Retrieve the line number of a specific character location in the text box (EM_LINEFROMCHAR).
- Retrieve the amount of lines before a specified character position in the text box (EM_LINEINDEX).
- Retrieve the amount of characters in a specified line in the text box (EM_LINELENGTH).
- Replace specified text with another text string (EM_REPLACESEL).

For a separate article that explains how to specify the amount of text allowable within a text control, query on the following word in the Microsoft Knowledge Base:

   EM_LIMITTEXT

MORE INFORMATION
================

Note that as of 3/25/92, the code below corrects the VBKNOWLG.HLP file Knowledge Base shipped with the Microsoft Professional Toolkit for Visual Basic version 1.0 for Windows.

Note also that using the SelStart, SelLength, and SelText properties may be easier than using EM_GETSEL and EM_REPLACESEL below.

The Windows API file USER.EXE defines the SendMessage function that will return or perform a specific event on your edit control. To create an example that will display specific information about your edit control, do the following:

1. Create a form (Form1), and add the following controls and properties:

Control	Control Name	Height	Left	Top	Width
Label	aGetLine		360	120	
Label	aGetLineCount		360	480	
Label	aGetSel		360	840	
Label	aLineFromChar		360	1200	
Label	aLineIndex		360	1560	
Label	aLineLength		360	1920	
Label	aReplaceSel		360	2280	
Command	Command1	375	360	2640	1815
Text	Text1	1815	2640	480	3495
Text	Text2	375	2520	2640	3615

2. Set each label's AutoSize property to True.

3. Set the Text1.MultiLine property to True.

4. Change the Command1.Caption to "Insert this text --->".

5. Add the following code to the global Declarations section:

```
Declare Function GetFocus% Lib "user" ()
' Enter the following Declare as one, single line:
Declare Function SendMessage& Lib "user"(ByVal hWnd%, ByVal wMsg%,
      ByVal wParam%, ByVal lParam As Any)
' lParam is actually a double word, or long, but declaring
' lParam "As Any" allows flexibility for certain cases of
' using SendMessage.
```

6. After adding the code listed below to your form, run the program. Whenever a key is released, the labels will be updated with the new information about your text box.

```
Sub Form_Load ()
   Show
   X% = fReplaceSel("")   '* Used to display the correct text.
End Sub

Sub Text1_KeyUp (KeyCode As Integer, Shift As Integer)
   '* Update the text control information whenever the key
   '* is pressed and released.
   CharPos& = fGetSel()
   LineNumber& = fLineFromChar(CharPos&)
   X% = fGetLine(LineNumber&)
   X% = fGetLineCount()
   X% = fLineIndex(LineNumber&)
   X% = fLineLength(CharPos&)
End Sub

Sub Command1_Click ()
   '* This routine will insert a line of text at the current location
   '* of the caret.

   D$ = Text2.text
   CharPos& = fGetSel()
   X% = fReplaceSel(D$)
```

```
   X% = fSetSel(CharPos&)

   '* Text has been inserted at the caret location.  No update the
   '* text controls information.
   Call Text1_KeyUp(0, 0)
   Text1.SetFocus
End Sub

Function fGetLineCount& ()
   '* This function will return the number of lines in the edit control.
   Const EM_GETLINECOUNT = &H400 + 10

   Text1.SetFocus
   ' In versions 2.00 and 3.00, you need to use a long integer to avoid
   ' a bad DLL calling convention error message. As an alternative,
   ' you can use the new HWND property instead of GetFocus():
   Pos& = SendMessage(GetFocus(), EM_GETLINECOUNT, 0&, 0&)
   ' Use the following Pos& if you have Visual Basic version 1.0:
   ' Pos& = SendMessage(GetFocus(), EM_GETLINECOUNT, 0%, 0%)
   aGetLineCount.Caption = "GetLineCount = " + Str$(Pos&)
   fGetLineCount = Pos&
End Function

Function fGetLine (LineNumber As Long)
   '* This function copies a line of text specified by LineNumber
   '* from the edit control. The first line starts at zero.

   Const MAX_CHAR_PER_LINE = 80
   Const EM_GETLINE = &H400 + 20

   byteLo% = MAX_CHAR_PER_LINE And (255)  '[changed 3/25/92]
   byteHi% = Int(MAX_CHAR_PER_LINE / 256) '[changed 3/25/92, two lines:]
   Buffer$ = chr$(byteLo%) + chr$(byteHi%) + Space$(MAX_CHAR_PER_LINE-2)

   Text1.SetFocus
   Pos& = SendMessage(GetFocus(), EM_GETLINE, CINT(LineNumber), Buffer$)
   aGetLine.Caption = "GetLine = " + Buffer$
   fGetLine = Pos&

End Function

Function fGetSel& ()
   '* This function returns the starting/ending position of the
   '* current selected text.  This is the current location of the
   '* cursor if start is equal to ending.
   '* LOWORD-start position of selected text
   '* HIWORD-first no selected text

   Const EM_GETSEL = &H400 + 0

   Text1.SetFocus
   location& = SendMessage(GetFocus(), EM_GETSEL, 0, 0&)
   ending% = location& \ &H10000
   starting% = location& And &H7FFF
   aGetSel.Caption = "Caret Location = " + Str$(starting%)
   fGetSel = location& mod 65536
End Function
```

```
Function fLineFromChar& (CharPos&)
   '* This function will return the line number of the line that
   '* contains the character whose location(index) specified in the
   '* third argument of SendMessage. If the third argument is -1,
   '* then the number of the line that contains the first character
   '* of the selected text is returned. If start = end from GetSel,
   '* then the current caret location is used. Line numbers start
   '* at zero.

   Const EM_LINEFROMCHAR = &H400 + 25

   Text1.SetFocus
   Pos& = SendMessage(GetFocus(), EM_LINEFROMCHAR, CINT(CharPos&), 0&)
   aLineFromChar.Caption = "Current Line = " + Str$(Pos&)
   fLineFromChar = Pos&
End Function

Function fLineIndex (LineNumber As Long)
   '* This function will return the number of bytes that
   '* precede the given line. The returned number reflects the CR/LF
   '* after each line. The third argument to SendMessage specifies
   '* the line number, where the first line number is zero. If the
   '* third argument to SendMessage is -1, then the current line
   '* number is used.

   Const EM_LINEINDEX = &H400 + 11

   Text1.SetFocus
   Pos& = SendMessage(GetFocus(), EM_LINEINDEX, CINT(LineNumber), 0&)
   aLineIndex.Caption = "#Char's before line = " + Str$(Pos&)
   fLineIndex = Pos&
End Function

Function fLineLength& (CharPos As Long)
   '* This function will return the length of a line in the edit
   '* control. CharPos specifies the index of the character that
   '* is part of the line that you would like to find the length. If
   '* this argument is -1, the current selected character is used as
   '* the index.

   Const EM_LINELENGTH = &H400 + 17
   Text1.SetFocus
   Pos& = SendMessage(GetFocus(), EM_LINELENGTH, CINT(CharPos), 0&)
   aLineLength.Caption = "LineLength = " + Str$(Pos&)
   fLineLength = Pos&
End Function

Function fSetSel& (Pos&)
   '* This function selects all characters in the current text that
   '* are within the starting and ending positions given by
   '* Location. The low word is the starting position and the high
   '* word is the ending position. If you set start to end, this
   '* can be used to position the cursor within the edit control.

   Const EM_SETSEL = &H400 + 1
   location& = Pos& * 2 ^ 16 + Pos&
```

```
      Text1.SetFocus
      X% = SendMessage(GetFocus(), EM_SETSEL, 0%, location&)
      fSetSel = Pos&
   End Function

   Function fReplaceSel (Buffer$)
      '* This function will replace the current selected text with the
      '* new text specified in Buffer$. You must call SendMessage with
      '* the EM_GETSEL constant to select text.

      Const EM_REPLACESEL = &H400 + 18
      Text1.SetFocus
      Pos& = SendMessage(GetFocus(), EM_REPLACESEL, 0%, Buffer$)
      aReplaceSel.Caption = "String inserted = " + Buffer$
      fReplaceSel = Pos&
   End Function
```

Reference(s):

"Programming Windows: The Microsoft Guide to Writing Applications for Windows 3," Charles Petzold, Microsoft Press, 1990

"Microsoft Windows Software Development Kit: Reference Volume 1," version 3.0

WINSDK.HLP file shipped with Microsoft Windows 3.0 Software Development Kit

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

PRB: No Events Generated When Msgbox Active

Article ID: Q76557

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SYMPTOMS

Visual Basic does not invoke the Paint event or any other event for controls or forms while a MsgBox or InputBox is active.

CAUSE

This behavior is by design. All events (including the timer control event) are disabled while a MsgBox is showing. The purpose is to block operations that could cause problems.

WORKAROUND

To compensate for Paint events not firing, you can set the form AutoRedraw property to True and paint the form from the Form_Load event.

To display a MsgBox-like dialog and allow all events to occur, you can:

- Call the Windows API function MessageBox.

- Display a modal form (formN.Show 1), which looks like the MsgBox dialog.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Demonstrate Behavior

1. Start Visual Basic or from the File menu, choose New Project if Visual Basic is already running.

2. Add the following code to the general declarations section:

```
   ' Enter the following Declare statement on one, single line:
   Declare Function MessageBox Lib "User" (ByVal hWnd As Integer, ByVal
lpText
      As String, ByVal lpCaption As String, ByVal wType As Integer) As
Integer
```

2. Add the following code to the Form_Click event:

```
Sub Form_Click ()
   MsgBox "move me", 0, "MsgBox"
   unused = MessageBox(hWnd, "move me", "MessageBox", 0)
End Sub
```

3. Add the following code to the Form_Paint event:

```
Sub Form_Paint ()
   Line (0, 0)-(ScaleWidth - 1, ScaleHeight - 1), &HFF, BF
End Sub
```

4. Run the application. Click the form to display the MsgBox dialog. When you drag this dialog box around on the form, the Paint event is not fired and the area previously occupied by the MsgBox is not updated.

5. Click OK to display the MessageBox API dialog. When you drag this dialog box around, the form is repainted.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Create and Use a Custom Cursor in Visual Basic; Win SDK

Article ID: Q76666

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Using a graphics editor, the Microsoft Windows Software Development Kit (SDK), and the Microsoft C compiler, you can create a dynamic-link library (DLL) containing mouse cursors that can be used in a Microsoft Visual Basic for Windows application. By making calls to the Windows API functions LoadLibrary, LoadCursor, SetClassWord, and GetFocus, you can display a custom cursor from within a Visual Basic for Windows application. Below are the steps necessary to a create a custom cursor and a Visual Basic for Windows application to use this custom cursor.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Setting a custom cursor in a Visual Basic for Windows application requires a call to the Windows API function LoadLibrary to load the custom DLL containing the cursor resource(s). A call to LoadCursor is then required to load a single cursor contained in the DLL. The return value of the LoadCursor function is a handle to the custom cursor. This handle can be passed as an argument to the API function SetClassWord with the constant GCW_HCURSOR. SetClassWord also requires a window handle (hWnd) to the object (form or control) for which the cursor is to be set. The hWnd of a form is available via the hWnd run-time method. For example, the statement FWnd = Form1.hWnd will return the hWnd of Form1 to the variable FWnd. The hWnd of a control can be obtained by first using the SetFocus method on the control to give it the input focus and then calling the API function GetFocus. GetFocus returns the hWnd of the object with the current input focus.

A custom cursor always takes the place of the system cursor. The MousePointer property of a form or control to receive the custom cursor must be set to zero (system). Any other value for this property will result in the selected cursor being displayed, not the custom cursor.

Because the cursor is defined as part of a window class, any change to the window class will be reflected across any control or form that uses that class. For example, if the MousePointer property for two command buttons is zero (system) and a custom cursor is set for one of the command buttons, both of the command buttons will receive the custom cursor. To guarantee a custom cursor for each control requires

that the cursor be set by calling SetClassWord in the MouseMove event procedure of the control.

Some controls, such as command buttons, do not contain a MouseMove event procedure. A custom mouse pointer for these types of controls can be set by initiating a timer event. Within the timer event, calls to the API functions GetCursorPos and WindowFromPoint can be made to determine if the mouse is over the control or not. If the WindowFromPoint API call returns the hWnd of the control, then the mouse is over the control. A call to SetClassWord can then be made to set the custom cursor for the control.

Below is an example of using the Windows SDK and C Compiler to create a DLL containing cursor resources. Further below are the steps necessary to create a Visual Basic for Windows program to use the cursor resources.

If you do not have the Windows SDK but have a pre-compiled DLL containing cursor resources, skip to the steps below outlining how to create a Visual Basic application to use the custom cursor resources.

1. Using a graphics editor such as Microsoft Windows SDK Paint program, create two cursor images. Save the images separately as CURS1.CUR and CURS2.CUR, respectively.

2. Using any text editor, create a C source file containing the minimum amount of code for a Windows DLL. The source code must contain the functions LibEntry and WEP (Windows exit procedure). Below is an example of the C source file:

```
#include <windows.h>
int _far _pascal LibMain (HANDLE hInstance,
                          WORD wDataSeg,
                          WORD cbHeapSize,
                          LPSTR lpszCmdLine)
{
   return(1);
}

int _far _pascal WEP (int nParameter)
{
   return(1);
}
```

3. Save the file created in step 2 above as CURSORS.C.

4. Using any text editor, create a definition file (.DEF) for the DLL. Enter the following as the body of the .DEF file:

```
LIBRARY CURSORS

DESCRIPTION 'DLL containing cursor resources'

EXETYPE WINDOWS

STUB 'WINSTUB.EXE'
```

```
CODE MOVEABLE DISCARDABLE

DATA MOVEABLE SINGLE

HEAPSIZE 0

EXPORTS
   WEP   @1 RESIDENTNAME
```

5. Save the file created in step 4 above as CURSORS.DEF.

6. Using a text editor, create a resource file for the cursors created in step 1 above. Enter the following as the body of the .RC file:

```
Cursor1 CURSOR CURS1.CUR
Cursor2 CURSOR CURS2.CUR
```

7. Save the file created in step 6 above as CURSORS.RC.

8. Compile CURSORS.C from the MS-DOS command line as follows:

```
CL /AMw /c /Gsw /Os /W2 CURSORS.C
```

9. Link the program from the MS-DOS command line as follows (enter the following two lines on a single line):

```
LINK /NOE /NOD cursors.obj +
           LIBENTRY.OBJ,,,MDLLCEW.LIB+LIBW.LIB,CURSORS.DEF;
```

   This will create the file CURSORS.EXE.

10. Add the cursor resources created in step 1 above to the .EXE file created in step 9 above by invoking the Microsoft Resource Compiler (RC.EXE) from the MS-DOS command line as follows:

```
RC CURSORS.RC
```

11. Rename CURSORS.EXE to CURSORS.DLL from the MS-DOS command line as follows:

```
REN CURSORS.EXE CURSORS.DLL
```

Below are the steps necessary to create a Visual Basic for Windows application that uses the cursor resources created in the steps above.

Important

When running the Visual Basic for Windows program created by following the steps below, it is important to terminate the application from the system menu, NOT the Run End option from the file menu. When Run End is chosen from the file menu, the unload event procedure is not executed. Therefore, the system cursor is not restored and the custom cursor will remain present at design time. Using Visual Basic version 1.0 for Windows, avoid terminating the program from the Program Manager (PROGMAN.EXE) task list. The unload event procedure is also not called when a program is terminated from the task list in Visual Basic version 1.0 for Windows.

1. Start Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 will be created by default.

2. Put a picture control (Picture1) on Form1.

3. Put a command button (Command1) on Form1.

4. Put a timer control (Timer1) on Form1.

5. Enter the following code in the Global Module:

```
Type PointType
    x As Integer
    y As Integer
End Type
```

6. Enter the following code in the General Declaration section of Form1:

```
DefInt A-Z
' Each of the following Declare statements must appear on one line.
Declare Function LoadLibrary Lib "kernel" (ByVal LibName$)
Declare Function LoadCursor Lib "user"    (ByVal hInstance, ByVal
                                           CursorName$)
Declare Function SetClassWord Lib "user" (ByVal hWnd, ByVal
                                          nIndex, ByVal NewVal)
Declare Function DestroyCursor Lib "user" (ByVal Handle)
Declare Function GetFocus Lib "user" ()
Declare Function PutFocus Lib "user" Alias "SetFocus" (ByVal hWnd)
Declare Sub GetCursorPos Lib "user" (p As PointType)
Declare Function WindowFromPoint Lib "user" (ByVal y, ByVal x)
Const GCW_HCURSOR = (-12)
Dim SysCursHandle
Dim Curs1Handle
Dim Curs2Handle
Dim Pic1hWnd
Dim Command1hWnd
Dim p As PointType
```

7. Enter the following code in the Form_Load event procedure of Form1:

```
Sub Form_Load ()
    Form1.Show
    DLLInstance = LoadLibrary("CURSORS.DLL")
    Curs1Handle = LoadCursor(DLLInstance, "Cursor1")
    Curs2Handle = LoadCursor(DLLInstance, "Cursor2")
    SysCursHandle=SetClassWord(Form1.hWnd,GCW_HCURSOR,Curs2Handle)

    ' Get the current control with the input focus.
    CurrHwnd = GetFocus()

    ' Get the Window handle of Picture1.
    Picture1.SetFocus
    Pic1hWnd = Picuture1.GetFocus()
```

```
        ' Get the Window handle of Command1.
        Command1.SetFocus
        Command1hWnd = GetFocus()

        ' Restore the focus to the control with the input focus.
        r = PutFocus(CurrHwnd)
        timer1.interval = 1   ' One millisecond.
        timer1.enabled = -1
    End Sub
```

8. Enter the following code in the Form_Unload event procedure of Form1:

```
  Sub Form_Unload (Cancel As Integer)
   ' Restore the custom cursors to the system cursor:
     LastCursor =SetClassWord(Form1.hWnd, GCW_HCURSOR, SysCursHandle)
     LastCursor = SetClassWord(Pic1hWnd, GCW_HCURSOR, SysCursHandle)
     LastCursor=SetClassWord(Command1hWnd, GCW_HCURSOR,SysCursHandle)
   ' Delete the cursor resources from memory:
     Success = DestroyCursor(Curs1Handle)
     Success = DestroyCursor(Curs2Handle)
   End Sub
```

9. Enter the following code in the Timer1_Timer event procedure of Timer1:

```
    Sub Timer1_Timer ()

        ' Get the current (absolute) cursor position.
        Call GetCursorPos(p)

        ' Find out which control the midpoint of the cursor is over.
        ' The cursor is 32 x 32 pixels square. Change the class word
        ' of the control to the appropriate cursor.
        Select Case WindowFromPoint(p.y + 16, p.x + 16)

         Case Form1.hWnd
         ' Each of the following statements must appear on one line.
             LastCursor = SetClassWord(Form1.hWnd, GCW_HCURSOR,
                                       Curs2Handle)
             LastCursor = SetClassWord(Command1hWnd, GCW_HCURSOR,
                                       Curs2Handle)
             LastCursor = SetClassWord(Pic1hWnd, GCW_HCURSOR,
                                       Curs2Handle)

          Case Command1hWnd

             LastCursor = SetClassWord(Form1.hWnd, GCW_HCURSOR,
                                       Curs1Handle)
             LastCursor = SetClassWord(Command1hWnd, GCW_HCURSOR,
                                       Curs1Handle)

          Case Pic1hWnd

             LastCursor = SetClassWord(Form1.hWnd, GCW_HCURSOR,
                                         Curs1Handle)
```

```
                LastCursor = SetClassWord(Pic1hWnd%, GCW_HCURSOR,
                                          Curs1Handle)
        End Select
    End Sub
```

Run the program. The form should receive the "Cursor2" cursor and the controls Command1 and Picture1 should receive the "Cursor1" cursor as the mouse cursor is moved about the form.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap APrgOther

Terminating Windows from a Visual Basic Application

Article ID: Q76981

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Visual Basic SendKeys function cannot be used to close Program Manager in order to terminate Windows. To correctly close Program Manager, you must invoke the ExitWindows API function, as shown below.

Many software setup or installation programs are designed to exit Windows, and then restart Windows when the setup or installation is complete. You can make a Visual Basic program automatically exit Windows and then restart Windows by passing the EW_RESTARTWINDOWS value to the ExitWindows API function.

MORE INFORMATION

You may want to terminate the current Windows session by closing the Program Manager from within a Visual Basic application. You may think that you can activate the Program Manager control menu and send the appropriate key sequences using the Visual Basic SendKeys function. However, this method will not work because after the Close menu item is chosen, a system modal dialog box is opened that prompts you to save changes to Program Manager. A system modal dialog box locks out ALL other programs until it is satisfied. Therefore, the keystroke you send by using the SendKeys function will never arrive in the dialog box.

To correctly close Program Manager, you must use the ExitWindows API function. You can declare this API function in the GLOBAL.BAS module. For example:

1. Start a new project in Visual Basic.

2. Draw a command button on the form.

3. Add the following as a single line to GLOBAL.BAS:

```
Declare Function ExitWindows Lib "user" (ByVal dwReserved&,
   ByVal wReturnCode%) as integer
```

4. Add the following line of code to the command button's Click procedure:

```
RetVal% = ExitWindows(0,0)
```

5. Run the program.

6. Click the command button.

The ExitWindows API call initiates the standard Windows shutdown procedure. If all applications agree to terminate, the windows session is terminated and control returns to MS-DOS. If the ExitWindows API call fails due to an open MS-DOS session or for some other reason, FALSE is returned. You should check for this and handle it appropriately.

Steps to Reproduce Incident
---
1. Start a New Project in Visual Basic.

2. Draw a command button on the form.

3. In the command button Click event procedure, add this code:

```
AppActivate("Program Manager")
SendKeys "%{ }{DOWN 5}{ENTER 2}", 0  'ALT, SPACE, DOWN 5, ENTER 2
```

4. Run the program.

Note that the Program Manager does not close. If you choose the OK button with the mouse, you'll see a message stating, "Can't quit at this time." If you choose the Cancel button, you'll see a message stating, "Cannot start more than one copy of the specified program." These messages are misleading, but are the result of attempting an unsupported action.

Additional reference words: 1.00 2.00 3.00 restart start exit windows
KBCategory:
KBSubcategory: APrgOther APrgWindow

How to Print a VB Picture Control Using Windows API Functions

Article ID: Q77060

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article explains how to print a Visual Basic picture control to a printer using several Windows API function calls.

NOTE: this example will not work correctly on PostScript printers. Instead of the picture control printing, two blank sheets are ejected from the printer when using a printer configured to use the PostScript printer driver. For the example to work correctly, the printer must use a standard non-PostScript laser printer configuration (such as PCL/HP.)

MORE INFORMATION

To print a picture control from Visual Basic, you must use the PrintForm method. Although this can very useful, there is no straightforward method of printing just a picture control without the use of API function calls. Printing a picture control to the printer is useful when you want to control the location or size of the printed image. Calling API functions to print a picture control is also useful if you want to include other images or text along with the picture image on a single sheet of paper.

To print a bitmap, you need to do the following:

1. Create a memory device context that is compatible with the bitmap (CreateCompatibleDC). A memory device context is a block of memory that represents a display surface. It is used to prepare images before copying them to the actual device surface of the compatible device.

2. Save the present object (SelectObject) and select the picture control using the handle from the memory device context.

3. Use the BitBlt or StretchBlt function to copy the bitmap from the memory device context to the printer.

4. Remove the bitmap from the memory device context (SelectObject) and delete the device context (DeleteDC).

Step-by-Step Example

The following steps demonstrate this process:

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a picture control (Picture1) to Form1 and set the AutoRedraw property to True.

3. Add a command button (Command1).

4. Display some graphics in Picture1 by loading from a picture file or pasting from the Clipboard at design time. You can load a picture from a file as follows:

   a. Select the Picture property from the Properties bar.

   b. Click the arrow at the right of the Settings box, then select the desired picture file (such as a .BMP or .ICO file supplied with Microsoft Windows) from the dialog box.

5. Add the following declarations to the global Declarations section of the Code window. Enter each Declare statement as one, single line.

```
Declare Function CreateCompatibleDC% Lib "GDI" (ByVal hDC%)

Declare Function SelectObject% Lib "GDI" (ByVal hDC%, ByVal hObject%)

Declare Function StretchBlt% Lib "GDI" (ByVal hDC%, ByVal X%,
   ByVal Y%, ByVal nWidth%, ByVal nHght%, ByVal hSrcDC%, ByVal XSrc%,
   ByVal YSrc%, ByVal nSrcWidth%, ByVal nSrcHeight%, ByVal dwRop&)

Declare Function DeleteDC% Lib "GDI" (ByVal hDC%)

Declare Function Escape% Lib "GDI" (ByVal hDC As Integer,
   ByVal nEscape As Integer, ByVal nCount As Integer,
   LpInData As Any, LpOutData As Any)
```

6. Add the following code to the Command_Click event:

```
Sub Command1_Click ()
   Const SRCCOPY = &HCC0020
   Const NEWFRAME = 1
   Const PIXEL = 3

   '* Display hour glass.
   MousePointer = 11
   Picture1.Picture = Picture1.Image

   '* StretchBlt requires pixel coordinates.
   Picture1.ScaleMode = PIXEL
   Printer.ScaleMode = PIXEL

   Printer.Print " "

   hMemoryDC% = CreateCompatibleDC(Picture1.hDC)
   hOldBitMap% = SelectObject(hMemoryDC%, Picture1.Picture)

   'Enter the following three lines as one, single line:
```

```
      ApiError% = StretchBlt(Printer.hDC, 0, 0, Printer.ScaleWidth,
         Printer.ScaleHeight, hMemoryDC%, 0, 0, Picture1.ScaleWidth,
         Picture1.ScaleHeight, SRCCOPY)

      hOldBitMap% = SelectObject(hMemoryDC%, hOldBitMap%)
      ApiError% = DeleteDC(hMemoryDC%)

      Result% = Escape(Printer.hDC, NEWFRAME, 0, 0&, 0&)

      Printer.EndDoc

      MousePointer = 1
   End Sub
```

7. Run the program to copy the bitmap to the printer. If you have selected a low resolution from the Print Manager, printing the bitmap will proceed quickly (the lower the resolution, the faster the print time). While designing your software, you may want to keep this at the lowest possible resolution. The print resolution can be changed from the Windows Control Manager.

REFERENCES
==========

"Programming Windows: The Microsoft Guide to Writing Applications for Windows 3," Charles Petzold, Microsoft Press, 1990

"Microsoft Windows Software Development Kit: Reference Volume 1," version 3.0

"Microsoft Windows Software Development Kit: Guide to Programming," version 3.0.

WINSDK.HLP file shipped with Microsoft Windows 3.0 Software Development Kit

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgPrint

How to Invoke GetSystemMetrics Windows API Function from VB

Article ID: Q77061

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Windows API GetSystemMetrics function can return useful information about the Windows system. GetSystemMetrics can be called directly from Visual Basic for Windows or from the Control Development Kit (CDK) to get system metrics for a particular display adapter, retrieve information about the Windows debug mode, or retrieve information about a mouse configuration.

The Visual Basic for Windows CDK is shipped as part of the Professional Edition of Microsoft Visual Basic versions 2.0 and 3.0 for Windows.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The Windows GetSystemMetrics function call retrieves information about the system metrics. The system metrics are the widths and heights of various display elements of the particular window display. The GetSystemMetrics function can also return flags that indicate whether the current Windows version is a debugging version, whether a mouse is present, or whether the meaning of the left and right mouse button has been changed. System metrics depend on the system display, and may vary from display to display.

The Visual Basic for Windows declaration for GetSystemMetrics is:

```
Declare Function GetSystemMetrics% Lib "user" (ByVal nIndex%)
```

The value nIndex% specifies the system measurement to be retrieved. All measurements are in pixels.

The value returned from the GetSystemMetrics% function specifies the system metrics.

Below is a sample call to determine if the present version of Windows is a debugging version:

```
ScaleMode = 3   ' Select pixel.
Print "Debugging version : "; GetSystemMetrics(SM_DEBUG)
```

The constants and meaning for nIndex% are as follows:

```
  Constant Name(Value)  Description
  --------------------  -----------
  SM_CXSCREEN(0).........Width of screen
  SM_CYSCREEN(1).........Height of screen
  SM_CXFRAME(32).........Width of window frame that can be sized
  SM_CYFRAME(33).........Height of window frame that can be sized
  SM_CXVSCROLL(2)........Width of arrow bitmap on vertical scroll bar
  SM_CYVSCROL(20)........Height of arrow bitmap on vertical scroll bar
  SM_CXHSCROLL(21).......Width of arrow bitmap on horizontal scroll bar
  SM_CYHSCROLL(3)........Height of arrow bitmap on horizontal scroll bar
  SM_CYCAPTION(4)........Height of caption
  SM_CXBORDER(5).........Width of window frame that cannot be sized
  SM_CYBORDER(6).........Height of window frame that cannot be sized
  SM_CXDLGFRAME(7).......Width of frame when window has WS_DLGFRAME style
  SM_CYDLGFRAME(8).......Height of frame when window has WS_DLGFRAME style
  SM_CXHTHUMB(10)........Width of thumb on horizontal scroll bar
  SM_CYHTHUMB(9).........Height of thumb on horizontal scroll bar
  SM_CXICON(11)..........Width of icon
  SM_CYICON(12)..........Height of icon
  SM_CXCURSOR(13)........Width of cursor
  SM_CYCURSOR(14)........Height of cursor
  SM_CYMENU(15)..........Height of single-line menu
  SM_CXFULLSCREEN(16)....Width of window client area for full-screen window
  SM_CYFULLSCREEN(17)....Height of window client area for full-screen
                         window (height - caption)
  SM_CYKANJIWINDOW(18)...Height of Kanji window
  SM_CXMINTRACK(34)......Minimum tracking width of window
  SM_CYMINTRACK(35)......Minimum tracking height of window
  SM_CXMIN(28)...........Minimum width of window
  SM_CYMIN(29)...........Minimum width of window
  SM_CXSIZE(30)..........Width of bitmaps contained in the title bar
  SM_CYSIZE(31)..........Height of bitmaps contained in the title bar
  SM_MOUSEPRESENT(19)....Mouse present
  SM_DEBUG(22)...........Nonzero if Windows debug version
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther APrgINI

Examples of Copying a Disk File in Visual Basic for Windows

Article ID: Q77315

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Visual Basic for Windows does not have a command to copy a disk file such as the MS-DOS COPY command. However, you can write the necessary code to copy a file. Two examples of copying a file are provided in this article. Note that in Visual Basic versions 2.0 and 3.0, you can use the FileCopy statement instead of the code shown in this article.

MORE INFORMATION

The following Visual Basic for Windows sample subprograms, CopyFile1 and CopyFile2, provide two different ways to copy a disk file in a way similar to the MS-DOS COPY command. The first example uses only Visual Basic for Windows code, while the second example includes Window API functions. CopyFile2 runs faster than CopyFile1, especially for large files (up to twice as fast).

Subprogram: CopyFile1

```
Sub CopyFile1 (ByVal Source As String, ByVal Destination As String)
    Dim Index As Integer, NumBlocks As Integer
    Dim FileLength As Long, LeftOver As Long
    Dim FileData As String

    Const BlockSize = 32768

    ' Source and Destination are strings containing filenames:
    Open Source For Binary Access Read As #1
    ' Opening then immediately closing the destination file with
    ' "For Output" access erases the file if it exists (which is
    ' necessary in case the copied Source file is shorter than the
    ' existing Destination file, which would leave some of the old
    ' file's characters at the end of the new Destination file).
    ' You can use this technique to erase the Destination file in place
    ' of the Kill statement to avoid a Kill statement error if the
    ' Destination file doesn't exist:
    Open Destination For Output As #2
    Close #2
    Open Destination For Binary As #2

    FileLength = LOF(1)
```

```
    NumBlocks = FileLength \ BlockSize
    LeftOver = FileLength Mod BlockSize

    FileData = String$(LeftOver, 32)

    Get #1, , FileData
    Put #2, , FileData

    FileData = String$(BlockSize, 32)

    For Index = 1 To NumBlocks
        Get #1, , FileData
        Put #2, , FileData
    Next Index

    Close #1, #2
End Sub
```

Subprogram: CopyFile2

Note that CopyFile2 (below) copies files faster than CopyFile1 (above). Because CopyFile2 uses several API functions, you must include the Visual Basic Declare statements shown below. Place these declarations in the global file or in the (general) (declarations) section of a form or module file that contains the CopyFile2 subprogram:

```
DefInt A-Z
' All Declare statements must be on one line when added to a program:
Declare Function fWrite Lib "kernel" Alias "_lwrite" (ByVal hFile,
    ByVal lpBuff As Long, ByVal nBuff)
Declare Function fRead Lib "kernel" Alias "_lread" (ByVal hFile,
    ByVal lpBuff As Long, ByVal nBuff)
Declare Function GLobalAlloc Lib "kernel" (ByVal wFlags, ByVal nBuff
    As Long)
Declare Function GLobalFree Lib "kernel" (ByVal hMem)
Declare Function GLobalLock Lib "kernel" (ByVal hMem) As Long
Declare Function GLobalUnlock Lib "kernel" (ByVal hMem)

Sub CopyFile2 (ByVal Source As String, ByVal Destination As String)
    Dim lpBuff As Long
    Dim DestFile As Integer, SourceFile As Integer
    Dim DestDOS As Integer, SourceDOS As Integer
    Dim ApiErr As Integer, AmtRead As Integer
    Dim hMem As Integer
    Const nBuff = 32767
    Const wFlags = &H20

    hMem = GLobalAlloc(wFlags, nBuff)
    lpBuff = GLobalLock(hMem)

    DestFile = FreeFile
    Open Destination For Output As #DestFile Len = 1

    SourceFile = FreeFile
    Open Source For Input As #SourceFile Len = 1
```

```
    DestDOS = FileAttr(DestFile, 2)
    SourceDOS = FileAttr(SourceFile, 2)

    Do
        AmtRead = fRead(SourceDOS, ByVal lpBuff, nBuff)
        ApiErr = fWrite(DestDOS, ByVal lpBuff, AmtRead)
    Loop Until AmtRead = 0

    Close #SourceFile, #DestFile

    lpBuff = GLobalUnlock(hMem)
    hMem = GLobalFree(hMem)
End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Determine Display State of a VB Form, Modal or Modeless

Article ID: Q77316

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Show method in the Visual Basic for Windows language can display a form either as modal or modeless. No direct support exists in the language to determine the display state of the form without maintaining global variables that contain the display state of the form. However, the Windows API function GetWindowLong can be used to check the display state of the form.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

When Visual Basic for Windows displays a modal form (.Show 1), all other forms will be modified to contain the Window Style WS_DISABLED. The Windows API function GetWindowLong can be used to return the Window Style of another form to check for the WS_DISABLED style.

The following code demonstrates this process:

Add the following to the General Declarations section of Form1 and Form2:

```
DefInt A-Z
Global Const GWL_STYLE = (-16)
Global Const WS_DISABLED = &H8000000
Declare Function GetWindowLong& Lib "user" (ByVal hWnd, ByVal nIndex)
```

Form1.Frm

```
Sub Form_Click ()
  ' Flip between "Modeless" and "Modal" display states.
  Static ShowStyle
  Unload form2
  form2.Show ShowStyle
  ShowStyle = (ShowStyle + 1) Mod 2
End Sub
```

Form2.Frm

```
Sub Form_Paint ()
  ' Get the Window Style for Form1.
```

```
   WinStyle& = GetWindowLong(Form1.hWnd, GWL_STYLE)
   If WinStyle& And WS_DISABLED Then
      ' The WS_DISABLED style is set on "FORM1" when "FORM2"
      ' is displayed with the Modal flag (Show 1).
      Print "Modal    - Show 1"
   Else
      ' The WS_DISABLED style is not set on "FORM1" when "FORM2"
      ' is displayed with the Modeless flag (Show or Show 0).
      Print "Modeless - Show"
   End If
End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgWindow PrgOptTips

Example of How to Read and Write Visual Basic Arrays to Disk

Article ID: Q77317

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Microsoft Visual Basic for Windows does not provide a command to read or write an entire array all at once to a disk file. Using Visual Basic for Windows alone, you must transfer each element of the array to the disk. However, using two Windows API functions, _lread and _lwrite, you can save an entire array to disk in one statement when the array is less then 64K.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The ReadArray and WriteArray functions provided below allow you to read and write a Visual Basic for Windows array to or from a disk file. These functions will work with arrays of Integers, Longs, Singles, Doubles, Currency, and user-defined types, but not with variable-length strings (as an array or as a member of a user-defined type) or variants. These functions can work with fixed length strings when the strings are a member of a user-defined type. Arrays greater than 64K are supported in Visual Basic versions 2.0 and later for Windows, however the _lread and _lwrite functions can only handle arrays up to 64K. Arrays greater than 64K can be written to disk using the standard I/O statements built into Visual Basic for Windows.

The two functions, ReadArray and WriteArray, require two parameters: the array to be transferred, and the Visual Basic for Windows file number to be written to or read from. The functions also return the number of bytes transferred, or -1 when an error occurs with the API function. The file number is the Visual Basic for Windows file number of a file that has already been opened with the Open statement, and will be used in the Visual Basic for Windows Close statement.

The following function examples use a user-defined type named "Mytype". An example of this type is as follows:

```
Type MyType
     Field1 As String * 10
     Field2 As Integer
     Field3 As Long
     Field4 As Single
     Field5 As Double
```

```
        Field6 As Currency
   End Type

Declarations of API Functions
-----------------------------
DefInt A-Z
' Each Declare statement must appear on one line:
Declare Function fWrite Lib "kernel" Alias "_lwrite" (ByVal hFile,
   lpBuff As Any, ByVal wBytes)
Declare Function fRead Lib "kernel" Alias "_lread" (ByVal hFile,
   lpBuff As Any, ByVal wBytes)

Function: ReadArray
-------------------
Function ReadArray (An_Array() As MyType, VBFileNumber As Integer) As Long

   Dim ApiErr As Integer
   Dim ArraySize As Long
   Dim DOSFileHandle As Integer
   Dim ReadFromDisk As Integer

   ArraySize = Abs(UBound(An_Array) - LBound(An_Array)) + 1
   ArraySize = ArraySize * Len(An_Array(LBound(An_Array)))

   If ArraySize > 32767 Then
      ReadFromDisk = ArraySize - 2 ^ 15
      ReadFromDisk = ReadFromDisk * -1
   Else
      ReadFromDisk = ArraySize
   End If

   DOSFileHandle = FileAttr(VBFileNumber, 2)
   ApiErr=fRead(DOSFileHandle,An_Array(LBound(An_Array)),ReadFromDisk)

   ReadArray = ApiErr
End Function

Function: WriteArray
--------------------
Function WriteArray (An_Array() As MyType, VBFileNumber As Integer) As Long

   Dim ApiErr As Integer
   Dim ArraySize As Long
   Dim DOSFileHandle As Integer
   Dim WriteToDisk As Integer

   ArraySize = UBound(An_Array) - LBound(An_Array) + 1
   ArraySize = ArraySize * Len(An_Array(LBound(An_Array)))

   If ArraySize > 32767 Then
      WriteToDisk = ArraySize - 2 ^ 15
      WriteToDisk = WriteToDisk * -1
   Else
      WriteToDisk = ArraySize
   End If

   DOSFileHandle = FileAttr(VBFileNumber, 2)
```

```
   ApiErr=fWrite(DOSFileHandle,An_Array(LBound(An_Array)),WriteToDisk)

   WriteArray = ApiErr
End Function
```

The following are the function header changes to allow the ReadArray and WriteArray functions to work with different data types (Integer, Long, Single, Double, Currency, and user-defined type). Each Function statement must be on a single line:

```
Function ReadArray  (An_Array() As Integer, VBFileNumber As Integer)
   As Long
Function WriteArray (An_Array() As Integer, VBFileNumber As Integer)
   As Long

Function ReadArray  (An_Array() As Long, VBFileNumber As Integer) As
   Long
Function WriteArray (An_Array() As Long, VBFileNumber As Integer) As
   Long

Function ReadArray  (An_Array() As Single, VBFileNumber As Integer) As
   Long
Function WriteArray (An_Array() As Single, VBFileNumber As Integer) As
   Long

Function ReadArray  (An_Array() As Double, VBFileNumber As Integer) As
   Long
Function WriteArray (An_Array() As Double, VBFileNumber As Integer) As
   Long

Function ReadArray  (An_Array() As Currency, VBFileNumber As Integer)
   As Long
Function WriteArray (An_Array() As Currency, VBFileNumber As Integer)
   As Long
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Get Windows Master List (Task List) Using Visual Basic

Article ID: Q78001

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

By calling the Windows API functions GetWindow, GetWindowText, and GetWindowTextLength, you can get the window titles of all windows (visible and invisible) currently loaded.

The list of all of the window titles is known as the master list. The Windows Task Manager contains a list of the window titles for each of the top-level windows (normally one per application). This list is known as the task list.

The sample program listed below demonstrates how to activate an application by using a list of the top-level windows (a task list).

MORE INFORMATION

This information is included with the Help file provided with Microsoft Professional Toolkit for Visual Basic version 1.0, Microsoft Visual Basic version 2.0, and Microsoft Visual Basic version 3.0.

The task list is generally a subset of the master list. The Windows API functions only support methods of getting the master list, not the task list. However, from the master list you can get a list of all top-level windows closely resembling the task list. The only difference is that the list containing the top-level windows may have more entries than the task list because it is possible for an application to remove itself from the task list even though it is part of the master list.

The example below demonstrates how to get the names of all top-level windows. The names of child windows can also be obtained by calling the GetWindow API function using the GW_CHILD constant. Although the code example only provides an example of using the constants GW_HWNDFIRST and GW_HWNDNEXT as arguments to GetWindow, the value of the other constants such as GW_CHILD are provided in the code.

Here are the steps necessary to construct a sample program that demonstrates how to load the task list into a Visual Basic combo box:

1. Start Visual Basic or choose New Project from the File menu (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Change the caption property of Form1 to AppActivate.

3. Add the following controls to Form1, and change the Name property as indicated:

Control	Default Name	Name
Label Control	Label1	Label1
Combo Box	Combo1	Combo_ListItem
Command Button	Command1	Command_Ok

4. Change the Caption properties of the controls as follows:

Control	Name	Caption
Label Control	Label1	Application to AppActivate:
Command Button	Command_OK	OK

5. Add the following code to the general declarations section of Form1:

```
DefInt A-Z

'Windows API function declarations
'Enter each entire Declare statement on one, single line:
Declare Function GetWindow Lib "user" (ByVal hWnd, ByVal wCmd)
   As Integer
Declare Function GetWindowText Lib "user" (ByVal hWnd, ByVal lpSting$,
   ByVal nMaxCount) As Integer
Declare Function GetWindowTextLength Lib "user" (ByVal hWnd) As Integer

'Declare constants used by GetWindow
Const GW_CHILD = 5
Const GW_HWNDFIRST = 0
Const GW_HWNDLAST = 1
Const GW_HWNDNEXT = 2
Const GW_HWNDPREV = 3
Const GW_OWNER = 4
```

6. Add the following code to the Form_Load event procedure of Form1:

```
Sub Form_Load ()
   Call LoadTaskList

   'If no items are in the task list, end the program.
   If Combo_ListItem.ListCount > 0 Then
      Combo_ListItem.Text = Combo_ListItem.List(0)
   Else
      MsgBox "Nothing found in task list", 16, "AppActivate"
      Unload Form1
   End If
End Sub
```

7. Add the following code to the Click event procedure of the Command_Ok button:

```
Sub Command_Ok_Click ()
   'Get the item selected from the text portion of the combo box.
   f$ = Combo_ListItem.Text
```

```
      'Resume if "Illegal function call" occurs on AppActivate statement.
      On Local Error Resume Next

      AppActivate f$
   End Sub
```

8. Add the following code to the general declarations section of Form1:

```
   Sub LoadTaskList ()
      'Get the hWnd of the first item in the master list
      'so we can process the task list entries (top-level only).
      CurrWnd = GetWindow(Form1.hWnd, GW_HWNDFIRST)

      'Loop while the hWnd returned by GetWindow is valid.
      While CurrWnd <> 0
         'Get the length of task name identified by CurrWnd in the list.
         Length = GetWindowTextLength(CurrWnd)

         'Get task name of the task in the master list.
         ListItem$ = Space$(Length + 1)
         Length = GetWindowText(CurrWnd, ListItem$, Length + 1)

         'If there is a task name in the list, add the item to the list.
         If Length > 0 Then
            Combo_ListItem.AddItem ListItem$
         End If

         'Get the next task list item in the master list.
         CurrWnd = GetWindow(CurrWnd, GW_HWNDNEXT)

         'Process Windows events.
         x = DoEvents()
      Wend
   End Sub
```

9. From the Run menu, choose Start (ALT, R, S) to run the program.

From the combo box, select the window title of an application currently running in Windows. Choose the OK button to activate the application.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

Use Common Dialog or Escape() API to Specify Number of Copies

Article ID: Q78165

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can use the Common Dialog in the Professional Edition of Visual Basic version 2.0 or 3.0, or you can call the Windows API Escape() function in other versions to tell the Windows Print Manager how many copies of a document you want to print.

MORE INFORMATION

The Windows API constant SETCOPYCOUNT (value 17) can be used as an argument to the Escape() function to specify the number of uncollated copies of each page for the printer to print.

The arguments for Escape() are as follows:

```
r% = Escape(hDC, SETCOPYCOUNT, Len(Integer), lpNumCopies, lpActualCopies)
```

Parameter	Type/Description
hDC	hDC. Identifies the device context. Usually referenced by Printer.hDC.
lpNumCopies	Long pointer to integer (not ByVal). Point to a short-integer value that contains the number of uncollated copies to print.
lpActualCopies	Long pointer to integer (not ByVal). Points to a short integer value that will receive the number of copies that where printed. This may be less than the number requested if the requested number is greater than the device's maximum copy count.

The return value specifies the outcome of the escape -- 1 if the escape is successful, a negative number if the escape is not successful, or zero if the escape is not supported.

The following example code demonstrates how to print three copies of a line of text on the printer. To recreate this example, choose New Project from the Visual Basic File menu. Then add a command button to the form and paste the following code into the appropriate event procedures:

```
REM   Below is GLOBAL.BAS:
```

```
' The following Declare statement must be typed on one, single line:
Declare Function Escape% Lib "GDI" (ByVal hDc%, ByVal nEsc%, ByVal nLen%,
   lpData%, lpOut%)

REM   Below is the click procedure for a command button on Form1:

Sub Command1_Click ()
   Const SETCOPYCOUNT = 17
      Printer.Print ""
      x% = Escape(Printer.hDC, SETCOPYCOUNT, Len(I%), 3, actual%)
      Printer.Print " Printing three copies of this"
      Printer.EndDoc
End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgPrint

Lstrcpy API Call to Receive LPSTR Returned from Other APIs

Article ID: Q78304

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Because Microsoft Visual Basic does not support a pointer data type, you cannot directly receive a pointer (such as a LPSTR) as the return value from a Windows API or DLL function.

You can work around this by receiving the return value as a long integer data type. Then use the lstrcpy Windows API function to copy the returned string into a Visual Basic string.

MORE INFORMATION

This information is included with the Help file provided with Microsoft Professional Toolkit for Visual Basic version 1.0, Microsoft Visual Basic version 2.0, and Microsoft Visual Basic version 3.0.

An LPSTR Windows API data type is actually a far pointer to a null-terminated string of characters. Because LPSTR is a far pointer, it can be received as a four byte data type, such as a Visual Basic long integer. Using the Visual Basic ByVal keyword, you can pass the address stored in a Visual Basic long integer back to the Windows API lstrcpy routine to copy the characters at that address into a Visual Basic string variable.

Because lstrcpy expects the target string to be long enough to hold the source string, you should pad any Visual Basic string passed to lstrcpy to have a size large enough to hold the source string before passing it to lstrcpy. Failure to allocate enough space in the Visual Basic string may result in an Unrecoverable Application Error (UAE) or general protection (GP) fault when you call lstrcpy.

The following is an example program that demonstrates how to use lstrcpy to retrieve an LPSTR pointer returned from the Windows API GetDOSEnvironment routine.

Note that the capability of the Windows API GetDOSEnvironment routine is already available through the Environ function built into Visual Basic. Therefore, so the program is useful only to demonstrate how to use lstrcpy.

```
'*** General declarations ***
Declare Function GetDosEnvironment Lib "Kernel" () As Long

' Enter the following Declare statement as one, single line:
```

```
Declare Function lstrcpy Lib "Kernel" (ByVal lpString1 As Any,
   ByVal lpString2 As Any) As Long

'*** Form Click event code ***
Sub Form_Click()
   Dim lpStrAddress As Long,  DOSEnv$

   ' Allocate space to copy LPSTR into
   DOSEnv$ = Space$(4096)

   ' Get address of returned LPSTR into a long integer
   lpStrAddress = GetDOSEnvironment()

   ' Copy LPSTR into a Visual Basic string
   lpStrAddress = lstrcpy(DOSEnv$, lpStrAddress)

   ' Parse first entry in environment string and print
   DOSEnv$ = Trim$(DOSEnv$)
   DOSEnv$ = Left$(DOSEnv$, Len(DOSEnv$) - 1)
   Form1.Print DOSEnv$
End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

PRB: Format$ Using # for Digit Affects Right Alignment

Article ID: Q79094

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

The pound (#) sign does not serve as a place holder for blank spaces when used with the Format$ function to reformat numbers as strings. If a pound sign place holder is not filled by a digit, Format$ truncates that digit position and will not replace that position with a space. This may be undesirable behavior if you are attempting to right justify the numeric digits within the string.

CAUSE

Visual Basic Format$ function handles the pound sign (#) place holder differently from the way the it's handled in the Print Using statement found in other Basic products. In the Print Using statement, a pound sign place holder is replaced by a space when no numeric digit occupies that position. By using the Print Using statement, you can right justify a formatted numeric string by using the pound sign as place holders for the number. Visual Basic does not support the Print Using statement, so you need to use additional code to right justify a string using the Visual Basic Format$ function. An example is given below.

WORKAROUND

To work around the problem, use a monospaced font, such as Courier, and use the Len function to determine how many spaces need to be added to the left of the string representation of the number to right justify the result. Here is the example code:

```
Sub Form_Click ()
   desired = 5   'longest number expected
   a = 1.23
   b = 44.56
   FontName = "Courier"   'Select a fixed-spaced font
   num1$ = Format$(a, "#0.00")  'This converts number to a string
   num2$ = Format$(b, "#0.00")  '2 decimal places and a leading 0
   If (desired - Len(num1$)) > 0 Then
       num1$ = Space$(desired - Len(num1$)) + num1$
   End If
   If (desired - Len(num2$)) > 0 Then
       num2$ = Space$(desired - Len(num2$)) + num1$
   End If
   Print num1$
```

```
   Print num2$
End Sub
```

STATUS
======

This behavior is by design.

MORE INFORMATION
================

Page 121 of the "Microsoft Visual Basic: Language Reference" for version 1.0 regarding the Format$ function doesn't specify how the pound sign is handled. When there is no numeric digit to fill the pound sign place holder, the manual does not specify whether the pound sign is replaced by a space or truncated. The documentation should reflect how the pound sign is handled by the Format$ function.

The Print Using statement supported in other Basic products allows the use of the pound sign as a place holder for leading or trailing spaces, as follows:

```
   Print Using "##0.00"; myvar
```

The above example causes two leading spaces to be added to the resulting string representation of the variable myvar when the value of myvar is printed to the screen.

However, when used with the Visual Basic Format$ function, the same pound sign format switch (#) does not work as a placeholder for spaces:

```
   mystring$ = Format$(myvar , " ##.## ")
```

The Visual Basic Format$ function yields a formatted string representation of myvar with no leading spaces. This may not be the result you expected (for example, when myvar = 1.23). You may have expected the formatted result to have one leading space allowing you to right justify the number, but no leading space is added.

The following code sample produces an output of right justified numbers in Microsoft QuickBasic version 4.5:

```
   a = 1.23
   b = 44.56
   Print Using "##.##"; a
   Print Using "##.##"; b
```

The following code sample produce an output of left justified numbers in Visual Basic:

```
   Sub Form_Click ()
      a = 1.23
      b = 44.56
      num1$ = Format$(a, "##.##")
      num2$ = Format$(b, "##.##")
      Print num1$
      Print num2$
```

```
   End Sub
```

Click the form to print the numbers. These numbers will be left justified, instead of right justified as may be desired.

Additional reference words: 1.00 2.00 3.00 4.50 alignment aligned align right-justify
KBCategory:
KBSubcategory: APrgOther

Use SetHandleCount to Open More than 15 Files at Once in VB

Article ID: Q79764

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Microsoft Visual Basic for Windows programs normally may not have more than 15 files open at once. Visual Basic for Windows displays the error message "Too many files" (error code 67) when you attempt to open more than the maximum number of files at once. You can increase the maximum number of open files by calling the Windows API function SetHandleCount.

MORE INFORMATION

The Windows API function SetHandleCount requests Windows to change the maximum number of files a program can open. SetHandleCount returns the actual number of handles that the program can use, which may be less than the number requested.

The FILES= statement in the CONFIG.SYS file does not limit the number of files available to a Microsoft Windows program.

Do not attempt to increase the number of files with MS-DOS interrupt 21 hex with function 67 hex. This interrupt does not record information needed by Windows.

Example

The following code example demonstrates how to use SetHandleCount:

```
'*** In the global module: ***
Declare Function SetHandleCount% Lib "kernel" (ByVal n%)

'*** In the form: ***
Sub Form_Load ()
   n% = SetHandleCount(60)  ' Request 60 file handles.
   MsgBox "Maximum number of open files: " + Format$(n%)
End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Set Landscape or Portrait for Printer from VB App

Article ID: Q80185

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Some printers support changing the orientation of the paper output to landscape. With the Windows API Escape() function, you can change the settings of the printer to either landscape or portrait. In addition, if you have one of the following products, you can use the Common Dialog box to allow users to set the mode inside a Visual Basic Application:

- Visual Basic version 1.0 Professional Toolkit
- Professional Edition of Visual Basic version 2.0
- Standard or Professional Edition of Visual Basic version 3.0

Below is an example showing how to invoke the Windows API Escape() function from Microsoft Visual Basic.

Note that the Windows API Escape() function is provided in Windows versions 3.0 and 3.1 for backward compatibility with earlier versions of Microsoft Windows. Applications are supposed to use the GDI DeviceCapabilities() and ExtDeviceMode() functions instead of the Escape() function, but neither DeviceCapabilities() nor ExtDeviceMode() can be called directly from Visual Basic. This is because they are exported by the printer driver, not by the Windows GDI. The only way to use ExtDeviceMode() or DeviceCapabilities() in Visual Basic is to create a DLL and call them from there.

MORE INFORMATION

Normally, output for the printer is in portrait mode, where output is printed horizontally across the narrower dimension of a paper. In landscape mode, the output is printed horizontally across the longer dimension of the paper.

You can use the Escape() function to change the orientation of the printer by passing GETSETPAPERORIENT as an argument. When you initially print text to the printer, Visual Basic will use the currently selected orientation. Sending the Escape() function will not take effect until you perform a Printer.EndDoc. After you perform a Printer.EndDoc, output will print in the orientation that you have selected.

To determine if your printer supports landscape mode, do the following:

1. From the Windows Program Manager, run Control Panel.

2. From the Control Panel, select the Printers icon.

3. From the Printers dialog box, choose the Configure button.

4. The Configure dialog box will contain an option for landscape orientation if landscape is supported on your printer.

The example below demonstrates how to change the printer orientation to landscape. Please note that your printer must support landscape mode for these commands to have any effect.

Example
----

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a command button (Command1) to Form1.

3. Add the following code to the global module:

```
Type OrientStructure
   Orientation As Long
   Pad As String * 16
End Type
' Enter the following Declare statement on one, single line:
Declare Function Escape% Lib "GDI" (ByVal hDc%, ByVal nEsc%,
            ByVal nLen%, lpData As OrientStructure, lpOut As Any)
```

4. Add the following code to the Command1_Click event procedure of the Command1 button:

```
Sub Command1_Click ()
   Const PORTRAIT = 1
   Const LANDSCAPE = 2
   Const GETSETPAPERORIENT = 30

   Dim Orient As OrientStructure

   '* Start the printer
   Printer.Print ""

   '* Specify the orientation
   Orient.Orientation = LANDSCAPE

   '* Send escape sequence to change orientation
   x% = Escape(Printer.hDC, GETSETPAPERORIENT,
             Len(Orient), Orient, NULL)
   '* The EndDoc will now re-initialize the printer
   Printer.EndDoc

   Printer.Print "Should print in landscape mode"
   Printer.EndDoc
End Sub
```

Additional reference words: 1.00 2.00 3.00

KBCategory:
KBSubcategory: APrgPrint

How to Kill an Application with System Menu Using Visual Basic

Article ID: Q80186

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Visual Basic for Windows can use the Windows API SendMessage function to close any active window that has a system menu (referred to as control box within Visual Basic for Windows) with the Close option.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

You can use the Windows API SendMessage function to post a message to any window in the environment as long as the handle to the window is known. You can use the API FindWindow function to determine the handle associated with the window the user wants to close.

Query on the following words in the Microsoft Knowledge Base for more information on the FindWindow function:

```
FindWindow and Visual Basic
```

To create a program to close an occurrence of the Windows version 3.0 Calculator program, do the following:

1. Create a form called Form1.

2. Create two command buttons called Command1 and Command2.

3. Within the Command1 Click event, add the following code:

```
Sub Command1_Click()
   X% = Shell("Calc.exe")
End Sub
```

4. Within the Command2 Click event, add the following code:

```
Sub Command2_Click()
   Const NILL = 0&
   Const WM_SYSCOMMAND = &H112
   Const SC_CLOSE = &HF060

   lpClassName$ = "SciCalc"
   lpCaption$ = "Calculator"
```

```
      '* Determine the handle to the Calculator window.
      Handle = FindWindow(lpClassName$, lpCaption$)

      '* Post a message to Calc to end it's existence.
      X& = SendMessage(Handle, WM_SYSCOMMAND, SC_CLOSE, NILL)

   End Sub
```

5. In the Declarations section, declare the following two API functions:

```
   ' Enter each of the following Declare statements on one, single line:
   Declare Function FindWindow% Lib "user" (ByVal lpClassName As Any,
      ByVal lpCaption As Any)
   Declare Function SendMessage& Lib "user" (ByVal hwnd%, ByVal wMsg%,
      ByVal wParam%, ByVal lParam As Long)
```

6. Run the program. Click the Command1 button to bring up an instance of the Calculator program. Click the Command2 button to close the window.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Reset the Parent of a Visual Basic Control

Article ID: Q80189

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Visual Basic version 1.0 does not support overlapping controls. This can be a problem if you want to drag and drop a control from one parent control to another parent control. Using the Windows API SetParent() function call, you can change a control's parent within Visual Basic.

Visual Basic versions 2.0 and 3.0 support overlapping controls with the z-order method. For more information on the z-order method, search for the z-order topic in the Visual Basic Help menu.

MORE INFORMATION

A frame, picture box, and form can act as parent controls. Creating a control on top of any of these parent controls creates that control as a child of the parent. When you use the Drag operations, there may be times when you want to move a child control from one parent control to another parent. If you allow the movement and don't change the child's parent, you are creating overlapping controls, which are not supported in Visual basic.

The SetParent function changes the parent of a child control. SetParent has the following description:

```
SetParent%(ByVal hWndChild, ByVal hWndParen%)
```

Parameter	Type/Description
hWndChild	HWnd/Identifies the child window
hWndParent	HWnd/Identifies the parent window

The returned value identifies the previous parent window.

Step-by-Step Example

The example below demonstrates how to drag and drop a text box between the form and a picture box on the form. The parent controls are the picture box and the form. The child control is the text box.

1. Start Visual Basic or from the File menu, choose New Project if Visual Basic is already running. Form1 is created by default.

2. Add a Text box (Text1) to Form1.

3. Add a Picture box (Picture1) to Form1.

4. Add a Command button (Command1) to Form1.

5. Add the following code to the Global module:

```
'============= GLOBAL.BAS ==================
Declare Function SetParent% Lib "user" (ByVal h%, ByVal h%)
Declare Function GetFocus% Lib "user" ()
' GetFocus will be used to obtain the handles to the
' controls. This is not build into every control of Visual Basic
```

6. Add the following code to the general declarations section of Form1:

```
'============= FORM1 =======================
Dim hWndText As Integer
Dim hWndPicture As Integer
```

7. Add the following code to the Form_Load event procedure of Form1:

```
Sub Form_Load ()
   'form has to be shown to access any of the controls
   Show

   'get the handle to the text box
   Text1.SetFocus
   hWndText = GetFocus()

   'get the handle to the picture box
   Picture1.SetFocus
   hWndPicture = GetFocus()
End Sub
```

8. Add the following code to the appropriate event procures:

```
Sub Picture1_DragDrop (Source As Control, X As Single, Y As Single)
   G% = SetParent(hWndText, hWndPicture)
   Source.Move X - Source.Width / 2, Y - Source.Height / 2
   Source.DragMode = 0
End Sub

Sub Form_DragDrop (Source As Control, X As Single, Y As Single)
   G% = SetParent(hWndText, Form1.hwnd)
   Source.Move X - Source.Width / 2, Y - Source.Height / 2
   Source.DragMode = 0
End Sub

Sub Command1_Click ()
   'start the dragging process
   Text1.DragMode = 1
End Sub
```

9. Run the program. The Command1 button is used to start the dragging operation.

Demonstration Steps

Try the following steps when running the application:

1. Press the command button.

2. Place the cursor over the text box.

3. Press the left mouse button and drag the text box either over the picture control or over the form.

4. Once the text box is over the control, release the mouse button.

For better control of where the text box is placed, turn off Grid Setting from the Edit menu of Visual Basic.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Add a Horizontal Scroll Bar to Visual Basic List Box

Article ID: Q80190

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The normal list box that comes with Visual Basic for Windows does not have a horizontal scroll bar. This can be a problem when the item in a list box extends past the boundaries of the list box. To add a horizontal scroll bar to the control, you can call the Windows API SendMessage function with the LB_SETHORIZONTALEXTENT (WM_USER + 21) constant.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

To add a horizontal scroll bar to a list box, perform a SendMessage function call with the LB_SETHORIZONTALEXTENT constant.

This message sets the width in pixels by which a list box can scroll horizontally. If the size of the list box is smaller than this value, the horizontal scroll bar will horizontally scroll items in the list box. If the list box is large as or larger than this value, the horizontal scroll bar is disabled.

The parameters for the SendMessage function are as follows:

```
SendMessage(hWnd%, LB_SETHORIZONTALEXTENT, wParam%, lParam&)
-------------------------------------------------------------
   hWnd%   - Handle to the list box
   wParam% - Specifies the number of pixels by which the list
             box can be scrolled
   lParam% - Is not used
```

To make a program example that will only allow the user to scroll a specified distance, create a form with the following controls:

Control	Name (CtlName in Visual Basic 1.0 for Windows)
Command button	Command1
List box	List1

Add the following code in the described locations in your code:

```
'======== General Declarations for Form1  ==================
' Enter the following Declare as one, single line:
```

```
Declare Function SendMessage& Lib "user" (ByVal hWnd%, ByVal wMsg%,
   ByVal wParam%, ByVal lParam&)
Declare Function GetFocus Lib "User" () as Integer

'======== Form1 ======================
'Note: each command must appear on one, single line.

Sub Command1_Click ()
   Const LB_SETHORIZONTALEXTENT = &H400 + 21
   Const NUL = 0&
   ' wParam is in PIXEL(3).
   ScaleMode = 3

   ' Get the handle.
   List1.SetFocus
   ListHwnd% = GetFocus()

   ' This string will show up initially.
   ListString1$ = "Derek is a great "

   ' You can scroll to see this portion.
   ListString2$ = "little boy "

   ' You cannot scroll to see this string.
   ListString3$ = "but can be a problem sometimes"

   ExtraPixels% = TextWidth(ListString2$)
   BoxWidth% = TextWidth(ListString1$)

   ' Resize the text box.
   List1.Move List1.Left, List1.Top, BoxWidth%

   ' Add the scroll bar.
   X& = SendMessage(ListHwnd%, LB_SETHORIZONTALEXTENT,
      BoxWidth% + ExtraPixels%, NUL)

   ' Add the example string to the list box.
   List1.AddItem ListString1$ + ListString2$ + ListString3$
End Sub
```

Additional reference words: 1.00 2.00 3.00 scrollbar
KBCategory:
KBSubcategory: APrgOther

How to Print VB Form Borders and Menus

Article ID: Q80409

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The methods to print a form provided by Visual Basic print only the client area, not the non-client area of a form. This is a design feature of Visual Basic. The client area of a form includes the form's controls and picture. The non-client area includes the form's borders and menus, which cannot be printed directly from Visual Basic.

To print both the client and non-client areas of a form, copy an image of the form into the Clipboard, paste it into a graphics editor such as Paintbrush, and print. Two methods for this procedure are provided below.

MORE INFORMATION

The Print option from the File menu, and the Visual Basic statement [form.]PrintForm only print the client area of a form. The client area includes the form's picture and controls. Both methods of printing do not print the non-client area, which includes the form's title bar, Minimize and Maximize buttons, borders and menus. To print both the client and non-client areas, you must print the form from an application outside of Visual Basic. If you want to print a form that either

- Has submenu items, but you do not wish to print the submenus

  -or-

- Has menus without submenus

  -or-

- Does not have menus

then use Method 1 below to print the form.

If you want to print a form that contains submenus in their pulled-down state, use Method 2 below.

Method 1

To print a form without pulled-down submenus, do the following:

1. From the Visual Basic editing environment, create the form you want to print. Include all controls, titles, menus, pictures, borders, and so on that you want to print, and size them appropriately.

2. Set focus to the form you want to print.

3. Press ALT+PRINT SCREEN. This key combination is an operation in Windows that copies the active window (your form in this case) to the Windows Clipboard.

4. From the Windows Program Manager, launch Paintbrush (or the graphics editor of your choice) and maximize it.

5. From the Paintbrush Edit menu, choose Paste. The image of your form should appear in Paintbrush. If the form is too large for Paintbrush, try either a larger screen resolution (such as 800-by-600 or 1024-by-768), another editor with a larger work screen, or slightly decrease the size of your form for the printing process.

6. Once the form is correctly pasted into Paintbrush, from the File menu, choose Print to print it.

Method 2

To print a form with pulled-down submenus, do the following:

1. In the Visual Basic editing environment, create the form you want to print. Include all controls, titles, menus, pictures, borders, and so on that you want to print and size them appropriately.

2. Set focus to the form to be printed.

3. Move the form to the upper left corner of the screen. When the Clipboard pastes its image into Paintbrush, it starts at the upper left corner. If the image is too large for the Paintbrush edit screen, the image is truncated on the right and bottom edges. Placing the form in the upper left corner helps to ensure that the full form fits into Paintbrush.

4. Choose the menu option you want to be pulled down when the form is printed. The menu option should appear pulled down on the screen. Only one menu option can be pulled down at a time, but submenu options can be selected.

5. Press SHIFT+PRINT SCREEN. This keystroke is an option in Windows that copies an image of the entire screen into the Windows Clipboard. When the pull-down menus are open, Visual Basic traps the ALT key and closes the menus, thus making the ALT+PRINT SCREEN keystroke in Method 1 ineffective when printing pull-down menus on a form.

6. From Program Manager, launch Paintbrush (or the graphics editor of your choice) and maximize it.

7. From the Paintbrush Edit menu, choose Paste. Your form should appear in Paintbrush. If the form is too large for Paintbrush, try either a larger screen resolution (for example, 800-by-600 or 1024-by-768), another editor with a larger work screen, or slightly decrease the size of your form for the printing process.

8. Using one of the cutting tools at the top of the Paintbrush toolbox, outline your form.

9. From the Edit menu, choose Copy. This places the graphics area contained in the cutting region into the Clipboard.

10. From the File menu, choose New to bring up a new editor screen. A dialog box will appear to ask if you want to save the current image. You will need to select Yes or No before a new editor screen will appear.

11. Once the editor screen is empty, from the Edit menu, choose Paste to paste your form into the editor.

12. From the File menu, choose Print to print the image.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgPrint

How to Clear VB Picture Property at Run Time Using LoadPicture

Article ID: Q80488

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

During execution of a Visual Basic program, you can clear the Picture property of a form or picture control by using the LoadPicture function. Calling LoadPicture with no parameters and assigning the result to the Picture property of a form or control will clear the Picture property.

MORE INFORMATION

This information is documented in the Visual Basic Help menu under the LoadPicture function.

Code Example

To clear the picture property at run time, do the following:

1. Start Visual Basic.

2. Make a picture box called Picture1.

3. Assign a bitmap or icon the picture1.picture property.

4. Add the following code to the form1.click event by double-clicking the form:

```
Sub Form_Click ()
   picture1.picture = LoadPicture()
End Sub
```

5. Run the program.

6. Click the form.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd APrgGrap

How to Get Windows Version Number in VB with GetVersion API

Article ID: Q80642

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

From a Microsoft Visual Basic for Windows program, you can find out which version of Windows is running by calling the Windows API GetVersion() function from the Windows Kernel module. The GetVersion() function can help your application accommodate any known differences, if any, in the way API calls operate between different versions of Windows (such as differences
between API parameters or return values).

MORE INFORMATION

The step-by-step example given below demonstrates how to make the GetVersion() function call. GetVersion() takes no parameters, and the return value is a WORD value -- which translates to an integer in Visual Basic for Windows.

The return value specifies the major and minor version numbers of Windows. The high order byte specifies the minor version and the low order byte specifies the major version number.

Step-by-Step Example

1. Create a form with a text box and a command button.

2. Add the following declaration to the General Declarations section:

```
Declare Function GetVersion Lib "kernel" () As Integer
```

3. Add following code to the command button Click event:

```
Sub Command1_Click ()
   i% = GetVersion()
   ' Lowbyte is derived by masking off high byte.
   lowbyte$ = Str$(i% And &HFF)
   ' Highbyte is derived by masking off low byte and shifting.
   highbyte$ = LTrim$(Str$((i% And &HFF00) / 256))
   ' Assign Windows version to text property.
   text1.text = lowbyte$ + "." + highbyte$
End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:

KBSubcategory: APrgWindow EnvtRun

How to Copy Entire Screen into a Picture Box in Visual Basic

Article ID: Q80670

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

Using the Windows API call BitBlt, you can capture the entire Microsoft Windows screen and place the image into a Microsoft Visual Basic for Windows picture box.

First, get the handle to the desktop. Then use the desktop window handle to get the handle to the desktop's device context (hDC). Finally, use the Windows API call BitBlt to copy the screen into the Picture property of a Visual Basic for Windows picture box control.

MORE INFORMATION

Step-by-Step Example

1. Start Visual Basic for Windows (VB.EXE). Form1 is created by default.

2. Add a picture box (Picture1) to Form1.

3. Set the following properties:

Control	Property	Value
Picture1	AutoRedraw	True
Picture1	Visible	False

4. Add the following code to the GLOBAL.BAS file in version 1.0 or to the general declarations section of Form1 in versions 2.0 and 3.0:

```
Type lrect
   left As Integer
   top As Integer
   right As Integer
   bottom As Integer
End Type
Declare Function GetDesktopWindow Lib "user" () As Integer
Declare Function GetDC Lib "user" (ByVal hWnd%) As Integer
' Enter the following Declare on one, single line:
Declare Function BitBlt Lib "GDI" (ByVal hDestDC%, ByVal X%, ByVal Y%,
   ByVal nWidth%, ByVal nHeight%, ByVal hSrcDC%, ByVal XSrc%,
   ByVal YSrc%, ByVal dwRop&) As Integer
' Enter the following Declare on one, single line:
Declare Function ReleaseDC Lib "User"(ByVal hWnd As Integer, ByVal hDC
   As Integer) As Integer
```

```
Declare Sub GetWindowRect Lib "User" (ByVal hWnd%, lpRect As lrect)
Global Const True = -1
Global Const False = 0
Global TwipsPerPixel As Single
```

5. Add the following code to the Form1 Click event procedure:

```
Sub Form_Click ()
   Call GrabScreen
End Sub

Sub GrabScreen ()

   Dim winSize As lrect

   ' Assign information of the source bitmap.
   ' Note that BitBlt requires coordinates in pixels.
   hwndSrc% = GetDesktopWindow()
   hSrcDC% = GetDC(hwndSrc%)
   XSrc% = 0: YSrc% = 0
   Call GetWindowRect(hwndSrc%, winSize)
   nWidth% = winSize.right             ' Units in pixels.
   nHeight% = winSize.bottom           ' Units in pixels.

   ' Assign informate of the destination bitmap.
   hDestDC% = Form1.Picture1.hDC
   x% = 0: Y% = 0

   ' Set global variable TwipsPerPixel and use to set
   ' picture box to same size as screen being grabbed.
   ' If picture box not the same size as picture being
   ' BitBlt'ed to it, it will chop off all that does not
   ' fit in the picture box.
   GetTwipsPerPixel
   Form1.Picture1.Top = 0
   Form1.Picture1.Left = 0
   Form1.Picture1.Width = (nWidth% + 1) * TwipsPerPixel
   Form1.Picture1.Height = (nHeight% + 1) * TwipsPerPixel

   ' Assign the value of the constant SRCOPYY to the Raster operation.
   dwRop& = &HCC0020

   ' Note function call must be on one line:
   Suc% = BitBlt(hDestDC%, x%, Y%, nWidth%, nHeight%,
                 hSrcDC%, XSrc%, YSrc%, dwRop&)

   ' Release the DeskTopWindow's hDC to Windows.
   ' Windows may hang if this is not done.
   Dmy% = ReleaseDC(hwndSrc%, hSrcDC%)

   'Make the picture box visible.
   Form1.Picture1.Visible = True
End Sub

Sub GetTwipsPerPixel ()
   ' Set a global variable with the Twips to Pixel ratio.
   Form1.ScaleMode = 3
```

```
      NumPix = Form1.ScaleHeight
      Form1.ScaleMode = 1
      TwipsPerPixel = Form1.ScaleHeight / NumPix
   End Sub
```

5. Run the program. Click Form1.

6. Using the mouse, change the size of the form to see more of the picture box. With a little work, you can use this as a screen saver program.

Additional reference words: 1.00 print printer
KBCategory:
KBSubcategory: APrgWindow APrgGrap

VB Custom Controls Support only Certain Picture Formats

Article ID: Q80779

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Load Picture dialog box for the 3-D Command Button, 3-D Group Push Button, Gauge, and Picture Clip custom controls include the extensions for picture formats that are invalid formats for these controls.

MORE INFORMATION

The 3-D Command Button, 3-D Group Push Button, Gauge, and Picture Clip custom controls use the same dialog box that Visual Basic uses to assign pictures to certain properties. However, not all .BMP, .ICO, and .WMF files are valid picture formats for the properties of these controls.

The following table lists the valid formats for the picture properties of custom controls and the error messages displayed if an invalid picture format is used:

Control	Property	Valid Formats	Error Message if Invalid Format
3-D Command Button	Picture	.BMP, .ICO	"Only Picture Formats '.BMP' and '.ICO' supported."
3-D Group Push Button	PictureUp, PictureDn, PictureDisabled	.BMP	"Only Picture Format '.BMP' supported."
Gauge	Picture	.BMP, .ICO	"Invalid Picture."
Picture Clip	Picture	.BMP	"Only Picture Format '.BMP' supported."

For additional information on Visual Basic version 2.0 custom controls, review the Professional Features manual.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus APrgGrap

How to Print Multiline Text Box Using Windows API Functions

Article ID: Q80867

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Printing the Text property of a multiline text box while maintaining the line structure requires attention to word wrapping and carriage return/line feeds. The programmer can either track the number of characters and lines in code or use Windows API functions to manipulate the Text property. This article demonstrates these techniques in a Visual Basic example.

MORE INFORMATION

The example below demonstrates how to use the API function SendMessage() to track the number of lines in a multiline text box and to select and print the lines the way they appear -- with line breaks or word wrapping intact. This code will work without modification even if the form and controls are resized at run time. The actual position of word wrapping will change.

For more information about API functions relating to text boxes, query on the following words in the Microsoft Knowledge Base:

```
API and text and box and manipulate
```

Step-by-Step Example

1. Create a form and place a label, text box, and command button on it.

2. Set the following properties at design time:

Control	Property	Setting
Text box	TabIndex	0 (zero, or first in tab order)
Text box	MultiLine	True
Label	AutoSize	True
Label	Name	aGetLineCount

3. Add the following code to the Global module:

```
Declare Function GetFocus% Lib "user" ()
' Enter the following Declare statement on one, single line:
Declare Function SendMessage% Lib "user" (ByVal hWnd%, ByVal wMsg%,
   ByVal wParam%, ByVal lParam As Any)
Global Buffer As String
```

```
Global resizing As Integer
Global Const EM_GETLINE = &H400 + 20
Global Const EM_GETLINECOUNT = &H400 + 10
Global Const MAX_CHAR_PER_LINE = 80  ' Scale this to size of text box
```

4. Add the following code to the Form_Load procedure:

```
Sub Form_Load ()
   ' Size form relative to screen dimensions.
   ' Could define all in move command but recursive definition causes
   ' extra paints.
   form1.width = screen.width * .8
   form1.height = screen.height * .6
   ' Enter the following form1.Move method on one, single line:
   form1.Move screen.width\2-form1.width\2,
      screen.height\2-form1.height\2
End Sub
```

5. Add the following code to the Form_Resize procedure:

```
Sub Form_Resize ()
   resizing = -1      ' Global  flag for fGetLineCount function call
   ' Dynamically scale and position the controls in the form.
   ' This code also is executed on first show of form.
   Text1.Move 0, 0, form1.width, form1.height \ 2
   Text1.SelStart = Text1.SelStart    ' To avoid UAE -see Q80669
   ' Enter the following two lines as one, single line:
   command1.Move form1.width\2-command1.width\2,
      form1.height-form1.height\4
   ' Enter the following two lines as one, single line:
   aGetLineCount.Move form1.width \ 2 - command1.width \ 2,
      Text1.height
   X% = fGetLineCount() ' Update to reflect change in text box size
   resizing = 0
End Sub
```

5. Add the following code to the Command1_Click event:

```
Sub Command1_Click ()
   '* Pop up an inputbox$ to allow user to specify which line
   '* in the text box to print or print all lines.
   '* Also check bounds so that a valid line number is printed
   OK = 0                     ' Zero the Do Loop flag
   NL$ = Chr$(13) + Chr$(10)
   prompt$ = "Which line would you like to print?"
   prompt1$ = prompt$ + NL$ + "Enter -1 for all"
   prompt2$ = "Too many lines" + NL$ + "Try again!" + NL$ + prompt1$
   prompt$ = prompt1$
   Do
      response$ = InputBox$(prompt$, "Printing", "-1")
      If response$ = "" Then Exit Sub  ' if user hits cancel then exit
      If Val(response$) > fGetLineCount&() Then
         prompt$ = prompt2$
      Else
         OK = -1        ' Line chosen is in valid range so exit DO
      End If
   Loop Until OK
```

```
   If Val(response$) = -1 Then  ' Print all lines
      ndx& = fGetLineCount&()
      For N& = 1 To ndx&
         Buffer = fGetLine(N& - 1)
         printer.Print Buffer       ' or print to the screen
      Next N&
   Else                          ' Print a line
      Buffer = fGetLine(Val(response$) - 1)
      printer.Print Buffer       ' or print to the screen
   End If
End Sub
```

6. Add the following code to the general Declarations section of the form's code:

```
Function fGetLine$ (LineNumber As Long)
  ' This function fills the buffer with a line of text
  ' specified by LineNumber from the text box control.
  ' The first line starts at zero.
   byteLo% = MAX_CHAR_PER_LINE And (255)  '[changed 5/15/92]
   byteHi% = Int(MAX_CHAR_PER_LINE / 256) '[changed 5/15/92]
   Buffer$ = chr$(byteLo%) + chr$(byteHi%)+Space$(MAX_CHAR_PER_LINE-2)
   ' [Above line changed 5/15/92 to correct problem.]
   text1.SetFocus 'Set focus for API function GetFocus to return handle
   x% = SendMessage(GetFocus(), EM_GETLINE, LineNumber, Buffer$)
   fGetLine$ = Left$(Buffer$,X%)
End Function

Function fGetLineCount& ()
   ' This function will return the number of lines
   ' currently in the text box control.
   ' Setfocus method illegal while in resize event
   ' so use global flag to see if called from there
   ' (or use setfocus prior to this function call in general case).
   If Not resizing Then
      Text1.SetFocus   ' Set focus for following function GetFocus
      resizing = 0
   End If
   lcount% = SendMessage(GetFocus(), EM_GETLINECOUNT, 0&, 0&)
   aGetLineCount.caption = "GetLineCount = " + Str$(lcount%)
   fGetLineCount& = lcount%
End Function
```

7. Add the following code to the Text1_Change event:

```
Sub Text1_Change ()
   X% = fGetLineCount() '* Update label to reflect current line
End Sub
```

8. Save the project. Then run the application.

9. Enter text into the text box and either let it wrap or use the ENTER key to arrange lines.

10. Choose the button or TAB and press ENTER.

11. Choose the default (which prints all lines) or enter the line

desired. If you choose Cancel, nothing will print.

12. Resize the form and repeat steps 9 to 11 above. The text will appear on the printed page as you saw it in the text box. Modify the example to print to the screen, write to a file, and so forth.

Reference(s):

"Microsoft Windows Programmer's Reference Book and Online Resource"
(Visual Basic Add-on kit number 1-55615-413-5)

Additional reference words: 1.00 2.00 3.00 textbox
KBCategory:
KBSubcategory: PrgCtrlsStd APrgWindow

How to Use FillPolygonRgn API to Fill Shape in Visual Basic

Article ID: Q81470

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY
=======

Microsoft Visual Basic versions 2.0 and later for Windows include the Shape control which can be used for creating and filling six different geometric shapes. Alternatively, you can create a polygon region on a form or picture and fill it with a color, using the CreatePolygonRgn and FillRgn Windows API calls to draw and fill areas of the screen with color. Geometric shapes not provided with the Shape control, such as a triangle, can be created using this method.

More Information:

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

To draw a polygon on a form or picture control, you can use the Polygon API call; this will draw the edge of the polygon. You can then use CreatePolygonRgn to create an area that you can paint and use FillRgn to fill it with a color. Using these Windows API calls allows you to pick the points, the number of points, and to choose the color or brush to fill with.

The API calls used in the following example should be declared in the general Declarations section of your form. They are as follows:

API Call	Description
CreatePolygonRgn	Creates a polygonal region
GetStockObject	Retrieves a handle to one of the predefined stock pens, brushes, or fonts
FillRgn	Fills the region specified by the hRgn parameter with the brush specified by the hBrush parameter
Polygon	Draws a polygon consisting of two or more points connected by lines

Code Example

The following code example shows how to create a black triangle on a form. To change the program to create other shapes, add points to the array.

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. From the File menu, choose New Module (press ALT, F, M). Module1 is created by default.

3. Add the following code to the general declarations section of Module1 (in Visual Basic version 1.0 for Windows, add it to GLOBAL.BAS):

```
Type Coord       ' This is the type structure for the x and y
   x As Integer  ' coordinates for the polygonal region.
   y As Integer
End Type

' Enter each Declare statement as one, single line:
Declare Function CreatePolygonRgn Lib "gdi" (lpPoints As Any,
   ByVal nCount As Integer, ByVal nPolyFillMode As Integer) As Integer
Declare Function Polygon Lib "gdi" ByVal hDC As Integer,
   lpPoints As Any, ByVal nCount As Integer) As Integer
Declare Function FillRgn Lib "gdi" (ByVal hDC As Integer,
   ByVal hRgn As Integer, ByVal hBrush As Integer) As Integer
Declare Function GetStockObject Lib "gdi" (ByVal nIndex As Integer)
   As Integer
Declare Function DeleteObject Lib "gdi" (ByVal hndobj As Integer)
   As Integer

Global Const ALTERNATE = 1 ' ALTERNATE and WINDING are
Global Const WINDING = 2   ' constants for FillMode.
Global Const BLACKBRUSH = 4' Constant for brush type.
```

2. Add the following code to the Form_Click event for Form1:

```
Sub Form_Click ()
   ' Dimension coordinate array.
   ReDim poly(1 To 3) As Coord
   ' Number of vertices in polygon.
   NumCoords% = 3
   ' Set scalemode to pixels to set up points of triangle.
   form1.scalemode = 3
   ' Assign values to points.
   poly(1).x = form1.scalewidth / 2
   poly(1).y = form1.scaleheight / 2
   poly(2).x = form1.scalewidth / 4
   poly(2).y = 3 * form1.scaleheight / 4
   poly(3).x = 3 * form1.scalewidth / 4
   poly(3).y = 3 * form1.scaleheight / 4
   ' Sets background color to red for contrast.
   form1.backcolor = &HFF
   ' Polygon function creates unfilled polygon on screen.
   ' Remark FillRgn statement to see results.
   bool% = Polygon(form1.hdc, poly(1), NumCoords%)
   ' Gets stock black brush.
   hbrush% = GetStockObject(BLACKBRUSH)
   ' Creates region to fill with color.
   hrgn% = CreatePolygonRgn(poly(1), NumCoords%, ALTERNATE)
   ' If the creation of the region was successful then color.
```

```
      If hrgn% Then bool% = FillRgn(form1.hdc, hrgn%, hbrush%)
      ' Print out some information.
      Print "FillRgn Return : ";bool%
      Print "HRgn : "; hrgn%
      Print "Hbrush : "; hbrush%
      Trash% = DeleteObject(hrgn%)
   End Sub
```

3. Run the program.

You should initially see a blank form. Click the form; a red background with a black triangle on it should be displayed. You can try different numbers of vertices by adding elements to the poly array and updating NumCoords. Different colors and brushes can be substituted as desired.

Note: If you try to fill a region with coordinates beyond the visible form, the CreatePolygonRgn function call will return a zero, meaning it was unsuccessful. The FillRgn will not work if the CreatePolygonRgn function was unsuccessful. All you will see is the outline created by the Polygon function. You should make certain that the vertices are all within the viewable form.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap

How to Set Windows System Colors Using API and Visual Basic

Article ID: Q82158

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article describes how to use the GetSysColor and SetSysColors API functions to set the system colors for various parts of the display in Microsoft Windows. This allows you to change the Windows display programmatically, instead of using the Windows Control Panel.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Windows maintains an internal array of 19 color values that it uses to paint the different parts of the Windows display. Changing any of these values will affect all windows for all applications running under Windows. Note that the SetSysColors routine only changes the internal system list. This means that any changes made using SetSysColors will only be valid for the current Windows session. To make these changes permanent, you need to change the [COLORS] section of the Windows initialization file, WIN.INI.

For more information on modifying the Windows initialization file programmatically, query on the following words in the Microsoft Knowledge Base:

   GetProfileString and WriteProfileString

To use the GetSysColor and SetSysColors functions within a Visual Basic for Window application, you must first declare them in the Declarations section of your Code window.

Declare the Function statement as follows:

```
Declare Function GetSysColor Lib "User" (ByVal nIndex%) As Long

Declare Sub SetSysColors Lib "User" (ByVal nChanges%,
                                     lpSysColor%,
                                     lpColorValues&)
```

Note: Each Declare statement above must be written on one line.

The parameters are defined as follows:

Parameter	Definition
nIndex%	Specifies the display element whose color is to be retrieved. See the list below to find the index value for the corresponding display element.
nChanges%	Specifies the number of system colors to be changed.
lpSysColor%	Identifies the array of integer indexes that specify the elements to be changed.
lpColorValues&	Identifies the array of long integers that contain the new RGB color values for each element to be changed.

The following system color indexes are defined using the predefined constants found in the WINDOWS.H file supplied with the Microsoft Windows Software Development Kit (SDK). The corresponding value is the value placed in the lpSysColor% array.

List of System Color Indexes

Windows.H Definition	Value	Description
COLOR_SCROLLBAR	0	Scroll-bar gray area
COLOR_BACKGROUND	1	Desktop
COLOR_ACTIVECAPTION	2	Active window caption
COLOR_INACTIVECAPTION	3	Inactive window caption
COLOR_MENU	4	Menu background
COLOR_WINDOW	5	Window background
COLOR_WINDOWFRAME	6	Window frame
COLOR_MENUTEXT	7	Text in menus
COLOR_WINDOWTEXT	8	Text in windows
COLOR_CAPTIONTEXT	9	Text in caption, size box, scroll bar arrow box
COLOR_ACTIVEBORDER	10	Active window border
COLOR_INACTIVEBORDER	11	Inactive window border
COLOR_APPWORKSPACE	12	Background color of multiple document interface (MDI) applications
COLOR_HIGHLIGHT	13	Items selected item in a control
COLOR_HIGHLIGHTTEXT	14	Text of item selected in a control
COLOR_BTNFACE	15	Face shading on push button
COLOR_BTNSHADOW	16	Edge shading on push button
COLOR_GRAYTEXT	17	Grayed (disabled) text. This color is set to 0 if the current display driver does not support a solid gray color.
COLOR_BTNTEXT	18	Text on push buttons

The following is an example of how to set the system colors for different parts of the Windows display:

1. Start Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Create the following controls for Form1:

Control	Name	Property Setting
Command button	Command1	Caption = "Change all Colors"
Command button	Command2	Caption = "Change selected Colors"

   (In Visual Basic version 1.0 for Windows, set the CtlName Property for the above objects instead of the Name property.)

3. Add the following code to the general Declarations section of Form1:

```
Declare Function GetSysColor Lib "User" (ByVal nIndex%) As Long

Declare Sub SetSysColors Lib "User" (ByVal nChanges%,
                                     lpSysColor%,
                                     lpColorValues&)
' Note: The above declaration must be on one line.

Const COLOR_BACKGROUND = 1
Const COLOR_ACTIVECAPTION = 2
Const COLOR_WINDOWFRAME = 6

Dim SavedColors(18) As Long
```

4. Add the following code to the Form_Load event procedure of Form1:

```
Sub Form_Load ()

' ** Save current system colors.
   For i% = 0 To 18
      SavedColors(i%) = GetSysColor(i%)
   Next i%

End Sub
```

5. Add the following code to the Form_Unload event procedure of Form1:

```
Sub Form1_Unload ()

' ** Restore system colors.
   ReDim IndexArray(18) As Integer
   For i% = 0 To 18
      IndexArray(i%) = i%
   Next i%
   SetSysColors 19, IndexArray(0), SavedColors(0)

End Sub
```

6. Add the following code to the Command1_Click event procedure of Form1:

```
Sub Command1_Click ()

' ** Change all display elements.
   ReDim NewColors(18) As Long
   ReDim IndexArray(18) As Integer
   For i% = 0 to 18
      NewColors(i%) = QBColor(Int(16 * Rnd))
      IndexArray(i%) = i%
   Next i%
   SetSysColors 19, IndexArray(0), NewColors(0)

End Sub
```

7. Add the following code to the Command2_Click event procedure of Form1:

```
Sub Command2_Click ()

' ** Change desktop, window frames, and active caption.
    ReDim NewColors(18) As Long
   ReDim IndexArray(18) As Integer
   For i% = 0 to 18
      NewColors(i%) = QBColor(Int(16 * Rnd))
      IndexArray(i%) = i%
   Next i%
   SetSysColors 19, IndexArray(0), NewColors(0)

End Sub
```

8. From the Run menu, choose Start, or press the F5 key, to run the program.

Choosing the Change All Colors button will cause all the different parts of the Windows display to be assigned a randomly generated color. Choosing the Change Selected Elements button will cause only the desktop, active window caption, and window frames to be assigned a random color. To restore the original system colors, double-click the Control-menu box to end the application.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap APrgOther

VB AniButton Control: Cannot Resize if PictDrawMode=Autosize

Article ID: Q82159

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Resizing an Animated Button custom control by setting the Width or Height property at run time will not work if the PictDrawMode property is set to Autosize (1). This is by design. When the PictDrawMode property is in autosize mode, the size is determined by the size of the images loaded, not by the design time setting of Width or Height nor the run time setting of those values.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Steps to Reproduce Behavior

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. From the Files menu, choose Add File. In the Files box, select the ANIBUTON.VBX custom control file. The Animated Button tool appears in the toolbox.

3. Add the following code to the Form_Load procedure:

```
Sub Form_Load ()
  Form1.BackColor = &HFFFF00  ' To make the size of the control more
                              ' visible.
  AniButton1.Move Form1.Width \ 4, 0, 1600, 1600
  AniButton1.TextPosition = 3  ' Put caption at top for clarity.
End Sub
```

4. Add the following code to the Form_Click procedure:

```
   Sub Form_Click ()
       AniButton1.Caption = "This is a very very long caption"
       AniButton1.PictDrawMode = 1   ' Autosize control.
      'AniButton1.PictDrawMode = 0   ' As Defined.
      'AniButton1.PictDrawMode = 2   ' Stretches image to fit.
   End Sub
```

4. Add the following code to the Form_DoubleClick event:

```
Sub Form_DblClick ()
   Print AniButton1.Width
   AniButton1.Width = 400
   Print AniButton1.Width
   Print AniButton1.PictDrawMode
End Sub
```

5. Run the project with the PictDrawMode setting of 0 uncommented and the other two commented out.

6. Click once to see the effect of changing the mode. Then double-click the form to see the changes due to changing the Width property. Because the caption is the largest object in an unloaded Animated Button, the autosize adjusts to it.

7. Access the Frame property and load a bitmap into the first frame and an icon in the second, or vice versa.

8. Repeat steps 5 and 6. Notice that the larger object (the bitmap) causes the control to resize to it.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: PrgCtrlsCus APrgGrap

How to Disable Close Command in VB Control Menu (System Menu)

Article ID: Q82876

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

To modify an item in the Visual Basic Control menu (also known as the System menu), you need to call the API functions GetSystemMenu and ModifyMenu. This article describes how to disable the Close command in the Control menu.

MORE INFORMATION

If you do not want the user to be able to choose the Close command from the Control menu or to be able to double-click the Control-menu box to end the application, you can disable the Close command. GetSystemMenu returns the handle to the Control menu. That handle can be used by ModifyMenu to change the control menu.

The following code example disables (grays out) the Close command in the Visual Basic Control menu.

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Place a command button (Command1) on Form1. Change its Caption property to Disable Close.

3. Place another command button (Command2) on Form1. Change its caption to Exit.

4. Add the following declarations and constants to the general Declarations section of Form1:

```
' Enter each Declare statement as one, single line:
Declare Function GetSystemMenu Lib "User" (ByVal hWnd%,
   ByVal bRevert%) as Integer
Declare Function ModifyMenu Lib "User" (ByVal hMenu%, ByVal nPosition%,
   ByVal wFlags%, ByVal wIDNewItem%, ByVal lpNewItem as Any) as Integer

Const MF_BYCOMMAND = &H0
Const MF_GRAYED = &H1
Const SC_CLOSE = &HF060
```

   Note that other constants to disable other menu items in the Control menu are described in the CONSTANT.TXT file.

5. Add the following code to the Command1 Click event:

```
Sub Command1_Click ()
   ' See the notes at the end of this article for important additional
   ' information about this code.
   nPosition% = SC_CLOSE
   idNewItem%=-10
   s$ = "Close"
   hMenu% = GetSystemMenu(hWnd, 0)
   wFlags% = MF_BYCOMMAND Or MF_GRAYED
   success% = ModifyMenu(hMenu%, nPosition%, wFlags%, idNewItem%, s$)
End Sub
```

6. Add the following code to the Command2 Click event:

```
Sub Command2_Click ()
   End
End Sub
```

7. Press the F5 key to run the program.

8. Click the Control-menu box to see that all the menu items are available.

9. Click the Disable Close command button. Then click the Control-menu box. Notice that the Close menu command is unavailable.

The user cannot end the application by either choosing Close from the Control menu or by double-clicking the Control-menu box. The only way to end this program is to choose the Exit command button.

Notes on the Use of ModifyMenu() in the Code

The code listed above uses the ModifyMenu() function, but the EnableMenuItem() may be more appropriate in your particular situation.

Here's the syntax for ModifyMenu():

```
ModifyMenu(hMenu, SC_CLOSE, MF_BYCOMMAND|MF_GRAYED, SC_CLOSE, "Close")
```

Here's the syntax for EnableMenuItem():

```
EnableMenuItem(hMenu, SC_CLOSE, MF_BYCOMMAND|MF_GRAYED)
```

Both functions work. However it appears that Visual Basic re-enables the menu item whose ID is SC_CLOSE. This is why it may appear as if the ModifyMenu() or EnableMenuItem() function failed.

To work around this problem in the code listed above, the second to last argument (idNewItem%) is set to -10 (0 would also work):

```
ModifyMenu(hMenu, SC_CLOSE, MF_BYCOMMAND|MF_GRAYED, -10, "Close")
```

This works because Visual Basic looks for a menu item with ID SC_CLOSE to re-enable and cannot find one because it has been changed to 0 or -10. So Visual Basic can't re-enable the Close menu item.

However, because of this workaround, another limitation is introduced. The problem is that the ID of the Close menu item is changed to -10. If you want the program to be able to re-enable the Close item, you'll need to use this alternative version:

```
   ModifyMenu(hMenu, -10, MF_BYCOMMAND|MF_GRAYED, SC_CLOSE, "Close")
```

This is a good workaround to Visual Basic's re-enabling of Close. Don't use 0 because menu separators also have the ID 0 and you will run into problems when you try to re-enable Close. 0xFFFF is a good ID to use.

Another alternative solution is to use DeleteMenu to remove Close and the separator above it, and use InsertMenu to add the Close and the separator.

REFERENCES
==========

"Microsoft Windows Programmer's Reference Book and Online Resource"
(Visual Basic Add-on kit number 1-55615-413-5)

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

PRB: Can't Change Minimized/Maximized MDIChild's Position/Size

Article ID: Q82878

The information in this article applies to:

- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

SYMPTOMS

When a MDI Child custom control is minimized (reduced to an icon), attempting to change its position or size at run time by setting the Top, Left, Height, or Width property will generate the following Visual Basic error message:

Cannot Change MDIChild Position Or Size While Minimized Or Maximized.

This valid error message will also be generated if the MDI child window is maximized and you attempt to change the size of position of the MDI child.

RESOLUTION

This article does not apply to later versions of Visual Basic. The MDI Child custom control shipped only with version 1.0. Multiple-document interface (MDI) forms are built into Visual Basic version 2.0 and later, making the MDI custom control obsolete.

You cannot change the position or size of a Visual Basic version 1.0 MDI child window when it is minimized or maximized. These properties can be set at run time in code or at design time for any MDI child window that is not maximized or minimized to an icon.

However, you can set the properties in Visual Basic version 2.0 for Windows. You do not get an error. Note though that MDI is different in Visual Basic version 2.0 because it is built in to both the Standard and Professional Editions rather than being a separate custom control, as it is in Visual Basic version 1.0.

MORE INFORMATION

The following steps demonstrate that an error message is generated in Visual Basic version 1.0 when you attempt to change (at run time in code) the Left property of an MDI child window that has been either reduced to an icon or maximized (to the full size of the parent form).

Steps to Reproduce Problem

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the MDICHILD.VBX custom control file. The MDI Child tool appears in the toolbox.

3. Place an MDI Child window control on Form1.

4. Double-click the form outside the MDI child window to open the Code window.

5. Add the following code to the Form1 Click event:

```
Sub Form_Click ()
   MDIchild1.Left = 0
End Sub
```

6. Press F5 to run the application.

7. Click the Control-menu box (in the upper left corner) of the MDI child window, and choose Minimize.

8. Click directly on the form.

The following error message dialog box is generated:

```
Cannot Change MDIChild Position Or Size While Minimized Or Maximized
```

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: PrgCtrlsCus APrgGrap

How to Create a Form with no Title Bar in VB for Windows

Article ID: Q83349

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

To create a Microsoft Visual Basic for Windows form with a border but with no title bar, the Caption property of a form must be set to a zero-length string; the BorderStyle property must be set to Fixed Single (1), Sizable (2) or Fixed Double; and the ControlBox, MaxButton and MinButton properties must be set to False (0). If any text (including spaces) exists for the Caption property or if the ControlBox, MaxButton, or MinButton property is set to True, a title bar will appear on the form. Note that setting the BorderStyle property to None (0) will always result in a form with no title bar.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Even with the ControlBox, MaxButton, and MinButton properties of a form set to False (0) and the BorderStyle set to Fixed Single (1), Sizable (2) or Fixed Double (3), the form will still have a title bar unless the Caption property is set to null. Setting the Caption to blanks will leave a title bar with no title.

Steps to Reproduce Behavior

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. From the Properties bar, set the ControlBox, MaxButton, and MinButton properties to False.

3. Set the Caption property to at least one space.

4. Press the F5 key to run the program. The form will have a title bar without a title.

5. Press CTRL+BREAK to return to design mode.

6. Set the Caption property to a zero-length string (that is, delete all characters including spaces).

7. Press the F5 key to run the program. There should be no title bar on

the form.

You can also have a form with no title bar by setting the BorderStyle property to None (0).

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap

How to Call LoadModule() API Function from Visual Basic

Article ID: Q83350

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article shows how to call the Windows LoadModule() API function from a Visual Basic program. The LoadModule() API function loads and executes a Windows program or creates a new instance of an existing Windows program. The code example below shows an example of calling WINVER.EXE with the LoadModule() function call, but you can change it to any executable file.

Note that the Shell function provided in Visual Basic provides a functionality similar to and simpler than the technique explained in this article.

MORE INFORMATION

The LoadModule() API function call has only two parameters, but the second parameter is a pointer to a structure with an embedded structure in it.

The two parameters are as follows:

lpModuleName	Points to a null terminated string that contains the filename of the application to be run.
lpParameterBlock	Points to a data structure consisting of four fields that define a parameter block. The data structure consists of the following fields:
wEnvSeg:	Specifies the segment address of the environment under which the module is to run; 0 indicates that the Windows environment is to be copied.
lpCmdLine:	Points to a NULL terminated character string that contains a correctly formed command line. This string must not exceed 120 bytes in length.
lpCmdShow:	Points to a data structure containing two WORD length values. The first value must be set to 2, and the second value in this example will be set to 5.
dwReserved:	Reserved and must be NULL.

Steps to Reproduce Behavior

---

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add the following code to the GLOBAL.BAS file (or any module in Visual basic version 2.0):

```
Type CmdShow
   fp As Integer   ' first parameter
   sp As Integer   ' second parameter
End Type

Type lpParameterBlock
   wEnvSeg As Integer
   lpCmdLine As Long      ' This line modified 6/25/93
   lpCmdShow As Long      ' This line modified 5/27/92
   dwReserved As Long
End Type

Declare Function lstrcpy Lib "Kernel" (lp1 As Any, lp2 As Any) As Long
' Enter the following Declare statement on one, single line
Declare Function LoadModule% Lib "kernel" (ByVal lpModuleName As String,
   lpParameterBlock As Any)
```

3. Add a command button to Form1, and add the following code to the Command1_Click procedure:

```
Sub Command1_Click ()

   Dim cs As CmdShow
   Dim pb As lpParameterBlock
   ' assign values to the CmdShow structure
   pb.lpCmdShow = lstrcpy(cs, cs)           ' Line added 5/27/92
   cs.fp = 2
   cs.sp = 5
   ' assign values to the lpParameterBlock structure
   pb.wEnvSeg = 0
   ' append null to end of path
   ' Following two lines added 6/25/93 replacing previous line:
   lpCmdLine$ = "c:\windows\winver.exe" + Chr$(0)
   pb.lpCmdLine = lstrcpy(ByVal lpCmdLine$, ByVal lpCmdLine$)
   pb.dwReserved = 0&
   ' make sure to append null to end of .EXE name
   m% = LoadModule%("winver.exe" + Chr$(0), pb)

End Sub
```

4. Save the program and run it.

When you run the program and press the command button, the WinVer program will run as it would with the Run command on the Windows Program Manager File menu.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Draw an Ellipse with Circle Statement in VB

Article ID: Q83906

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article describes how to use the Circle statement to draw an ellipse of a specified width and height by calculating the radius and aspect ratio appropriate for the dimensions of the ellipse and the units of measurement, determined by the ScaleMode property.

MORE INFORMATION

The Circle statement takes two arguments that determine the shape of the ellipse drawn: the radius and the aspect ratio. For example:

```
Circle (x, y), radius,,,, aspect
```

The aspect ratio is the y-radius divided by the x-radius of the ellipse drawn. An aspect ratio of 1.0 (the default) yields a perfect (non-elliptical) circle. If the aspect ratio is less than one, the radius argument specifies the x-radius. If the aspect ratio is greater than one, the radius argument specifies the y-radius. Both the x-radius and the y-radius are measured in units of the x-axis.

Steps to Create Example Program

1. Run Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Enter the following code in the Form1_Click event procedure:

```
Sub Form_Click ()
    Cls

    ' Set x-axis units different from y-axis to demonstrate
    ' that the ellipse still comes out right.
    Form1.ScaleWidth = Rnd * 100
    Form1.ScaleHeight = Rnd * 100
    Print "ScaleWidth  = "; Format$(Form1.ScaleWidth, "#")
    Print "ScaleHeight = "; Format$(Form1.ScaleHeight, "#")

    ' Print the dimensions of the ellipse.
    ' Draw an ellipse centered on the form and touching the
    ' borders.
    w = Form1.ScaleWidth / 2
```

```
    h = Form1.ScaleHeight / 2
    Call ellipse(Form1, w, h, w, h)
End Sub
```

3. Enter the following code in the general Declarations section:

```
' ellipse(frm, x, y, w, h)
'   Purpose
'     Draws an ellipse on a form.
'   Parameters
'     frm  -- the form to draw on
'     x, y -- specify the center of the ellipse.
'     w, h -- specify the width and height.
'
Sub ellipse (frm As Form, ByVal x!, ByVal y!, ByVal w!, ByVal h!)
    Dim swt As Long    ' ScaleWidth  in twips
    Dim sht As Long    ' ScaleHeight in twips
    Dim k As Double    ' conversion factor for x-units to y-units
    Dim ar As Double   ' aspect ratio
    Dim r As Single    ' radius
    Dim save_mode As Integer   ' for saving and restoring ScaleMode
    Dim save_width As Single   ' for saving and restoring ScaleWidth
    Dim save_height As Single  ' for saving and restoring ScaleHeight

    ' Check arguments.
    If w <= 0 Or h <= 0 Then Stop

    ' Determine form dimensions in twips.
    save_mode = frm.ScaleMode  ' save Scale... properties
    save_width = frm.ScaleWidth
    save_height = frm.ScaleHeight
    frm.ScaleMode = 1          ' set units to twips
    swt = frm.ScaleWidth
    sht = frm.ScaleHeight
    frm.ScaleMode = save_mode  ' restore Scale... properties
    If frm.ScaleMode = 0 Then
        frm.ScaleWidth = save_width
        frm.ScaleHeight = save_height
    End If

' Compute conversion factor of x-axis units to y-axis units.
    k = frm.ScaleWidth / frm.ScaleHeight * sht / swt

' Compute aspect ratio and radius.
    ar = k * h / w
    If ar <= 1 Then
        r = w
    Else
        r = k * h
    End If

    ' Draw the ellipse.
    frm.Circle (x, y), r, , , , ar
End Sub
```

4. Press F5 to run the program. Then click the form.

The program draws an ellipse centered in the form, and touching the sides of the form. Resize the form and/or click the form again to repeat the demonstration.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap

UCase$/LCase$ in Text Box Change Event Inverts Text Property

Article ID: Q84059

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

When using the UCase$ or LCase$ functions in Microsoft Visual Basic for Windows to capitalize text or make text lower case from within the change procedure of a text box, you may encounter unexpected results if the following conditions are true:

- The text property of the text box is being updated by the UCase$ or LCase$ statement.
- The resulting string created by UCase$ or LCase$ is assigned to the text property of the text box.
- The above statements appear in the Change event procedure of the text box.

Every time a key is pressed, the text contents are changed, and the cursor is placed at the beginning of the line. This causes the character for your next key press to be inserted at the beginning of the line rather than the end.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

When allowing users to enter text into text boxes, it is often desirable to control whether the user enters all uppercase or all lowercase letters. To do this, it would seem that putting a UCase$ or LCase$ statement in a text box Change event would allow you to enter only uppercase or lowercase letters into the text box. However, each time you press a key, the Change event fires and the cursor is brought back to the beginning of the text box as a result of assigning the Text property a new string.

Steps to Reproduce Behavior

1. Start Visual Basic for Windows or from the File menu, select New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Put a text box (Text1) on Form1 by either double-clicking the text box control or single clicking the text box control and drawing

it on Form1.

3. Add the following code to the Text1_Change event procedure:

```
Sub Text1_Change ()
  text1.text = UCase$(text1.text)
End Sub
```

4. Press the F5 key to run the program.

Notice that when you try to type information into the text box that it is entered in reverse order of what you would expect.

An alternative method of changing all contents of the text box to capital letters is to change the KeyAscii code of the typed information in the text box KeyPress event as follows:

```
Sub Text1_KeyPress (KeyAscii As Integer)

' Check to see if key pressed is a lower case letter.
  If KeyAscii >= 97 And KeyAscii <= 122 Then

    'If it is lowercase, change it to uppercase.
    KeyAscii = KeyAscii - 32

  End If

End Sub
```

When you run the above code, the letters typed into the text box are immediately changed to capital letters and are entered correctly as you type them in.

Another alternative method of changing the contents of the text box to uppercase letters is to add the following code to the Change event for the text box:

```
Sub Text1_Change ()

' Get the current position of the cursor.
  CurrStart = Text1.SelStart

' Change the text to capitals.
  Text1.Text = UCase$(Text1.Text)

' Reset the cursor position.
  Text1.SelStart = CurrStart

End Sub
```

SelStart sets or returns the starting point of text selected, and indicates the position of the insertion point if no text is selected.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Print Entire VB Form and Control the Printed Size

Article ID: Q84066

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Visual Basic for Windows PrintForm method provides a way to print the client area of a form. However, PrintForm does not allow you to control the size or proportion of the printed output, or to print the non-client area (the caption and border) of the form.

The following code example uses Windows API functions to print the entire form, and provides a method to control the size of the output. This method can also be used to print only the client area to a specific size and to control the position of the printed form to allow text or other graphics to be printed on the same page as the image of the form. The method is also applicable to printing all the forms in a project.

Note that this example will not work correctly on PostScript printers. For the example to work correctly, the printer must use a standard non-PostScript laser printer configuration (such as PCL/HP).

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Combining the Windows API functions BitBlt, StretchBlt, CreateCompatibleDC, DeleteDC, SelectObject, and Escape allows greater control over the placement and size of the printed form than the PrintForm method. In a two-part process, the image of the entire form is captured by using BitBlt to make an invisible picture, and is turned into a persistent bitmap using the AutoRedraw property. Then the picture is printed using the method of printing a picture control (outlined in a separate article, found by querying for the following word in the Microsoft Knowledge Base):

   CreateCompatibleDC

This method works on maximized forms as well as any smaller forms. The use of GetSystemMetrics allows a general procedure to handle different window border styles passed to it by querying the video driver for the size of windows standard borders in pixels.

The example below requires a single form with an invisible picture control.

Example
----
1. Add the following code to the general Declarations level of the
   form in a new project:

   Note: All Declare statements below must be on one line each.

```
DefInt A-Z
Declare Function BitBlt Lib "gdi" (ByVal hDestDC, ByVal X, ByVal Y,
         ByVal nWidth, ByVal nHeight, ByVal hSrcDC, ByVal XSrc,
         ByVal YSrc, ByVal dwRop&)
Declare Function CreateCompatibleDC Lib "GDI" (ByVal hDC)
Declare Function SelectObject Lib "GDI" (ByVal hDC, ByVal hObject)
Declare Function StretchBlt Lib "GDI" (ByVal hDC, ByVal X, ByVal Y,
         ByVal nWidth, ByVal nHght, ByVal hSrcDC, ByVal XSrc,
         ByVal YSrc, ByVal nSrcWidth, ByVal nSrcHeight, ByVal dwRop&)
Declare Function DeleteDC Lib "GDI" (ByVal hDC)
Declare Function Escape Lib "GDI" (ByVal hDC, ByVal nEscape,
         ByVal nCount, lplnData As Any, lpOutData As Any)
Declare Function GetSystemMetrics Lib "User" (ByVal nIndex)

Const SM_CYCAPTION = 4
Const SM_CXBORDER = 5
Const SM_CYBORDER = 6
Const SM_CXDLGFRAME = 7
Const SM_CYDLGFRAME = 8
Const SM_CXFRAME = 32
Const SM_CYFRAME = 33

Const TWIPS = 1
Const PIXEL = 3
Const NILL = 0&
Const SRCCOPY = &HCC0020
Const NEWFRAME = 1

Dim ModeRatio, XOffset, YOffset As Integer
```

2. Set the following properties at design time:

Control	Property	Setting
Form1	Name	Form1 (default)
Form1.Picture1	Name	Picture1 (default)
Form1.Picture2	Name	Picture2 (default)
Form1.File1	Name	File1 (default)

   (In Visual Basic version 1.0 for Windows, set the CtlName/FormName
    Property for the above objects instead of the Name property.)

You can add any other control(s) to the form to print. If a picture control is drawn at run time, be sure to set its AutoRedraw property to True so that the graphics will be transferred by the Windows API call BitBlt and eventually printed by StretchBlt.

3. Add the following code to the Form_Load procedure of Form1:

```
Sub Form_Load ()
' Size the form explicitly to match parameters of StretchBlt.
' Or use design time size to set coordinates.
        Form1.Move 1095, 1200, 8070, 5280

' Size two example controls.
        File1.Move 4080, 120, 2775, 2535
        Picture1.Move 240, 120, 2775, 2535

' Put up a caption to indicate how to print the form.
    Form1.Caption = "Double Click to Print Form And Text"

' The following *optional* code illustrates creating a persistent
' bitmap that will successfully StretchBlt to the printer.
    Picture1.AutoRedraw = -1  ' Create persistent bitmap of picture
                              ' contents.
    Picture1.Line (0, 0)-(Picture1.ScaleWidth / 2,
    Picture1.ScaleHeight / 2), , BF
    Picture1.AutoRedraw = 0   ' Toggle off.

' Make sure the temporary workspace picture is invisible.
    Picture2.visible = 0
End Sub
```

4. Add the following code to the general procedure level of the form:

```
Sub FormPrint (localname As Form)
' Display cross.
    screen.MousePointer = 2
' Calculate ratio between ScaleMode twips and ScaleMode pixel.
    localname.ScaleMode = PIXEL
    ModeRatio = localname.height \ localname.ScaleHeight
    localname.ScaleMode = TWIPS

XOffset = (localname.width - localname.ScaleWidth) \ ModeRatio
YOffset = (localname.height - localname.ScaleHeight) \ ModeRatio
CapSize% = GetSystemMetrics(SM_CYCAPTION) ' The height of the caption.

  ' The size of the fixed single border:
FudgeFactor% = GetSystemMetrics(SM_CYBORDER)
' The fudgefactor is due to inevitable mapping errors when converting
' logical pixels to screen pixels. This example is coded for 640X480
' screen resolution. For 800X600, remove the fudgefactor.
' For other resolutions, tweak for perfection!

Select Case localname.BorderStyle
Case 0      ' None.
        XOffset = 0
        YOffset = 0
Case 1      ' Fixed Single.
        XOffset = GetSystemMetrics(SM_CXBORDER)
        YOffset = GetSystemMetrics(SM_CYBORDER) + CapSize% - FudgeFactor%
Case 2      ' Sizeable.
        XOffset = GetSystemMetrics(SM_CXFRAME)
        YOffset = GetSystemMetrics(SM_CYFRAME) + CapSize% - FudgeFactor%
Case 3      ' Fixed Double.
        XOffset = GetSystemMetrics(SM_CXDLGFRAME) + FudgeFactor%
```

```
        YOffset = GetSystemMetrics(SM_CYDLGFRAME) + CapSize%
End Select

' Size the picture to the size of the form's non-client (complete)
' area.
   Picture2.Move 0, 0, localname.Width, localname.Height

' Note that Bitblt requires coordinates in pixels.
   Picture2.ScaleMode = PIXEL
' Clear Picture property of any previous BitBlt image.
   Picture2.Picture = LoadPicture("")
' -1 equals true: Must Have This!!!
   Picture2.AutoRedraw = -1
' Assign information of the destination bitmap.
   hDestDC% = Picture2.hDC
        X% = 0: Y% = 0
        nWidth% = Picture2.ScaleWidth
        nHeight% = Picture2.ScaleHeight

' Assign information of the source bitmap.
' Source is entire client area of form (plus non-client area)
' XOffset and YOffset settings depend on the BorderStyle chosen for
' the form.
        hSrcDC% = localname.hDC
        XSrc% = -XOffset: YSrc% = -YOffset
' Show transition to BitBlt by changing MousePointer.
   Screen.MousePointer = 4
' Assign the SRCCOPY constant to the Raster operation.
   dwRop& = SRCCOPY
   ' The following statement must appear on one line.
   Suc% = BitBlt(hDestDC%, X%, Y%, nWidth%, nHeight%, hSrcDC%, XSrc%,
               YSrc%, dwRop&)

' Start the StretchBlt process now.
' Assign persistent bitmap to Picture property:
   Picture2.Picture = Picture2.Image
' StretchBlt requires pixel coordinates.
       Picture2.ScaleMode = PIXEL
       Printer.ScaleMode = PIXEL
' * The following is an example of mixing text with StretchBlt.
       Printer.Print "This is a test of adding text and bitmaps "
       Printer.Print "This is a test of adding text and bitmaps "
       Printer.Print "This is a test of adding text and bitmaps "
' * If no text is printed in this procedure,
' * then you must add minimum: Printer.Print " "
' * to initialize Printer.hDC.

' Now display hour glass for the StretchBlt to printer.
   screen.MousePointer = 11

   hMemoryDC% = CreateCompatibleDC(Picture2.hDC)
   hOldBitMap% = SelectObject(hMemoryDC%, Picture2.Picture)
' You adjust the vertical stretch factor of the form in the
' argument "Printer.ScaleHeight - 1000":
   ApiError% = StretchBlt(Printer.hDC, 0, 192,
               Printer.ScaleWidth - 300, Printer.ScaleHeight - 1000,
               hMemoryDC%, 0, 0, Picture2.ScaleWidth,
```

```
               Picture2.ScaleHeight, SRCCOPY)  ' concatenate above
' The second parameter above allows for text already printed: modify
' accordingly.
   hOldBitMap% = SelectObject(hMemoryDC%, hOldBitMap%)
   ApiError% = DeleteDC(hMemoryDC%)
' * The following is an example of mixing text with StretchBlt.
' Set the printer currentY to allow for the size of the StretchBlt
' image. (This is relative to size of form and stretch factors chosen)
        Printer.currentY = 2392 ' In Twips.
        Printer.Print "This is for text after the StretchBlt"
        Printer.Print "This is for text after the StretchBlt"
        Printer.Print "This is for text after the StretchBlt"
   Printer.EndDoc
   ApiError% = Escape(Printer.hDC, NEWFRAME, 0, NILL, NILL)

' Reset MousePointer to default.
   Screen.MousePointer = 1
End Sub
```

5. Add the following code to the Double_Click event:

```
Sub Form_DblClick ()
   FormPrint Form1
End Sub
```

6. After saving the project, run the example.

Double-click the form to invoke the FormPrint procedure. Any form passed as a parameter to FormPrint will be printed. BitBlt will transfer the image to the Picture control, then StretchBlt transfers it to the printer DC, which will print the image that was transferred by BitBlt.

Optionally, you could place text or graphics in the picture (Form1.Picture2) before printing with StretchBlt or print directly to the page using Printer.Print or Printer.Line. If you choose the latter method, by adjusting the second and third parameters of StretchBlt, you can make the already printed content be followed by the image of the form on the same page.

Reference(s):

"Microsoft Windows Programmer's Reference Book and Online Resource" (Add-on kit number 1-55615-413-5)

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgPrint

Creating TOPMOST or "Floating" Window in Visual Basic

Article ID: Q84251

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can create a "floating" window such as that used for the Microsoft Windows version 3.1 Clock by using the SetWindowPos Windows API call.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

A floating (or TOPMOST) window is a window that remains constantly above all other windows, even when it is not active. Examples of floating windows are the Find dialog box in WRITE.EXE, and CLOCK.EXE (when Always on Top is selected from the Control menu).

There are two methods to produce windows that "hover" or "float," one of which is possible in Visual Basic for Windows. This method is described below:

Call SetWindowPos, specifying an existing non-topmost window and HWND_TOPMOST as the value for the second parameter (hwndInsertAfter):

Use the following declarations:

```
Declare Function SetWindowPos Lib "user" (ByVal h%, ByVal hb%,
        ByVal x%, ByVal y%, ByVal cx%, ByVal cy%, ByVal f%) As Integer
' The above Declare statement must appear on one line.

Global Const SWP_NOMOVE = 2
Global Const SWP_NOSIZE = 1
Global Const FLAGS = SWP_NOMOVE Or SWP_NOSIZE
Global Const HWND_TOPMOST = -1
Global Const HWND_NOTOPMOST = -2
```

To set the form XXXX to TOPMOST, use the following code:

```
success% = SetWindowPos (XXXX.hWnd, HWND_TOPMOST, 0, 0, 0, 0, FLAGS)
REM success% <> 0 When Successful
```

To reset the form XXXX to NON-TOPMOST, use the following code:

```
success% = SetWindowPos (XXXX.hWnd, HWND_NOTOPMOST, 0, 0, 0, 0, FLAGS)
REM success% <> 0 When Successful
```

Note: This attribute was introduced in Windows, version 3.1, so remember to make a GetVersion() API call to determine whether the application is running under Windows, version 3.1.

Reference(s):

"Page 892 of Microsoft Windows 3.1 Programmer's Reference, Volume 2, Functions,"

Additional reference words: Win31 Float Topmost Notopmost Setwindowpos top most
KBCategory:
KBSubcategory: APrgWindow

Property or Control Not Found When Use Form/Control Data Type

Article ID: Q84383

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You do not need to prefix a control name with the parent form name when you are accessing the property of a control from a Sub or Function to which the control is passed as a parameter. If you use the syntax

```
form.control.property
```

to access the property of the control, you will get a "Property or Control not found" error.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The full syntax to access a property of a control on a form is as follows:

```
form.control.property
```

If the control whose property you are accessing is on the form where the code resides, you do not need to prefix the control name with the form name. For example, if command button Command1 is on Form1 and you want to set its Enabled property to False (0) in the event procedure Command1_Click, you can use the following:

```
Command1.Enabled = 0
```

You can use the same syntax if the statement is in the general Declarations section of Form1. However, if you want to access the Enabled property of Command1 on a form other than its parent form, or from a Sub or Function in a module, you need to use the full syntax (with the form name).

The property of the control can also be accessed in a module by using the full syntax. For example, to disable Command1 (which is on Form1) in MODULE1.BAS, add the following:

```
Sub AccessProperty
    Form1.Command1.Enabled = 0
End Sub
```

However, if you are passing the control as an argument to a Sub or Function procedure in a general module, you do not need to use the full syntax. For example

```
   Sub AccessProperty (ThisForm as Form, ThisControl as Control)
       ThisForm.ThisControl.Enabled = 0
   End Sub
```

will give you a "Property or Control 'ThisControl' not found" error. You only need to pass the control name as an argument to the procedure. For example:

```
   Sub AccessProperty (ThisControl as Control)
       ThisControl.Enabled = 0
   End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

PRB: DateValue Argument Gives "Illegal Function Call" Error

Article ID: Q84547

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

You will correctly get an "Illegal function call" error for the DateValue argument if the short date (the three numbers representing the month, day, and year such as 4/24/92) does not follow the order of the date format selected under International settings in the Windows Control Panel.

STATUS

This behavior is by design.

MORE INFORMATION

The DateValue function returns a serial number that represents the date of the string argument. The string argument can be a date in abbreviated form (three numbers that represent the month, day, and year). However, this has to conform to the Short Date Format selected in the International settings of the Control Panel.

By default, the order is MDY or the month followed by the day and then the year separated by a slash (/) or a hyphen (-). An example of a valid argument is 4/24/92 for the date April 24, 1992. Using 24/4/92 would produce an "Illegal function call" error.

Note that for the long form of the date, DateValue recognizes April 24, 1992, Apr 24, 1992, 24-Apr-1992, and 24 April 92".

Remember that you will have to restart Windows for any changes made in the International settings to take effect.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How VB Can Get Windows Status Information via API Calls

Article ID: Q84556

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY
=======

The Visual Basic for Windows program example below demonstrates how you can obtain system status information similar to the information displayed in the Windows Program Manager About box. The example program displays the following information using the Windows API function(s) indicated:

- The Windows version number with GetVersion
- The kind of CPU (80286, 80386, or 80486) and whether a math coprocessor is present with GetWinFlags
- Whether Windows is running in enhanced mode or standard mode with GetWinFlags
- The amount of free memory with GetFreeSpace and GlobalCompact
- The percentage of free system resources with SystemHeapInfo

Note: The API function SystemHeapInfo is new to Windows version 3.1 and is not available in Windows, version 3.0. All other API functions listed above are available in both Windows versions 3.0 or 3.1.

MORE INFORMATION
================

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Steps to Create Example Program
----

1. Run Visual Basic for Windows, or if Visual Basic for Windows is already running, choose New Project from the File menu (press ALT, F, N). Form1 will be created by default.

2. From the File menu, choose Add Module (press ALT, F, M). Module 1 is created by default (In Visual Basic version 1.0 for Windows, this step is unnecessary).

3. Enter the following code into the general declarations section of a code module (In Visual Basic version 1.0 for Windows, place the following in the Global module):

```
' Constants for GetWinFlags.
Global Const WF_CPU286 = &H2
Global Const WF_CPU386 = &H4
Global Const WF_CPU486 = &H8
Global Const WF_80x87 = &H400
Global Const WF_STANDARD = &H10
Global Const WF_ENHANCED = &H20
Global Const WF_WINNT = &H4000

' Type for SystemHeapInfo.
Type SYSHEAPINFO
   dwSize As Long
   wUserFreePercent As Integer
   wGDIFreePercent As Integer
   hUserSegment As Integer
   hGDISegment As Integer
End Type

Declare Function GetVersion Lib "Kernel" () As Integer
Declare Function GetWinFlags Lib "Kernel" () As Long
'Enter each of the following Declare statements as one, single line:
Declare Function GetFreeSpace Lib "Kernel" (ByVal wFlags As Integer)
   As Long
Declare Function GlobalCompact Lib "Kernel" (ByVal dwMinFree As Long)
   As Long
Declare Function SystemHeapInfo Lib "toolhelp.dll" (shi As
   SYSHEAPINFO) As Integer
```

4. Enter the following code into the Form_Load procedure of Form1:

```
Sub Form_Load ()
   Dim msg As String         ' Status information.
   Dim nl As String          ' New-line.
   nl = Chr$(13) + Chr$(10)  ' New-line.

   Show
   MousePointer = 11   ' Hourglass.
   ver% = GetVersion()
   status& = GetWinFlags()

   ' Get operating system and version.
   If status& And WF_WINNT Then
      msg = msg + "Microsoft Windows NT "
   Else
      msg = msg + "Microsoft Windows "
   End If
   ver_major$ = Format$(ver% And &HFF)
   ver_minor$ = Format$(ver% \ &H100, "00")
   msg = msg + ver_major$ + "." + ver_minor$ + nl

   ' Get CPU kind and operating mode.
   msg = msg + "CPU: "
   If status& And WF_CPU286 Then msg = msg + "80286"
   If status& And WF_CPU386 Then msg = msg + "80386"
   If status& And WF_CPU486 Then msg = msg + "80486"
   If status& And WF_80x87 Then msg = msg + " with 80x87"
   msg = msg + nl
```

```
      msg = msg + "Mode: "
      If status& And WF_STANDARD Then msg = msg + "Standard" + nl
      If status& And WF_ENHANCED Then msg = msg + "Enhanced" + nl

      ' Get free memory.
      memory& = GetFreeSpace(0)
      msg = msg + "Memory free: "
      msg = msg + Format$(memory& \ 1024, "###,###,###") + "K" + nl
      memory& = GlobalCompact(&HFFFFFFFF)
      msg = msg + "Largest free block: "
      msg = msg + Format$(memory& \ 1024, "###,###,###") + "K" + nl

      ' Get free system resources.
      ' The API SystemHeapInfo became available in Windows version 3.1.
      msg = msg + "System resources: "
      If ver% >= &H310 Then
         Dim shi As SYSHEAPINFO
         shi.dwSize = Len(shi)
         If SystemHeapInfo(shi) Then
            If shi.wUserFreePercent < shi.wGDIFreePercent Then
               msg = msg + Format$(shi.wUserFreePercent) + "%"
            Else
               msg = msg + Format$(shi.wGDIFreePercent) + "%"
            End If
         End If
      Else
         msg = msg + "n/a"
      End If

      MsgBox msg, 0, "About " + Caption
      MousePointer = 0
   End Sub
```

5. Press the F5 key to run the program.

Additional reference words: 1.00 2.00 3.00 3.10 286 386 486
KBCategory:
KBSubcategory: APrgWindow

How to Determine the Number of VB Applications Running at Once

Article ID: Q84836

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

To determine the total number of Microsoft Visual Basic for Windows applications running at any given time, you can use the Microsoft Windows API functions GetModuleHandle and GetModuleUsage.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The following code fragment demonstrates a technique to find the total number of Visual Basic for Windows applications currently executing by determining the number of instances of the Visual Basic run-time module (VBRUN100.DLL) with the Windows API functions GetModuleHandle and GetModuleUsage. Remember that Visual Basic for Windows itself is not counted; only applications created with Visual Basic for Windows are included.

Steps to Create Example Program

1. Start several Visual Basic for Windows applications and leave them running.

2. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

3. Enter the following Windows API function declarations into the General Declarations section of Form1:

   ```
   Declare Function GetModuleUsage% Lib "kernel" (ByVal hModule%)
   Declare Function GetModuleHandle% Lib "kernel" (ByVal FileName$)
   ```

4. Place a command button (Command1) on Form1. Double-click that button to open the Code window. In the Command1_Click procedure, add the following code:

   ```
   Sub Command1_Click ()
     msg$ = "Number of executing VB Apps: "

     hModule% = GetModuleHandle("VBRUN300.DLL")
       ' For Visual Basic versions 1.0 and 2.0 for Windows, use
   ```

```
        ' VBRun100.DLL and VBRun2.00.DLL respectively.
      nInstances% = GetModuleUsage(hModule%)

      msg$ = msg$ + Str$(nInstances%)
      MsgBox msg$
    End Sub
```

5. From the File menu, choose Make EXE File.

6. Press the F5 key to run the file.

7. Click the command button.

A message box displays the total number of executing Visual Basic for Windows applications.

Note: This program itself will count as one application.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgWindow

VB "Bad DLL Calling Convention" Means Stack Frame Mismatch

Article ID: Q85108

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

When you call a dynamic link library (DLL) function from Visual Basic for Windows, the "Bad DLL Calling Convention" error is often caused by incorrectly omitting or including the ByVal keyword from the Declare statement or the Call statement. The ByVal keyword affects the size of data placed on the stack. Visual Basic for Windows checks the change in the position of the stack pointer to detect this error.

When Visual Basic for Windows generates the run time error "Bad DLL Calling Convention," the most common cause when calling API functions is omitting the ByVal keyword from the Declaration of the external function or from the call itself. It can also occur due to including the ByVal keyword when the function is expecting a 4 byte pointer to the parameter instead of the value itself. This changes the size (number of bytes) of the values placed on the stack, and upon return from the DLL, Visual Basic for Windows detects the change in the position of the stack frame and generates the error.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

There are two calling conventions, or inter-language protocols: the Pascal/Basic/FORTRAN calling convention, and the C calling convention. Visual Basic for Windows uses the Pascal calling convention, as do the Microsoft Window API functions and other Microsoft Basic language products. Under the Pascal convention, it is the responsibility of the called procedure to adjust or clean the stack. (In addition, parameters are pushed onto the stack in order from the leftmost parameter to the rightmost.) Because the DLL function is responsible for adjusting the stack based on the type and number of parameters it expects, Visual Basic for Windows checks the position of the stack pointer upon return from the function. If the called routine has adjusted the stack to an unexpected position, then Visual Basic for Windows generates a "Bad DLL Calling Convention" error. Visual Basic for Windows assumes a stack position discrepancy because the DLL function uses the C calling convention. With the C calling convention, the calling program is responsible for adjusting the stack immediately after the called routine returns control.

Steps to Reproduce Behavior

Create a simple DLL using Microsoft Quick C for Windows or any compiler capable of creating Windows DLLs. The following example is in C and written for Quick C for Windows:

Stacking.C

```
#include <windows.h>
long far pascal typecheck (long a, float b, short far *c, char far *buff)
{
short retcode;
a = a * 3;
retcode = MessageBox(NULL, "I am in the DLL", "BOX", MB_OK);
return (a);
}
```

Stacking.DEF

```
LIBRARY      STACKING
EXETYPE     WINDOWS
STUB        'winstub.exe'
STACKSIZE   5120
HEAPSIZE    1024
DATA PRELOAD MOVEABLE SINGLE        ; ADD THESE TWO LINES
CODE PRELOAD MOVEABLE DISCARDABLE   ; TO AVOID WARNINGS.
EXPORTS
    typecheck   @1
     WEP           @2
```

Add the following code to the general Declarations module in a Visual Basic for Windows form:

```
Declare Function typecheck Lib "d\stacking.dll" (ByVal a As Long,
      ByVal b As Single, c As Integer, ByVal s As String) As Long
```

Note: The above declaration must be placed on one line.

In the Form_Click event:

```
Sub Form_Click ()
Dim a As Long   ' Explicitly type the variables.
Dim b As Single
Dim c As Integer
Dim s As String
a = 3                      ' Initialize the variables.
b = 4.5
c = 6
s = "Hello there! We've been waiting for you!"
Print typecheck(a, b, c, s)
End Sub
```

Running the program as written above will not generate the error. Now add the ByVal keyword before the variable named c in the Visual Basic for Windows Declaration. Run the program. Note that the MessageBox function pops a box first, and then the error box pops up indicating that Visual Basic for Windows checks the stack upon return to see if it has been correctly adjusted. Because the DLL expected a 4-byte pointer and received a 2-byte value, the stack has not adjusted back

to the initial frame.

As another test, first remove the ByVal keyword before the variable 'c' that you added in the previous test. Declare the parameter 'a As Any' instead of As Long. Change the type of the variable 'a' in the Form_Click to Integer. Run the program again. Using As Any turns off type checking by Visual Basic for Windows. Because the program passed an integer ByVal instead of the long that the DLL expected, the stack frame is off and the error is generated.

Reference(s):

"Microsoft BASIC 7.0: Programmer's Guide" for versions 7.0 and 7.1, pages 423-426

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

Print Form or Client Area to Size on PostScript or PCL Printer

Article ID: Q85978

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article demonstrates two Visual Basic procedures: PrintWindow and PrintClient. Both procedures allow you to print a control or form at a specified size and location (a printed page, another form, or a picture control).

The PrintWindow procedure allows you to print the entire control including the border, caption, and menus.

The PrintClient procedure prints everything contained in the form or control excluding the border, caption, and menus. When passed a form, the PrintClient procedure works just like Visual Basic's PrintForm method.

Both procedures (PrintWindow and PrintClient) print all child controls contained in the form or control. And both use the StretchDIBits Window API function as well as other Windows API functions to print a form or control. These functions will print to both Postscript and PCL (printer control language) or HP-type LaserJet printers.

MORE INFORMATION

This information is included with the Help file provided with Microsoft Visual Basic version 3.0.

Step-by-Step Example

The following steps show you how to create a program that prints a form on the printer.

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add two command buttons (Command1 and Command2) to Form1.

3. Load the WINLOGO.BMP file (or some other large bitmap) into the Picture property of Form1. WINLOGO.BMP should be in the \WINDOWS directory.

4. From the File menu, choose New Module (ALT, F, M). Module1 is created by default.

5. Add the following code to Module1:

```
'*************************************************************************
'* Project
'*          PrintAll.MAK
'*
'* Contents
'*          PrintALL.FRM  (Form1)
'*          PrintALL.BAS
'* Structure
'*          Form1 can contain any number of controls.
'*          The minimum number to demonstrate both client area
'*          printing and entire form printing is two command buttons.
'*          For illustration, assign a large bitmap to the picture
'*          property of Form1.
'*
'* Description:
'*          This example successfully prints on both PostScript and
'*          PCL (Printer Control Language: the non-PostScript type)
'*          printers. The printer output is of the same resolution as
'*          you would expect from the PrintForm method or from
'*          printing the form from the VB environment. Both the
'*          PrintClient and PrintWindow procedures are generic in
'*          that they can be used to print any visible window. To
'*          incorporate the code into your project, add PrintAll.BAS
'*          in the project and paste the code in the PrintAll.GLB
'*          program into a code module. The code in the
'*          Command1_Click and Command2_Click events demonstrates how
'*          to call the two procedures PrintWindow and PrintClient. To
'*          print any active window, use the Appactivate and the
'*          GetFocus functions to get the handle to the window to pass
'*          to the procedures.
'*
'*************************************************************************
'*************************************************************************
'* Module
'*      PrintAll.BAS
'*
'* Description:
'*      Contains all the necessary Windows API function and Type
'*      structure declarations
'*************************************************************************
DefInt A-Z

Type BITMAPINFOHEADER_Type
    biSize As Long
    biWidth As Long
    biHeight As Long
    biPlanes As Integer
    biBitCount As Integer
    biCompression As Long
    biSizeImage As Long
    biXPelsPerMeter As Long
    biYPelsPerMeter As Long
    biClrUsed As Long
    biClrImportant As Long
End Type

Type BITMAPINFO_Type
```

```
    BitmapInfoHeader As BITMAPINFOHEADER_Type
    bmiColors As String * 1024
End Type

Type RectType
    Left As Integer
    Top As Integer
    Right As Integer
    Bottom As Integer
End Type

Type PointType
    x As Integer
    y As Integer
End Type

' DC related API
Declare Function CreateCompatibleDC Lib "gdi" (ByVal hDC)
Declare Function GetWindowDC Lib "user" (ByVal hWnd)
Declare Function GetDC Lib "user" (ByVal hWnd)
Declare Function ReleaseDC Lib "user" (ByVal hWnd, ByVal hDC)
Declare Function DeleteDC Lib "gdi" (ByVal hDC)

' Graphics related API
' Enter the following Declare statement as one, single line:
Declare Function BitBlt Lib "gdi" (ByVal hDC, ByVal x, ByVal y,
   ByVal w, ByVal h, ByVal hDC, ByVal x, ByVal y, ByVal o As Long)
' Enter the following Declare statement as one, single line:
Declare Function GetDIBits Lib "gdi" (ByVal hDC, ByVal hBitmap,
   ByVal nStartScan, ByVal nNumScans, ByVal lpBits As Long,
   BitmapInfo As BITMAPINFO_Type, ByVal wUsage)
' Enter the following Declare statement as one, single line:
Declare Function StretchDIBits Lib "gdi" (ByVal hDC, ByVal DestX,
   ByVal DestY, ByVal wDestWidth, ByVal wDestHeight, ByVal SrcX,
   ByVal SrcY, ByVal wSrcWidth, ByVal wSrcHeight, ByVal lpBits&,
   BitsInfo As BITMAPINFO_Type, ByVal wUsage, ByVal dwRop&)

' General attribute related API
Declare Function GetDeviceCaps Lib "gdi" (ByVal hDC, ByVal nIndex)
Declare Function GetWindowRect Lib "user" (ByVal hWnd, lpRect As RectType)
Declare Function GetClientRect Lib "user" (ByVal hWnd, lpRect As RectType)

' Memory allocation related API
Declare Function GlobalAlloc Lib "kernel" (ByVal wFlags, ByVal lMem&)
Declare Function GlobalLock Lib "kernel" (ByVal hMem) As Long
Declare Function GlobalUnlock Lib "kernel" (ByVal hMem)
Declare Function GlobalFree Lib "kernel" (ByVal hMem)

' Graphics object related API
' Enter the following Declare statement as one, single line:
Declare Function CreateCompatibleBitmap Lib "gdi" (ByVal hDC, ByVal nWidth,
   ByVal nHeight)
Declare Function DeleteObject Lib "gdi" (ByVal hObject)
Declare Function SelectObject Lib "gdi" (ByVal hDC, ByVal hObject)
Declare Function ClientToScreen Lib "user" (ByVal hWnd%, p As PointType)
Declare Function LPToDP Lib "gdi" (ByVal hDC, p As PointType, ByVal nCount)
```

```
' Include the following constant declarations if using Visual Basic
' version 1.0
' Const False = 0
' Const True = Not False

Const HORZRES = 8
Const VERTRES = 10

Const SRCCOPY = &HCC0020
Const NEWFRAME = 1
Const BITSPIXEL = 12
Const PLANES = 14

Const BI_RGB = 0
Const BI_RLE8 = 1
Const BI_RLE4 = 2

Const DIB_PAL_COLORS = 1
Const DIB_RGB_COLORS = 0

Const GMEM_MOVEABLE = 2
```

4. Add the following function (PrintWindow) to Module1:

```
'********************************************************************
'* Title
'*      PrintWindow()
'*
'* Description
'*
'*      Copies the entire window (form or control) to another
'*      window (form or control) or device such as a printer. This
'*      routine is capable of printing complete form images on any
'*      printer that has Windows drivers loaded including Postscript.
'*
'*      The API functions GetDIBits and StretchDIBits are used to copy
'*      the client area image to the destination window or device.
'*
'* Parameters:
'*      hDC_Dest        Handle to the DC of the destination device or
'*                       window.
'*      DestX           X position of where the image will be
'*                       displayed on the destination device.
'*      DestY           Y position of where the image will be
'*                       displayed on the destination device.
'*      DestDevWidth    Pixel width of the destination device.
'*      DestDevHeight   Pixel height of the destination device.
'*      hWnd_SrcWindow  Window handle of the source window to be
'*                       displayed on the destination device.
'********************************************************************
' Enter the following statement as one, single line:
Function PrintWindow (ByVal hDC_Dest, ByVal DestX, ByVal DestY,
   ByVal DestDevWidth, ByVal DestDevHeight, ByVal hWnd_SrcWindow)

   Dim Rect As RectType
   Dim BitmapInfo As BITMAPINFO_Type
```

```
cr$ = Chr$(13)

' Get the DC for the entire window including the non-client area.
hDC_Window = GetWindowDC(hWnd_SrcWindow)
hDC_Mem = CreateCompatibleDC(hDC_Window)

' Get the pixel dimensions of the screen.  This is necessary so
' that we can determine the relative size of the window compared to
' the screen
ScreenWidth = GetDeviceCaps(hDC_Window, HORZRES)
ScreenHeight = GetDeviceCaps(hDC_Window, VERTRES)

' Get the pixel dimensions of the window to be printed.
r = GetWindowRect(hWnd_SrcWindow, Rect)
Window_Width = Abs(Rect.Right - Rect.Left)
Window_Height = Abs(Rect.Bottom - Rect.Top)

' Create a bitmap compatible with the window DC. Enter the following
' statement as one, single line:

hBmp_Window = CreateCompatibleBitmap(hDC_Window, Window_Width,
   Window_Height)

' Select the bitmap to hold the window image into the memory DC.
hPrevBmp = SelectObject(hDC_Mem, hBmp_Window)

' Copy the image of the window to the memory DC. Enter the following
' statement as one, single line:
r1 = BitBlt(hDC_Mem, 0, 0, Window_Width, Window_Height, hDC_Window,
   0, 0, SRCCOPY)

BitsPerPixel = GetDeviceCaps(hDC_Mem, BITSPIXEL)
ColorPlanes = GetDeviceCaps(hDC_Mem, PLANES)

BitmapInfo.BitmapInfoHeader.biSize = 40
BitmapInfo.BitmapInfoHeader.biWidth = Window_Width
BitmapInfo.BitmapInfoHeader.biHeight = Window_Height
BitmapInfo.BitmapInfoHeader.biPlanes = 1
BitmapInfo.BitmapInfoHeader.biBitCount = BitsPerPixel * ColorPlanes
BitmapInfo.BitmapInfoHeader.biCompression = BI_RGB
BitmapInfo.BitmapInfoHeader.biSizeImage = 0
BitmapInfo.BitmapInfoHeader.biXPelsPerMeter = 0
BitmapInfo.BitmapInfoHeader.biYPelsPerMeter = 0
BitmapInfo.BitmapInfoHeader.biClrUsed = 0
BitmapInfo.BitmapInfoHeader.biClrImportant = 0

' Calculate the ratios based on the source and destination
' devices. This will help to cause the size of the window image
' to be approximately the same proportion on another device
' such as a printer.
WidthRatio! = Window_Width / ScreenWidth
HeightAspectRatio! = Window_Height / Window_Width

PrintWidth = WidthRatio! * DestDevWidth
PrintHeight = HeightAspectRatio! * PrintWidth

' Calculate the number of bytes needed to store the image assuming
```

```
   ' 8 bits/pixel.
   BytesNeeded& = CLng(Window_Width + 1) * (Window_Height + 1)

   ' Allocate a buffer to hold the bitmap bits.
   hMem = GlobalAlloc(GMEM_MOVEABLE, BytesNeeded&)

   ' Enter the following If statement as one, single line:
   If hDC_Window <> 0 And hBmp_Window <> 0 And hDC_Dest <> 0 And
      hMem <> 0 Then

      lpBits& = GlobalLock(hMem)

      ' Get the bits that make up the image and copy them to the
      ' destination device.
      ' Enter the following r2 statement as one, single line:
      r2 = GetDIBits(hDC_Mem, hBmp_Window, 0, Window_Height, lpBits&,
         BitmapInfo, DIB_RGB_COLORS)
      ' Enter the following r3 statement as one, single line:
      r3 = StretchDIBits(hDC_Dest, DestX, DestY, PrintWidth, PrintHeight,
         0, 0, Window_Width, Window_Height, lpBits&, BitmapInfo,
         DIB_RGB_COLORS, SRCCOPY)
   End If

   ' Reselect in the previous bitmap and select the source image bitmap.
   r = SelectObject(hDC_Mem, hPrevBmp)
   r = DeleteDC(hDC_Mem)

   ' Release or delete DC's, memory and objects.
   r = GlobalUnlock(hMem)
   r = GlobalFree(hMem)
   r = DeleteDC(hDC_Window)
   r = DeleteObject(hBmp_Window)

   ' Return true if the window was successfully printed.
   If r2 <> 0 And r3 <> 0 Then
      PrintWindow = True
   Else
      PrintWindow = False
   End If

End Function
```

6. Add the following function, PrintClient to Module1:

```
'*********************************************************************
'* Title
'*      PrintClient()
'*
'* Description
'*
'*      Copies the client area of a window visible on the desktop to
'*      another window or device such as a printer. This routine is
'*      capable of printing client area images on any printer that has
'*      Windows drivers loaded including PostScript.
'*
'*      The API functions GetDiBits and StretchBits are used to copy
'*      the client area image to the destination device.
```

```
'*
'* Parameters:
'*      hDC_Dest        Handle to the DC of the destination device or
'*                       window.
'*      DestX           X position of where the image will be
'*                       displayed on the destination device.
'*      DestY           Y position of where the image will be
'*                       displayed on the destination device.
'*      DestDevWidth    Pixel width of the destination device.
'*      DestDevHeight   Pixel height of the destination device.
'*      hWnd_SrcWindow  Window handle of the source window to be
'*                       displayed on the destination device.
'*****************************************************************

' Enter the following Function statement as one, single line:
Function PrintClient (ByVal hDC_Dest, ByVal DestX, ByVal DestY,
   ByVal DestDevWidth, ByVal DestDevHeight, ByVal hWnd_SrcWindow)

   Dim Rect As RectType, RectClient As RectType
   Dim BitmapInfo As BITMAPINFO_Type
   '*
   Dim pWindow As PointType, pClient As PointType, pDiff As PointType
   '*

   cr$ = Chr$(13)

   ' Get the DC for the entire window including the non-client area.
   hDC_Window = GetWindowDC(hWnd_SrcWindow)
   hDC_Mem = CreateCompatibleDC(hDC_Window)

   ' Get the pixel dimensions of the screen.
   ScreenWidth = GetDeviceCaps(hDC_Window, HORZRES)
   ScreenHeight = GetDeviceCaps(hDC_Window, VERTRES)

   ' Get the pixel dimensions of the window to be printed.
   r = GetWindowRect(hWnd_SrcWindow, Rect)
   Window_Width = Abs(Rect.Right - Rect.Left)
   Window_Height = Abs(Rect.Bottom - Rect.Top)

   ' Create a bitmap compatible with the window DC.
   ' Enter the following statement as one, single line:
   hBmp_Window = CreateCompatibleBitmap(hDC_Window, Window_Width,
      Window_Height)

   ' Select the bitmap to hold the window image into the memory DC.
   hPrevBmp = SelectObject(hDC_Mem, hBmp_Window)

   ' Copy the image of the window to the memory DC.
   ' Enter the following statement as one, single line:
   r1 = BitBlt(hDC_Mem, 0, 0, Window_Width, Window_Height,
      hDC_Window, 0, 0, SRCCOPY)

   ' Get the dimensions of the client area.
   r = GetClientRect(hWnd_SrcWindow, RectClient)
   Client_Width = Abs(RectClient.Right - RectClient.Left)
   Client_Height = Abs(RectClient.Bottom - RectClient.Top)
```

```
' Calculate the pixel difference (x and y) between the upper-left
' corner of the non-client area and the upper-left corner of the
' client area.
pClient.x = RectClient.Left
pClient.y = RectClient.Top
r = ClientToScreen(hWnd_SrcWindow, pClient)

xDiff = Abs(pClient.x - Rect.Left)
yDiff = Abs(pClient.y - Rect.Top)

' Create a DC and bitmap to represent the client area of the window.
hDC_MemClient = CreateCompatibleDC(hDC_Window)

' Enter the following statement as one, single line:
hBmp_Client = CreateCompatibleBitmap(hDC_Window, Client_Width,
   Client_Height)

hBmpClientPrev = SelectObject(hDC_MemClient, hBmp_Client)

' Bitblt client area of window to memory bitmap representing the client
' area.
' Enter the following statement as one, single line:
r = BitBlt(hDC_MemClient, 0, 0, Client_Width, Client_Height,
   hDC_Mem, xDiff, yDiff, SRCCOPY)

' Reselect in the previous bitmap and select the source image bitmap.
r = SelectObject(hDC_Mem, hPrevBmp)

' Delete the DC a and bitmap associated with the window.
r = DeleteDC(hDC_Window)
r = DeleteObject(hBmp_Window)

BitsPerPixel = GetDeviceCaps(hDC_MemClient, BITSPIXEL)
ColorPlanes = GetDeviceCaps(hDC_MemClient, PLANES)

BitmapInfo.BitmapInfoHeader.biSize = 40
BitmapInfo.BitmapInfoHeader.biWidth = Client_Width
BitmapInfo.BitmapInfoHeader.biHeight = Client_Height
BitmapInfo.BitmapInfoHeader.biPlanes = 1
BitmapInfo.BitmapInfoHeader.biBitCount = BitsPerPixel * ColorPlanes
BitmapInfo.BitmapInfoHeader.biCompression = BI_RGB
BitmapInfo.BitmapInfoHeader.biSizeImage = 0
BitmapInfo.BitmapInfoHeader.biXPelsPerMeter = 0
BitmapInfo.BitmapInfoHeader.biYPelsPerMeter = 0
BitmapInfo.BitmapInfoHeader.biClrUsed = 0
BitmapInfo.BitmapInfoHeader.biClrImportant = 0

' Calculate the ratios based on the source and destination
' devices. This will help to cause the size of the window image to
' be approximately the same proportion on another device such as
' a printer.
WidthRatio! = Client_Width / ScreenWidth
HeightAspectRatio! = Client_Height / Client_Width

PrintWidth = WidthRatio! * DestDevWidth
PrintHeight = HeightAspectRatio! * PrintWidth
```

```
   ' Calculate the number of bytes needed to store the image assuming
   ' 8 bits/pixel.
   BytesNeeded& = CLng(Window_Width + 1) * (Window_Height + 1)

   ' Allocate a buffer to hold the bitmap bits.
   hMem = GlobalAlloc(GMEM_MOVEABLE, BytesNeeded&)

   If hDC_Window <> 0 And hBmp_Window <> 0 And hDC_Dest <> 0 And
      hMem <> 0 Then

      lpBits& = GlobalLock(hMem)

      ' Get the bits that make up the image and copy them to the
      ' destination device.

      ' Enter the following r2 statement as one, single line:
      r2 = GetDIBits(hDC_MemClient, hBmp_Client, 0, Client_Height,
         lpBits&, BitmapInfo, DIB_RGB_COLORS)

      ' Enter the following r3 statement as one, single line:
      r3 = StretchDIBits(hDC_Dest, DestX, DestY, PrintWidth, PrintHeight,
         0, 0, Client_Width, Client_Height, lpBits&, BitmapInfo,
         DIB_RGB_COLORS, SRCCOPY)
   End If

   ' Select in the previous bitmap.
   r = SelectObject(hDC_MemClient, hBmpClientPrev)

   ' Release or delete DC's, memory and objects.
   r = GlobalUnlock(hMem)
   r = GlobalFree(hMem)
   r = DeleteDC(hDC_MemClient)
   r = DeleteObject(hBmp_Client)
   r = ReleaseDC(hWnd_SrcWindow, hDC_Form)

   ' Return true if the window was successfully printed.
   If r2 <> 0 And r3 <> 0 Then
      PrintClient = True
   Else
      PrintClient = False
   End If

End Function
```

7. Add DefInt A-Z to the general declarations level of Form1.

8. Add the following code to the Command1_Click event:

```
   Sub Command1_Click ()
      ' The ScaleMode must be set to pixels for the PrintWindow
      ' routine to print correctly.
      Printer.ScaleMode = 3

      ' Change MousePointer to an hourglass.
      Screen.MousePointer = 11

      ' Initialize the printer.
```

```
      Printer.Print ""

      ' Copy the image of the form to the printer.
      ' To print Command1 instead, you can substitute Command1.hWnd for
      ' Form1.hWnd as the last argument.
      ' Enter the following statement as one, single line:
      r = PrintClient(Printer.hDC, 100, 100, Printer.ScaleWidth,
         Printer.ScaleHeight, Form1.hWnd)

      ' Display an error if the return value from PrintWindow is zero.
      If Not r Then
         MsgBox "Unable to print the form"
      Else
         Printer.EndDoc
      End If

      Screen.MousePointer = 0
   End Sub
```

9. Add the following code to the Command2_Click event:

```
   Sub Command2_Click ()
      ' The ScaleMode must be set to pixels for the PrintWindow
      ' routine to print correctly.
      Printer.ScaleMode = 3

      ' Change MousePointer to an hourglass.
      Screen.MousePointer = 11

      ' Initialize the printer.
      Printer.Print ""

      ' Copy the image of the form to the printer.
      ' To print Command1 instead, you can substitute Command1.hWnd for
      ' Form1.hWnd as the last argument.
      ' Enter the following statement as one, single line:
      r = PrintWindow(Printer.hDC, 100, 100, Printer.ScaleWidth,
         Printer.ScaleHeight, Form1.hWnd)

      ' Display an error if the return value from PrintWindow is zero.
      If Not r Then
         MsgBox "Unable to print the form"
      Else
         Printer.EndDoc
      End If

      Screen.MousePointer = 0
   End Sub
```

10. Run the program.

Choose the Command1 button to print only the client area of Form1. Choose the Command2 button to print the entire area of the form.

Note that you can print any of the forms or controls in a project by using this method. Control the size and placement of the forms by changing the second, third, fourth, and fifth parameters of the call to StretchDIBits.

In the example shown above, the form or control is sized in proportion to the size of the screen.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgPrint

How to Play a Waveform (.WAV) Sound File in Visual Basic

Article ID: Q86281

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can play a waveform (.WAV) sound file from Microsoft Visual Basic for Windows by calling the sndPlaySound API function from the MMSYSTEM.DLL file. In order to be able to call the sndPlaySound API function, you must be using either Microsoft Windows, version 3.1 or the Microsoft Multimedia Extensions for Windows, version 3.0. The following information discusses the sndPlaySound parameters, and includes an example of how to use this function from Visual Basic for Windows.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

To use the sndPlaySound API from within a Visual Basic for Windows application, you must Declare the sndPlaySound function in either the global module or from within the Declarations section of your Code window. To declare the function, enter the following Declare statement as one, single line:

```
Declare Function sndPlaySound Lib "MMSYSTEM.DLL"
   (ByVal lpszSoundName$, ByVal wFlags%) As Integer
```

The parameters listed above are explained as follows:

Parameters
---

lpszSoundName$

Specifies the name of the sound to play. The function first searches the [sounds] section of the WIN.INI file for an entry with the specified name, and plays the associated waveform sound file. If no entry by this name exists, then it assumes the specified name is the name of a waveform sound file. If this parameter is NULL, any currently playing sound is stopped.

wFlags%

Specifies options for playing the sound using one or more of the following flags:

SND_SYNC
   The sound is played synchronously and the function does
   not return until the sound ends.

SND_ASYNC
   The sound is played asynchronously and the function
   returns immediately after beginning the sound.

SND_NODEFAULT
   If the sound cannot be found, the function returns
   silently without playing the default sound.

SND_LOOP
   The sound will continue to play repeatedly until
   sndPlaySound is called again with the lpszSoundName$
   parameter set to null. You must also specify the
   SND_ASYNC flag to loop sounds.

SND_NOSTOP
   If a sound is currently playing, the function will
   immediately return False without playing the requested
   sound.

The sndPlaySound function returns True (-1) if the sound is played, otherwise it returns False (0).

The following code example illustrates how to use the sndPlaySound API function to play a waveform (.WAV) sound file.

Add the following code to the global module or general Declarations section of your form:

```
' The following Declare statement must appear on one line.
Declare Function sndPlaySound Lib "MMSYSTEM.DLL" (ByVal
              lpszSoundName$, ByVal wFlags%) As Integer

Global Const SND_SYNC      = &H0000
Global Const SND_ASYNC     = &H0001
Global Const SND_NODEFAULT = &H0002
Global Const SND_LOOP      = &H0008
Global Const SND_NOSTOP    = &H0010
```

Add the following line of code to the appropriate function or subroutine in your application:

```
SoundName$ = "c:\windows\tada.wav"
wFlags% = SND_ASYNC Or SND_NODEFAULT
x% = sndPlaySound(SoundName$,wFlags%)
```

Note that if a large waveform (.WAV) sound file is specified and the above call fails to play the file in its entirety, you will need to adjust the settings on the appropriate sound driver.

For more information on adjusting the sound driver settings, query on the following words in the Microsoft Knowledge Base:

  Speaker and Sound and Driver and Settings and .Wav and File

Reference(s):

"Microsoft Multimedia Development Kit: Programmer's Reference"
version 1.0

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

VB for Windows Line Method Does Not Paint Last Pixel

Article ID: Q86770

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

The Line method does not paint the last pixel specified in the coordinates passed to it. Therefore, when trying to draw one line that ends on top of another, you must add to the coordinates.

STATUS

This behavior is by design and is the same as the LineTo statement in the Windows API; however, this information is not included in the Visual Basic documentation or Help menu.

MORE INFORMATION

Steps to Reproduce Behavior

1. Start Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Put a picture box (Picture1) on Form1.

3. Set the BackColor property of the picture to blue (&H00C00000&) to view the picture box more clearly.

4. Add the following code to the Picture1.Click event:

```
Sub Picture1_Click ()
   Picture1.Line (100, 100)-(500, 100)
   Picture1.Line (500, 10)-(500, 200), QBColor(15)
   Picture1.Line (100, 100)-(500, 100)
End Sub
```

5. Press F5 to start the program. Click the picture box.

You may expect that the third line statement should overwrite the second line and the point of intersection should be black. However, no intersection occurs because the last pixel of the third line is not drawn, so the third line statement does not overwrite the second line at all.

Additional reference words: 1.00 2.00
KBCategory:

KBSubcategory: APrgGrap

How to Invoke Search in Windows Help from Visual Basic Program

Article ID: Q86771

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Windows versions 3.0 and 3.1

SUMMARY

You can invoke the Search feature of the Windows version 3.0 and 3.1 Help engine from a Visual Basic program. To do this, call the Windows API function WinHelp and pass the constant HELP_PARTIALKEY (&H105) as the wCommand parameter and any string that is a NON-valid topic as the dwData parameter.

MORE INFORMATION

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. In GLOBAL.BAS (or the .BAS file of your choice in Visual Basic version 2.0), add this code:

```
Global Const HELP_PARTIALKEY = &H105
' Enter the following Declare statement entirely on one, single line.
Declare Function WinHelp Lib "User"(ByVal hWnd As Integer,
   ByVal lpHelpFile As String, ByVal wCommand As Integer,
   ByVal dwData As Any) As Integer
```

3. In the Form1 Click event procedure, add this code:

```
Sub Form_Click ()
   DummyVal$ = ""
   ' Enter the following function call entirely on one, single line:
   Temp% = WinHelp(Form1.hWnd, "c:\Windows\winhelp.hlp",
      HELP_PARTIALKEY, DummyVal$)
End Sub
```

4. Press F5 to run this example. Click the form.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: APrgOther

How to Use LZCOPYFILE Function to Decompress or Copy Files

Article ID: Q88257

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Included with Windows versions 3.0 and 3.1 is a dynamic link library (DLL) named LZEXPAND.DLL that contains routines to manipulate compressed files. The functions in LZEXPAND.DLL manipulate files that compressed by the COMPRESS.EXE utility supplied with the Windows Software Development Kit (SDK) versions 3.0 and 3.1. These functions allow you to expand (decompress) a compressed file.

The following example demonstrates how to use the LZCOPYFILE function included in LZEXPAND.DLL. This function is used to expand a compressed file or to copy a file.

MORE INFORMATION

The following is a small program that will copy or decompress a file in Visual Basic for Windows:

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. In the general Declarations section of Form1, add the following declaration entirely on one, single line:

   ```
   Declare Function LZCOPY Lib "LZEXPAND.DLL" (ByVal SOURCEHANDLE As
      Integer, ByVal DESTHANDLE As Integer) As Long
   ```

3. In the Form1 Click event procedure, add the following code:

   ```
   Open "Source" For Input As #1 ' Insert the name and path of the
                                 ' file to be decompressed, or copied.

   Open "Dest" For Output As #2  ' Insert the name and path of the
                                 ' destination file here.
   SOURCEHANDLE% = Fileattr(1,2)
   DESTHANDLE% = Fileattr(2,2)
   RETURNCODE& = Lzcopy(Sourcehandle%, Desthandle%)
   Close
   ```

4. From the Run menu, choose Start (ALT, R, S) to run the program.

The return code will be set to the number of bytes copied or set to the following value if an error occurs:

```
   -1  invalid input handle
   -2  invalid output handle
   -3  corrupt compressed file format
   -4  out of space for output file
   -5  insufficient memory for LZFile struct
   -6  bad global handle
   -7  input parameter out of range
   -8  compression algorithm not recognized

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther
```

How to Hide a Non-Visual Basic Window or Icon

Article ID: Q88476

The following information applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Visual Basic 1.0 for Windows

---

SUMMARY

Occasionally, it is desirable to hide a window from a Visual Basic for Windows application that is not owned by the Visual Basic for Windows application. For example, when using the GRAPH.VBX custom control provided with the Microsoft Professional Toolkit for Visual Basic version 1.0 for Windows and with the Professional Edition of Visual Basic version 2.0 for Windows, an icon appears at the bottom of the screen for the graphics server. This icon represents a program that is a support module for the graph control and so serves no direct purpose for the user. You can hide the icon by issuing two Windows API calls.

MORE INFORMATION

The FindWindow and ShowWindow Windows APIs can be used to hide a window. FindWindow uses the title on the top of the window to get a handle that can then be used by ShowWindow. ShowWindow can perform several different operations. In this case it makes a window invisible.

The following example hides the Graphics Server icon started by the Graph control. You can use this same technique to hide any window currently active in Windows.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the GRAPH.VBX custom control file. The GRAPH tool appears in the Toolbox. This starts the Graphics Server at the bottom of your screen.

3. Add a command button (Command1) to Form1.

4. Enter the following code into the global module taking care to enter each Declare statement entirely on one, single line:

```
Declare Function FindWindow Lib "User" (ByVal lpClassName As Any,
    ByVal lpWindowName As Any) As Integer
Declare Function ShowWindow Lib "User" (ByVal hWnd As Integer,
    ByVal nCmdShow As Integer) As Integer
```

5. Enter the following code into the Command1 click event procedure:

```
   Sub Command1_Click()
      Dim Handle As Integer
      Dim WindowName As String

      WindowName = "Graphics Server"
      Const SW_Hide = 0

      Handle = FindWindow(0&, WindowName)
      X% = ShowWindow(Handle, SW_Hide)
   End Sub
```

6. Press F5 to run the application.

When you choose the Command1 button, the Graphics Server icon becomes invisible.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Compare User-Defined Type Variables in Visual Basic

Article ID: Q88551

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The relational operators (=, <>, and so on) do not support the comparison of user-defined type variables. However, you can compare user-defined type variables by converting the variables to strings, and then comparing the strings. The Windows version 3.1 API hmemcpy can be used to convert a user-defined type variable to a string.

The hmemcpy API was introduced in Microsoft Windows version 3.1, so this technique requires Windows version 3.1 or later.

MORE INFORMATION

If you attempt to compare user-defined type variables using the relational operators, the error "Type mismatch" is displayed.

The following steps demonstrate how to compare user-defined type variables by first converting the variables to strings and then comparing the strings by using the relational operators.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Enter the following code into the global module:

```
Type myType
   f1 As String * 2
   f2 As Single
End Type

' Enter the following Declare statement entirely as one, single line:
Declare Sub hmemcpy Lib "kernel" (hpvDest As Any, hpvSource As Any,
   ByVal cbCopy As Long)
```

3. Enter the following code into the general Declarations section of Form1:

```
' type2str converts a user-defined type variable to a string.
Function type2str (t As myType) As String
   Dim s As String
   s = Space$(Len(t))
```

```
      Call hmemcpy(ByVal s, t, Len(t))
      type2str = s
   End Function
```

4. Enter the following code into the Form1 Click event procedure:

```
   Sub Form_Click ()
      Dim x As myType
      Dim y As myType

      x.f1 = "ab"
      x.f2 = 2
      y = x

      If type2str(x) = type2str(y) Then
         Print "x = y"
      Else
         Print "x <> y"
      End If

      y.f1 = "ba"
      If type2str(x) > type2str(y) Then
         Print "x > y"
      Else
         Print "x <= y"
      End If
   End Sub
```

5. Press the F5 key to run the program.

The program prints "x = y" and "x <= y" on Form1.

Additional reference words: 1.00 2.00 3.00 3.10
KBCategory:
KBSubcategory: APrgOther

How to Extract a Windows Program Icon -- Running or Not

Article ID: Q88944

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Windows versions 3.0 and 3.1

SUMMARY

The example program included below demonstrates how to extract an icon from a Windows program, whether it is currently running or not. There are two different techniques depending on whether the program is run in Windows version 3.0 or 3.1. The API function ExtractIcon, introduced in Windows version 3.1, simplifies the process of extracting the icon. In Windows version 3.0, a different approach is required. Both methods are illustrated below.

MORE INFORMATION

The example program shown below displays the icon of an application in a picture box. The example demonstrates the handling of the hDC property of the picture box control, specifically the relationship between the Refresh method, the Image property, and the AutoRedraw property. The code in the Command3_Click event demonstrates how to transfer the captured icon image to the Picture property of a picture box (Picture2).

Step-by-Step Example

1. Start Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Create the following controls with the default property settings:

   - Picture1
   - Picture2
   - Command1
   - Command2
   - Command3

3. Place the code below into the general Declarations section of Form1 taking care to enter each Declare statement on one, single line:

```
' API declarations used in Windows version 3.0 method.
Declare Function GetActiveWindow Lib "User" () As Integer
Declare Function PostMessage Lib "User" (ByVal hWnd As Integer,
   ByVal wMsg As Integer, ByVal wParam As Integer, ByVal lParam As Any)
   As Integer
Declare Function FindWindow Lib "User" (ByVal lpClassName As Any,
```

```
   ByVal lpWindowName As Any) As Integer
Declare Function LoadLibrary Lib "Kernel" (ByVal lpLibFileName
   As String) As Integer
Declare Function GetWindowWord Lib "User" (ByVal hWnd As Integer,
   ByVal nIndex As Integer) As Integer
Declare Function LoadIcon Lib "User" (ByVal hInstance As Integer,
   ByVal lpIconName As Any) As Integer

' API declarations used in Windows version 3.1 method.
Declare Function GetModuleHandle Lib "Kernel" (ByVal lpModuleName
   As String) As Integer
Declare Function GetClassWord Lib "User" (ByVal hWnd As Integer,
   ByVal nIndex As Integer) As Integer
Declare Function ExtractIcon Lib "SHELL" (ByVal hInst As Integer,
   ByVal lpszexename As String, ByVal hIcon As Integer) As Integer

' API declaration used by both Windows version 3.0 and 3.1 methods.
Declare Function DrawIcon Lib "User" (ByVal hDC As Integer, ByVal x
   As Integer, ByVal Y As Integer, ByVal hIcon As Integer) As Integer

' Window field offsets for GetClassWord() and GetWindowWord().
Const GWW_HINSTANCE = (-6)
Const GCW_HMODULE = (-16)
' Constants for SendMessage and PostMessage.
Const WM_CLOSE = &H10
' If using Visual Basic version 1.0, remove the single quotation mark
' from the following line of code:
' Const NULL = 0&
```

4. Place the following code in the Form_Load event of Form1:

```
Sub Form_Load ()
   Command1.Caption = " 3.0 method "
   Command2.Caption = " 3.1 method "
   Command3.Caption = " Transfer "
   Form1.Caption = " Example of Extracting an Icon"
   Form1.Width = Screen.Width * 2 / 3
   Form1.Height = Screen.Height / 2

   ' Center the form on the screen.
   ' Enter the following two lines as one, single line:
   Form1.Move (Screen.Width - Form1.Width) / 2,
      (Screen.Height - Form1.Height) / 2
   ' Size and position the controls dynamically at run time.
   ' Enter the following two lines as one, single line:
   Picture1.Move 0, 0, Form1.Width / 2,
      Form1.Height - Command1.Height * 4
   ' Enter the following two lines as one, single line:
   Picture2.Move Form1.Width / 2, 0, Form1.Width,
      Form1.Height - Command2.Height * 4
   ' Enter the following two lines as one, single line:
   Command1.Move (Form1.Width / 2 - Command1.Width) / 2,
      Form1.Height - Command1.Height * 4
   ' Enter the following two lines as one, single line:
   Command2.Move (Form1.Width / 2 - Command1.Width) / 2,
      Form1.Height - Command1.Height * 3
   ' Enter the following two lines as one, single line:
```

```
   Command3.Move (Form1.Width * 3 / 2 - Command2.Width) / 2,
      Form1.Height - Command2.Height * 4
End Sub
```

5. Place the following code in the Command1_Click event. Configure the code to match your situation by removing the comment apostrophe from one of the three methods and adding comment apostrophes to the other two -- to effectively enable one of the methods and disable the other two.

```
Sub Command1_Click ()
Dim hInstance As Integer, handle As Integer, hIcon As Integer
Picture1.Picture = LoadPicture("") ' clear any previous image

   ' Three alternative ways to obtain the handle of the top-level window
   ' of the program whose icon you want to extract:

   ' Method 1: If the program is currently running and you don't know
   '           the class name.
   ' AppActivate ("Program Manager") ' Set focus to application.
   ' handle = GetActiveWindow()      ' Get handle to window.
   ' Command1.SetFocus               ' Return focus to button.

   ' Method 2: If program is running and you know the class name.
   ' Handle = FindWindow("Progman", "Program Manager")

   ' Method 3: If program is not running, use path and filename.
   ' Not_Running_Way "sysedit.exe" ' Call sub at general level.
   ' Exit Sub                      ' Bypass remaining code in this Sub.

   ' Now you have the handle -- use it to obtain the instance handle.
   hInstance = GetWindowWord(handle, GWW_HINSTANCE)
   Picture2.Print "3.0 method "
   Picture2.Print "handle="; Hex$(handle)
   Picture2.Print "hInstance= "; Hex$(hInstance)  ' Sanity check.

   ' Iterate through icon resource identifier values
   '  until you obtain a valid handle to an icon.
   Do
      hIcon = LoadIcon(hInstance, n&)
      n& = n& + 1
   Loop Until hIcon <> 0
   Picture2.Print "hIcon= "; Hex$(hIcon)
   Picture1.AutoRedraw = -1 ' Make hDC point to persistent bitmap.
   r = DrawIcon(Picture1.hDC, 19, 19, hIcon) 'Draw the icon.
   Picture1.Refresh         ' Refresh from persistent bitmap to Picture.
End Sub
```

6. Place the following code in the Command2_Click event. Note that the first two methods commented out are provided for information and contrast to the preferred method, method 3.

```
Sub Command2_Click ()
   Dim myhInst As Integer, hIcon As Integer
   Picture1.Picture = LoadPicture("") ' Clear the previous image.

   ' Listed below are three alternative methods that can be used to
   ' obtain the hInst of your program's module handle.
```

```
    ' Method 1: Use only with .EXE version of your program.
       ' myhInst = GetModuleHandle("Project1.exe")

    ' Method 2: Use only with your program running in the environment.
       ' myhInst = GetModuleHandle("VB.EXE")

    ' Method 3: The slick way that works in either case.
        myhInst = GetClassWord(hWnd, GCW_HMODULE)

    ' The path and filename of program to extract icon from.
    lpzxExeName$ = "moricons.dll"  ' Can also use an .EXE file here.

    ' Get handle to icon.
    hIcon = ExtractIcon(myhInst, lpzxExeName$, 0)
    Picture2.Print "3.1 method "
    Picture2.Print "myhInst= "; Hex$(myhInst) ' Sanity check.
    Picture2.Print "hIcon= "; Hex$(hIcon)   ' Sanity check.

    Picture1.AutoRedraw = -1  ' Make the picture's hDC point to the
                              ' persistent bitmap.
    r% = DrawIcon(Picture1.hDC, 19, 19, hIcon)
    Picture1.Refresh  ' Cause Windows to paint from the persistent bitmap
                      ' to show the icon.
End Sub
```

7. Place the following code in the form's general Declarations section:

```
Sub Not_Running_Way (appname As String)
    Dim hInstance As Integer, handle As Integer, hIcon As Integer
    Dim hWndShelledWindow As Integer
    Picture1.Picture = LoadPicture("") ' Clear any previous image.
    hInstance = Shell(appname, 2)
    Picture2.Print "3.0 method-application not running"
    Picture2.Print "hInstance= "; Hex$(hInstance)  ' Check return.
    r = DoEvents()  ' Allow time for shell to complete.

    ' The following technique is from another article that explains
    ' how to determine when a shelled process has terminated. It is
    ' used here to obtain the correct handle to the window of the
    ' application whose icon is being extracted. The handle is needed
    ' to close the application after the extraction is complete.
    TimeOutPeriod = 5
    fTimeOut = 0      ' Set to false.
    s! = Timer
    Do
       r = DoEvents()
       hWndShelledWindow = GetActiveWindow()
       ' Set timeout flag if time has expired.
       If Timer - s! > TimeOutPeriod Then fTimeOut = True
    Loop While hWndShelledWindow = Form1.hWnd And Not fTimeOut
    ' If a timeout occurred, display a timeout message and terminate.
    If fTimeOut Then
       MsgBox "Timeout waiting for shelled application", 16
       Exit Sub
    End If
```

```
      ' Iterate through icon resource identifier values
      ' until you obtain a valid handle to an icon.
      Do
         hIcon = LoadIcon(hInstance, n&)
         n& = n& + 1
      Loop Until hIcon <> 0

      Picture2.Print "HICON= "; Hex$(hIcon)
      Picture1.AutoRedraw = -1 ' Make hDC point to persistent bitmap.
      r = DrawIcon(Picture1.hDC, 19, 19, hIcon)
      Picture2.Print "return from DrawIcon="; r
      Picture1.Refresh         ' Refresh from persistent bitmap to picture.

      ' Now post a message to the window to close the application.
      r = PostMessage(hWndShelledWindow, WM_CLOSE, NULL, NULL)
      Picture2.Print "return from PostMessage="; r
   End Sub
```

8. Place the following code in the Command3_Click event:

```
   Sub Command3_Click ()
      ' This code transfers the extracted icon's image to Picture2's
      ' Picture property and demonstrates that DrawIcon assigns the image
      ' to the hDC of Picture1, which points to the persistent bitmap
      ' (Image property), not to the Picture property.
      Picture2.Picture = LoadPicture("")  ' Clear old icon.
      Picture2.currenty = 0               ' Reset coordinates for printing
                                          '  return values.
      Picture2.currentx = 0
      Picture2.Picture = Picture1.image   ' Transfer persistent bitmap
image
                                          '  to the Picture property.
   End Sub
```

9. Press ALT F, V to save the project. Then press F5 to run the program.
   Click "3.0 method" to run the code that works in Windows version 3.0.
   Click "3.1 method" to run the code that works in Windows version 3.1.
   Click Command3 to copy the icon in Picture1 to Picture2 so that the
   icon can be accessed as Picture2.Picture.

   Both methods extract the first icon in the file. This can be
   modified to find the second or succeeding icons by:

   - Storing the value of n& in the Do Loop from the first extraction
     and plugging that in as the starting point of the next search
     in Windows version 3.0.

                                - Or -

   - Setting the third parameter of the ExtractIcon function to a
     specific index number in Windows version 3.1.

   You could do this in a loop to find and examine each icon in the file.

   The Windows version 3.0 method may take slightly longer to iterate and
   find the icon resource ID number.

Reference(s):

"Microsoft Windows Software Development Kit Volume 2"
"Microsoft Press Programmer's Reference Library Volume 2"

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

Diagnosing "Error in loading DLL" with LoadLibrary

Article ID: Q90753

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The error "Error in loading DLL" (code 48) occurs when you call a dynamic-link library (DLL) procedure and the file specified in the procedure's Declare statement cannot be loaded. You can use the Microsoft Windows API function LoadLibrary to find out more specific information about why a DLL fails to load.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The API function LoadLibrary loads a DLL and returns either a handle or an error code. If the return value is less than 32, it indicates one of the errors listed below. A return value greater than or equal to 32 indicates success and you should call the FreeLibrary function to unload the library.

LoadLibrary Error Codes

 0 System was out of memory, executable file was corrupt, or relocations were invalid.

 2 File was not found.

 3 Path was not found.

 5 Attempt was made to dynamically link to a task, or there was a sharing or network-protection error.

 6 Library required separate data segments for each task.

 8 There was insufficient memory to start the application.

10 Windows version was incorrect.

11 Executable file was invalid. Either it was not a Windows application or there was an error in the .EXE image.

12 Application was designed for a different operating system.

13 Application was designed for MS-DOS 4.0.

14 Type of executable file was unknown.

15 Attempt was made to load a real-mode application (developed for an earlier version of Windows).

16 Attempt was made to load a second instance of an executable file containing multiple data segments that were not marked read-only.

19 Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.

20 Dynamic-link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.

21 Application requires Microsoft Windows 32-bit extensions.

Steps to Create Example Program

The following program demonstrates how to call LoadLibrary to load a library and display a resulting error code.

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Enter the following code into the general declarations section:

```
Declare Function LoadLibrary Lib "kernel" (ByVal f$) As Integer
Declare Sub FreeLibrary Lib "Kernel" (ByVal h As Integer)
```

3. Enter the following code into the Form Click event handler:

```
Sub Form_Click ()
   Dim hInst As Integer
   ' Enter the name of your DLL file inside the quotes below.
   ' The file WIN.COM is not a valid DLL and demonstrates an error.
   hInst = LoadLibrary("win.com")
   If hInst > 32 Then
      MsgBox "LoadLibrary success"
      FreeLibrary (hInst)
   Else
      MsgBox "LoadLibrary error " + Format$(hInst)
   End If
End Sub
```

4. Press the F5 key to run the program. Then click Form1. The program displays the error code returned from LoadLibrary. Look up this error code in the list of errors above to find an explanation.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

Converting an Icon (.ICO) to Bitmap (.BMP) Format

Article ID: Q90872

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can convert a Windows icon file (.ICO) to a Windows bitmap (.BMP) file by storing the icon in a picture box, and then using the SavePicture statement with the Image property of the picture control.

MORE INFORMATION

You may wish to convert an icon format file to the bitmap format to perform operations that cannot be performed on icon format files, such as loading the image into Microsoft Windows Paintbrush.

To convert an icon format file to a bitmap format file, assign the icon to the Picture property of a picture box property (at design-time or run-time). At run-time, use the following statement:

```
SavePicture Picture1.Image, "filename.bmp"
```

When you convert an icon to a bitmap, you lose device independence for resolution characteristics. Windows bitmap format files, which usually have a .BMP extension, represent an image with device independent color information. Windows icon files, which usually have an .ICO extension, can contain information for both color and resolution device independence.

The steps listed below demonstrate how to convent an icon format file to a bitmap format file:

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 will be created by default.

2. Place a picture box named Picture1 on Form1.

3. Enter the following code into the form's Click event:

```
Sub Form_Click ()
   Picture1.AutoSize = -1
   Picture1.Picture = LoadPicture("icons\arrows\arw01dn.ico")
   SavePicture Picture1.Image, "arw01dn.bmp"
End Sub
```

4. Press the F5 key to run the program. Click Form1 to convert the file.

Additional Reference(s):

Chapter 19 File Formats of "Microsoft Windows Programmer's Reference"

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap

Visual Basic 3.0 Programming Questions & Answers

Article ID: Q92550

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, version 3.0

---

1. Q. I use the picture control to group other controls. However when I select the picture control, the other controls do not remain on top of the picture control. How can I correct this problem?

   A. This problem occurs if you place the controls on the form in the same place as the picture control but not in the picture control. itself. To group the controls in a picture control, you must first select the Picture control and then draw the desired control within the Picture control. For more information, please see Chapter 3 of the "Microsoft Visual Basic version 3.0 Programmer's Guide."

2. Q. How can I make calls from Visual Basic to the functions in the Windows Application Programming Interface (API) or other dynamic link libraries (DLLs)?

   A. To call a subroutine or function from one of the Windows APIs or any other DLL, you need to first provide a Declare statement for that subroutine or function in your Visual Basic application. The exact syntax for the declaration for each Windows API function can be found in the WIN31API.HLP help file included with the Professional Edition of Visual Basic. For more information, please see Chapter 24 of the "Microsoft Visual Basic version 3.0 Programmer's Guide."

3. Q. Is there a reference available that lists the correct Visual Basic declarations for the Multimedia API functions?

   A. Yes, the file is called WINMMSYS.TXT. It comes with the Professional edition of Visual Basic. You can find it in the \VB\WINAPI directory.

4. Q. Is there a reference available that lists the correct Visual Basic declarations for the Windows for Workgroups API functions?

   A. No, at this time such a file is not available from Microsoft. However, you can obtain a copy of the Windows for Workgroups SDK from the WINEXT forum on CompuServe.

5. Q. I followed the examples in the manuals and in the help file on how to use Domain functions such as DSum and DCount, but I keep receiving this error:

      Reference to undefined function or array.

      Why?

   A. The examples provided for the Domain Aggregate functions are

incorrect. These functions must be used within an SQL Statement just as SQL Aggregate functions such as Sum and Count are used. Please look at the SQL Aggregate examples to see how to use these functions within an SQL Statement. For more information, query on the following words in the Microsoft Knowledge Base:

DOMAIN and FUNCTION and SQL

6. Q. I want to sort the records referenced by the Data Control in my application. I tried to use the Index Property as described in the example in the manual and in the help file, but I receive the following error message:

   Property 'Index' not found

   Why?

   A. The examples provided in the Index Property are incorrect. The Index property does not apply to the Data Control. To sort the records referenced by the Data Control, use the ORDER BY Clause within an SQL Statement in the RecordSource property of the Data Control.

7. Q. Is there a better way than the Print Form method to print Forms and Controls in a program?

   A. Yes, it is possible to print forms and/or controls and specify the printed size by using various Windows API function calls. This process is documented in Microsoft Knowledge Base article Q85978. You can also find this article in the top 10 Microsoft Knowledge Base articles that are in the Visual Basic help file. To view these articles, select "Technical Support" from the Contents screen in the Visual Basic help file. Then select "Knowledge Base Articles on Visual Basic."

Additional reference words: 3.00 ivrfax fasttips
KBCategory:
KBSubcategory: PrgCtrlsStd APrgOther TlsCDK

How to Get Windows 3.1 Version Number in VB with GetVersion

Article ID: Q92936

This information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

From a Visual Basic program, you can determine the Windows version by calling the Windows 3.1 API function GetVersion from the Windows kernel module. The GetVersion function can help your program accommodate differences in the way API calls operate between different versions of Windows (such as differences between API parameters or return values).

MORE INFORMATION

The code example below shows how to make the GetVersion function call, which takes no parameters. The return value is a DWORD (double-word) value, which translates into a long integer (32-bit value) in Visual Basic.

The GetVersion function changed in Windows 3.1 from a WORD value to a DWORD (double-word) value. The low-order word returns the major (low byte) and minor (high byte) version numbers of Windows, and the high-order word returns the major (high byte) and minor (low byte) versions of MS-DOS, if the function is successful.

For details on the GetVersion function, see pp. 469-470 in the "Microsoft Windows Software Development Kit Programmer's Reference Vol. 2: Functions."

Step-by-Step Example

1. Create a new form and add two text boxes (Text1 and Text2) and a command button (Command1).

2. Add the following declaration to the General Declarations section:

   ```
   Declare Function GetVersion Lib "kernel" () As Long
   ```

3. Add following code to the command button's Click event:

   ```
   Sub Command1_Click ()
      I& = GetVersion()

      Windows& = I& And &HFFFF&
      Dos& = (I& And &HFFFF0000) / 65536

      ' The low byte is derived by masking off high byte.
      Lowbyte$ = Str$(Dos& And &HFF)
      ' The high byte is derived by masking off low byte and shifting.
   ```

```
      Highbyte$ = LTrim$(Str$((Dos& And &HFF00) / 256))
      ' Assign MS-DOS version to Text property.
      Text1.Text = Highbyte$ + "." + Lowbyte$

      Lowbyte$ = Str$(Windows& And &HFF)
      ' The high byte is derived by masking off low byte and shifting.
      Highbyte$ = LTrim$(Str$((Windows& And &HFF00) / 256))
      ' Assign Windows version to Text property.
      Text2.Text = Lowbyte$ + "." + Highbyte$
   End Sub
```

REFERENCES
==========

"Microsoft Windows Software Development Kit Programmer's Reference Vol. 2: Functions", pp. 469-470.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Establish a Network DDE Link Using Visual Basic

Article ID: Q93160

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article demonstrates how to establish a network Dynamic Data Exchange (DDE) link between two computers running Microsoft Windows for Workgroups.

MORE INFORMATION

Under DDE, a destination (or client) application sends commands through DDE to the source (or server) application to establish a link. Through DDE, the source provides data to the destination at the request of the destination or accepts information at the request of the destination. When you use DDE with Windows version 3.0 or 3.1 based applications, the source and destination applications are both located on the same computer.

When you use Network DDE with Windows for Workgroups based applications, DDE functions exactly the same way as standard DDE except that the source and destination applications are located on different computers.

Before establishing a network DDE link, you must first establish a network DDE share for the conversation by calling the API NDdeShareAdd() function located in the NDDEAPI.DLL file. Here is the Visual Basic declaration:

```
' Enter the following as one, single line:
Declare Function NDdeShareAdd Lib "NDDEAPI.DLL" (Server As Any, ByVal Level
   As Integer, ShareInfo As NDDESHAREINFO, ByVal nSize As Long) As Integer
```

Enter the entire statement as a single line. The first parameter is always a 0 and is passed with ByVal 0& from Visual Basic. The second parameter is always 2. The next parameter is a filled ShareInfo structure (given below). The last parameter is the size of the ShareInfo structure.

Here is The structure of the NDDESHAREINFO structure:

```
   Type NDDESHAREINFO
      szShareName As String * MAX_NDDESHARENAME_PLUSONE
      lpszTargetApp As Long    'LPSTR lpszTargetApp
      lpszTargetTopic As Long  'LPSTR lpszTargetTopic
      lpbPassword1 As Long     'LPBYTE lpbPassword1
      cbPassword1 As Long      'DWORD  cbPassword1;

      dwPermissions1 As Long   'DWORD  dwPermissions1;
      lpbPassword2 As Long     'LPBYTE lpbPassword2;
      cbPassword2 As Long      'DWORD  cbPassword2;
```

```
      dwPermissions2  As Long  'DWORD  dwPermissions2;
      lpszItem As Long         'LPSTR  lpszItem;
      cAddItems As Long        'LONG   cAddItems;
      lpNDdeShareItemInfo As Long
   End Type
```

The following table describes each field of the NDDESHAREINFO type:

Field Name	Purpose
szShareName	Name of the share to add.
lpszTargetApp	Pointer to null-terminated string containing the service or application name.
lpszTargetTopic	Pointer to null-terminated string holding the topic name
lpbPassword1	Pointer to the read-only password -- uppercase, null-terminated string. If null, pass null string, not zero.
cbPassword1	Length of read-only password
dwPermissions1	Full access password
cbPassword2	Length of the full access password
dwPermissions2	Permissions allowed by the full access password

Here are the permissions allowed for dwPermissions:

Name	Value	Function
NDDEACCESS_REQUEST	&H1	Allows LinkRequest
NDDEACCESS_ADVISE	&H2	Allows LinkAdvise
NDDEACCESS_POKE	&H4	Allows LinkPoke
NDDEACCESS_EXECUTE	&H8	Allows LinkExecute
NDDEACCESS_START_APP	&H10	Starts source application on connect

Here are the possible return values from NDdeShareAdd():

Name	Value	Meaning
NDDE_NO_ERROR	0	No error.
NDDE_BUF_TOO_SMALL	2	Buffer is too small to hold information.
NDDE_INVALID_APPNAME	13	Application name is not valid.
NDDE_INVALID_ITEMNAME	9	Item name is not valid.
NDDE_INVALID_LEVEL	7	Invalid level; nLevel parameter must be 2.
NDDE_INVALID_PASSWORD	8	Password is not valid.
NDDE_INVALID_SERVER	4	Computer name is not valid; lpszServer parameter must be NULL.
NDDE_INVALID_SHARE	5	Share name is not valid.
NDDE_INVALID_TOPIC	10	Topic name is not valid.
NDDE_OUT_OF_MEMORY	12	Not enough memory to complete request.
NDDE_SHARE_ALREADY_EXISTS	15	Existing shares cannot be replaced.

There are two steps to establish a network Dynamic Data Exchange (DDE) link between two computers running Microsoft Windows for Workgroups. First, create the DDE source application. Second, create the DDE destination application.

Step One -- Create DDE source application

The following steps show you how to create a Visual Basic DDE source and destination application that communicates through a network DDE link.

1. From the DDE source computer, start Visual Basic or if Visual Basic is already running, from the File menu, choose New Project (ALT, F, N). Form1 is created by default.

2. Change the LinkTopic property of Form1 to VBTopic.

3. If you are running Visual Basic version 2.0 or 3.0 for Windows, change the LinkMode property of Form1 to 1 - Source. In Visual Basic version 1.0, this property is already set to 1 - Server; don't change it.

4. Add a text box (Text1) to Form1.

5. Change the Name property (CTlName in version 1.0) of Text1 to VBItem.

6. Add a timer (Timer1) to Form1.

7. From the File menu, choose New Module (ALT, F, M). Module1 is created.

8. Add the following code to the general declarations section of Module1, and enter all lines as a single line even though they may be shown on multiple lines for readability:

```
' DDE access options
Global Const NDDEACCESS_REQUEST = &H1
Global Const NDDEACCESS_ADVISE = &H2
Global Const NDDEACCESS_POKE = &H4
Global Const NDDEACCESS_EXECUTE = &H8
Global Const NDDEACCESS_START_APP = &H10
Global Const MAX_NDDESHARENAME_PLUSONE = 65
Type NDDESHAREINFO
   szShareName As String * MAX_NDDESHARENAME_PLUSONE
   lpszTargetApp As Long    'LPSTR lpszTargetApp
   lpszTargetTopic As Long  'LPSTR lpszTargetTopic
   lpbPassword1 As Long     'LPBYTE lpbPassword1
   cbPassword1 As Long      'DWORD  cbPassword1;
   dwPermissions1 As Long   'DWORD  dwPermissions1;
   lpbPassword2 As Long     'LPBYTE lpbPassword2;
   cbPassword2 As Long      'DWORD  cbPassword2;
   dwPermissions2  As Long  'DWORD  dwPermissions2;
   lpszItem As Long         'LPSTR  lpszItem;
   cAddItems As Long        'LONG   cAddItems;
   lpNDdeShareItemInfo As Long
End Type
Declare Function NDdeShareAdd Lib "NDDEAPI.DLL" (Server As Any, ByVal
   Level As Integer, ShareInfo As NDDESHAREINFO,
   ByVal Size As Long) As Integer
Declare Function lstrcpy Lib "KERNEL" (szDest As Any, szSource As Any)
   As Long
'If using Visual Basic version 1.0, add the following declarations
'Global Const False = 0
'Global Const True = Not False
```

9. Add the following code to the Form_Load event of Form1:

```
Sub Form_Load ()
   Dim r As Integer
   Dim szShareName As String       ' Net DDE share name
   Dim szTargetName As String      ' Net DDE target name
   Dim szTopicName As String       ' Net DDE source topic name
   Dim szItemName As String
   Dim szReadOnlyPassword As String ' Read-only pw Net DDE share
   Dim szFullAccessPassword As String ' Full access password
   Dim ShareInfo As NDDESHAREINFO

   Dim ShareInfoSize As Long
   Dim Result As Integer
   szShareName = "VBDDESource$" + Chr$(0)
   szTargetName = "VBTARGET" + Chr$(0)
   szTopicName = "VBTopic" + Chr$(0)
   szItemName = Chr$(0)                    'All items are allowed
   szReadOnlyPassword = Chr$(0)            'No password
   szFullAccessPassword = Chr$(0)
   'Provide the share, target, topic, and item names along with
   'passwords that identify the network DDE share
   ShareInfo.szShareName = szShareName
   ShareInfo.lpszTargetApp = lstrcpy(ByVal szTargetName,
      ByVal szTargetName)
   ShareInfo.lpszTargetTopic = lstrcpy(ByVal szTopicName,
      ByVal szTopicName)
   ShareInfo.lpszItem = lstrcpy(ByVal szItemName, ByVal szItemName)

   ShareInfo.cbPassword1 = 0
   ShareInfo.lpbPassword1 = lstrcpy(ByVal szReadOnlyPassword,
      ByVal szReadOnlyPassword)
   ShareInfo.dwPermissions1 = NDDEACCESS_REQUEST Or NDDEACCESS_ADVISE Or
      NDDEACCESS_POKE Or NDDEACCESS_EXECUTE Or NDDEACCESS_START_APP
   ShareInfo.cbPassword2 = 0
   ShareInfo.lpbPassword2 = lstrcpy(ByVal szFullAccessPassword,
      ByVal szFullAccessPassword)
   ShareInfo.dwPermissions2 = NDDEACCESS_REQUEST Or NDDEACCESS_ADVISE Or
      NDDEACCESS_POKE Or NDDEACCESS_EXECUTE Or NDDEACCESS_START_APP
   ShareInfo.lpNDdeShareItemInfo = 15
   Result = NDdeShareAdd(ByVal 0&, 2, ShareInfo, Len(ShareInfo))
   ' Start the timer that will continually update the text box and
   ' the DDE link item with random data.
   timer1.Interval = 1000
   timer1.Enabled = True

End Sub
```

10. Add the following code to the Timer1_Timer event procedure:

```
Sub Timer1_Timer ()
   ' Display random value 0 - 99 in the text box (DDE source data).
   Randomize Timer
   VBItem.Text = Format$(Rnd * 100, "0")
End Sub
```

11. From the File menu, choose Make EXE File...

12. Name the file VBTARGET.EXE and choose OK to create the .EXE file.

13. From the File Manager or Program Manager, run VBTARGET.EXE to display a random value in the text box every second.

Step Two -- Create the DDE destination application
---

14. From the DDE destination computer, start Visual Basic or if Visual Basic is already running, from the File menu, choose New Project (ALT, F, N). Form1 is created by default.

15. Add a text box (Text1) to Form1.

16. Add the following code to the Form_Load event of Form1:

```
Sub Form_Load ()
   Dim r As Long
   Dim szComputer As String     ' Network server name.
   Dim szTopic As String
   ' Identify the network server where the DDE source application
   ' is running. The following statement assumes the source computer
   ' name is COMPUTER1. Change it to your source computer name.
   szComputer = "\\COMPUTER1"
   ' Identify the DDE share established by the source application
   szTopic = "VBDDESource$"
   Text1.LinkMode = 0
   ' The link topic identifies the computer name and link topic
   ' as established by the DDE source application
   Text1.LinkTopic = szComputer + "\" + "NDDE$" + "|" + szTopic
   Text1.LinkItem = "VBItem" ' Name of text box in DDE source app

   Text1.LinkMode = 1        ' Automatic link.
End Sub

'For this program to work, set the szComputer variable (above) to the
'computer name that holds the DDE source application. Find the name
'in the Network section of Windows for Workgroups Control Panel.
```

17. From the Run menu, choose Start to run the program.

You should see the same random values generated on the source computer displayed in the text box of the destination computer. If you receive the error message "DDE method invoked with no channel open" on the Text1.LinkMode = 1 statement in Step 16, make sure the szComputer variable is set correctly.

Additional reference words: 1.00 2.00 3.00 NETDDE
KBCategory:
KBSubcategory: APrgNet IAPDDE

Form Cannot Be Larger Than the Screen

Article ID: Q94665

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- The Standard and Professional Editions of Microsoft Visual Basic for MS-DOS, version 1.0

SUMMARY
=======

The maximum size of a form in Microsoft Visual Basic for Windows and Microsoft Visual Basic for MS-DOS is limited to the size of the screen you are using.

In both Microsoft Visual Basic for MS-DOS and for Windows, the following code sizes a form to maximum size:

```
TOP = 0
LEFT = 0
WIDTH = SCREEN.WIDTH
HEIGHT = SCREEN.HEIGHT
```


Additional reference words: 1.00 2.00 3.00 3.10
KBCategory:
KBSubcategory: APrgWindow

How to Connect to a Network Drive by Using WNetAddConnection

Article ID: Q94679

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

Windows version 3.1 provides a new API Call, WNetAddConnection, which will redirect a local device to a shared resource or network server.

WNetAddConnection requires the name of the local device, the name of the network resource, and the password necessary to use that resource.

This article explains in detail the arguments and potential error messages for the Windows version 3.1 WNetAddConnection function call.

MORE INFORMATION

To use WNetAddConnection within a Visual Basic application, declare the WNetAddConnection function in the General Declarations Section of your code window. (In Visual Basic version 1.0 you can also put the declaration in the Global Module.) Declare the function as follows entering the entire Declare statement on one, single line:

```
   Declare Function WnetAddConnection% Lib "user" (ByVal lpszNetPath As
Any,
      ByVal lpszPassword As Any,
      ByVal lpszLocalName As Any)
```

Here are definitions for the formal parameters:

Formal Parameter	Definition
lpszNetPath	Points to a null-terminated string specifying the shared device or remote server.
lpszPassword	Points to a null-terminated string specifying the network password for the given device or server.
lpszLocalName	Points to a null-terminated string specifying the local drive or device to be redirected. All lpszLocalName strings (such as LPT1) are case independent. Only the drive names A through Z and device names LPT1 through LPT3 are used.

Below are the possible return values as defined on page 990 of the Microsoft Windows version 3.1 Programmer's Reference:

Value	(Hex Value)	Meaning

```
   ---------------------------------------------------------------------
   WN_SUCCESS              (&H0)        Function was successful.
   WN_NOT_SUPPORTED        (&H1)        Function was not supported.
   WN_OUT_OF_MEMORY        (&HB)        System was out of memory.
   WN_NET_ERROR            (&H2)        An error occurred on the network.
   WN_BAD_POINTER          (&H4)        Pointer was invalid.
   WN_BAD_NETNAME          (&H32)       Network resource name was invalid.
   WN_BAD_LOCALNAME        (&H33)       Local device name was invalid.
   WN_BAD_PASSWORD         (&H6)        Password was invalid.
   WN_ACCESS_DENIED        (&H7)        A security violation occurred.
   WN_ALREADY_CONNECTED    (&H34)       Local device was already connected
                                        to a remote resource.
```

Below is an example of how to redirect a local device to a network resource:

1. Start Visual Basic (VB.EXE). Form1 is created by default.

2. Create the following controls with the indicated properties on Form1:

```
   Default Name   Caption             CtlName
   ------------------------------------------
   Text1          (Not applicable)    NetPath
   Text2          (Not applicable)    Password
   Command1       &Connect            Connect
   Drive1         (Not applicable)    Drive1
```

3. Add the following code to the general declaration section of Form1. Enter the Declare statement as one, single line:

```
   Declare Function WnetAddConnection% Lib "user"
     (ByVal lpszNetPath as Any, ByVal lpszPassword as Any,
     ByVal lpszLocalName as Any)
   Const WN_Success = &H0
   Const WN_Not_Supported = &H1
   Const WN_Net_Error = &H2
   Const WN_Bad_Pointer = &H4
   Const WN_Bad_NetName = &H32
   Const WN_Bad_Password = &H6
   Const WN_Bad_Localname = &H33
   Const WN_Access_Denied = &H7
   Const WN_Out_Of_Memory = &HB
   Const WN_Already_Connected = &H34
```

   If you're using Visual Basic version 1.0, add the following to the general declarations also:

```
   Const True = -1
   Const False = 0
```

4. Add the following code to the procedure Connect_Click:

```
   Sub Connect_Click ()

      ServerText$ = UCase$(NetPath.Text) + Chr$(0) ' Network resource name
```

```
      PasswordText$ = Password.Text + Chr$(0) ' Password for the resource
      driveletter$ = "N:" + Chr$(0) ' Substitute your own drive letter

      Succeed% = WnetAddConnection(ServerText$, PasswordText$,
driveletter$)

      If IsSuccess(Succeed%, msg$) = True Then ' Call Function to parse
                                               ' potential error messages.
         Drive1.Refresh
         NetPath.Text = "" ' Reset the contents following connection
      Else
         MsgBox msg$
      End If

   End Sub
```

5. Create a Sub within the (Declarations) section of the Code window and add the following code:

```
   Function IsSuccess% (ReturnCode%, Msg$)

   If ReturnCode% = WN_Success Then
      IsSuccess% = True
   Else
      IsSuccess% = False
      Select Case ReturnCode%

         Case WN_Success:
            Drive1.Refresh
         Case WN_Not_Supported:
            msg$ = "Function is not supported."
         Case Wn_Out_Of_Memory:
            msg$ = "Out of Memory."
         Case WN_Net_Error:
            msg$ = "An error occurred on the network."
         Case WN_Bad_Pointer:
            msg$ = "The Pointer was Invalid."
         Case WN_Bad_NetName:
            msg$ = "Invalid Network Resource Name."
         Case WN_Bad_Password:
            msg$ = "The Password was Invalid."
         Case WN_Bad_Localname:
            msg$ = "The local device name was invalid."
         Case WN_Access_Denied:
            msg$ = "A security violation occurred."
         Case WN_Already_Connected:
            msg$ = "The local device was connected to a remote resource."
         Case Else:
            msg$ = "Unrecognized Error " + Str$(ReturnCode%) + "."

      End Select
   End If

   End Function
```

6. Run the program. Type in the name of a network resource in the edit box and press the Connect button. The drive box will be updated with the

new resource if the call was successful.

Reference(s):

"Microsoft Windows Software Development Kit: Reference Volume 2," version 3.1 and the WIN31WH.HLP file that shipped with the Microsoft Visual Basic version 2.0 Professional Version for Windows.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: APrgNet

Using Lstrcpy() API Function to Get Far Address of a Variable

Article ID: Q94700

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

You can use the Windows API function Lstrcpy() to get the far address of a variable as a Long integer.

The Lstrcpy() function returns the same value as its first argument, which is the address of a variable. Usually you would use the Lstrcpy() function to copy strings that are terminated by a zero byte. However, if you pass the same variable as both the source and the destination, Lstrcpy() copies the variable to itself, which has no effect.

MORE INFORMATION

Basic cannot deal with pointers directly. All Basic can do with a pointer is pass it as a parameter to a DLL function.

Basic variables may move in memory. You should take the address of a variable immediately before you use it.

The following steps demonstrate how to get the address of an integer and a variable-length string.

1. Run Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Enter the following code into the general declarations section:

   ```
   Declare Function Lstrcpy Lib "kernel" (p1 As Any, p2 As Any) As Long
   ```

3. Enter the following code into the Click event handler:

   ```
   Sub Form_Click ()
      Dim ptr As Long     ' pointer value
      Dim x1 As Integer   ' variable to take address of
      Dim x2 As String    ' variable to take address of
      x1 = 123
      ptr = Lstrcpy(x1, x1)
      MsgBox "The address of x1 is: " + Hex$(ptr)
      x2 = "x2"
      ' must use ByVal on variable length strings
      ptr = Lstrcpy(ByVal x2, ByVal x2)
      MsgBox "The address of x2 is: " + Hex$(ptr)
   End Sub
   ```

4. Press the F5 key to run the program. It displays the address of the variable in hexadecimal.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Pass Numeric Variables to a C DLL

Article ID: Q94960

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft QuickC for Windows, version 1.0

SUMMARY

This article shows by example how to pass numeric variables from Visual Basic for Windows to a C DLL. The first example shows how to call C functions with single parameters of all numeric types. The second example shows how to pass multiple parameters and how to pass variables by reference so they can be manipulated on the C side.

MORE INFORMATION

Example One

1. Start Visual Basic or if you are in Visual Basic, choose New Project from the File menu (ALT, F, N). Form1 is created by default.

2. Place five command buttons (Command1, Command2, Command3, Command4, and Command5) on Form1.

3. Add two Text boxes (Text1 and Text2) to Form1.

4. Add the following code to the General Declarations section of Form1:

```
Declare Function noparams Lib "passnums.dll" () As Integer
Declare Function passint Lib "passnums.dll" (ByVal x%) As Integer
Declare Function passlong Lib "passnums.dll" (ByVal x&) As Long
Declare Function passfloat Lib "passnums.dll" (ByVal x!) As Single
Declare Function passdouble Lib "passnums.dll" (ByVal x#) As Double
```

5. Add the following code to the click event of each command buttons:

```
Sub Command1_Click ()
   text1.Text = Str$(noparams())
   text2.Text = "Noparams"
End Sub

Sub Command2_Click ()
   i% = 21
   text1.Text = Str$(passint(55))
   text2.Text = Str$(passint(i%))
End Sub

Sub Command3_Click ()
```

```
   i& = 45000
   text1.Text = Str$(passlong(40000))
   text2.Text = Str$(passlong(i&))
End Sub

Sub Command4_Click ()
   i! = 1.35
   text1.Text = Str$(passfloat(1.23))
   text2.Text = Str$(passfloat(i!))
End Sub

Sub Command5_Click ()
   i# = 1234.5678
   text1.Text = Str$(passdouble(1.23456))
   text2.Text = Str$(passdouble(i#))
End Sub
```

6. Start Microsoft QuickC for Windows, or if it's already running, choose New from the File menu.

7. Add the following code to the new file:

```
#include <windows.h>
#include <stdio.h>
/*  Noparams takes no parameters and returns a 2 */
extern int far pascal noparams()
{
return(2);
}
/* add 32 to the integer passed in */
extern int far pascal passint(int a)
{
a += 32;
return(a);
}
/* passlong() takes a long integer and adds 7 to it */
extern long far pascal passlong(long x)
{
x += 7;
return(x+7);
}
// passfloat passes a floating point number
extern float far pascal passfloat(float x)
{
   return (x += (float) 1.45927);
}
// passdouble passes a floating point number
extern double far pascal passdouble(double x)
{
   return (x+=(double) 1.45927);
}
```

NOTE: Microsoft C and Borland C return values of type Double differently. Therefore the passdouble example above won't work in Borland C. Use the following code in Borland C:

```
// return a value through y
```

```
void FAR PASCAL _export passdouble(double x, double *y)
{
 // do processing here
 // use '*y =' instead of a return statement
    *y = x;
}
```

Borland C is manufactured by a vendor independent of Microsoft; Microsoft makes no warranty, implied or otherwise, regarding Borland C's performance or reliability.

8. From the File menu, choose Save As, and save the file as PASSNUMS.C.

9. From the File menu, choose New, and Type these .DEF file lines:

```
LIBRARY     PASSNUMS
EXETYPE     WINDOWS 3.1
DATA        PRELOAD MOVABLE SINGLE
CODE        PRELOAD MOVABLE DISCARDABLE
EXPORTS
            noparams   @1
            passint    @2
            passlong   @3
            passfloat  @4
            passdouble @5
```

10. From the File menu, choose Save As, and save the file as PASSNUMS.DEF.

11. From the Project Menu, choose Open and enter PASSNUMS.

12. Choose the OK button. Add PASSNUMS.C and PASSNUMS.DEF to the project.

13. From the Options menu, choose Project. Set the program type to Windows DLL and set the compiler memory model to Large.

14. From the Project menu, choose Rebuild All. This creates PASSNUMS.DLL.

15. Return to Visual Basic and run the program. Pressing any of the command buttons will change the contents of the two text boxes.

Example Two
---

1. Start Visual Basic, or if Visual Basic is already running, from the File menu, choose New Project (ALT, F, N). Form1 is created by default.

2. Place two command buttons (Command1, Command2) on Form1.

3. Add 2 Text boxes (Text1, Text2) to Form1.

4. Add the following code to the General Declarations section of Form1:

```
' Enter each of the following Declare statements on one, single line:
Declare Function bunchparam Lib "multvars.dll" (ByVal w%,
   ByVal x&, ByVal y!, ByVal z#) As Double
Declare Function bunchbyref Lib "multvars.dll"
   (x%, y&, z!, a#) As Double
```

5. Add the following code to the click events of the Command buttons:

```
Sub Command1_Click ()
   i% = 123
   j& = 40000
   k! = 1.234
   l# = 1234.567
   text1.Text = Str$(bunchparam(123, 40000, 1.2345, 1.2345))
   text2.Text = Str$(bunchparam(i%, j&, k!, l#))
End Sub

Sub Command2_Click ()
   i% = 12
   j& = 40000
   k! = 123.455
   l# = 123455.678
   x# = bunchbyref(i%, j&, k!, l#)
   text1.Text = Str$(i%) + Str$(j&) + Str$(k!) + Str$(l#)
   text2.Text = Str$(x#)
End Sub
```

6. Start Microsoft QuickC for Windows or choose New from the File menu.

7. Add the following code to the new file:

```
#include <windows.h>
#include <stdio.h>
/* bunchparam() adds double-precision values and an integer. */
extern double far pascal bunchparam(int a, long b, float c, double d)
{
return(a+b+c+d);
}
extern double far pascal bunchbyref(int *a, long *b, float *c, double *d
)
{
*a += 55;
*b += 77;
*c += (float) 123.456;
*d += 12345.678;
return(*a+*b);
}
```

8. From the File menu, choose Save As, and save the file as MULTVARS.C.

9. From the File menu, choose New, and type these .DEF file lines:

```
LIBRARY     MULTVARS
EXETYPE     WINDOWS 3.1
DATA        PRELOAD MOVABLE SINGLE
CODE        PRELOAD MOVABLE DISCARDABLE
EXPORTS
            bunchparam @1
            bunchbyref @2
```

10. From the File menu, choose Save As, and save the file as MULTVARS.DEF.

11. From the Project Menu, choose Open and enter MULTVARS.

12. Choose the OK button. Add MULTVARS.C and MULTVARS.DEF to the project.

13. From the Options menu, choose Project. Set the program type to Windows DLL and set the compiler memory model to Large.

14. From the Project menu, choose Rebuild All. This creates MULTVARS.DLL.

15. Return to Visual Basic and run the program. Pressing either Command button will change the contents of the text boxes.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Create a Transparent Bitmap Using Visual Basic

Article ID: Q94961

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 2.0 and 3.0

SUMMARY

A transparent image shows the background behind it instead of the image itself. You can use an icon editor such as the IconWorks sample program provided with Visual Basic to create icons that contain transparent parts. This article shows you how to make certain parts of a bitmap transparent.

MORE INFORMATION

Here are the six general steps required to create a transparent bitmap:

1. Store the area, or background, where the bitmap is going to be drawn.

2. Create a monochrome mask of the bitmap that identifies the transparent areas of the bitmap by using a white pixel to indicate transparent areas and a black pixel to indicate non-transparent areas of the bitmap.

3. Combine the pixels of the monochrome mask with the background bitmap using the And binary operator. The area of the background where the non-transparent portion of the bitmap will appear is made black.

4. Combine an inverted copy of the monochrome mask (step 2) with the source bitmap using the And binary operator. The transparent areas of the source bitmap will be made black.

5. Combine the modified background (step 3) with the modified source bitmap (step 4) using the Xor binary operator. The background will show through the transparent portions of the bitmap.

6. Copy the resulting bitmap to the background

Example Code

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add the following controls to Form1 with the associated property values:

Control	Name (or CtlName)	Property Settings
Picture	pictSource	Picture ="WINDOWS\THATCH.BMP"
Picture	pictDest	Picture ="WINDOWS\ARCHES.BMP"
Command button	cmdCopy	Caption ="Copy"

3. From the File menu, choose New Module (ALT, F, M). Module1 is created.

4. Add the following code to the cmdCopy_Click event procedure of Form1. This code calls the TransparentBlt() function to copy a source bitmap to a destination (background) picture control. White (QBColor(15)) areas of the bitmap are made transparent against the background bitmap.

```
Sub cmdCopy_Click ()
  Call TransparentBlt(pictDest, pictSource.Picture, 10, 10, QBColor(15))
End Sub
```

5. Add the following code the general declarations section of Module1. Enter each Declare statement as one, single line:

```
Type bitmap
   bmType As Integer
   bmWidth As Integer
   bmHeight As Integer
   bmWidthBytes As Integer
   bmPlanes As String * 1
   bmBitsPixel As String * 1
   bmBits As Long
End Type
Declare Function BitBlt Lib "GDI" (ByVal srchDC As Integer, ByVal srcX
   As Integer, ByVal srcY As Integer, ByVal srcW As Integer, ByVal srcH
   As Integer, ByVal desthDC As Integer, ByVal destX As Integer, ByVal
   destY As Integer, ByVal op As Long) As Integer
Declare Function SetBkColor Lib "GDI" (ByVal hDC As Integer, ByVal
   cColor As Long) As Long
Declare Function CreateCompatibleDC Lib "GDI" (ByVal hDC As Integer)
   As Integer
Declare Function DeleteDC Lib "GDI" (ByVal hDC As Integer) As Integer
Declare Function CreateBitmap Lib "GDI" (ByVal nWidth As Integer, ByVal
   nHeight As Integer, ByVal cbPlanes As Integer, ByVal cbBits As
   Integer, lpvBits As Any) As Integer
Declare Function CreateCompatibleBitmap Lib "GDI" (ByVal hDC As Integer,
   ByVal nWidth As Integer, ByVal nHeight As Integer) As Integer
Declare Function SelectObject Lib "GDI" (ByVal hDC As Integer, ByVal
   hObject As Integer) As Integer
Declare Function DeleteObject Lib "GDI" (ByVal hObject As Integer) As
   Integer
Declare Function GetObj Lib "GDI" Alias "GetObject" (ByVal hObject As
   Integer, ByVal nCount As Integer, bmp As Any) As Integer
Const SRCCOPY = &HCC0020
Const SRCAND = &H8800C6
Const SRCPAINT = &HEE0086
Const NOTSRCCOPY = &H330008
```

6. Add the following Sub procedure to the general declarations section of Module1. TransparentBlt() accepts six parameters: a destination picture control (dest), a source bitmap to become transparent (srcBmp), the X,Y coordinates in pixels where you want to place the source bitmap on the destination (destX and destY), and the RGB value for the color you want to be transparent. TransparentBlt() copies the source bitmap to any X,Y location on the background making areas transparent.

```
Sub TransparentBlt (dest As Control, ByVal srcBmp As Integer, ByVal
```

```
destX As Integer, ByVal destY As Integer, ByVal TransColor As Long)
Const PIXEL = 3
Dim destScale As Integer
Dim srcDC As Integer  'source bitmap (color)
Dim saveDC As Integer 'backup copy of source bitmap
Dim maskDC As Integer 'mask bitmap (monochrome)
Dim invDC As Integer  'inverse of mask bitmap (monochrome)
Dim resultDC As Integer 'combination of source bitmap & background
Dim bmp As bitmap 'description of the source bitmap
Dim hResultBmp As Integer 'Bitmap combination of source & background
Dim hSaveBmp As Integer 'Bitmap stores backup copy of source bitmap
Dim hMaskBmp As Integer 'Bitmap stores mask (monochrome)
Dim hInvBmp As Integer  'Bitmap holds inverse of mask (monochrome)
Dim hPrevBmp As Integer 'Bitmap holds previous bitmap selected in DC
Dim hSrcPrevBmp As Integer  'Holds previous bitmap in source DC
Dim hSavePrevBmp As Integer 'Holds previous bitmap in saved DC
Dim hDestPrevBmp As Integer 'Holds previous bitmap in destination DC
Dim hMaskPrevBmp As Integer 'Holds previous bitmap in the mask DC
Dim hInvPrevBmp As Integer 'Holds previous bitmap in inverted mask DC
Dim OrigColor As Long 'Holds original background color from source DC
Dim Success As Integer 'Stores result of call to Windows API
If TypeOf dest Is PictureBox Then 'Ensure objects are picture boxes
  destScale = dest.ScaleMode 'Store ScaleMode to restore later
  dest.ScaleMode = PIXEL 'Set ScaleMode to pixels for Windows GDI
  'Retrieve bitmap to get width (bmp.bmWidth) & height (bmp.bmHeight)
  Success = GetObj(srcBmp, Len(bmp), bmp)
  srcDC = CreateCompatibleDC(dest.hDC)    'Create DC to hold stage
  saveDC = CreateCompatibleDC(dest.hDC)   'Create DC to hold stage
  maskDC = CreateCompatibleDC(dest.hDC)   'Create DC to hold stage
  invDC = CreateCompatibleDC(dest.hDC)    'Create DC to hold stage
  resultDC = CreateCompatibleDC(dest.hDC) 'Create DC to hold stage
  'Create monochrome bitmaps for the mask-related bitmaps:
  hMaskBmp = CreateBitmap(bmp.bmWidth, bmp.bmHeight, 1, 1, ByVal 0&)
  hInvBmp = CreateBitmap(bmp.bmWidth, bmp.bmHeight, 1, 1, ByVal 0&)
  'Create color bitmaps for final result & stored copy of source
  hResultBmp = CreateCompatibleBitmap(dest.hDC, bmp.bmWidth,
     bmp.bmHeight)
  hSaveBmp = CreateCompatibleBitmap(dest.hDC, bmp.bmWidth,
     bmp.bmHeight)
  hSrcPrevBmp = SelectObject(srcDC, srcBmp)     'Select bitmap in DC
  hSavePrevBmp = SelectObject(saveDC, hSaveBmp) 'Select bitmap in DC
  hMaskPrevBmp = SelectObject(maskDC, hMaskBmp) 'Select bitmap in DC
  hInvPrevBmp = SelectObject(invDC, hInvBmp)    'Select bitmap in DC
  hDestPrevBmp = SelectObject(resultDC, hResultBmp) 'Select bitmap
  Success = BitBlt(saveDC, 0, 0, bmp.bmWidth, bmp.bmHeight, srcDC,
     0, 0, SRCCOPY) 'Make backup of source bitmap to restore later
  'Create mask: set background color of source to transparent color.
  OrigColor = SetBkColor(srcDC, TransColor)
  Success = BitBlt(maskDC, 0, 0, bmp.bmWidth, bmp.bmHeight, srcDC,
     0, 0, SRCCOPY)
  TransColor = SetBkColor(srcDC, OrigColor)
  'Create inverse of mask to AND w/ source & combine w/ background.
  Success = BitBlt(invDC, 0, 0, bmp.bmWidth, bmp.bmHeight, maskDC,
    0, 0, NOTSRCCOPY)
  'Copy background bitmap to result & create final transparent bitmap
  Success = BitBlt(resultDC, 0, 0, bmp.bmWidth, bmp.bmHeight,
     dest.hDC, destX, destY, SRCCOPY)
```

```
        'AND mask bitmap w/ result DC to punch hole in the background by
        'painting black area for non-transparent portion of source bitmap.
        Success = BitBlt(resultDC, 0, 0, bmp.bmWidth, bmp.bmHeight,
           maskDC, 0, 0, SRCAND)
        'AND inverse mask w/ source bitmap to turn off bits associated
        'with transparent area of source bitmap by making it black.
        Success = BitBlt(srcDC, 0, 0, bmp.bmWidth, bmp.bmHeight, invDC,
           0, 0, SRCAND)
        'XOR result w/ source bitmap to make background show through.
        Success = BitBlt(resultDC, 0, 0, bmp.bmWidth, bmp.bmHeight,
           srcDC, 0, 0, SRCPAINT)
        Success = BitBlt(dest.hDC, destX, destY, bmp.bmWidth, bmp.bmHeight,
           resultDC, 0, 0, SRCCOPY) 'Display transparent bitmap on backgrnd
        Success = BitBlt(srcDC, 0, 0, bmp.bmWidth, bmp.bmHeight, saveDC,
           0, 0, SRCCOPY) 'Restore backup of bitmap.
        hPrevBmp = SelectObject(srcDC, hSrcPrevBmp) 'Select orig object
        hPrevBmp = SelectObject(saveDC, hSavePrevBmp) 'Select orig object
        hPrevBmp = SelectObject(resultDC, hDestPrevBmp) 'Select orig object
        hPrevBmp = SelectObject(maskDC, hMaskPrevBmp) 'Select orig object
        hPrevBmp = SelectObject(invDC, hInvPrevBmp) 'Select orig object
        Success = DeleteObject(hSaveBmp)   'Deallocate system resources.
        Success = DeleteObject(hMaskBmp)   'Deallocate system resources.
        Success = DeleteObject(hInvBmp)    'Deallocate system resources.
        Success = DeleteObject(hResultBmp) 'Deallocate system resources.
        Success = DeleteDC(srcDC)          'Deallocate system resources.
        Success = DeleteDC(saveDC)         'Deallocate system resources.
        Success = DeleteDC(invDC)          'Deallocate system resources.
        Success = DeleteDC(maskDC)         'Deallocate system resources.
        Success = DeleteDC(resultDC)       'Deallocate system resources.
        dest.ScaleMode = destScale 'Restore ScaleMode of destination.
      End If
   End Sub
```

7. From the Run menu, choose Start (ALT, R, S) to run the program.

8. Click the Copy button. The thatched pattern in the first picture is copied onto the second picture (an image of arches) making the arches show through areas of the previously white thatched pattern.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgGrap

How Windows Versions 3.0 and 3.1 Activate Apps Differently

Article ID: Q95463

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.00

SUMMARY

When activating a multiple-window application in Windows version 3.0, only the window that was activated comes to the top. In Windows version 3.1, all the windows relating to the application come to the top.

Microsoft has confirmed this to be a problem in Microsoft Windows version 3.0. This problem was corrected in Microsoft Windows version 3.1.

Because many Visual Basic applications are written as multiple-window applications, this problem is apparent when these applications are run with both Windows versions.

MORE INFORMATION

Steps to Reproduce Problem

1. Start Windows version 3.1.

2. Run Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

3. From the File menu, choose New Form (ALT, F, F) twice to create two forms,
   Form2 and Form3.

4. Add the following code to Form_Load of Form1:

```
Sub Form_Load
   Form1.Show
   Form2.Show
   Form3.Show
End Sub
```

5. From the File menu, chose Make EXE File (ALT, F, K) and choose the OK button to create an executable using the default name (PROJECT1.EXE).

6. From outside the Visual Basic environment, Run PROJECT1.EXE.

7. Run NOTEPAD.EXE.

8. Click Form1 in PROJECT1.EXE. All three forms for PROJECT1.EXE, which are currently being clipped by NOTEPAD.EXE, come to the top.

9. Close Windows version 3.1, saving all necessary data in open applications.

10. Start Windows version 3.0.

11. Repeat steps 6 through 8 to see that only Form1 comes to the top.

Additional reference words: 2.00 3.10 3.00
KBCategory:
KBSubcategory: APrgWindow

How to Use Windows 3.1 APIs to Play Videos in Visual Basic

Article ID: Q96090

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

You can play video .AVI files in Visual Basic by using Windows version 3.1 APIs.

MORE INFORMATION

Use the following procedure to position and size the window where you want to play the .AVI file and to play the .AVI file:

1. Run Visual Basic, or if Visual Basic is already running, choose New Project from the File menu (ALT, F, N). Form1 is created by default.

2. Add a command button control (Command1) to Form1.

3. Add the following code to the Command1_Click event of Form1:

```
DIM CmdStr as String
DIM ret as Integer

'*** This will open the AVIVideo and create a child window on the
'*** form where the video will display. Animation is the device_id.
CmdStr = ("open c:\rbtndog.avi type AVIVideo alias Animation parent "
   + LTrim$(Str$(form1.hWnd)) + " style " + LTrim$(Str$(WS_CHILD)))
Ret = mciSendString(CmdStr, 0&, 0, 0)

'*** Put the window at location 10 10 relative to the parent window
'*** with a size of 200 200
Ret = mciSendString("put Animation window at 10 10 200 200", 0&, 0, 0)

'*** The wait tells the MCI command to complete before returning
'*** control to the application.
Ret = mciSendString("play Animation wait", 0&, 0, 0)

'*** Close windows so they don't crash when you exit the application.
Ret = mciSendString("close Animation", 0&, 0, 0)
```

4. Choose New Module from the File menu (ALT, F, M). MODULE1.BAS is created by default. Add the following code to Module1. Enter the entire Declare on a single line:

```
Global Const WS_CHILD = &H40000000
Declare Function mciSendString Lib "mmsystem" (ByVal lpstrCommand$,
   ByVal lpstrReturnStr As Any, ByVal wReturnLen%, ByVal hCallBack%)
```

```
      As Long
```

5. From the Run menu, choose Start (ALT, R, S) or press the F5 key to run the program.

For more information on the sndSendString() function and command strings, see pages 3-26 and 7-23 to 7-93 in the "MultiMedia Programmer's Reference."

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

Using PASSTHROUGH Escape to Send Data Directly to Printer

Article ID: Q96795

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

By using the Windows API Escape() function, your application can pass data directly to the printer. If the printer driver supports the PASSTHROUGH printer escape, you can use the Escape() function and the PASSTHROUGH printer escape to send native printer language codes to the printer driver.

Printer escapes such as PASSTHROUGH allow applications to access certain facilities of output devices that are not directly available through the graphics device interface (GDI). The PASSTHROUGH printer escape allows the application to send data directly to the printer, bypassing the standard print-driver code.

MORE INFORMATION

A printer driver that supports the PASSTHROUGH printer escape does not add native printer language codes to the data stream sent to the printer, so you can send data directly to the printer. However, Microsoft recommends that applications not perform functions that consume printer memory, such as downloading a font or a macro.

The sample program listed below sends native PCL codes to the printer to change the page orientation and the paper bin. A Hewlett-Packard LaserJet is the assumed default printer.

An Important Note

Note that the Windows API Escape() function is provided in Windows versions 3.0 and 3.1 for backward compatibility with earlier versions of Microsoft Windows. Applications are supposed to use the GDI DeviceCapabilities() and ExtDeviceMode() functions instead of the Escape() function, but neither DeviceCapabilities() nor ExtDeviceMode() can be called directly from Visual Basic. This is because they are exported by the printer driver, not by the Windows GDI. The only way to use ExtDeviceMode() or DeviceCapabilities() in Visual Basic is to create a DLL and call them from there.

Steps to Create Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add the following code to the general declarations section of Form1:

```
' Enter the entire Declare statement on one, single line.
Declare Function Escape Lib "Gdi" (ByVal Hdc%, ByVal nEscape%,
   ByVal ncount%, ByVal indata$, ByVal oudata as Any) As Integer

Const PASSTHROUGH = 19

Const RevLandScape = "&l3O" ' PCL command to change Paper
                            ' orientation to Reverse Landscape.
Const Portrait = "&l0O"     ' PCL command to change paper
                            ' orientation to Portrait.
Const ManualFeed = "&l3H"   ' PCL command to change Paper Bin
                            ' to Manual Feed Envelope.
Const AutoFeed = "&l1H"     ' PCL command to change Paper Bin
                            ' to Paper Tray AutoFeed
```

3. Add a list box (List1) to Form1.

4. Add the following code to Form1's Form_Load event procedure:

```
Sub Form_Load ()
  List1.AddItem "HP/PCL Reverse Landscape"
  List1.AddItem "HP/PCL Portrait"
  List1.AddItem "HP/PCL Manual Feed Envelope"
  List1.AddItem "HP/PCL Paper Tray Auto Feed"
End Sub
```

5. Add the following code to the List1_Click event procedure:

```
Sub List1_Click
Select Case List1.ListIndex
   Case 0:
      PCL_Escape$ = Chr$(27) + RevLandScape
   Case 1:
      PCL_Escape$ = Chr$(27) + Portrait
   Case 2:
      PCL_Escape$ = Chr$(27) + ManualFeed
   Case 3:
      PCL_Escape$ = Chr$(27) + AutoFeed
End Select

' Enter the following two lines as one, single line:
PCL_Escape$ = Chr$(Len(PCL_Escape$) MOD 256)
   + Chr$(Len(PCL_Escape$) \ 256) + PCL_Escape$

Printer.Print ""
Result% = Escape%(Printer.hDC, PASSTHROUGH, 0, PCL_Escape$, 0&)

Select Case Result%
   ' Enter each Case statement on one, single line.
   Case Is < 0: MsgBox "The PASSTHROUGH Escape is not
      supported by this printer driver.", 48
   Case 0: MsgBox "An error occurred sending the escape
      sequence.", 48
   Case Is > 0: MsgBox "Escape Successfully sent.
      Sending test printout to printer."
```

```
      Printer.Print "Test case of "; List1.Text
   End Select
   End Sub
```

6. From the Run menu, choose Start (ALT, R, S) to run the program. List1 is filled with four escape sequences to send to the printer.

7. Select any of the options in the list box. A message box appears to indicate the success of the operation.

If the printer driver does not support the PASSTHROUGH printer escape, you must use the DeviceCapabilities() and ExtDevMode() functions instead.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgPrint

Using an Escape to Obtain and Change Paper Size for Printer

Article ID: Q96796

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

By using the Windows API Escape() function, an application can change the paper size on the printer and obtain a list of available paper metrics for the default printer.

To get the list of available paper metrics, pass the ENUMPAPERMETRICS printer escape constant to the Escape() function. The function will return either an array containing the paper metrics or the number of paper metrics available. Note that paper metrics differ from the physical paper sizes in that paper metrics delineate the actual region that can be printed to, whereas paper size is the physical size of the paper including the non-printable regions.

To change the paper size, pass the GETSETPAPERMETRICS printer escape constant along with the paper metrics to the Escape() function.

MORE INFORMATION

The example program listed below demonstrates how to use both printer escape constants (ENUMPAPERMETRICS and GETSETPAPERMETRICS) with the Windows API Escape() function.

An Important Note

Note that the Windows API Escape() function is provided in Windows versions 3.0 and 3.1 for backward compatibility with earlier versions of Microsoft Windows. Applications are supposed to use the GDI DeviceCapabilities() and ExtDeviceMode() functions instead of the Escape() function, but neither DeviceCapabilities() nor ExtDeviceMode() can be called directly from Visual Basic. This is because they are exported by the printer driver, not by the Windows GDI. The only way to use ExtDeviceMode() or DeviceCapabilities() in Visual Basic is to create a DLL and call them from there. To execute the ExtDeviceMode() function, you need to obtain a function pointer to it from the current printer driver. Visual Basic does not support pointers.

Steps to Create Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose New Module (ALT, F, M). Module1 is created by default.

3. Add the following code to the general declarations section of Module1:

```
Type Rect
   Left As Integer
   Top As Integer
   Right As Integer
   Bottom As Integer
End Type

' Enter each Declare as one, single line.
Declare Function EnumPaperMetricsEscape% Lib "GDI" Alias "Escape"
   (ByVal hDC%, ByVal nEscape%, ByVal IntegerSize%, lpMode%,
   lpOutData As Rect)
Declare Function SetPaperMetricsEscape% Lib "GDI" Alias "Escape"
   (ByVal hDC%, ByVal nEscape%, ByVal RectSize%, NewPaper As Rect,
   PrevPaper As Rect)
Declare Function GetDeviceCaps% Lib "gdi" (ByVal hDC%, ByVal nIndex%)

Global Const ENUMPAPERMETRICS = 34
Global Const GETSETPAPERMETRICS = 35
Global Const LOGPIXELSX = 88    '  Logical pixels/inch in X
Global Const LOGPIXELSY = 90    '  Logical pixels/inch in Y
```

4. Add the following code to the General Declarations section of Form1:

```
Dim RectArray() As Rect
```

5. Add a command button (Command1) to Form1.

6. Add a list box (List1) to Form1.

7. Add the following code to the Command1_Click event procedure. For readability some lines of code are shown as two lines but must be entered as a single line of code.

```
Sub Command1_Click ()
   ReDim RectArray(1)
   mode% = 0
   ' Enter the entire Result% statement as one, single line.
   Result% = EnumPaperMetricsEscape(Printer.hDC, ENUMPAPERMETRICS,
      2, mode%, RectArray(0))
   If Result% = 0 Then    ' If Result = 0, the call failed
      MsgBox "Printer Driver does not Support EnumPaperMetrics", 48
      Command1.Enabled = False
      Exit Sub
   End If

   ReDim RectArray(Result% - 1) ' Result% contains num paper sizes
   mode% = 1
   ' Enter the entire Result2% statement as one, single line.
   Result2% = EnumPaperMetricsEscape(Printer.hDC, ENUMPAPERMETRICS,
      2, mode%, RectArray(0))
   HorzRatio% = GetDeviceCaps(Printer.hDC, LOGPIXELSX)
   VertRatio% = GetDeviceCaps(Printer.hDC, LOGPIXELSY)
```

```
      ' Add Paper Sizes (Listed by actual printing region) in inches
      ' to the list box. Enter each of the PWidth$ and PHeight$ statements
      ' as one, single line.
      For i% = 0 To Result% - 1
         PWidth$ = Format$((RectArray(i%).Right - RectArray(i%).Left)
            / HorzRatio%) + Chr$(34) ' Enter as a single line
         PHeight$ = Format$((RectArray(i%).Bottom - RectArray(i%).Top)
            / VertRatio%) + Chr$(34) ' Enter as a single line
         List1.AddItem PWidth$ + " X " + PHeight$
      Next i%
   End Sub
```

8. Add the following code to the List1_Click event procedure:

```
   Sub List1_Click ()
      Dim PrevPaperSize As Rect
      ' Enter the entire Result% statement as one, single line.
      Result% = SetPaperMetricsEscape(Printer.hDC, GETSETPAPERMETRICS,
         Len(PrevPaperSize), RectArray(List1.ListIndex), PrevPaperSize)

      If Result% = 0 Then
         MsgBox "Printer Driver does not support this Escape.", 48
      ElseIf Result% < 0 Then
         MsgBox "Error in calling Escape with GETSETPAPERMETRICS."
      Else
         MsgBox "Paper size successfully changed!"
      End If
   End Sub
```

9. From the Run menu, choose Start (ALT, R, S) to run the program.

10. Choose the Command1 button to display a list of available paper metrics in the List1 box. The paper metrics represent the size of the printable regions supported by the printer, not the physical paper sizes.

11. Select one of the paper metrics shown in the List1 box. A message box appears indicating whether or not the paper size was successfully changed using the paper metrics you selected.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: APrgPrint

How to Obtain & Change the Paper Bins for the Default Printer

Article ID: Q96797

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

By using the Windows API Escape() function, an application can change the paper bin on the printer and obtain a list of available paper bins for the default printer.

To return a list of paper bin names and a list of corresponding of bin numbers, pass the ENUMPAPERBINS printer escape constant to the Escape() function. You can use the first list to display the available paper bins for the user, and use the second list to change the paper bin.

To change the paper bin, pass the GETSETPAPERBINS printer escape constant along with the bin number to the Escape() function. GETSETPAPERBINS returns the current bin and the number of bins supported by the default printer.

MORE INFORMATION

The example code listed below demonstrates how to use both ENUMPAPERBINS and GETSETPAPERBINS with the Windows API Escape() function.

An Important Note

Note that the Windows API Escape() function is provided in Windows versions 3.0 and 3.1 for backward compatibility with earlier versions of Microsoft Windows. Applications are supposed to use the GDI DeviceCapabilities() and ExtDeviceMode() functions instead of the Escape() function, but neither DeviceCapabilities() nor ExtDeviceMode() can be called directly from Visual Basic. This is because they are exported by the printer driver, not by the Windows GDI. The only way to use ExtDeviceMode() or DeviceCapabilities() in Visual Basic is to create a DLL and call them from there.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose New Module (ALT, F, M). Module1 is created by default.

3. Add the following code to the general declarations section of Module1:

   ```
   Global Const MaxBins = 6
   ```

```
Type PaperBin          ' Used for EnumPaperBins
   BinList(1 To MaxBins) As Integer
   PaperNames(1 To MaxBins) As String * 24
End Type

Type BinInfo                ' Used for GetSetPaperBins
   CurBinNumber As Integer  ' Current Bin
   NumBins As Integer       ' Number of bins supported by printer
   Reserved1 As Integer     ' Reserved
   Reserved2 As Integer     ' Reserved
   Reserved3 As Integer     ' Reserved
   Reserved4 As Integer     ' Reserved
End Type
' Enter each of the following Declare statements on one, single line.
Declare Function EnumPaperBinEscape% Lib "GDI" Alias "Escape"
   (ByVal hDC%, ByVal nEscape%, ByVal nCount%, NumBins%,
   lpOutData As Any)
Declare Function GetPaperBinEscape% Lib "GDI" Alias "Escape"
   (ByVal hDC%, ByVal nEscape%, ByVal nCount%, InBinInfo As Any,
   OutBinInfo As Any)

Global Const ENUMPAPERBINS = 31
Global Const GETSETPAPERBINS = 29
```

4. Add a command button (Command1) to Form1.

5. Add a list box (List1) to Form1.

6. Add the following code to the Command1_Click event procedure:

```
Sub Command1_Click ()
   Dim InPaperBin As PaperBin
   Dim InBinInfo As BinInfo
   ' Enter each of the following result% statements on one, single line:
   result% = GetPaperBinEscape(Printer.hDC, GETSETPAPERBINS, 0,
      ByVal 0&, InBinInfo)
   result% = EnumPaperBinEscape(Printer.hDC, ENUMPAPERBINS, 2,
      MaxBins, InPaperBin)

   List1.Clear
   For I% = 1 To InBinInfo.NumBins ' Fill list1 with available bins
      List1.AddItem InPaperBin.PaperNames(I%)
      List1.ItemData(List1.NewIndex) = InPaperBin.BinList(I%)
   Next I%

End Sub
```

7. Add the following code to the List1_Click event procedure:

```
Sub List1_Click ()
   Dim InBinInfo As BinInfo
   Dim NewBinInfo As BinInfo

   NewBinInfo.CurBinNumber = List1.ItemData(List1.ListIndex)
   ' Enter the following result% statement on one, single line.
   result% = GetPaperBinEscape(Printer.hDC, GETSETPAPERBINS,
      Len(NewBinInfo), NewBinInfo, NewBinInfo)
```

```
      MsgBox "Sending Sample Output to printer using Bin: " + List1.Text
      Printer.Print "This should of have come from Bin: "; List1.Text
      Printer.EndDoc
   End Sub
```

8. From the Run menu, choose Start (ALT, R, S) to run the program.

9. Choose the Command1 button to see a list of available paper bins for the default printer listed in the List1 box.

10. Select one of the paper bins listed in the List1 box. A message box appears to tell you that a sample printout is being sent to the printer using the paper bin you selected.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgPrint

How to Determine When a Shelled Process Has Terminated

Article ID: Q96844

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Executing the Shell() function in a Visual Basic for Windows program starts another executable program asynchronously and returns control to the Visual Basic application. The shelled program continues to run indefinitely until the user closes it -- not until your Visual Basic program terminates. However, your program can wait until the shelled program has finished by polling the return value of the Windows API GetModuleUsage() function. This article describes the method and provides a code example.

MORE INFORMATION

This information is included with the Help file provided with Microsoft Visual Basic version 3.0 for Windows.

Technique Also Works with MS-DOS Programs

The technique described in this article also works for MS-DOS programs. The return value from the Visual Basic Shell() function is a unique instance handle to the MS-DOS session that was started in the Shell(). If you call GetModuleUsage() with that handle after the MS-DOS session in question has ended, GetModuleUsage() will return 0 because the handle is no longer valid. This can be verified with the following code:

```
Debug.Print Shell("EDIT.COM")
Debug.Print Shell("EDIT.COM")
Debug.Print Shell("EDIT.COM")
```

Executing this code will show that the Shell() return value is unique for each of the shelled MS-DOS programs. Using GetModuleUsage() on one of these handles after the associated EDIT.COM program has been terminated will return zero (because the handle isn't valid anymore) and take you out of the wait loop.

Monitoring the Status of a Shelled Process

By using the Windows API GetModuleUsage() function, your Visual Basic program can monitor the status of a shelled process. The return value from the Shell() function can be used to call the GetModuleUsage() function continuously within a loop to find out if the shelled program has finished.

If the Shell() function is successful, the return value is the instance handle for the shelled program. This instance handle can be passed to the GetModuleUsage() function to determine the reference count for the module. When the GetModuleUsage() function returns a value of 0 or less, the shelled program has finished.

This algorithm works correctly regardless of the WindowStyle used to shell the program. In addition, this method works correctly when:

- Shelling to Windows programs.
- Shelling to MS-DOS programs.
- Shelling to applications that do not display a window.

Below are the steps necessary to build a Visual Basic for Windows program that uses the Shell() function to execute the Windows Notepad accessory (NOTEPAD.EXE). The code shows by example how to use the Windows API GetModuleUsage() function to wait until a shelled process terminates before resuming execution.

Step-by-Step Example
---

1. Start Visual Basic for Windows or from the File menu, choose New Project (ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Add the following code to the general declarations section of Form1:

```
Declare Function GetModuleUsage% Lib "Kernel" (ByVal hModule%)
```

3. Add the following code to the Form_Click event procedure of Form1:

```
Sub Form_Click ()
   x% = Shell("NOTEPAD.EXE")       ' Modify the path as necessary.

   While GetModuleUsage(x%) > 0    ' Has Shelled program finished?
      z% = DoEvents()              ' If not, yield to Windows.
   Wend
   MsgBox "Shelled application just terminated", 64
End Sub
```

4. From the Run menu, choose Start (ALT, R, S) to run the program.

5. Using the mouse, click in the Form1 window. At this point, the Notepad application is shelled.

The MsgBox statement following the Shell() Function is not executed because the While loop prevents it. The message box does not appear until Notepad is closed when the user chooses Exit from Notepad's File menu (ALT, F, X).

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

Using the Printer Object to Print a Grid Control's Contents

Article ID: Q96941

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

The example program in this article shows you how to print the contents of a grid control using the Printer object.

MORE INFORMATION

The example code prints a line border around the grid if the grid control BorderStyle is set to 1 and prints grid lines between the cells if GridLines is set to True.

Steps to Create Example Program

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running.

2. From the File menu, choose Add File. Select GRID.VBX. The Grid tool appears in the Toolbox.

3. Place a grid (Grid1) on Form1. Set the Cols and Rows properties to 6.

4. Add the following code to the Form1 Click event:

```
Sub Form_Click ()
    ' Add sample data to the grid
    Dim i, j
    For i = 0 To Grid1.Cols - 1
       For j = 0 To Grid1.Rows - 1
          Grid1.Col = i
          Grid1.Row = j
          Grid1.Text = Format$(i + j + i ^ j)
       Next
    Next
    ' Print the data
    Call Grid_Print(Grid1)
    Printer.EndDoc
End Sub
```

5. Add the following code to the general declarations section:

```
Sub Grid_Print (grid As Control)
    Dim tppx As Integer  ' alias TwipsPerPixelX
    Dim tppy As Integer  ' alias TwipsPerPixelY
    tppx = Printer.TwipsPerPixelX
```

```
tppy = Printer.TwipsPerPixelY
Dim Col As Integer   ' index to grid columns
Dim Row As Integer   ' index to grid rows
Dim x0 As Single     ' upper left corner
Dim y0 As Single     '   "
Dim x1 As Single     ' position of text
Dim y1  As Single    '   "
Dim x2  As Single    ' position of grid lines
Dim y2  As Single    '   "

' set upper left corner
x0 = Printer.CurrentX
y0 = Printer.CurrentY

' draw the border around the grid
If grid.BorderStyle <> 0 Then
   Printer.Line -Step(grid.Width - tppx, grid.Height - tppy), , B
   x0 = x0 + tppx
   y0 = y0 + tppy
End If

' draw the text in the grid
x1 = x0
For Col = 0 To grid.Cols - 1
   ' skip non-visible columns
   If Col >= grid.FixedCols And Col < grid.LeftCol Then
      Col = grid.LeftCol
   End If
   ' stop if outside grid
   If x1 + grid.ColWidth(Col) >= grid.Width Then Exit For
   y1 = y0
   For Row = 0 To grid.Rows - 1
      ' skip non-visible columns
      If Row >= grid.FixedRows And Row < grid.TopRow Then
         Row = grid.TopRow
      End If
      ' stop if outside grid
      If y1 + grid.RowHeight(Row) >= grid.Height Then Exit For
      ' set position to print the cell
      Printer.CurrentX = x1 + tppx * 2
      Printer.CurrentY = y1 + tppy
      ' print cell text
      grid.Col = Col
      grid.Row = Row
      Printer.Print grid.Text
      ' advance to next row
      y1 = y1 + grid.RowHeight(Row)
      If grid.GridLines Then
         y1 = y1 + tppy
      End If
   Next
   ' advance to next column
   x1 = x1 + grid.ColWidth(Col)
   If grid.GridLines Then
      x1 = x1 + tppx
   End If
Next
```

```
      ' draw grid lines
      If grid.GridLines Then
         x2 = x0
         y2 = y0
         For Col = 0 To grid.Cols - 1
            ' skip non-visible columns
            If Col >= grid.FixedCols And Col < grid.LeftCol Then
               Col = grid.LeftCol
            End If
            x2 = x2 + grid.ColWidth(Col)
            ' stop if outside grid
            If x2 >= grid.Width Then Exit For
            Printer.Line (x2, y0)-Step(0, y1 - tppy)
            x2 = x2 + tppx
         Next
         For Row = 0 To grid.Rows - 1
            ' skip non-visible rows
            If Row >= grid.FixedRows And Row < grid.TopRow Then
               Row = grid.TopRow
            End If
            y2 = y2 + grid.RowHeight(Row)
            ' stop if outside grid
            If y2 >= grid.Height Then Exit For
            Printer.Line (x0, y2)-Step(x1 - tppx, 0)
            y2 = y2 + tppy
         Next
      End If
   End Sub
```

6. Press the F5 key to run the program. Click Form1 to fill the grid with sample data and print the grid.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgPrint

How to Use SystemParametersInfo API for Control Panel Settings

Article ID: Q97142

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

The SystemParametersInfo API call can be used to get and set Windows settings that are normally set from the Desktop by using the Control Panel.

MORE INFORMATION

You can call the SystemParametersInfo API to set and get all the settings controlled by the Windows Control Panel. Normally a user would have to choose the Windows Control Panel to view or change system settings such as granularity, wallpaper, or icon title wrap. Instead of forcing the user to set things manually using the Control Panel you can have your program call the SystemParametersInfo API to set them automatically.

Use the following Visual Basic for Windows Declare for the API. Enter it all as one, single line:

```
Declare Function SystemParametersInfo Lib "User" (ByVal uAction
   As Integer, ByVal uparam As Integer, lpvParam As Any, ByVal fuWinIni
   As Integer) As Integer
```

Here are the formal arguments to the function:

```
uAction    system parameter to query or set
uParam     depends on system parameter
lpvParam   depends on system parameter
fuWinIni   WIN.INI update flag
```

The uAction argument can be one of the following constants:

```
CONST SPI_GETBEEP=1
CONST SPI_SETBEEP=2
CONST SPI_GETMOUSE=3
CONST SPI_SETMOUSE=4
CONST SPI_GETBORDER=5
CONST SPI_SETBORDER=6
CONST SPI_GETKEYBOARDSPEED=10
CONST SPI_SETKEYBOARDSPEED=11
CONST SPI_LANGDRIVER=12
CONST SPI_ICONHORIZONTALSPACING=13
CONST SPI_GETSCREENSAVETIMEOUT=14
CONST SPI_SETSCREENSAVETIMEOUT=15
CONST SPI_GETSCREENSAVEACTIVE=16
```

```
CONST SPI_SETSCREENSAVEACTIVE=17
CONST SPI_GETGRIDGRANULARITY=18
CONST SPI_SETGRIDGRANULARITY=19
CONST SPI_SETDESKWALLPAPER=20
CONST SPI_SETDESKPATTERN=21
CONST SPI_GETKEYBOARDDELAY=22
CONST SPI_SETKEYBOARDDELAY=23
CONST SPI_ICONVERTICALSPACING=24
CONST SPI_GETICONTITLEWRAP=25
CONST SPI_SETICONTITLEWRAP=26
CONST SPI_GETMENUDROPALIGNMENT=27
CONST SPI_SETMENUDROPALIGNMENT=28
CONST SPI_SETDOUBLECLKWIDTH=29
CONST SPI_SETDOUBLECLKHEIGHT=30
CONST SPI_GETICONTITLELOGFONT=31
CONST SPI_SETDOUBLECLICKTIME=32
CONST SPI_SETMOUSEBUTTONSWAP=33
CONST SPI_SETICONTITLELOGFONT=34
CONST SPI_GETFASTTASKSWITCH=35
CONST SPI_SETFASTTASKSWITCH=36
```

The UParam argument should be 0 when used with a GET constant, and it should contain the new value of the setting when used with a SET constant. The exceptions to these rules are documented in the Windows version 3.1 Software Development Kit (SDK) help file.

When used with a GET constant, the lpvParam argument returns the current value of the setting. When used with a SET constant, it is a NULL. The exceptions to these rules are documented in the Windows version 3.1 SDK help file.

The fuWinIni argument updates the WIN.INI file:

```
Const SPIF_SENDWININICHANGE = &H2
Const SPIF_UPDATEINIFILE = &H1
```

Example One

One exception to the rules given above occurs with a call to set or get the icon spacing setting. The following example gives the correct arguments to use to set and get the horizontal spacing:

1. Create a Visual Basic project, and add the following controls to a form:

```
Control Name   Caption
----------------------
Command1       Read
Command2       Set
Text1
Label1         Icon Horizontal Spacing
```

2. Add the following code to the general declarations section of the form:

```
Const SPIF_SENDWININICHANGE = &H2
Const SPIF_UPDATEINIFILE = &H1
Const SPI_ICONHORIZONTALSPACING = 13
Dim uAction As Integer
```

```
Dim uparam As Integer
' Enter the following Declare as one, single line:
Declare Function SystemParametersInfo Lib "User" (ByVal uAction As
   Integer, ByVal uparam As Integer, lpvParam As Any, ByVal fuWinIni As
   Integer) As Integer
```

3. Add the following code to the Command1_Click event:

```
uAction = 0
uparam = 0
ret% = SystemParametersInfo(SPI_ICONHORIZONTALSPACING, uAction,
uparam, SPIF_UPDATEINIFILE Or SPIF_SENDWININICHANGE)
text1.Text = uparam
```

4. Add the following code to the Command2_Click event:

```
uAction = Val(text1.Text)
uparam = 0
' Enter the following as one, single line:
x% = SystemParametersInfo(SPI_ICONHORIZONTALSPACING, uAction,
   ByVal 0&, SPIF_UPDATEINIFILE Or SPIF_SENDWININICHANGE)
```

5. Run the program, and click the Read button. The current setting of the icon horizontal spacing will be displayed in the Text1 box. Enter a new number(32 is the lowest setting accepted) in the Text1 box, and click the Read button. The spacing will be reset. To see the new setting, bring up the Windows Task list, and choose Arrange Icons.

Example Two

The example follows the general parameter rules. It demonstrates how to turn icon title wrapping on and off by using SETICONTITLEWRAP.

1. Create a Visual Basic project and add the following controls to a form:

Control Name	Caption
Command1	Wrapping True
Command2	Wrapping False

2. Add the following code to the general declarations section of the form:

```
' Enter the following Declare as one, single line:
Declare Function SystemParametersInfo Lib "User" (ByVal uAction As
   Integer, ByVal uparam As Integer, lpvParam As Any, ByVal fuWinIni As
   Integer) As Integer
Const SPI_SETICONTITLEWRAP = 26
Const SPIF_SENDWININICHANGE = &H2
Const SPIF_UPDATEINIFILE = &H1
```

3. Add the following code to the Command1 Click event:

```
' Enter the following as one, single line:
x% = SystemParametersInfo(SPI_SETICONTITLEWRAP, True, 0&,
   SPIF_UPDATEINIFILE Or SPIF_SENDWININICHANGE)
```

4. Add the following code to the Command2 Click event:

```
   ' Enter the following as one, single line:
   x% = SystemParametersInfo(SPI_SETICONTITLEWRAP, False, 0&,
      SPIF_UPDATEINIFILE Or SPIF_SENDWININICHANGE)
```

5. Run the program and watch the icon titles as you click the two buttons.

Example Three

This example follows the general parameter rules. It demonstrates how to change your desktop's wallpaper with the SPI_SETDESKWALLPAPER.

1. Create a Visual Basic project and add the following controls to a form:

Control Name	Caption
Command1	Change Wallpaper to Rivets

2. Add the following code to the general declarations section of the form:

```
   Const SPIF_UPDATEINIFILE = &H1
   Const SPI_SETDESKWALLPAPER = 20
   Const SPIF_SENDWININICHANGE = &H2

   ' Enter the following Declare as one, single line:
   Declare Function SystemParametersInfo Lib "User" (ByVal uAction As
      Integer, ByVal uparam As Integer, ByVal lpvParam As String, ByVal
      fuWinIni As Integer) As Integer
```

3. Add the following code to the Command1 Click event:

```
   Sub Command1_Click ()
      filenm$ = "C:\Windows\rivets.bmp"

      ' Enter the following two lines as one, single line:
      x% = SystemParametersInfo(SPI_SETDESKWALLPAPER, 0&,
         filenm$, SPIF_UPDATEINIFILE Or SPIF_SENDWININICHANGE)
   End Sub
```

4. Run the program and watch the wallpaper change to RIVETS.BMP.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

Example of calling EnumFontFamilies from a DLL

Article ID: Q98577

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, versions 2.0 and 3.0

SUMMARY

This article demonstrates how to obtain a list of available fonts for a device by calling EnumFontFamilies or EnumFonts from a DLL.

Visual Basic already provides a Fonts property for obtaining a list of available font names for a device. Microsoft recommends that you use the Fonts property instead of the function provided in this article to obtain a list of available fonts. Use the technique shown in this article only if you have encountered a bug or limitation when using the Fonts property.

To create the example shown below, you need a C compiler capable of creating Windows dynamic link libraries (DLLs), and you need to have the Visual Basic Control Development Kit (CDK) version 2.0 or 3.0. The CDK is provided with the Professional Edition of Visual Basic version 2.0 and 3.0 for Windows.

MORE INFORMATION

Below are the steps necessary to create a sample DLL that demonstrates using EnumFontFamilies:

STEP ONE: Create Example .DLL File

1. Create a source file called FONTNAME.C and add the following code:

```
#include <windows.h>
#include <vbapi.h>
#include <string.h>

int FAR PASCAL _export EnumFontNames (HDC, HAD);
int FAR PASCAL  _export GetNextFont (LPLOGFONT, LPNEWTEXTMETRIC,
                                     int, LPARAM);
BOOL Win31OrGreater (VOID);

int giFontCount;
float gfVersion;

//====================================================================
// Title
//      EnumFontNames()
//
// Parameters
//      hdc      Device context for which fonts will be enumerated
```

```
//      had      Handle to Visual Basic string array where the
//               font names will be placed.
//
// Returns
//      The number of fonts enumerated.
//=================================================================
int FAR PASCAL EnumFontNames (HDC hdc, HAD had)
{
   giFontCount = 0;

   if ( Win31OrGreater() )
      //Use EnumFontFamilies under Win 3.1 and later
      while (EnumFontFamilies(hdc, NULL, GetNextFont, had));
   else
      //Need to use EnumFonts under Win 3.0
      while (EnumFonts(hdc, NULL, GetNextFont, had));

   return giFontCount;
}

//=================================================================
// Title
//      GetNextFont()
//
// Parameters
//      lplf     Far pointer to LOGFONT structure
//      lpntm    Far pointer to NEWTEXTMETRIC structure
//      FontType Type of font
//      lp       User-defined.  In this case it holds the handle
//               to a Visual Basic string array.
//
// Returns
//      TRUE as a signal to enumerate the next font
//      FALSE as a signal to stop enumeration
//=================================================================
int FAR PASCAL GetNextFont
(
   LPLOGFONT lplf,
   LPNEWTEXTMETRIC lpntm,
   int FontType,
   LPARAM lp
)
{
   static char szFirstFont[LF_FACESIZE + 1];
   char szFaceName[LF_FACESIZE + 1];
   int iElements, lbound;

   HAD had = (HAD) lp;
   LONG lBounds = VBArrayBounds(had, 1);

   //Get out if there are no elements in the array
   if (lBounds == AB_INVALIDINDEX)
      return FALSE;

   // Store the lower bound of the array for index 1
   lbound = LOBOUND(lBounds);
```

```
//Get number of elements in the array
iElements = HIBOUND(lBounds) - lbound + 1;

//Initialize the vars holding the font face names
if (giFontCount == 0)
    szFirstFont[0] = '\0';

szFaceName[0] = '\0';

if (giFontCount <= iElements)
{
   HLSTR hlstr;
   SHORT indexes[1];

   //Copy the face size into a buffer so that we can insure its
   //null terminated
   if ( Win31OrGreater() )
       lstrcpyn((LPSTR) szFaceName, lplf->lfFaceName,
                LF_FACESIZE - 1);
   else
       //Need to use C runtime routine fmemcpy instead of
       //lstrcpyn under Win 3.0
      _fmemcpy((LPVOID) szFaceName, lplf->lfFaceName,
                LF_FACESIZE - 1);

   szFaceName[LF_FACESIZE] = '\0';

   if (giFontCount == 0)

      //Store the first font retrieved.  If we see this font
      //again, we know we've enumerated all the fonts
      lstrcpy((LPSTR) szFirstFont, szFaceName);

   else if (!lstrcmp(szFirstFont, szFaceName))
   //If we see the same face name again, get out and stop
   //enumerating
      return FALSE;

   //Assume a single index array
   indexes[0] = lbound + giFontCount;

   //Get the VB string handle from the VB array
   hlstr = VBArrayElement(had, VBArrayIndexCount(had),
                          indexes);

   //Make sure the string handle is valid
   if (HIWORD(hlstr))
   {
      //Add the fontname to the array
      VBSetHlstr(&hlstr, (LPSTR) szFaceName, lstrlen((LPSTR)
                 szFaceName));

      //Return and get the next font
      giFontCount++;
   }

   return TRUE;
```

```
   }

   else
      //Can't fit all font names into the array provided, so get
      //out.
      return FALSE;
}

//==========================================================================
// Title
//      Win31OrGreater ()
//
// Returns
//      TRUE if we're running under Windows 3.1 or better
//      FALSE if we're running under Windows 3.0
//==========================================================================
BOOL Win31OrGreater ( VOID )
{
    DWORD dVersion;

    //Check which version of Windows we're running under
    dVersion = GetVersion();
    if (LOBYTE(LOWORD(dVersion)) > 3 || (LOBYTE(LOWORD(dVersion))
        == 3 && HIBYTE(LOWORD(dVersion)) > 0))
        return TRUE;
    else
        return FALSE;
}

//--------------------------------------------------------------------------
// Initialize library. This routine is called when the first
// client loads
// the DLL.
//--------------------------------------------------------------------------
int FAR PASCAL LibMain
(
   HANDLE hModule,
   WORD   wDataSeg,
   WORD   cbHeapSize,
   LPSTR  lpszCmdLine
)
{
   // Avoid warnings on unused (but required) formal parameters
   wDataSeg = wDataSeg;
   cbHeapSize = cbHeapSize;
   lpszCmdLine = lpszCmdLine;

   return 1;
}

//--------------------------------------------------------------------------
// WEP
//--------------------------------------------------------------------------
int FAR PASCAL WEP(int fSystemExit);

//--------------------------------------------------------------------------
// Performs cleanup tasks when the DLL is unloaded.  WEP() is
```

```
// called automatically by Windows when the DLL is unloaded (no
// remaining tasks still have the DLL loaded).  It is strongly
// recommended that a DLL have a WEP() function, even if it does
// nothing but returns success (1), as in this example.
//--------------------------------------------------------------
int FAR PASCAL WEP
(
    int fSystemExit
)
{
    // Avoid warnings on unused (but required) formal parameters
    fSystemExit = fSystemExit;

    return 1;
}
```

2. Create a module-definition file (DEF) called FONTNAME.DEF and add the following:

```
LIBRARY FONTNAME

DESCRIPTION 'Example of how to enumerate all font names for
             specific device'

EXETYPE WINDOWS

CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE SINGLE

EXPORTS
   WEP @1 RESIDENTNAME
   ENUMFONTNAMES @2
   GETNEXTFONT @3
```

3. Compile FONTNAME.C from the MS-DOS command line as follows:

```
CL /c /ASw /W3 FONTNAME.C
```

4. Link the resulting FONTNAME.OBJ file as follows:

```
LINK /NOE /NOD
   FONTNAME.OBJ+LIBENTRY.OBJ,FONTNAME.DLL,,
   LIBW+SDLLCEW+VBAPI.LIB,FONTNAME.DEF;
```

5. Resource compile FONTNAME.DLL to make it Windows 3.0 compatible as follows:

```
RC /30 FONTNAME.DLL
```

6. Copy FONTNAME.DLL to the \WINDOWS\SYSTEM directory.

STEP TWO: Create Visual Basic Sample Program
---

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a list box (List1) to Form1.

3. Add the following Declare statement as one, single line to the general-declarations section of Form1:

```
Declare Function EnumFontNames Lib "FONTNAME.DLL" (ByVal hDC As
   Integer, FontNames() As String) As Integer
```

4. Add the following code to the Form_Click event of Form1:

```
Sub Form_Click ()

   Dim i As Integer
   Dim FontCount As Integer
   ReDim FontNames(255) As String  'Make the array intentionally
                                   'large to hold any number of
                                   'font names

   'For Screen fonts, pass Form1.hDC instead.  If using the
   'Common Dialog control, you can also pass the hDC property
   'of the Common Dialog control.
   FontCount = EnumFontNames(Printer.hDC, FontNames())

   List1.Clear
   For i = 0 To FontCount - 1
      List1.AddItem FontNames(i)
   Next

End Sub
```

5. From the Run menu, choose Start (ALT, R, S) or press F5 to run the program.

6. Click Form1.

The available font names for the selected printer will be displayed in the list box.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Print Text Sideways in Picture Control with Windows API

Article ID: Q99874

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic for Windows, version 1.0

SUMMARY

The example below shows how to print text sideways in a picture control using Windows API function calls. The text prints vertically in the picture control, rotated by 90 degrees.

MORE INFORMATION

Step-by-Step Example

1. Start Visual Basic. Form1 is created by default.

2. Draw a large picture box (Picture1) on the form.

3. From the File menu, choose New Module to create a new module. Put the following code in the module:

```
DefInt A-Z
global Const LF_FACESIZE = 32
Type LOGFONT
   lfheight As Integer
   lfwidth As Integer
   lfescapement As Integer
   lforientation As Integer
   lfweight As Integer
   lfitalic As String * 1
   lfunderline As String * 1
   lfstrikeout As String * 1
   lfcharset As String * 1
   lfoutprecision As String * 1
   lfclipprecision As String * 1
   lfquality As String * 1
   lfpitchandfamily As String * 1
   lffacename As String * LF_FACESIZE
End Type

' Enter each of the following 7 Declare statement on one, single line:
Declare Function CreateFont% Lib "GDI" (ByVal h%, ByVal w%, ByVal e%,
   ByVal o%, ByVal n%, ByVal i%, ByVal u%, ByVal s%, ByVal c%,
   ByVal op%, ByVal cp%, ByVal q%, ByVal j%, ByVal f$)
Declare Function createfontindirect Lib "GDI" (lplogfont As LOGFONT)
   As Integer
Declare Function selectobject Lib "GDI" (ByVal hdc%, ByVal object%)
```

```
   As Integer
Declare Function textout Lib "GDI" (ByVal hdc%, ByVal x%, ByVal y%,
   ByVal text$, ByVal ncount%) As Integer
Declare Sub deleteobject Lib "GDI" (ByVal object%)
Declare Function getdevicecaps Lib "GDI" (ByVal hdc%, ByVal nindex%)
   As Integer
Declare Function gettextface Lib "GDI" (ByVal hdc As Integer,
   ByVal ncount As Integer, ByVal lpname As String) As Integer

Global Const PROOF_QUALITY = 2
Global Const FW_NORMAL = 400
```

4. Add the following code to the Form_Click event:

```
picture1.Cls
Dim hfont As Integer, holdfont As Integer
Dim font As LOGFONT
nvalue = getdevicecaps(picture1.hDC, 34)
font.lfheight = 12
font.lfwidth = 0
font.lfescapement = 900
font.lforientation = 900
font.lfweight = 400    'This is normal
font.lfitalic = Chr$(0)
font.lfunderline = Chr$(0)
font.lfstrikeout = Chr$(0)
font.lfcharset = Chr$(0)
font.lfoutprecision = Chr$(0)
font.lfclipprecision = Chr$(0)
font.lfquality = Chr$(2)
font.lfpitchandfamily = Chr$(33)
font.lffacename = "Courier New" + Chr$(0)

hfont = createfontindirect(font)
holdfont = selectobject(picture1.hDC, hfont)
szfacename$ = Space$(80)
retval% = gettextface(picture1.hDC, 79, szfacename$)

nchars = Len(sometext$)
picture1.CurrentX = 200
picture1.CurrentY = 2000
picture1.Print Left$(RTrim$(szfacename$), Len(RTrim$(szfacename$)) - 1)
deleteobject hfont
```

5. Run the program. Click the form, not the picture. You'll see the phrase "Courier New" print sideways in the picture control, from the lower left to the upper left.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgPrint APrgWindow

How to Play MIDI Files Using API Calls from Visual Basic

Article ID: Q99898

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

This article demonstrates how to play a MIDI (.MID) file from Visual Basic using Windows version 3.1 APIs.

If you have the Professional Edition of Visual Basic version 2.0 or 3.0, or if you have the Professional Toolkit for Visual Basic version 1.0, you can use the MCI control to play a MIDI file. You don't need to use the APIs

MORE INFORMATION

Step by Step to an Application that Plays an .MID file

1. Start Visual Basic, or if Visual Basic is already running, choose New Project from the File menu (ALT, F, N). Form1 is created by default.

2. Add a Command Button (Command1) to Form1.

3. Add the following code to the Command1_Click event of Form1:

```
DIM ret as Integer

'*** The following will open the sequencer with the C:\WIN31\CANYON.MID
'*** file. Canyon is the device_id. Enter the entire statement on one,
'*** single line.
ret = mciSendString("open CANYON.MID type sequencer alias canyon",
   0&, 0, 0)

'*** The wait tells the MCI command to complete before returning control
'*** to the application.
Ret = mciSendString("play canyon wait", 0&, 0, 0)

'***  Close CANYON.MID file and sequencer device
Ret = mciSendString("close Animation", 0&, 0, 0)
```

4. Add the following code to the general declarations section of Form1:

```
' Enter the following Declare statement on one, single line:
Declare Function mciSendString Lib "mmsystem" (ByVal lpstrCommand$,
   ByVal lpstrReturnStr As Any, ByVal wReturnLen%, ByVal hCallBack%)
   As Long
```

5. From the Run menu, choose Start (ALT, R, S) or press F5 to run the program.

More information about sndSendString() can be found in:

 - the MultiMedia Programmer's Reference on page 3-26.
 - Command strings described on pages 7-23 to 7-93 and in the WIN31MWH.HLP
   file shipped with the Windows 3.1 Software Development Kit (SDK).

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgWindow

How to Read a Large File into Memory by Calling API Functions

Article ID: Q100513

The information in this article applies to:

- The Visual Basic programming system for Windows, versions 2.0 and 3.0
- Microsoft Windows, version 3.1 or higher

SUMMARY

This article demonstrates how to call Windows API functions to read a file of any size (including a huge file such as a bitmap) into memory and how to write a block of memory (including a huge memory block) out to a file.

The information in this article applies only to Windows version 3.1 or higher because it uses Windows API functions introduced in Windows version 3.1.

MORE INFORMATION

Perform the following steps to create a sample program that demonstrates how to read a large file into memory and write that memory back out to a file:

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose New Module (ALT, F, M). Module1 is created.

3. Add the following code to the general-declarations section of Module1:

```
' OpenFile() Structure
Type OFSTRUCT
   cBytes As String * 1
   fFixedDisk As String * 1
   nErrCode As Integer
   reserved As String * 4
   szPathName As String * 128
End Type

' OpenFile() Flags
Global Const OF_READ = &H0
Global Const OF_WRITE = &H1
Global Const OF_READWRITE = &H2
Global Const OF_SHARE_COMPAT = &H0
Global Const OF_SHARE_EXCLUSIVE = &H10
Global Const OF_SHARE_DENY_WRITE = &H20
Global Const OF_SHARE_DENY_READ = &H30
Global Const OF_SHARE_DENY_NONE = &H40
Global Const OF_PARSE = &H100
Global Const OF_DELETE = &H200
Global Const OF_VERIFY = &H400
```

```
Global Const OF_CANCEL = &H800
Global Const OF_CREATE = &H1000
Global Const OF_PROMPT = &H2000
Global Const OF_EXIST = &H4000
Global Const OF_REOPEN = &H8000

' Enter each of the following Declare statements on one, single line:
Declare Function OpenFile Lib "Kernel" (ByVal lpFilename As
   String, lpReOpenBuff As OFSTRUCT, ByVal wStyle As Integer) As Integer
Declare Function hRead Lib "kernel" Alias "_hread" (ByVal hFile As
   Integer, lpMem As Any, ByVal lSize As Long) As Long
Declare Function hWrite Lib "Kernel" Alias "_hwrite" (ByVal hFile
   As Integer, lpMem As Any, ByVal lSize As Long) As Long
Declare Function lClose Lib "kernel" Alias "_lclose" (ByVal hFile
   As Integer) As Integer

' Global Memory Flags
Global Const GMEM_FIXED = &H0
Global Const GMEM_MOVEABLE = &H2
Global Const GMEM_NOCOMPACT = &H10
Global Const GMEM_NODISCARD = &H20
Global Const GMEM_ZEROINIT = &H40
Global Const GMEM_MODIFY = &H80
Global Const GMEM_DISCARDABLE = &H100
Global Const GMEM_NOT_BANKED = &H1000
Global Const GMEM_SHARE = &H2000
Global Const GMEM_DDESHARE = &H2000
Global Const GMEM_NOTIFY = &H4000
Global Const GMEM_LOWER = GMEM_NOT_BANKED

Global Const GHND = (GMEM_MOVEABLE Or GMEM_ZEROINIT)
Global Const GPTR = (GMEM_FIXED Or GMEM_ZEROINIT)

' Enter each of the following Declare statements on one, single line:
Declare Function GlobalAlloc Lib "Kernel" (ByVal wFlags As
    Integer, ByVal dwBytes As Long) As Integer
Declare Function GlobalLock Lib "Kernel" (ByVal hMem As Integer)
    As Long
Declare Function GlobalUnlock Lib "Kernel" (ByVal hMem As Integer)
    As Integer
Declare Function GlobalFree Lib "Kernel" (ByVal hMem As Integer)
    As Integer
```

4. Add the following code to the Form_Load event procedure of Form1:

```
Sub Form_Load ()

   Dim InpFile As String
   Dim OutFile As String
   Dim hFile As Integer
   Dim fileStruct As OFSTRUCT
   Dim FSize As Long
   Dim BytesRead As Long
   Dim BytesWritten As Long
   Dim hMem As Integer
   Dim lpMem As Long
   Dim r As Integer
```

```
    Me.Show

    'Insert the name of a bitmap or file that is greater than 64K.
    '256COLOR.BMP is less than 5K in size, however, the routine
    'below still demonstrates how to read and write a file of any
    'size
    InpFile = "C:\WINDOWS\256COLOR.BMP"
    OutFile = "C:\WINDOWS\TEST.BMP"

    'Get the size of the file to be read
    FSize = FileLen(InpFile)

    If FSize > 0 Then

       'Allocate a block of memory equal to the size of the input file.
       hMem = GlobalAlloc(GPTR, FSize)

       If hMem <> 0 Then
          lpMem = GlobalLock(hMem)

          'Read the file into memory
          hFile = OpenFile(InpFile, fileStruct, OF_READ Or
                           OF_SHARE_DENY_NONE)
          BytesRead = hRead(hFile, ByVal lpMem, FSize)

          MsgBox Format(BytesRead) & " bytes read into memory"

          r = lClose(hFile)

          'Write the file back to disk to verify the file was
          'read correctly
          hFile = OpenFile(OutFile, fileStruct, OF_CREATE Or
                           OF_WRITE Or OF_SHARE_DENY_NONE)
          BytesWritten = hWrite(hFile, ByVal lpMem, FSize)

          MsgBox Format(BytesWritten) & " bytes written to output file"

          r = lClose(hFile)

          'Free resources
          r = GlobalUnlock(hMem)
          r = GlobalFree(hMem)
       Else
           MsgBox "Not enough memory to store file"
       End If
    Else
       MsgBox "Input file is zero bytes in length"
    End If
    End
End Sub
```

5. From the Run menu, choose Start (ALT, R, S) or press F5 to run the program. Form1 will be displayed and the program will end.

6. Use PaintBrush or some other bitmap editor to open C:\WINDOWS\TEST.BMP to verify that it is the same bitmap as C:\WINDOWS\256COLOR.BMP.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgWindow

How to Find Next Available Drive Letter (for Network Connect)

Article ID: Q100834

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

The Visual Basic program in this article shows by example how to find the next available (unused) drive letter in Windows. This is useful when making network connections to a new drive letter.

MORE INFORMATION

Step-by-Step Example

The Freedrive function defined below returns the next drive letter available in Windows, followed by a colon (:).

1. Start Visual Basic. Form1 is created by default.

2. Add the following code to the General Declarations section of Form1:

```
' Enter the following Declare statement as one, single line:
Declare Function GetDriveType Lib "kernel"
   (ByVal nDrive As Integer) As Integer

Function Freedrive ()
   Dim DriveNum As Integer, FirstFreeDrive As String
   Dim FirstDrive As Integer
   DriveNum = -1
   Do
      DriveNum = DriveNum + 1   ' start at drive zero.
      FirstDrive% = GetDriveType(DriveNum)
      ' GetDriveType returns zero if it cannot determine drive
      ' type or returns 1 if the specified drive does not exist.
   Loop Until FirstDrive% = 0
   ' DriveNum of 0 means Drive A, 1=B, 2=C, 3=D, 4=E, 5=F, and so on:
   FirstFreeDrive = Chr$(DriveNum + 65) + ":"
   Freedrive = FirstFreeDrive
End Function
```

3. In the Form_click event, add the following statements:

```
Sub Form_Click ()

   Cls
   Print "The next available (unused) drive letter is: "; Freedrive()
```

```
      ' More handy tips: The "App" object below is found in VB 2.0
      ' and 3.0 (but not 1.0).
      Print "The title for the EXE in Windows Task Manager: "; app.Title
      Print "The name of this EXE, or project in VB, is: "; app.EXEName
      Print "The path to this application is: "; app.Path

   End Sub
```

4. Run the program, and click the form.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgNet

How to Set the Formatting Rectangle of a TextBox

Article ID: Q101162

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

You can change the formatting rectangle of a text box to control when scrolling occurs.

The formatting rectangle determines the range of positions allowed for the caret (text cursor). The limiting rectangle is independent of the size of the edit-control window. By default, the formatting rectangle is the same as the client area of the edit-control window.

MORE INFORMATION

Using the SendMessage API call and the EM_SETRECT message, you can set the formatting rectangle of a text box. If you do not send the EM_SETRECT message, the formatting rectangle defaults to the size of the client area of the text box.

You can use this API call to control where the scrolling starts in a text box. The default scrolling starts when the cursor reaches the left side of the text box. This API can make that rectangle smaller than the actual text box forcing the scrolling to start before the cursor reaches the left side of the text box.

Note the following if you do not use the message until after text has been entered into the text box:

If the text box does not have a horizontal scroll bar, and the formatting rectangle is set to be larger than the text box window, lines of text exceeding the width of the text box (but smaller than the width of the formatting rectangle) are clipped instead of wrapped.

Step-by-Step Demonstration

1. Start Visual Basic, or if you are in Visual Basic, start a new project.

2. Add a text box (Text1) to your form.

3. Set the Text1 MultiLine Property to True and the ScrollBars Property to 3 (Both).

4. From the File menu, choose New Module (Module1.bas).

5. Add the following code to Module1.bas:

```
   Type recttype
      l As Integer ' left of rectangular region
      t As Integer ' top of region
      r As Integer ' right of region
      b As Integer ' bottom of region
   End Type
   ' Note the following Declare must be on one, single line:
   Declare Function SendMessage Lib "user" (ByVal hwnd%, ByVal wMsg%,
      ByVal wp%, lp As Any) As Long
```

6. Add the following code to the Form_Load event for Form1:

```
   Sub Form_Load ()
      EM_SETRECT = &H403      ' Set EM_SETRECT variable
      Dim rect As recttype    ' dim variable as rectype
      rect.l = 0              ' Set left to upper left corner
      rect.t = 0              ' Set top to upper left corner
      rect.r = 200            ' Set right of region
      rect.b = 200            ' Set bottom of region
      x% = SendMessage(text1.hwnd, EM_SETRECT, 0, rect)
   End Sub
```

7. Run the program.

Start typing in the text box. Scrolling will begin when you reach the edge of your region. You can change the size of your region by changing the values of the rect type.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgWindow

Adjusting Form Size for Different Video Screen Resolutions

Article ID: Q103646

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 2.0 and 3.0

SUMMARY

Different display devices can have different resolutions (twips per pixel ratios). These differences can cause form and control sizes and locations to appear differently than when they were created. Two solutions to this problem are:

- Set the ScaleMode on all forms and picture boxes to Pixels (3). This unit of measurement does not depend on screen resolution, so forms and controls will always appear the same size and location relative to each other.

- Adjust your form and control sizes and locations at run time to match visual elements which are not affected by the screen resolution. For example, the sample program given below adjusts the width of the client area of a form to match a bitmap which is a fixed number of pixels wide and is therefore not affected by screen resolution.

MORE INFORMATION

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add two labels, one command button, and one picture control to Form1.

3. Set Picture1's picture property to C:\WINDOWS\WINLOGO.BMP.

4. Add the following code in the Form Load event procedure of Form1:

```
Sub Form_Load ()

    ' Set up a picture box:
    Picture1.AutoSize = True
    Picture1.Move 0, 0

    ' Set up the labels and command button:
    Xtwips& = Screen.TwipsPerPixelX
    Ytwips& = Screen.TwipsPerPixelY
    Ypixels& = Screen.Height / Ytwips&
    Xpixels& = Screen.Width / Xtwips&
    label1.Caption = "Below is resolution that you are running in"
```

```
      label2.Caption = Str$(Xpixels&) + " by " + Str$(Ypixels&)
      label1.Width = Picture1.Width
      label2.Width = Picture1.Width
      label1.Left = 0
      label2.Left = 0
      label1.Top = Picture1.Height + 10
      label2.Top = label1.Top + label1.Height + 10
      command1.Top = label2.Top + label2.Height + 10
      command1.Left = (Picture1.Width - command1.Width) / 2

      ' Size the form to fit the picture box, labels, and command button
      ScaleMode = 1  ' twips
      Width = Width - ScaleWidth + Picture1.Width

      ' Enter the Height statement as one, single line:
      Height = Height - ScaleHeight + Picture1.Height + label1.Height
         + label2.Height + command1.Height
   End Sub
```

5. Add the following code in the Command1 Click event procedure:

```
   Sub Command1_Click ()
      End
   End Sub
```

6. Press the F5 key to run the program. Click the Command1 button to exit from the example.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgWindow

How to Play an .AVI Video File in Full Screen in Visual Basic

Article ID: Q104123

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

This article shows by example how to play an .AVI (video) file in full screen from Visual Basic for Windows. When you play an .AVI file using the full screen, the color palette focus is set to the .AVI file only. No dithering of colors occurs because there are no other windows in the background to capture the color palette.

MORE INFORMATION

The example uses the mciSendString application programming interface (API) from Microsoft Windows version 3.1 or Microsoft Windows version 3.0 with Multimedia Extensions.

For the example to work, your computer must be able to play .AVI files and you need either Microsoft Windows version 3.1 or Microsoft Windows version 3.0 with Multimedia Extensions.

The .AVI file included in the example (WNDSURF1.AVI) is the one from Microsoft Video for Windows.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a command button (Command1) to Form1, and set its caption property to: Play Video.

3. Add the following line of code to the (general) (declarations) section of Form1:

```
' Enter the following Declare statement on one, single line:
Declare Function mciSendString Lib "mmsystem"
   (ByVal lpstrCommand$, ByVal lpstrReturnStr As Any,
   ByVal wReturnLen%, ByVal hCallBack%) As Long
```

4. Add the following lines of code to the Command1 Click event procedure:

```
Sub Command1_Click ()
   CmdStr$ = "play c:\winvideo\wndsurf1.avi fullscreen "
   ReturnVal& = mciSendString(CmdStr$, 0&, 0, 0&)
End Sub
```

5. From the Run menu, choose Start (ALT, R, S) to run the program. Click the Play Video button to watch the video full screen. The video will last for a few seconds and return back to the Visual Basic environment.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

Windows Debugging Tools for Use with Visual Basic

Article ID: Q104156

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

The Microsoft Windows debugging tools listed in this article may help you debug and troubleshoot problems such as general protection (GP) faults that you encounter while in Visual Basic or while executing a compiled Visual Basic program.

None of these tools are included as part of the Microsoft Visual Basic programming system for Windows, but they are readily available from other sources as listed in each tool's description. You can use these tools to debug many different problems, including but not limited to GP faults.

MORE INFORMATION

The following tools may help you debug your Visual Basic programs. A brief description of each tool is given below. For more information, review the Microsoft Windows Software Development Kit (SDK) documentation.

Dr. Watson for Windows (DRWATSON.EXE)

This tool comes with Microsoft Windows version 3.1. It is located in the \WINDOWS directory. This is a diagnostic tool for the Microsoft Windows operating system. It detects system and application failures caused by Windows applications and can store information in a disk file called a log file. There is more information in the Programming Tools Manual in Chapter 6 of the Microsoft Windows Software Development Kit for Windows 3.1.

CodeView for Windows (CVW.EXE)

This tool comes with the Microsoft Windows Software Development Kit for Windows version 3.1. You cannot use this tool to debug Visual Basic programs, but you can use it to debug dynamic link libraries (DLLs) used by Visual Basic. For example, you can use this tool to test the execution of your application and examine your data simultaneously.

You can isolate problems quickly because you can display any combination of variables, global or local, while you interrupt or trace an application's execution. For information on how to use CodeView for Windows, query on the following words in the Microsoft Knowledge Base:

```
codeview and visual and basic
```

Also, there is more information in the Programming Tools Manual in Chapter 4 of the Microsoft Windows Software Development Kit for Windows 3.1.

Heap Walker (HEAPWALK.EXE)

This tool comes with the Microsoft Windows Software Development Kit for Windows 3.1. Use it to test how memory is being allocated. It checks memory by examining the global heap (the system memory that the Windows operating system uses), local heaps used by active applications, and DLLs in your Windows system.

Heap Walker is useful for analyzing the effects your application has when it allocates memory from the global heap or when it creates user interface objects or graphics objects. There is more information in the Programming Tools Manual in Chapter 9 of the Microsoft Windows Software Development Kit for Windows 3.1.

Microsoft Windows SPY (SPY.EXE)

This tool comes with the Microsoft Windows Software Development Kit for Windows 3.1. Use it to test or monitor messages sent to one or more windows in Microsoft Windows and to examine the values of message parameters. For more information, see the Programming Tools Manual in Chapter 7 of the Microsoft Windows Software Development Kit for Windows 3.1.

Dynamic Data Exchange Spy (DDESPY.EXE)

This tool comes with the Microsoft Windows Software Development Kit for Windows 3.1. Use it to test or monitor dynamic data exchange messages and activity between two windows applications in the Microsoft Windows operating system. You can use DDESPY.EXE to trace DDE messages in Microsoft Windows. For more information, see the Programming Tools Manual in Chapter 8 of the Microsoft Windows Software Development Kit for Windows 3.1.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Get Control Dimensions from VBGetControlProperty

Article ID: Q104393

The information in this article applies to:

- Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

This article shows by example how to get the IPROP_STD_LEFT, IPROP_STD_TOP, IPROP_STD_WIDTH, and IPROP_STD_HEIGHT values using the VBGetControlProperty function.

MORE INFORMATION

The IPROP_STD_LEFT, IPROP_STD_TOP, IPROP_STD_WIDTH, and IPROP_STD_HEIGHT properties are stored as floats. The following code shows how to call VBGetControlProperty to get these properties from a VBX and DLL. It is assumed that the standard property indexes found in VBAPI.H were used to build the control.

```
/* VBGetControlProperty is prototyped in vbapi.h */
#include <vbapi.h>

*** You also need to add "vbapi.lib" to the libraries in the makefile. ***

 float fValue ;
 int nRet ;

  /* hctl would normally be passed in as a HCTL to the function using
     VBGetControlProperty */
  /* The third parameter must be the address of a float */
 nRet = VBGetControlProperty(hctl, IPROP_STD_TOP, &fValue) ;
```

Now fValue has the value of Top property for the hctl control.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

PRB: GP Fault if Uninitialized String Passed to API Function

Article ID: Q105807

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SYMPTOMS

When you incorrectly call a Windows API function as described in the CAUSE section further below, you can receive a general protection (GP) fault

```
Application error: VB caused a General Protection Fault in VB.EXE
at nnnn:nnnn
```

or one of the following error messages:

- Assertion failed
- Bad handle
- Bad heap block

CAUSE

Invoking a Windows API function in any of the following incorrect ways can give you a GP fault or another memory violation error:

- a passed string initialized to a value that is too short to receive the return value (See example below.)
- incorrect placement of ByVal in the Declare statement
- undefined parameters in the function declaration or invocation
- incorrect type or length of parameters in the function declaration or invocation

Windows requires you to ensure memory integrity when calling API functions.

WORKAROUND

If you get a GP fault or another memory error when calling a Windows API function, check that you have properly defined and passed all parameters.

STATUS

This behavior is by design.

MORE INFORMATION

String parameters passed from Visual Basic to Windows API functions must be initialized to at least the size of the data returned in them, or else you

may get a general protection fault.

The following example causes a GP fault by invoking the GetProfileString API function with a string initialized with a too-small value.

Steps to Reproduce Behavior
---

1. Start Visual Basic with a new, empty project.

2. Place the following correct Declare statement for GetProfileString in the General Declarations section:

```
'Enter the following Declare statement as one, single line:
Declare Function GetProfileString Lib "Kernel"
   (ByVal lpAppName As String, ByVal lpKeyName As String,
   ByVal lpDefault As String, ByVal lpReturnedString As String,
   ByVal nsize As Integer) As Integer

'NOTE: The GetProfileString function is located in the KERNAL.DLL file,
'which is usually located in the \WINDOWS\SYSTEM directory
```

3. Add a Command button to Form1, and add the following code to the Command1_Click event procedure:

```
dim size, str1 as string
str1 = ""    ' To avoid GP fault, initialize with a longer string:
             ' str1="abcdefghijklmnopqrstuvwxyz"
size = GetProfileString("intl", "sLongDate", "-1", str1, 1024)
```

   The above API function looks in the WIN.INI file under the [intl] section
   and retrieves the string after sLongDate=. The function returns a string in the address of its fourth argument. If you fail to define the fourth argument or you initialize it to a string that is smaller than the retrieved string, an assertion error occurs. If you define the fourth argument with a string that is larger than the retrieved string, the call
   will succeed. For example, change the line str1="" to str1="abcdefghijklmnopqrstuvwxyz" and the code will work.

4. Execute the code. VB.EXE will give the following error messages:

   > An error has occurred in your application. If you choose Ignore, you should save your work in a new file. If you choose Close your application will terminate. <Close> <Ignore>

   followed by:

   > Application error: VB caused a General Protection Fault in VB.EXE at 004A:0122

Windows API functions must be called with valid parameters. If you in effect tell Windows to overwrite some part of Visual Basic internal memory, this usually causes a GP fault or other memory problem. This usually ends the Visual Basic session. In Windows version 3.1, memory of other Windows applications should not be affected. But in Windows version 3.0, a GP fault

means you must restart Windows itself, thus ending all current applications in memory.

REFERENCES
==========

- "Visual Basic: Programmers Guide" for version 3.0, Chapter 24, "Calling Procedures in DLLs."

- The correct Declare statements for API functions are in the Visual Basic Professional Edition help file WIN31API.HLP, which is located in the WINAPI subdirectory of your Visual Basic directory. The WIN31API.HLP file contains function declarations, Type declarations, and the values for global constants used in the API functions.

Additional reference words: 2.00 3.00 GPF
KBCategory: APrg
KBSubcategory: APrgINI
.END:

Changing WIN.INI Printer Settings from VB using Windows API

Article ID: Q105839

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

From Visual Basic, you can call Windows API routines to change the default printer settings stored in the Windows WIN.INI file. You can also broadcast a message to all applications currently loaded in Windows to try to force them to use this WIN.INI change. However, most Windows version 3.0 and 3.1 applications are not designed to act on this broadcast message.

Applications that are started after you change WIN.INI will reflect your WIN.INI changes, as will applications that are currently loaded, if the user did not change Printer Setup in the application. But if the user changed Printer Setup in a currently loaded application, the application will ignore any changes to WIN.INI during that application's session.

There are only two ways to ensure that an application will take the changes made to the printer settings stored in WIN.INI. Either method will work:

- Exit and restart the application, or restart Windows.
- Choose options in the application Printer-Setup dialog box.

This article also describes how to add a Printer Setup dialog to a Visual Basic application that optionally changes WIN.INI, and it gives example code showing how to change the default printer in Windows.

MORE INFORMATION

The following steps change the printer settings in the WIN.INI file, and then broadcast a message to all programs currently loaded in Windows to make the change take effect:

1. Call the Windows API functions GetProfileString and WriteProfileString to change the printer setting device= in the [windows] section of the WIN.INI file to one of the printers listed in the [devices] section.

   For example, a WIN.INI file would contain the following settings to make an HP LaserJet the default printer:

   ```
   [windows]
   device=HP LaserJet IIISi PostScript,pscript,LPT1:

   [devices]
   Generic / Text Only=TTY,FILE:
   ```

HP LaserJet IIISi PostScript=pscript,LPT1:

For a detailed article that discusses how to change WIN.INI, search for the following words in the Microsoft Knowledge Base:

How to Access Windows Initialization Files Within Visual Basic

2. Call the Windows API WriteProfileString function using all NULL pointer parameters to force Windows to reload the WIN.INI file into memory. (WIN.INI is normally cached in memory and not reloaded until you restart Windows.) Pass all parameters By Value as type Long with value 0.

3. Call the SendMessage API function with hWnd% parameter set to HWND_BROADCAST (&hffff) to broadcast a message to all pop-up windows currently loaded in the system. Setting the wMsg% parameter to WM_WININICHANGE notifies all top-level windows of a WIN.INI change, and WM_DEVMODECHANGE notifies them of a device-mode change.

However, if you changed settings in the Printer-Setup dialog box of a loaded application earlier in this session of Windows, most applications ignore the SendMessage broadcast. By design, most Windows-based applications ignore this message, as explained in the Notepad example given below.

Example of How Notepad Uses WIN.INI Printer Settings

Under Windows, you can change default printer settings in the Printers section of the Control Panel program. This writes changes to the WIN.INI file on disk and in memory. Many other applications, such as Microsoft Word, also write changes to the WIN.INI file.

When Notepad starts, a global variable of type PRINTDLG provides the structure to initialize the Print dialog box. One of the members of that structure is hDevNames. It contains three strings that specify the driver name, printer name, and output port name. When Notepad starts, these three strings start with a NULL value. This tells Notepad to get its printer device context (DC) from the WIN.INI file.

If you choose Print Setup from within NotePad and make changes, NotePad will continue using those changes for the remaining NotePad session, and the three strings in hDevNames will no longer all be NULL. That session of NotePad will no longer look in the WIN.INI file, so it will ignore any WM_WININICHANGE and WM_DEVMODECHANGE messages. Many Windows-based applications work in this manner. Internally, they process only certain messages, and they pass all unrecognized messages to the default API DefWindowProc function, which does nothing.

Because you cannot rely on an application processing WM_WININICHANGE and WM_DEVMODECHANGE messages, an application such as Visual Basic cannot force the updated WIN.INI modifications onto another loaded application by sending Windows messages. To change printer parameters to those changed in the WIN.INI file, you must use one of these two techniques:

- Exit and restart the application, or restart Windows.
- Use the application's Printer-Setup dialog box to set the parameters.

Adding Printer Setup to a Visual Basic Application

To add a Printer-Setup dialog to a Visual Basic application, use the Common Dialog printer control provided with the following products:

- Visual Basic version 1.0 Professional Toolkit for Windows
- Professional Edition of Visual Basic version 2.0 for Windows
- Standard or Professional Edition of Visual Basic version 3.0 for Windows

Setting the PrinterDefault property to True writes any Printer Setup changes
to the WIN.INI file:

```
CMDialog1.PrinterDefault = True
```

You can use the Flags property of the Common Dialog printer control to specify various options, as described on page 208 of "Visual Basic 3.0: Language Reference." For example, you can have a print dialog with a button for Printer Setup. Or, you can give the Printer Setup its own dialog box by setting the Flags property to PD_PRINTSETUP as follows:

```
CMDialog1.Flags = PD_PRINTSETUP  ' PD_PRINTSETUP = &H40&
CMDialog1.Action = 5     ' Displays Printer Dialog for Printer Setup
```

To change printer settings from a Visual Basic application without user interaction, call a DLL written in C that calls the Windows API ExtDeviceMode function. Because Visual Basic does not support function pointers, you cannot call the ExtDeviceMode function directly from Visual Basic. A Windows-compatible C compiler is required to create a Windows DLL.

Code Example to Change Windows Default Printer in WIN.INI

The following program demonstrates how to change the default printer in the WIN.INI file by using Visual Basic code:

1. In Visual Basic, place a list box (List1) and a command button (Command1) on Form1.

2. Set the Caption property of Command1 to Set Default Printer.

3. Add the following code and three subprograms to the General Declarations section of Form1:

```
Option Explicit
' Enter each Declare statement on one, single line:
Declare Function GetProfileString Lib "Kernel"
   (ByVal lpAppName As String, ByVal lpKeyName As Any,
   ByVal lpDefault As String, ByVal lpReturnedString As String,
   ByVal nSize As Integer) As Integer
Declare Function WriteProfileString Lib "Kernel"
   (ByVal lpApplicationName As String, ByVal lpKeyName As Any,
   ByVal lpString As Any) As Integer
Declare Function SendMessage Lib "User" (ByVal hWnd As Integer,
   ByVal wMsg As Integer, ByVal wParam As Integer,
```

```
   lParam As Any) As Long
Const WM_WININICHANGE = &H1A
Const HWND_BROADCAST = &HFFFF

' Enter the following two lines as one, single line:
Sub GetDriverAndPort (ByVal Buffer As String, DriverName As String,
   PrinterPort As String)
   Dim r As Integer
   Dim iDriver As Integer
   Dim iPort As Integer
   DriverName = ""
   PrinterPort = ""

   'The driver name is first in the string terminated by a comma
   iDriver = InStr(Buffer, ",")
   If iDriver > 0 Then

      'Strip out the driver name
      DriverName = Left(Buffer, iDriver - 1)

      'The port name is the second entry after the driver name
      'separated by commas.
      iPort = InStr(iDriver + 1, Buffer, ",")

      If iPort > 0 Then
         'Strip out the port name
         PrinterPort = Mid(Buffer, iDriver + 1, iPort - iDriver - 1)
      End If
   End If
End Sub

Sub ParseList (lstCtl As Control, ByVal Buffer As String)
   Dim i As Integer
   Do
      i = InStr(Buffer, Chr(0))
      If i > 0 Then
         lstCtl.AddItem Left(Buffer, i - 1)
         Buffer = Mid(Buffer, i + 1)
      Else
         lstCtl.AddItem Buffer
         Buffer = ""
      End If
   Loop While i > 0
End Sub

' Enter the following two lines as one, single line:
Sub SetDefaultPrinter (ByVal PrinterName As String,
   ByVal DriverName As String, ByVal PrinterPort As String)
   Dim DeviceLine As String
   Dim r As Integer
   Dim l As Long
   DeviceLine = PrinterName & "," & DriverName & "," & PrinterPort
   ' Store the new printer information in the [WINDOWS] section of
   ' the WIN.INI file for the DEVICE= item
   r = WriteProfileString("windows", "Device", DeviceLine)
   ' Cause all applications to reload the INI file:
   l = SendMessage(HWND_BROADCAST, WM_WININICHANGE, 0, ByVal "windows")
```

```
   End Sub
```

4. Add the following code to the Command1_Click event procedure:

```
   Dim r As Integer
   Dim Buffer As String
   Dim DeviceName As String
   Dim DriverName As String
   Dim PrinterPort As String
   Dim PrinterName As String
   If List1.ListIndex > -1 Then
      'Get the printer information for the currently selected printer
      'in the list. The information is taken from the WIN.INI file.
      Buffer = Space(1024)
      PrinterName = List1.Text
      r=GetProfileString("PrinterPorts",PrinterName,"",Buffer,Len(Buffer))

      'Parse the driver name and port name out of the buffer
      GetDriverAndPort Buffer, DriverName, PrinterPort

      If DriverName <> "" And PrinterPort <> "" Then
         SetDefaultPrinter List1.Text, DriverName, PrinterPort
      End If
   End If
```

5. Add the following code to Form_Load event procedure:

```
   Dim r As Integer
   Dim Buffer As String

   'Get the list of available printers from WIN.INI
   Buffer = Space(8192)
   r = GetProfileString("PrinterPorts",ByVal 0&,"",Buffer,Len(Buffer))

   'Display the list of printer in the list box List1
   ParseList List1, Buffer
```

6. Run the program. The list box will display the printer choices from the WIN.INI file. By clicking the command button, you will set the default printer in the WIN.INI file.

REFERENCES
==========

"Microsoft Windows Programmer's Reference," Chapters 4 and 6, Microsoft Press, 1990.

Additional reference words: 2.00 3.00
KBCategory: APrg
KBSubcategory: APrgPrint

How to Create a Screen Saver in Visual Basic

Article ID: Q106239

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

You can create a Windows screen saver with Visual Basic by following the guidelines listed below.

MORE INFORMATION

Follow these guidelines when creating a Windows screen saver with Visual Basic:

- In the File Make EXE File dialog, insert the string SCRNSAVE: (in upper case) at the beginning of the Application Title. For example:

  SCRNSAVE:Flying Fish.

- In the File Make EXE File dialog, specify the program file name extension as .SCR instead of .EXE.

- Locate the .SCR program file in the \WINDOWS directory.

- Give your form the following property settings so that it occupies the entire screen and does not have a title bar:

```
Caption     = ""     (no caption)
ControlBox  = False
MaxButton   = False
MinButton   = False
WindowState = 2      (maximized)
```

- Place code in all MouseMove, MouseDown, and KeyDown event handlers that exit the program. Because Visual Basic may invoke the MouseMove event when the form is first loaded, you must write code to ignore the first MouseMove event. For example:

```
Sub Form_MouseMove (...)
   Static once As Integer
   If once Then
      End
   Else
      once = True
   End If
End Sub

Sub Form_MouseDown (...)
```

```
   End
End Sub

Sub Form_KeyDown (...)
   End
End Sub
```

Windows usually launches the screen saver program multiple times. To prevent more than one copy of your screen saver from running, add the following statements to the Form_Load event handler (or Sub Main, if used):

```
If App.PrevInstance Then
   SaveTitle$ = App.Title
   App.Title = "... duplicate instance."
   Form1.Caption = "... duplicate instance."
   AppActivate SaveTitle$
   SendKeys "% R", True
   End
End If
```

Windows takes care of launching. It keeps track of system idle time and launches the screen saver program.

You can use a timer control to periodically draw graphics on the form.

Screen savers are selected and configured from Windows Control Panel in the Desktop dialog. The screen saver section of this dialog has a button labeled Setup that invokes the screen saver program with the command line option /c. When your program is invoked with this option, you can display a configuration form to allow the user to select settings such as speed, number of objects, colors, and so on. Detect the /c command line parameter by checking the Command$ function. For example:

```
Sub Form_Load ()
   If Command$ = "/c" Then
      frmConfig.Show   ' display configuration form
      Unload Me        ' bypass regular form
   End If
End Sub
```

When Windows launches the screen saver, it usually specifies the command line option /s.

You may also want your program to appear on top of all other windows by making it a TOPMOST window.

For more information, use the following words to query in the Microsoft Knowledge Base:

```
TOPMOST and SETWINDOWPOS
```

Also, you can find two example programs and a complete explanation showing how to write your own screen savers in Visual Basic in the following book:

"Visual Basic Workshop 3.0" by John C. Craig, published by Microsoft Press.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgOther

How to Write C DLLs and Call Them from Visual Basic

Article ID: Q106553

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article outlines how to use DLLs with Visual Basic. It covers the following issues:

Section A

1.0 What Is a DLL
1.1 Why Use a DLL
1.2 Anatomy of a DLL
1.3 DLL Memory Management Issues
1.4 Building a DLL Using Visual C++
1.5 Example C DLL

Section B

2.0 Calling DLLs from Visual Basic
2.1 DLL Parameters
2.2 Trouble Shooting
2.3 Example Visual Basic Calling Program

MORE INFORMATION

SECTION A

1.0 What Is a DLL

DLLs (Dynamic Link Libraries) are an important aspect of Windows. A DLL contains functions that your executable program can call during execution. In other words, a DLL is a library of functions that your program can link with dynamically.

A link can be static or dynamic. Static links don't change. All the address information needed by your program to access the library function is fixed when the executable file is created and remains unchanged during execution.

Dynamic links are created as needed. When your program needs a function that is not in the executable file, Windows loads the dynamic link library (the DLL), making all of its functions available to your application. At that time, Windows resolves the address of each function and dynamically

links it to your application.

All Custom controls used in Visual Basic are DLLs. The only difference is that they require special handling in terms of messages received from Visual Basic.

1.1 Why Use DLLs

Here are four reasons why you might want to use a DLL:

- Access to C Run-Time Functions:

  The C run-time library has many useful functions that would not be available to Visual Basic programmers were it not for DLLs. For example, the _dos_getdiskfree function allows you to calculate the total amount of disk space and the free disk space available on a drive.

- Access to Windows API (Application Programming Interface) Functions that Require Callback Routines:

  Some Windows API functions require a callback function. A callback function is a function that Windows will call while executing the API call. An example of this sort of function is EnumTaskWindows, which will give the handle of all windows that are owned by a particular task.

- Speed:

  C is a fully compiled language that works at a level that is fairly close to native machine code. This means that the execution of programs that are well written in C will be fast.

- Load on Use:

  Code and data from a DLL are loaded only when needed. A DLL can be organized such that only required parts are loaded as opposed to the entire DLL. This reduces the amount of memory required and the time taken to load.

1.2 Anatomy of a DLL

Every DLL must contain a LibMain function and should contain a Windows Exit Procedure (WEP) in addition to the exported functions that can be called by an executable program.

- LibMain:

  A DLL must contain the LibMain function. The LibMain function is called by the system to initialize the DLL. LibMain is called only once -- when the first program that requires the DLL is loaded. The following are the parameters passed to LibMain:

  - HANDLE : Handle to the instance of the DLL.
  - WORD : Library's data segment.
  - WORD : Heap size.
  - LPSTR : Command line parameters.

- WEP:

  The WEP (Windows Exit Procedure) performs cleanup for a DLL before the library is unloaded. Although a WEP function was required for every DLL in previous versions of the Windows operating system, for version 3.1 it is optional. A WEP should be included in the module definition file (.DEF) in Visual C, for example:

  ```
  EXPORTS
     WEP
  ```

- Exported Functions:

  These are the functions you want to call from your DLL. They are denoted by _export. _export is used for backward compatibility. All the functions you want to call must also be listed in the (.DEF) file of your DLL.

1.3 DLL Memory management issues

Use the large memory model.

C stores all variables defined as static or global (defined outside of a function) in the program's heap space, and C stores all other variables on the stack.

In the small and medium model, all pointers are near by default. This means that the data is accessed by 16-bit offsets to either the data segment (DS) register, or the stack segment (SS) register. Unfortunately, the compiler has no way of knowing whether the offset is from the DS or the SS. In most programs this would not be a problem because the DS and SS point to the same segment. A DLL, however, is a special case.

A DLL has its own data segment but shares its stack with the calling program. This means that the DS and the SS do not point to the same location. The easiest solution to this problem is to build the DLL in the large memory model where all variables are referenced by a 32-bit value.

Why Allocate Memory Dynamically?

Allocating memory dynamically is a Windows-friendly technique. Declaring large arrays of data takes up space in either your program's stack, which is limited to 64K, or you program's Data Segment, which wastes disk space and Windows memory. It is better to ask Windows for the memory when you need it, and then free it when you have finished.

Allocating Memory

In Windows, you can dynamically allocate two types of memory, local and global. Local memory is limited to 64K, and in the case of a DLL, local memory is shared with the program that called the DLL. Global memory is all of the memory available to Windows after it has loaded.

Local memory is allocated and managed using the LocalAlloc, LocalLock LocalUnlock, and LocalFree functions -- as in this example:

```
char* pszBuffer;
....
pszBuffer = (char *) LocalAlloc (LPTR, 20);
...
LocalFree (pszBuffer);
```

It is faster to allocate local memory than it is to allocate global memory. But allocations from the local heap are limited to 64K, which must be shared amongst all programs that are calling the DLL. It is best to use local memory when small, short lived blocks of memory are required.

Global memory is allocated and managed using the GlobalAlloc, GlobalLock GlobalUnlock, and GlobalFree functions -- as in this example:

```
HGLOBAL hglb;
char* pszBuffer;

hglb = GlobalAlloc (GHND, 2048);
   // GHND allocates the memory as moveable and
   // initialized to 0
   // 2048 is the amount of memory to be allocated...
pszBuffer = GlobalLock (hglb);
...
GlobalUnlock (hglb);
GlobalFree (hglb);
```

The GlobalAlloc function allocates memory in multiples of 4K.

If you want to share memory allocated in the DLL with other programs, you should allocate it using the GMEM_SHARED flag. If you want to share the memory through DDE, you must allocate it by using the GMEM_DDESHARE flag.

Be Careful When Storing Data in Static Variables

If you try to store data in a DLL using global or static variables, don't be surprised if these values have changed when you next call your DLL. The data stored in this way will be common to all applications that access this DLL. No matter how many applications use a DLL, there is only one instance of the DLL. The best way to get around this is to return structures from the DLL and pass them in again when they are needed.

File Handles

It is not possible to share file handles between applications or DLLs. Each application has its own file-handle table. For two applications to use the same file using a DLL, they must both open the file individually.

1.4 Building a DLL Using Visual C++

Here are the steps necessary to build a DLL using Visual C++:

1. Start Visual C++.

2. Create a new project by choosing New from the Project menu. Select the following options:

   - Set the Project Type to "Windows dynamic-link library (.DLL)"
   - Clear the "Use Microsoft Foundation Classes" check box.

   You can also set or view these options later by choosing Project from the Options menu.

3. Add your existing .C and .DEF files to the project by using the dialog box that comes up when you choose Edit from the Project menu. Or enter your code directly in the Visual C++ editing window. (See the .C and .DEF example code listed below.)

4. From the Project menu, choose the Build <yourname>.DLL option.

1.5 Example C DLL

The following DLL contains the GetDiskInfo function, which can be called from Visual Basic. It will return the disk space available, the current drive name and the volume name.

C Code Example, DISKINFO.C:

```
#include <windows.h>
#include <dos.h>

int CALLBACK LibMain (HANDLE hInstance, WORD wDataSeg, WORD wHeapSize,
LPSTR lpszCmdLine)
{
   if (wHeapSize > 0)
      UnlockData (0);  //Unlocks the data segment of the library.
   return 1;
}

void __export CALLBACK GetDiskInfo (char *cDrive, char *szVolumeName,
unsigned long *ulFreeSpace)
{
   unsigned drive;
   struct _diskfree_t driveinfo;
   struct _find_t c_file;

   _dos_getdrive (&drive);
   _dos_getdiskfree( drive, &driveinfo );

   if (!_dos_findfirst( "*.*", _A_VOLID, &c_file ))
      wsprintf( szVolumeName, "%s", c_file.name);
   else
      wsprintf ( szVolumeName, "NO LABEL");

   *cDrive = drive + 'A' -1;

   *ulFreeSpace = (unsigned long) driveinfo.avail_clusters * (unsigned
      long) driveinfo.sectors_per_cluster * (unsigned long)
```

```
      driveinfo.bytes_per_sector;
}
```

Use the following DISKINFO.DEF file in Visual C++:

```
   LIBRARY  diskinfo
   DESCRIPTION 'GetDiskInfo Can be called from Visual Basic'
   EXETYPE WINDOWS 3.1
   CODE PRELOAD MOVEABLE DISCARDABLE
   DATA PRELOAD MOVEABLE SINGLE
   HEAPSIZE    4096
   EXPORTS
      GetDiskInfo @1
```

NOTE: The LIBRARY name in the .DEF file must be the same as the DLL file name, or else Visual Basic will give you "Error in loading DLL." For example, create the file DISKINFO.DLL using the LIBRARY DISKINFO statement in the .DEF file above.

SECTION B

2.0 Calling DLLs from Visual Basic

In Visual Basic, all functions, including DLL functions, that you want to call must first be declared by using the Declare statement. You can declare your functions in the declarations section of a Form or a Module. If you declare a DLL procedure or function in a Form, it is private to that Form. To make it public, you must declare it in a Module. The following is an example Declare statement:

```
   Declare Sub getdiskinfo Lib "c:\somepath\diskinfo.dll"
      (ByVal mydrive As String, ByVal myvolume As String, free As Long)
```

You must enter the entire Declare statement as one, single line. This particular Declare statement declares the user-defined procedure GETDISKINFO located in user-created DISKINFO.DLL file.

Once you declare the function, you can call and use the function just as you would call and use a Visual Basic function.

2.1 DLL Parameters

Because DLLs are typically written in C, DLLs can use a wide variety of parameters not directly supported by Visual Basic. As a result, when passing parameters, he programmer has to find the appropriate data type to pass.

Passing Arguments by Value or by Reference

By default, Visual Basic passes all arguments by reference. (When passing by reference, Visual Basic supplies a 32-bit far address.) However, many DLL functions expect an argument to be passed by value. This can be achieved by placing the ByVal keyword in front of the argument declaration.

The following sections show you how to convert parameters to Visual Basic.

8- to 16-Bit Numeric Parameters

Pass 8- to 16-bit numeric parameters (int, short, unsigned int, unsigned short, BOOL, and WORD) as Integer.

32-bit Numeric Parameters

Pass 32-bit numeric parameters (long, unsigned long, and DWORD) as LONG.

32-Bit Signed Integer Parameters

Pass 32-bit signed integer parameters as Currency or Double.

Object Handles

All handles are unique 16-bit integer values associated with a Window and are passed by value, so pass these parameters as Integer.

Strings

Strings include the LPSTR and LPBYTE data types (pointer to characters or pointer to unsigned characters). Pass these parameters as (ByVal paramname As String). DLL functions cannot return Visual Basic strings. They do sometimes return LPSTRs, which can be copied into Visual Basic strings by using API functions.

To pass Visual Basic strings directly, pass them as (param As String).

NOTE: Visual Basic strings require special handling, so don't pass strings directly unless the DLL explicitly requires it.

Pointers to Numeric Values

Pass pointers to numeric values by simply not using the ByVal keyword.

Structures

If the Visual Basic user-defined type matches the structure expected by the DLL, the structure can be passed by reference.

NOTE: Structures cannot be passed by value.

Pointers to Arrays

Pass the first element of the array by reference.

Pointers to functions

Visual Basic does not support callback functions, so DLL functions that have pointers to functions cannot be used with Visual Basic.

Null Pointers

If a DLL expects a Null pointer, pass it as (ByVal paramname As Any). You can use &0 or &0H as the value of paramname.

2.2 Trouble Shooting

Below are solutions to some problems you may encounter.

System Resources Keep Getting Lower After the DLL Is Called

If your DLL is using GDI objects, you must remember to free them after using them. This may not be obvious in Visual Basic, but when using the Windows SDK (software development kit) if you create a GDI object (for example, CreateBrushIndirect), you must delete it by using DeleteObject later on.

Bad DLL Calling Convention Error

This error is often caused by incorrectly omitting or including the ByVal keyword from the Declare statement. This error can also be caused if the wrong parameters are passed.

Error in loading DLL

This error occurs when you call a dynamic-link library procedure and the file specified in the procedure's Declare statement cannot be loaded. You can use the Microsoft Windows API function LoadLibrary to find out more specific information about why a DLL fails to load.

General Protection (GP) Fault

GP faults occur when your program writes to a block of memory that doesn't belong to it. The two most likely reasons for this are:

- You overstepped an array boundary. C does not check that the array subscript you are writing to is valid. Therefore, you can easily write to memory you don't own.

- You are using a pointer to a memory location that you have freed. The best option is to assign NULL to all pointers after you free their memory.

A GP fault can also occur when an incorrect variable type is passed to the DLL function.

2.3 Example Visual Basic Calling Program

There are two parts to calling a DLL in a Visual Basic program. First you declare the function, and then you use it in event code.

Here is an example of a Declare statement. The Declare statement should be put in a module or in a form's General Declarations section.

```
   ' Enter the following Declare as one, single line:
   Declare Sub getdiskinfo Lib "c:\dllartic\diskinfo.dll"
      (ByVal mydrive As String, ByVal myvolume As String, free As Long)
```

Specify ByVal statements exactly as shown, or else a GP fault may occur.

Once the function is declared, you can use it in event code. The following example uses a function from the DLL in the Command1 Click event code:

```
Sub Command1_Click ()
   Dim drive As String * 1
   Dim volume As String * 20
   Dim free As Long
   Call getdiskinfo(drive, volume, free)
   Text1.Text = drive
   Text2.Text = volume
   Text3.Text = Str$(free)
End Sub
```

Additional reference words: 3.00
KBCategory:
KBSubCategory: APrgOther RefsDoc

How to Pass User-Defined Structure Containing Strings to DLL

Article ID: Q107750

The information in this article applies to:

- Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This articles shows by example how to pass a user-defined structure that contains strings to a DLL. The example enables a DLL to read and write the strings in a user-defined structure.

MORE INFORMATION

The following step-by-step example passes a user-defined structure that contains strings to a DLL to manipulate.

1. Start a new project in Visual Basic.

2. From the File menu, choose New Module (ALT F M). MODULE1.BAS will be created by default. Add the following code to the .BAS module:

```
' Fixed-length string elements of a structure are packed in memory
' as are other values in Visual Basic. The following structure takes up
' 16 bytes of memory:
'
Type MYSTRINGSTRUCT
   str1 As String * 8
   str2 As String * 8
End Type
' Enter the following Declare statement as one, single line
Declare Sub MyStructProc Lib "Name of DLL your create"
   (lpStringStruct As MYSTRINGSTRUCT)
```

3. Add a command button (Command1) to Form1.

4. Add the following code to the Command1_Click event of Form1:

```
Sub Command1_Click ()
Dim StringStruct As MYSTRINGSTRUCT
   StringStruct.str1 = "str1"
   StringStruct.str2 = "str2"
   MyStructProc StringStruct
   TEXT1.Text = StringStruct.str1
   TEXT2.Text = StringStruct.str2
End Sub
```

5. Add two text controls (Text1 and Text2) to Form1.

6. Create the C code needed to make the DLL. In the .h file of the DLL a user-defined type will create a mirror image of the type you defined in

the Visual Basic .BAS file. Char str[8] is equivalent to Visual Basic declaration of str1 as String * 8. This structure definition takes up 16 bytes in memory as does the Visual Basic structure definition.

```
typedef struct STRINGSTRUCT{
char str1[8] ;
char str2[8] ;
} FAR * LPSTRINGSTRUCT ;

/* Declaration of the function */
void FAR PASCAL MyStructProc(LPSTRINGSTRUCT) ;
```

7. Add the following code to your .c file:

```
#include "The .h file where you added the code above"

void FAR PASCAL MyStructProc(LPSTRINGSTRUCT lpStringStruct)
{
/* You need to use lstrcpyn because the structure from Visual
Basic is packed, and the strings are not Null terminated. The way
structures are passed from Visual Basic to a DLL is fully described
beginning on page 566 in the Visual Basic version 3.0 for Windows
"Programmers Guide," Chapter 24, "Calling Procedures in DLLs," in
"User-Defined Types" under "Calling DLL Procedures with Specific Data
Types." */

   lstrcpyn(lpStringStruct->str1, "change11", 8) ;
   lstrcpyn(lpStringStruct->str2, "change22", 8) ;
}
```

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgOther

PRB: Printer.FontSize Return Value Is Not Requested Value

Article ID: Q108073

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

When you set the FontSize property of the Printer object in Visual Basic, the font size you get may differ from the size you requested. This behavior occurs with both Postscript printer fonts and Truetype fonts.

CAUSE

This behavior occurs because Windows doesn't store the requested point size. Instead, Windows stores the closest character cell height that is supported on the current printer device. The character cell height is rounded to an integer number of pixels (dots). Rounding is necessary because Windows doesn't support a fractional device unit such as half a dot (pixel).

STATUS

This behavior is by design.

MORE INFORMATION

As an example, if your printer prints at 96 dots per inch (DPI) and you request a 10 point font, Windows creates a logical printer font that is exactly 13 dots high:

```
13 dots = CInt( (10 points * 96 DPI) / (72 points per inch) )
```

The above CInt function rounds to the nearest integer. The conversion depends on the DPI resolution of your printer.

When you set the FontSize property to 10 points, the FontSize will actually be set to 9.75 points, which is the nearest point size the printer can support at 96 DPI:

```
9.75 points = 13 dots * (72 points per inch) / (96 DPI)
```

On 300 DPI printers, the minimum interval between supported font sizes is 0.24 points. You get 0.24 points per dot as a result of the following formula:

```
(72 points per inch) / (300 DPI)
```

On 600 DPI printers, such as with an HP LaserJet 4 driver, supported font sizes are at intervals of 0.12 points.

In typesetting, a point is 1/72 of an inch. The height of fonts is usually expressed in points.

Windows automatically maps the requested font or font size to the nearest one supported by the screen or printer device if that font or size does not exist on that device.

Steps to Reproduce Behavior

As an example, the Hewlett-Packard (HP) LaserJet with the HP PostScript cartridge supports point sizes down to a resolution of 0.25 point. In Visual Basic, you can set the Printer.FontSize property to a desired point size. You can set the Printer.FontName property to the PostScript font. However, the Printer.FontSize property displays font sizes at 0.24 point intervals instead of 0.25 points. You get font sizes such as 9.6, 9.84, 10.08, 10.32, 10.56, 10.8, 11.04, and so forth. Setting the font size to 11.0 points actually gives you a font with 10.8 points.

The following steps reproduce this behavior:

1. In the Windows Control Panel, select a printer that uses a 300 DPI driver, such as on the HP LaserJet IIIsi. Printers with different DPI settings will return fonts in different increments.

2. Start a new project in Visual Basic. Form1 is created by default.

3. Add the following to the Form Load event procedure:

```
Sub Form_Load ()
   For j = 9 To 12 Step .25
   Printer.FontSize = j  'Set printer font size in increments of .25
   Debug.Print Printer.FontSize
   Next
End Sub
```

4. Start the program (or press F5). Choose Debug from the Window menu. Although you requested fonts in increments of 0.25 points, you instead get fonts in increments of 0.24 points:

   9.36, 9.6, 9.84, 10.08, 10.32, 10.56, 10.8, 11.04, 11.28, 11.52, 12

Additional reference words: 3.00 H-P laser jet HPLJ PS Apple LaserWriter
KBCategory: APrg
KBSubcategory: APrgPrint

Category Keywords for All Visual Basic KB Articles

Article ID: Q108753

The information in this article applies to:

- Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Each article in the Visual Basic for Windows collection contains at least one keyword (called a KBSubcategory keyword) that places the article in an appropriate category. This article lists all the KBSubcategory keywords.

MORE INFORMATION

Category & Subcategory Description	KBSubcategory Keyword
Setup / Installation (Setins)	Setins
Environment-specific Issues (Envt)	
VB Design Environment	EnvtDes
Run-Time Environment	EnvtRun
Programming (Prg)	
Visual Basic Forms and Controls	
Standard Controls / Forms	PrgCtrlsStd
Custom Controls	PrgCtrlsCus
Third-Party Controls	PrgCtrlsThird
Optimization	
Memory Management	PrgOptMemMgt
General Optimization Tips	PrgOptTips
General VB Programming	PrgOther
Advanced programming (APrg)	
Network	APrgNet
Windows Programming (APIs / DLLs)	
Printing	APrgPrint
Graphics	APrgGrap
Windowing	APrgWindow
INI Files	APrgINI
Other API / DLL Programming	APrgOther
Data Access	
ODBC	APrgDataODBC
IISAM	APrgDataIISAM
Access	APrgDataAcc
General Database Programming	APrgDataOther
3rd Party DLL's	APrgThirdDLL

Inter-Application Programmability (IAP)

OLE	IAPOLE
DDE	IAPDDE
3rd Party Interoperability	IAPThird

Tools (Tls)

Setup Toolkit / Wizard	TlsSetWiz
Control Development Kit (CDK)	TlsCDK
Help Compiler (HC)	TlsHC

References (Refs)

Documentation / Help File Fixes	RefsDoc
Product Information	RefsProd
Third-Party Information	RefsThird
PSS-Only Information	RefsPSS

Using Keywords to Query the KB

At Microsoft, we use the subcategory keywords to organize the articles for Help files and for the FastTips Catalog. You can use them to query the Microsoft Knowledge Base for Visual Basic articles that apply to that category or subcategory. For example, you can find all the general database programming articles by querying on the following words in the Microsoft Knowledge Base:

```
visual and basic and APrgDataOther
```

Use the asterisk (*) wildcard to find articles that fall into the general categories or into an intermediate subcategory. The first element in each keyword is the category. For example, to find all the articles that apply to Visual Basic Forms and Controls regardless of whether they are standard, custom, or third-party controls, use the following words to query the Microsoft Knowledge Base:

```
visual and basic and PrgCtrls*
```

To find all advanced programming articles, query on these words:

```
visual and basic and APrg*
```

Add KBSubcategory Keyword to Each Article

When contributing an article to the Visual Basic Knowledge Base, add the appropriate KBSubcategory keyword to the bottom of the article on the KBSubcategory line. Each article in the Visual Basic for Windows collection contains the following section at the bottom of the article:

```
Additional reference words:
KBCategory:
KBSubcategory: <keyword>
```

An article usually has only one subcategory keyword, but it may have more.

If you are interested in contributing, please obtain the guidelines by

querying on the following words in the Microsoft Knowledge Base:

   visual and basic and kbguide and kbartwrite

Additional reference words: 3.00 dskbguide subcatkey
KBCategory:
KBSubcategory: RefsPSS

Popular Windows API Functions Used from Visual Basic 3.0

Article ID: Q109290

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

Below is a summary of Application Programmer's Interface (API) functions for Microsoft Windows that programmers commonly use to extend the abilities of Visual Basic.

MORE INFORMATION

Commonly-Used API Functions for Windows

The following Windows API functions are very useful for Visual Basic programmers:

- BitBlt: Move a bitmap from a source device context to a destination.
- ExtractIcon, DrawIcon, and LoadIcon: Manipulate icons.
- FindExecutable: Find and retrieve the executable filename that is associated with a specified filename.
- GetWindowsDirectory: Get the pathname of the Windows directory.
- GetSystemDirectory: Get the pathname of the Windows system subdirectory.
- GetSystemMetrics: Get widths and heights of the display elements of Microsoft Windows.
- GetTempFileName: Return a temporary filename and path using the TEMP environment variable.
- GetWindowPlacement and SetWindowPlacement: Get or set the show state and the normal (restored), minimized, and maximized positions of a window.
- GetProfileString, GetProfileInt, SetProfileString: Get or set the information stored in the initialization file for Windows (WIN.INI).
- GetPrivateProfileString, GetPrivateProfileInt, SetPrivateProfileString: Get or set the information in a given initialization file (ininame.INI).
- SendMessage: Send Windows messages to control applications. For example, the LB_SETTABSTOPS message sets tab stops in a list box. LB_FINDSTRING finds the first string in the list box which matches prefix text. Hundreds of other messages are available.

- SetCapture: Send all mouse input to the specified window, regardless of the cursor position.

- SetWindowPos: Changes the size, position, and ordering of child, pop-up, and top-level windows.

Additional Commonly-Used API Functions for Windows

The following API functions for Windows are also very useful for Visual Basic programmers:

- BringWindowToTop, SetActiveWindow: Activate a window.

- CreateCompatibleDC: Prepare image in memory, such as before copying an image to the compatible device.

- DeleteObject, ReleaseDC: Remove object or device context from memory.

- DragAcceptFiles, DragFinish: Support File Manager drag/drop file ability.

- FindWindow, ShowWindow: Check to see if given applications are currently running. This is useful before you perform Dynamic Data Exchange (DDE).

- GetActiveWindow, IsWindow: Find out when a Shell function has finished loading a program.

- GetDesktopWindow: Get a handle to the Windows desktop window, which covers the entire screen and is the area on top of which all icons and other windows are painted.

- GetFreeSpace, GetVersion, GetWinFlags: Check system settings for Microsoft Windows, such as for reporting in an About box under a Help menu.

- GetModuleFileName, GetModuleHandle, GetModuleUsage: Get full pathname of the executable file from which the specified module was loaded.

- GetPaletteEntries, CreatePen, SelectObject: Manipulate color palettes.

- GetParent: Get handle of specified window's parent.

- EnumChildWindows: Get list of child windows that belong to specified parent window.

- GetWindowLong, SetWindowLong: Get or set window style information.

- GetWindowText: Get the caption title of a window or the text in a control.

- IsAppLoaded, IsIconic, IsWindowVisible: Find the state of the windows of an application -- visible, loaded, or minimized to an icon.

- LoadCursor, DestroyCursor, GetCursorPos: Handle different mouse cursors.

- LZOpenFile, LZCopy, LZClose: Manipulate compressed files. See the LZEXPAND.DLL file that ships with Visual Basic version 3.0. The functions in LZEXPAND.DLL manipulate files that compressed by the COMPRESS.EXE utility supplied with the Windows Software Development Kit (SDK) versions 3.0 and 3.1.

- OpenComm, WriteComm, GetCommEventMask, SetCommState: Use the COMn: serial communications port.

- RoundRect, FillRect, ExtFloodFill, StretchBlt: Perform graphic operations beyond Visual Basic capabilities.

- SetSysModalWindow: Set a window to be system-modal, such as for a screen saver's password dialog box.

- SndPlaySound: Play waveform .WAV sound file sounds.

- WinExec: Run a Windows-based or non-Windows application, as an alternative to Basic's Shell function.

- WinHelp: Invoke WINHELP.EXE, the Windows Help application. Useful as an alternative to invoking WINHELP.EXE by setting the Common Dialog's Action property to a value of 6.

Visual Basic Setup Kit API Routines

- DiskSpaceFree: Get free space on specified disk. The Declare statement is found in SETUP1.GLB in the SETUPKIT subdirectory for Visual Basic:

  ```
  Declare Function DiskSpaceFree Lib "SETUPKIT.DLL" () As Long
  ```

- GetFileVersionInfoSize and GetFileVersionInfo in Windows API: Get file version information from the version-information resource that was added to an application using the VERSIONINFO statement.

  The VERSIONINFO statement is found in the file installation library found in Windows version 3.1. The resource contains such information about the file as its version number, its intended operating system, and its original filename. The resource is intended to be used with the Windows file installation library functions.

  The GetFileVersion function is defined in SETUP1.BAS in the SETUPKIT subdirectory for Visual Basic. The GetFileVersion function invokes the GetFileVersionInfo and GetFileVersionInfoSize Windows API functions. The following Declare statements are taken from SETUP1.GLB:

  ```
  Declare Function GetFileVersionInfoSize Lib "VER.DLL"
     (ByVal lpszFileName As String, lpdwHandle As Long) As Long

  Declare Function GetFileVersionInfo Lib "VER.DLL"
     (ByVal lpszFileName As String, ByVal lpdwHandle As Long,
     ByVal cbbuf As Long, ByVal lpvdata As String) As Integer
  ```

REFERENCES

- "Microsoft Windows Programmer's Reference," published by Microsoft Press.

- "Visual Basic - Game Programming with Windows," by Craig, published by Microsoft Press.

- "PC Magazine's Visual Basic Programmer's Guide to the Windows API" by Daniel Appleman (of Desaware), published by Ziff-Davis Press. This reference describes most Windows API functions that can be used from within Visual Basic.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgOther

How to Invoke MessageBeep API to Play System Alert .WAV Sounds

Article ID: Q110103

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0
- Microsoft Windows version 3.1

SUMMARY

This article describes how to invoke the MessageBeep API function to play the waveform sound associated with a given Windows system alert level.

This is useful for playing a sound such as when you display a message box with the MsgBox statement.

MORE INFORMATION

The sound for each Windows alert level is identified by an entry in the [sounds] section of the WIN.INI initialization file. You can use the Windows Control Panel to change this [sounds] section.

The MessageBeep API function returns control to Visual Basic immediately after queuing a sound. The Visual Basic program executes subsequent code while the MessageBeep sound plays asynchronously.

The MessageBeep API function accepts one parameter, which can have one of the following values:

Parameter Value	Meaning
-1	Produces a standard beep sound by using the computer speaker.
MB_ICONASTERISK	Plays the sound identified by the SystemAsterisk entry in the [sounds] section of WIN.INI.
MB_ICONEXCLAMATION	Plays the sound identified by the SystemExclamation entry in the [sounds] section of WIN.INI.
MB_ICONHAND	Plays the sound identified by the SystemHand entry in the [sounds] section of WIN.INI.
MB_ICONQUESTION	Plays the sound identified by the SystemQuestion entry in the [sounds] section of WIN.INI.
MB_OK	Plays the sound identified by the SystemDefault entry in the [sounds] section of WIN.INI.

Example: How to Invoke MessageBeep API Function

1. Start a new project in Visual Basic. Form1 is created by default.

2. Double click Form1. Add the following to the Form Load event code:

```
Sub Form_Load ()
   Const MB_ICONQUESTION = 32     ' Warning query. See CONSTANTS.TXT.
   Const MB_ICONEXCLAMATION = 48  ' Warning message.
   Const MB_YESNO = 4             ' Yes and No buttons
   MessageBeep MB_ICONEXCLAMATION      ' Plays waveform sound.
   MsgBox "Wow!", MB_ICONEXCLAMATION   ' Displays message box.
   MessageBeep MB_ICONQUESTION
   MsgBox "Yes or No?", MB_ICONQUESTION + MB_YESNO
   End
End Sub
```

NOTE: The MB_ICONQUESTION and MB_ICONEXCLAMATION values are the same for both the MessageBeep API function and the MsgBox statement. See the "Parameters for MsgBox Statement" section below.

3. Choose (general) from the Object menu. Add the following Declare to the general declarations section:

   ```
   Declare Sub MessageBeep Lib "User" (ByVal wType As Integer)
   ```

4. Start the program or press the F5 key. MessageBeep plays the appropriate sound waveform file as each message box displays.

Windows Sound Events Are Not Standardized

Windows version 3.1 allows you to assign waveform audio sounds to certain events through the Control Panel. These events are:

- Default Beep
- Exclamation
- Windows Start
- Windows Exit
- Critical Stop
- Question
- Asterisk

System sounds are dependent upon the application in which they occur. To produce a sound, an application needs to notify Windows that a sound is to occur, and then tell Windows which system sound to play. The application will specify one of the seven default system sounds or any sound event that it has added to this list.

This means that you cannot add sound events to the default list and have an application play that sound, unless the application has been specifically written to call that sound event.

Additionally, applications for Windows have not standardized on when these sound events should occur. Therefore, one application may play the Default Beep sound when an error occurs while another application might play the Critical Stop sound.

Parameters for MsgBox Statement

```
Const MB_ICONSTOP = 16          ' Critical message; displays STOP icon.
Const MB_ICONQUESTION = 32      ' Warning query; displays ? icon.
```

```
Const MB_ICONEXCLAMATION = 48   ' Warning message; displays ! icon.
Const MB_ICONINFORMATION = 64   ' Information message; displays i icon.

Const MB_OK = 0                 ' OK button only
Const MB_OKCANCEL = 1           ' OK and Cancel buttons
Const MB_ABORTRETRYIGNORE = 2   ' Abort, Retry, and Ignore buttons
Const MB_YESNOCANCEL = 3        ' Yes, No, and Cancel buttons
Const MB_YESNO = 4              ' Yes and No buttons
Const MB_RETRYCANCEL = 5        ' Retry and Cancel buttons

Const MB_APPLMODAL = 0          ' Application Modal Message Box
Const MB_DEFBUTTON1 = 0         ' First button is default
Const MB_DEFBUTTON2 = 256       ' Second button is default
Const MB_DEFBUTTON3 = 512       ' Third button is default
Const MB_SYSTEMMODAL = 4096      'System Modal
```

The above parameters for the MsgBox statement can also be found in any of the following sources:

- Visual Basic's Help menu; search for the MsgBox statement.
- The CONSTANTS.TXT file.
- Page 384-387 of the "Language Reference."

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgOther

Using MSGBLAST.VBX Control to Process Windows Messages from VB

Article ID: Q110104

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

The Message Blaster (MSGBLAST.VBX) file is a Visual Basic custom control that lets you catch and process Windows messages from Visual Basic. You can obtain Message Blaster from the Microsoft Developer Network (MSDN) CD-ROM disk number 5 or 6. On that MSDN CD-ROM disk, see the "Message Blaster: Processing Messages in Visual Basic" technical article.

MORE INFORMATION

Message Blaster: Processing Messages in Visual Basic

The Microsoft Visual Basic development environment is not based on a message-driven programming model, as is Microsoft Windows. In Microsoft Windows, messages control most everything that happens. Visual Basic, on the other hand, supports a predefined set of events for each object (form or control) that you create. An application written in Visual Basic cannot respond to messages from Microsoft Windows that are not handled directly by a Visual Basic event.

The Message Blaster (MSGBLAST.VBX) is a Visual Basic custom control that addresses this restriction by allowing you to catch and process Windows messages (except WM_CREATE and WM_NCCREATE messages) from Visual Basic.

Message Blaster was created for Visual Basic version 2.0 on April 30, 1993, by:

- Ed Staffin, Microsoft Consulting Services
- Kyle Marsh, Microsoft Developer Network Technology Group

You can obtain the Message Blaster from Microsoft Developer Network (MSDN) compact disk (CD) number 5 or 6. Run the "MS Development Library CD 6" icon that is installed in the "MS Development Library" folder in Program Manager. Click the Contents button. Double-click "Technical Articles." Double-click "Visual Basic Articles." Double-click the "Message Blaster: Processing Messages in Visual Basic" article.

If you prefer, you can download Message Blaster (MSGBLAST.VBX) as a self-extracting file from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL and download S14515.EXE

- Microsoft Download Service (MSDL)
    Dial (206) 936-6735 to connect to MSDL
    Download MSGBLAST.EXE

- Internet (anonymous FTP)
    ftp ftp.microsoft.com
    Change to the \softlib\mslfiles directory
    Get MSGBLAST.EXE

After downloading the file, run it to obtain MSGBLAST.VBX along with a number of example and source code files. For more information about these other files, please see the following article in the Microsoft Knowledge Base:

   ARTICLE-ID: Q103224
   TITLE   : SAMPLE: MSGBLAST - Sample Application

The sample program EX1 uses the Message Blaster control to process WM_MENUINIT and WM_MENUSELECT messages.

How to Join MSDN

The Microsoft Developer Network (MSDN) provides technical information and development toolkits for all developers who write applications for Microsoft operating systems. MSDN members receive a quarterly CD-ROM disk and a monthly newsletter.

To join MSDN:

- In the U.S. and Canada, call (800) 759-5474, 24 hours a day, 7 days a week.
- In France, call 05 90 59 04 (toll-free).
- In Germany, call 0130 81 02 1.
- In the Netherlands, call 06 022 24 80 (toll-free).
- In the United Kingdom, call 0800 96 02 79 (toll-free).
- In Japan, call 03 5461-2617.
- For any other country in Europe, call +31 10 258 88 64.
- Outside of Europe, the U.S., Canada, or Japan, call (402) 691-0173.

This MSDN information is taken from the "Microsoft Developer Network News" newsletter dated January 1994. This information is subject to change.

Additional reference words: 2.00 3.00
KBCategory: APrg
KBSubcategory: APrgOther

LONG: How to Call Windows API from VB - General Guidelines

Article ID: Q110219

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 2.0 and 3.0

SUMMARY

This article gives general guidelines and examples to introduce you to the process of calling Windows API functions from a Visual Basic application. The examples used in this article are discussed individually in other articles in the Microsoft Knowledge Base. They are repeated here as examples for those new to the process of calling Windows API functions.

One of the most powerful features of Microsoft Visual Basic is the Declare statement, which allows you, the Visual Basic programmer, to call the routines in any Dynamic Link Library (DLL). Microsoft Windows is itself a collection of DLLs, so Visual Basic can call almost any of the functions in the Microsoft Windows Application Programming Interface (API). By calling these routines you can perform tricks that are impossible in Visual Basic alone.

MORE INFORMATION

The Windows API can appear daunting at first. You need to approach it with a sense of adventure: for a Visual Basic programmer, the Windows API is a huge unexplored jungle of over five hundred functions. Fortunately, Visual Basic takes care of so many details for you that you will never have to learn anything about most of these functions. But some of them do things that are very hard to do in Visual Basic alone. And a few of them allow you to do things in your Visual Basic application that you can't do any other way. This article is your guide to the API jungle.

Backups Are Crucial

As with any good adventure, there are risks as well. When calling the Windows API, you may declare a function incorrectly or pass it the wrong values. As a result, you may get a general protection (GP) fault or an Unexpected Application Error (UAE). Fortunately, insurance for this adventure is cheap: always save your work before you run it. Keep backups for every version.

Additional Resources You Might Want

While reading this article and trying out the examples, you may find it helpful to keep the Visual Basic Programmer's Guide handy. Chapter 24, "Calling Procedures in DLLs" explains details only touched on here.

To go beyond this article, you need to get documentation for the Windows API. The Professional Edition of Visual Basic includes this information in two Help files (WIN31WH.HLP and WIN31API.HLP) and a text file (WINAPI.TXT) the complete list of Visual Basic DLL procedure, constant, and user-defined type declarations for the Windows API. You can search for the declaration you want in the WIN31API.HLP Help file; then copy and paste them into your code. Alternatively, you can copy the declarations from the WINAPI.TXT file. You'll find all three files in the \VB\WINAPI directory

Two-Step Process

There are two steps to using a DLL procedure in Visual Basic. First you declare it once. Then you call it as many times as it is needed. The remainder of this article provides a number of examples you can use to test this two-step process.

Declaring DLL Routines

Most DLL routines, including those in the Windows API, are documented using notation from the C programming language. This is only natural, as most DLLs are written in C. However, this poses something of a challenge for the intrepid Visual Basic programmer who wants to call these routines.

In order to translate the syntax of a typical API routine into a Visual Basic Declare statement, you have to understand something about how both C and Visual Basic pass their arguments. The usual way for C to pass numeric arguments is "by value": a copy of the value of the argument is passed to the routine.

Sometimes C arguments are pointers, and these arguments are said to be passed "by reference." Passing an argument by reference allows the called routine to modify the argument and return it. C strings and arrays are always passed by reference.

Visual Basic, on the other hand, usually passes all of its arguments by reference. In effect, when you pass arguments to a Visual Basic procedure you are actually passing "far" (32 bit) pointers to those values. In order to pass arguments to a C routine that expects its arguments to be passed by value, you have to use the ByVal keyword with the argument in the Declaration statement.

Obviously, if a DLL routine is expecting an argument to be passed by value and you pass a pointer instead, it is not going to behave as you expect. Likewise, if the routine is expecting a pointer to a value and you pass the value itself, the routine is going to attempt to access the wrong memory location and probably cause a GP fault or UAE. So be careful.

One added wrinkle to this is that Visual Basic strings do not use the same format as C strings. Visual Basic has overloaded the ByVal keyword to mean "pass a C string" when it is used with a string argument in a declare statement.

C argument types and their equivalent declarations in Visual Basic:

If the argument is	Declare it as

standard C string (LPSTR, char far *)	ByVal S$
Visual Basic string (see note)	S$
integer (WORD, HANDLE, int)	ByVal I%
pointer to an integer (LPINT, int far *)	I%
long (DWORD, unsigned long)	ByVal L&
pointer to a long (LPDWORD, LPLONG, DWORD far *)	L&
standard C array (A[ ])	base type array (no ByVal)
Visual Basic array (see note)	A()
struct (typedef)	S As Struct

NOTE: You will never pass a Visual Basic string or array to a DLL routine unless the DLL was written specifically for use with Visual Basic. Visual Basic strings and arrays are represented in memory by "descriptors" (not pointers), which are useless to DLL routines that were not written with Visual Basic in mind.

There is one more complication to this, however. Some Windows functions take a 32 bit argument that sometimes is a "far" (32 bit) pointer to something and sometimes is just a 32 bit value. The fourth argument in the SendMessage function is like this. If you are going to call it with just a pointer or with just a value, you can declare it appropriately. For example, you could declare SendMessage to take a pointer to a string:

```
' Enter the following Declare statement as one, single line:
Declare Function SendMessage Lib "user"
   (ByVal hWnd%, ByVal msg%, ByVal wp%, ByVal lp$) As Long
```

Or you could declare it to take a 32 bit value:

```
' Enter the following Declare statement as one, single line:
Declare Function SendMessage Lib "user"
   (ByVal hWnd%, ByVal msg%, ByVal wp%, ByVal lp&) As Long
```

Notice the fourth argument is declared ByVal lp$ in the first example and ByVal lp& in the second.

However, what if you want to call it with both kinds of arguments in the same program? If you declare it one way and call it another, you will get an error message from Visual Basic. The solution is to declare the argument As Any:

```
' Enter the following Declare statement as one, single line:
Declare Function SendMessage Lib "user"
   (ByVal hWnd%, ByVal msg%, ByVal wp%, lp As Any) As Long
```

The Any "data type" tells Visual Basic not to do any type checking for that argument. So now you can pass it anything as long as it is what the function is expecting. If an argument is declared As Any, you must specify whether the argument is passed by value or not -- when you actually call the function. You do this by using ByVal for strings and for arguments that should be passed by value, and omitting ByVal for arguments that should be passed by reference. Use the appropriate entry from the second column in the table shown above as the argument when you call the function.

For example, if you have declared the fourth argument in SendMessage As Any, you can pass a string in that argument:

```
buflen& = SendMessage(txthWnd, EM_GETLINE, lineNum%, ByVal buf$)
```

Notice you use the ByVal keyword in the call. This tells Visual Basic that you want to pass a standard C string. If you don't include the ByVal, you will pass a Visual Basic string descriptor, which is not something SendMessage knows how to handle.

You can also pass an array:

```
dummy& = SendMessage(any_hWnd%, EM_SETTABSTOPS, NumCol%, ColSizes(1))
```

Notice you do not use ByVal in this case because you want to pass a pointer (specifically a pointer to the indicated element in the array -- all subsequent array elements are packed into memory after it).

You can pass a long integer value as the fourth argument:

```
dummy& = SendMessage(txthWnd, EM_LINESCROLL, 0, ByVal ScrollAmount&)
```

Note the use of ByVal here. You want to pass the value itself, rather than a pointer to the value. It's very important that you pass a Long integer for this argument. If you pass a normal Integer Visual Basic will not convert it into Long.

If you're careful to match up what you're passing with what the routine expects, you should have no trouble calling the Windows API to get Visual Basic to do what you want as demonstrated in the examples that comprise the remainder of this article.

Scoping Out the System

One of the nice things about Windows is that it insulates you from a lot of the details of the system. You can print to the printer without knowing what kind it is; you can display things on the screen without knowing its resolution. However, there may be times when your application needs to know key information about the system. For example, you may want your application to perform different calculations depending on whether the system has a math coprocessor or not.

Fortunately, Windows provides several functions that you can use to obtain this kind of information. For example the GetWinFlags API function can give you a lot of information.

Place this declaration in the declarations section of a form or module, or in the global module:

```
Declare Function GetWinFlags Lib "kernel" () As Long
```

As functions in the Windows API go, this one is very simple. It is found in the Windows "kernel" DLL. It takes no arguments (hence the empty parentheses in the declaration) and returns a Long integer. This Long will have bits, or flags, set to indicate certain facts about the system. Here are some of the flags:

```
Const WF_CPU286 = &H2&
```

```
Const WF_CPU386 = &H4&
Const WF_CPU486 = &H8&
Const WF_STANDARD = &H10&
Const WF_ENHANCED = &H20&
Const WF_80x87 = &H400&
```

Place the constants in the Declarations section of the form or module where you declare the GetWinFlags function.

Now you can call GetWinFlags and use the And operator with these constants to test the value returned. For example:

```
Dim WinFlags As Long
WinFlags = GetWinFlags()
If WinFlags And WF_ENHANCED Then
   Print "Windows Enhanced Mode ";
Else
   Print "Windows Standard Mode ";
End If
If WinFlags And WF_CPU486 Then Print "on a 486"
If WinFlags And WF_CPU386 Then Print "on a 386"
If WinFlags And WF_CPU286 Then Print "on a 286"
If WinFlags And WF_80x87 Then Print "Math coprocessor available"
```

There's one important fact about the system that this function does not provide: the version of Windows. You can obtain that information with the GetVersion function:

```
Declare Function GetVersion Lib "Kernel" () as Long
```

This returns a Long integer containing the version numbers of MS-DOS and Windows. Here's the code that extracts the version information:

```
Dim Ver As Long, WinVer As Long, DosVer As Long
Ver = GetVersion()
WinVer = Ver And &HFFFF
Print "Windows " + Format$(WinVer Mod 256) + "." + Format$(WinVer \ 256)
DosVer = Ver \ &H10000
Print "MS-DOS " + Format$(DosVer \ 256) + "." + Format$(DosVer Mod 256)
```

GetSystemMetrics is another Windows function that provides useful system information. You declare it like this:

```
Declare Function GetSystemMetrics Lib "User" (ByVal nIndex%) As Integer
```

This function is located in the "User" DLL. It takes one argument: an integer indicating which item of system information you want it to return. This argument, like most arguments to Windows API functions, is passed by value. Because Visual Basic usually passes arguments by reference, you have to include the ByVal keyword to specify the argument should be passed by value. This is very important. Forgetting ByVal when it is needed or including it when it isn't often leads to problems.

GetSystemMetrics provides a potpourri of information. For example, you can use it to find out if a mouse is installed in the system with code like this:

```
Const SM_MOUSEPRESENT = 19
If GetSystemMetrics(SM_MOUSEPRESENT) Then Print "Mouse installed"
```

Some other useful information provided by GetSystemMetrics is the size of the arrow bitmaps used by standard horizontal and vertical scroll bars. This is important because the size of these bitmaps varies with the resolution of the display and the display driver installed. When you create an application that uses a horizontal scroll bar control, you usually give it a fixed height; likewise, when you create a form with a vertical scroll bar control, you usually give it a fixed width. You fix these sizes based on what looks good on your system. Unfortunately, what looks good on your display can look strange on a display that has a different resolution. If you are writing applications that need to look good on a variety of display resolutions, you need to write code that can determine the standard size of scroll bars on the current display and dynamically resize your scroll bar controls to match. You need to write code like this:

```
Const SM_CXVSCROLL = 2
Const SM_CYHSCROLL = 3
ScaleMode = 3 'Pixels
VScroll1.Width = GetSystemMetrics(SM_CXVSCROLL)
HScroll1.Height = GetSystemMetrics(SM_CYHSCROLL)
```

Notice that the values returned by the GetSystemMetrics function are always in pixels, so you need to set the ScaleMode of the form to 3 (pixels) before setting the sizes of the scroll bars.

There are a lot of other system values you can obtain using GetSystemMetrics, but not all of them are useful to Visual Basic programmers. Here are a few of the interesting ones:

```
Const SM_CXSCREEN = 0   'Width of screen in pixels
Const SM_CYSCREEN = 1   'Height of screen in pixels
Const SM_CYCAPTION = 4  'Height of form titlebar in pixels
Const SM_CXICON = 11    'Width of icon in pixels
Const SM_CYICON = 12    'Height of icon in pixels
Const SM_CXCURSOR = 13  'Width of mousepointer in pixels
Const SM_CYCURSOR = 14  'Height of mousepointer in pixels
Const SM_CYMENU = 15    'Height of top menu bar in pixels
```

Yet another function that provides system information is GetDeviceCaps. This function returns information about a particular device in the system, such as the printer or the display. Like many of the functions you will see in this article, the declaration for GetDeviceCaps is too long to fit on one line, but you must type it all on one, single line:

```
' Enter the following Declare statement as one, single line:
Declare Function GetDeviceCaps Lib "GDI" (ByVal hDC%, ByVal nIndex%)
   As Integer
```

GetDeviceCaps is found in the "GDI" DLL and it takes two arguments. The first allows you to specify the device for which you want information. When calling the function from Visual Basic, supply either the hDC property of a form or the hDC property of the Printer object. The second argument specifies the device information you want to get. There are a lot of possible values for this second argument, but only a couple of them are very interesting to the Visual Basic programmer. For example, you can find

out how many colors the screen or printer supports:

```
Const PLANES = 14
Const BITSPIXEL = 12
Dim Cols As Long
Cols = GetDeviceCaps(hDC, PLANES) * 2 ^ GetDeviceCaps(hDC, BITSPIXEL)
```

The number of colors a device supports is the product of the number of color planes it has and the number of bits per pixel in each plane. Because each bit can represent two colors, you have to raise 2 to the power of the number of bits per pixel, and then multiply that by the number of color planes, to get the total number of colors that the device can display.

Some Useful Tricks

That's enough poking about in the system. Here are some useful tricks. If you have used the Shell function in Visual Basic, you have probably discovered that it will only run files that have the extension .EXE, .COM, .PIF, or .BAT. But you can double-click almost any file in the File Manager, and Windows does the right thing. For example, if it is a .TXT file, Windows starts Notepad. How would you add this kind of functionality to your own applications?

Windows stores the association between data files and their related application (such as the association between a .TXT file and the NOTEPAD application) in the WIN.INI file in the Extensions section. Windows also provides a function called GetProfileString that reads the WIN.INI file for you. Here is the Declare for GetProfileString:

```
' Enter the following Declare statement on one, single line:
Declare Function GetProfileString Lib "Kernel"
   (ByVal Sname$, ByVal Kname$, ByVal Def$, ByVal Ret$, ByVal Size%)
   As Integer
```

GetProfileString searches the WIN.INI section specified in the first argument for the key specified in the second argument and returns the value for that key in the third argument. The fourth argument provides the length of the string passed as the third argument.

Therefore, once you know the extension of a file, you can use GetProfileString to find the parent application for files with that extension. Here's a function that does that:

```
Function FindApp (Ext As String) As String
   ' Find the parent app for a file with the given extension
   Dim Sname As String, Ret As String, Default As String
   Ret = String$(255, 0)
   Default = Ret
   Sname = "Extensions"
   nSize = GetProfileString(Sname, Ext, Default, Ret, Len(Ret))
   If Left$(Ret, 1) <> Chr$(0) Then
      FindApp = Mid$(Ret, 1, InStr(Ret, "^") - 1)
   End If
End Function
```

GetProfileString is an example of a Windows function that returns a string

by modifying one of its arguments. To use these kinds of functions, you must create a string and fill it with something (character code 0 in the example above) before you call the function. This is because Windows cannot enlarge strings the way Visual Basic can, so whenever you pass a string to Windows you must ensure that it is long enough to hold the largest possible string that Windows might return.

You can add new values to WIN.INI using the WriteProfileString function:

```
' Enter the following Declare statement on one, single line:
Declare Function WriteProfileString Lib "Kernel"
   (ByVal Sname$, ByVal Kname$, ByVal Set$) As Integer
```

This function searches the WIN.INI file for the section specified in the first argument and the key specified in the second argument. Then it replaces the key value with the value specified in the third argument. If the key is not found, it adds the key and its value to the specified section. If it does not find the section, it adds that to WIN.INI as well.

Some applications use their own private .INI files rather than using WIN.INI -- Visual Basic has its own VB.INI, for example. You can use the functions GetPrivateProfileString and WritePrivateProfileString to manipulate other .INI files:

```
' Enter each of the following Declare statements on one, single line:
Declare Function GetPrivateProfileString Lib "Kernel"
   (ByVal Sname$, ByVal Kname$, ByVal Def$, ByVal Ret$, ByVal Size%,
   ByVal Fname$) As Integer
Declare Function WritePrivateProfileString Lib "Kernel"
   (ByVal Sname$, ByVal Kname$, ByVal Set$, ByVal Fname$) As Integer
```

These work exactly like GetProfileString and WriteProfileString, except they have one additional argument that specifies the path and filename of the .INI file.

Now, where should you put that custom .INI file? One obvious place is the Windows directory. But people name that directory all sorts of things: \WINDOWS or \WIN3 or who knows what. It might not even be at the root level. How can you find it? Well, Windows knows where this directory is, and it provides a function to tell you:

```
' Enter the following Declare statement on one, single line:
Declare Function GetWindowsDirectory Lib "User"
   (ByVal P$, ByVal S%) As Integer
```

The first argument is a string that the function will fill with the path to the Windows directory; the second argument is the length of this string. Again, because you are passing a string to be filled by Windows, you must make sure it is large enough to accommodate whatever string Windows might provide. In this case, the Windows Reference warns that it should be at least 144 characters. On the other hand, 144 characters is the worst case so in most cases there will be a lot of unused characters that you will need to trim off. The GetWindowsDirectory function returns an integer value that indicates the actual length of the returned string. So here's some fancy code that calls the function and trims the returned string all in one line:

```
Dim WinPath As String
WinPath = String$(145, Chr$(0))
WinPath = Left$(WinPath, GetWindowsDirectory(WinPath, Len(WinPath)))
```

In addition, there is usually a \SYSTEM directory within the windows directory. Once again, that could be called anything, and once again Windows provides a function to find it: GetSystemDirectory.

This function is declared in exactly the same way as GetWindowsDirectory and can be called in the same way, so substitute it in the declaration and code above and try it out.

Another sensible place to put that .INI file is in the same directory as the application. It should be trivial to figure out what directory a running Visual Basic application is stored in, but it's not. This information isn't provided through the Command$ system variable, unfortunately. And even CurDir$ isn't reliable because the application could have been run using a full path without changing the current directory to the application's directory. Windows API calls to the GetModuleHandle and GetModuleFileName functions give you what you need. Here are the declarations:

```
' Enter each of the following Declare statement on one, single line:
Declare Function GetModuleHandle Lib "Kernel"
   (ByVal FileName$) As Integer
Declare Function GetModuleFileName Lib "Kernel"
   (ByVal hModule%, ByVal FileName$, ByVal nSize%) As Integer
```

GetModuleHandle takes the filename of a running program and returns a "module handle." All you need to know about the module handle is that it is an integer and you pass it to the GetModuleFileName function.

GetModuleFilename takes three arguments; the first is the module handle returned by GetModuleHandle. The second is a string that the function fills with the complete path and filename of the program specified by the module handle. The third is the size of this string. The value returned by GetModuleFilename is the length of the path that it placed in the string you passed.

Using these two functions, obtaining the path to a running program is easy:

```
Dim hMod As Integer, Path As String
hMod = GetModuleHandle%("MyApp.EXE")
Path = String$(145, Chr$(0))
Path = Left$(Path, GetModuleFileName%(hMod, Path, Len(Path))
```

Notice that you are again passing a large string and using the value returned by the function to trim the string down to size with Left$.

Another trick you can perform with GetModuleHandle is limiting your application to a single instance. Normally, Windows allows you to run multiple instances (copies) of the same program. Most of the time this is a handy feature, but sometimes it can cause problems. If your program uses data
files, having more than one instance of the program accessing those files at the same time can leave the files in an inconsistent state. You could write the program so that it works correctly even if there are multiple

copies running, but that's a lot of work and sometimes it's not even possible. An easier method is to just ensure that only one copy of the program can be run. Thanks to GetModuleHandle and another Windows function, GetModuleUsage, this is easy:

```
Declare Function GetModuleUsage Lib "Kernel" (ByVal hModule%) As Integer
```

GetModuleUsage returns how many instances of the specified program exist. The program is specified by passing GetModuleUsage a module handle, which is what GetModuleHandle returns. Putting code like this in the Form_Load event for your startup form (or in your Sub Main if you don't have a startup form) ensures that only a single instance of your application can be run:

```
If GetModuleUsage(GetModuleHandle("YOURAPP.EXE")) > 1 Then
   MsgBox "This program is already loaded!", 16
   End
End If
```

GetModuleHandle and GetModuleUsage work for DLLs as well as ordinary executable files, so you could use this technique to find out how many Visual Basic executables are running by using GetModuleUsage with the module handle for VBRUN100.DLL.

Sending Messages to Controls

Windows is built around messages. The Windows system sends messages to applications. You see these messages in your Visual Basic program as events. In addition, applications send messages to each other (this is the basis for DDE), and applications even send messages to themselves.

You can get in on the action. By sending messages to controls, you can get them to do things that would otherwise be impossible, such as setting tab stops in list boxes and getting a single line of text from a multi-line text box. You can also do things by sending a single message that would take a lot more Basic code to accomplish, such as emptying a list box. You send messages with SendMessage function:

```
' Enter the following Declare statement as one, single line:
Declare Function SendMessage Lib "User"
   (ByVal hWnd%, ByVal msg%, ByVal wp%, lp As Any) As Long
```

The first argument identifies the recipient of the message. Windows uses "handles" to keeps track of everything it uses. Handles are integer ID numbers that Windows assigns to things -- like the module handles used earlier to refer to programs.

Controls are just another kind of window as far as Windows is concerned. Controls are identified by their window handle, or hWnd. To send a message to a control you need its hWnd.

Visual Basic provides the hWnd for a form through the hWnd property. Unfortunately, there is no such property for any of the controls. The controls are windows and do have hWnds, but Visual Basic doesn't provide them for you. So you have to resort to some subterfuge. Windows has a function called GetFocus that will return the hWnd for the window that has

the focus:

```
Declare Function GetFocus Lib "user" () As Integer
```

And we can give the focus to any control using the Visual Basic SetFocus method. So to get the hWnd for a control, use code similar to this:

```
AnyControl.SetFocus
control_hWnd = GetFocus()
```

The second argument in the SendMessage function is the message number. All of the message numbers are some offset from the WM_USER message, which has the value 1024 (&H400 in hexadecimal notation). The complete list of message numbers is included in the WINAPI.TXT file.

The last two arguments in SendMessage supply additional information for a particular message. What they contain varies from message to message. Notice that the last argument was declared As Any. This is different from the way the SendMessage function is declared in the WINAPI.TXT file. It allows you to pass any data type as the fourth argument.

The Long integer value that SendMessage returns depends on what message you sent. Sometimes you send a message to tell a control to do something, and the return value is zero if the control could perform the action and non-zero if it could not. Sometimes you send a message to a control to find out something about that control, and in those cases the return value is the information you requested. And sometimes the return value means nothing at all.

Try out SendMessage by starting with something simple: emptying list boxes. If you've spent much time programming with list boxes, you are probably annoyed that there is no simple way to empty a list. Instead of telling the list box to simply empty itself, you have to loop through all the entries and use the RemoveItem method on each one. But there's a better way.

Windows provides a message (LB_RESETCONTENT) that you can send to a list box to make it empty itself in one step.

```
Const WM_USER = &H400
Const LB_RESETCONTENT = WM_USER + 5
```

Here is a procedure that uses this message to empty any list box:

```
Sub ClearListBox(Ctrl As Control)
   Ctrl.SetFocus
   dummy& = SendMessage(GetFocus(), LB_RESETCONTENT, 0, ByVal 0&)
End Sub
```

The RESETCONTENT message needs no additional information, so the last two arguments to SendMessage are zero. Notice that the last argument is ByVal 0& (zero followed by an ampersand character). The ampersand is very important; it ensures that a long (32 bit) zero is passed. There is no useful information returned when this message is sent, so the SendMessage function is assigned to a dummy variable.

You can also empty combo box lists in the same way; just use this constant for the message:

```
Const CB_RESETCONTENT = WM_USER+11
```

While on the topic of lists, there's an easy way to find strings in a list. You can loop through all the items in the list, but why bother when the LB_FINDSTRING message allows you to find a string in a list box with a single function call? When you send the LB_FINDSTRING message, the SendMessage function returns the index of the first item in the list that matches the string you specified (so obviously you should only use this message with sorted list boxes).

```
Const LB_FINDSTRING = WM_USER + 16
Dim itemNum As Long
itemNum = SendMessage(GetFocus(), LB_FINDSTRING, -1, ByVal "Visual")
Print "Windows is item: "; Format$(itemNum)
```

This finds the first list item that begins with "Visual" and returns its index in the list. It will match even if there are additional characters following the specified string, so the example above would match "Visual Basic" if it was the first string in the list beginning with "Visual." Again, this technique works as well with combo boxes as it works with list boxes. Just use the message:

```
Const CB_FINDSTRING = WM_USER + 12
```

And, speaking of combo boxes, here's a neat trick for a dropdown list combo box (a combo box with the Style property set to 2). This code drops the list automatically when the combo box gets the focus:

```
Sub Combo1_GotFocus ()
   Const CB_SHOWDROPDOWN = WM_USER + 15
   Dummy& =  SendMessage(GetFocus(), CB_SHOWDROPDOWN, 1, ByVal 0&)
End Sub
```

Sending messages in the GotFocus event for a control is a good technique because it allows you to avoid explicitly setting the focus to a control to get its hWnd.

Another useful message is LB_GETTOPINDEX. When you send this message to a list box, the SendMessage function returns the index of the first visible item in the list. This is valuable if the list has been scrolled and you want to determine which items are actually visible in the list (in a DragDrop event, for example).

```
Const LB_GETTOPINDEX = WM_USER+15
FirstItem& = SendMessage(GetFocus(), LB_GETTOPINDEX, 0, ByVal 0&)
```

You can also send a message to scroll a list box to make any item the first visible item in the list:

```
Const LB_SETTOPINDEX = WM_USER+24
Success& = SendMessage(GetFocus(), LB_SETTOPINDEX, item%, ByVal 0&)
```

You could combine this message with the LB_FINDSTRING message to scroll the list box so that the list item found by LB_FINDSTRING is at the top of the list.

One especially valuable use of SendMessage is to set the tabstops in a list or
text box. You have probably discovered that list boxes and multi-line text boxes handle tabs automatically, so if you assign text that contains tabs (character code 9) the columns line up automatically. Unfortunately, Visual Basic gives you no way to adjust where the columns fall -- except by sending a message. For a list box, the message is:

```
Const LB_SETTABSTOPS = WM_USER + 19
```

When you send this message, you must supply an array of integers that specify the new tab positions. (These positions are specified in terms of characters; when the list box or text box contains a proportional font, these are "average" characters.) This array is the fourth argument to the SendMessage function; the number of elements in the array is the third argument. Notice that to pass an array to a Windows function, you actually pass the first element of the array:

```
ReDim tabs(3) As Integer
tabs(1) = 10
tabs(2) = 50
tabs(3) = 90
List.SetFocus
dummy& = SendMessage(GetFocus(), LB_SETTABSTOPS, 0, ByVal 0&)
dummy& = SendMessage(GetFocus(), LB_SETTABSTOPS, 3, tabs(1))
```

The first call to SendMessage clears any existing tabstops; the second sets three tabstops as specified in the array.

You can set the tabstops in a multi-line text box as well; just send the message EM_SETTABSTOPS:

```
Const EM_SETTABSTOPS = WM_USER + 27
```

This won't work for a single-line text box. Speaking of multi-line text boxes, there are several useful messages you can send to a multi-line text box to get information that Visual Basic does not provide. For example, you can send the EM_GETLINECOUNT message to get the number of lines text in a multi-line text box:

```
Const EM_GETLINECOUNT = WM_USER+10
lineCount& = SendMessage(GetFocus(), EM_GETLINECOUNT, 0, ByVal 0&)
```

You can obtain the text contained in any line of a multi-line text box with the message:

```
Const EM_GETLINE = WM_USER + 20
```

When sending this message, you have to provide the number of the line you want to retrieve in the third argument to SendMessage, and the string to be filled with the contents of the line as the fourth argument. There's one odd thing about this string. Normally, when you pass a string to a Windows function you also supply the size of the string as an argument. However, the usual place for that information is in the third argument, and that is already being used to specify which line you want retrieved. So you have to place the length of the string in the first two bytes of the string, using code like this:

```
   Dim LineNum As Integer, linelength As Integer, buf As String
   'Set linelength to some reasonable value
   buf = String$(linelength, chr$(0))
   buf = Chr$(linelength Mod 256) + Chr$(linelength \ 256) + Buf
   ' Enter the following two lines as one, single line:
   buf = Left$(buf,
      SendMessage(GetFocus(), EM_GETLINE, lineNum, ByVal buf))
```

Another handy message for multi-line text boxes is EM_LINESCROLL, which allows you to scroll them horizontally and vertically. You specify the amount to scroll in the fourth argument of the SendMessage function: place the number of characters to scroll horizontally in the high word (by multiplying by 65536) and the number of lines to scroll vertically in the low order word. For example:

```
   Sub ScrollIt (ctl As Control, chars As Integer, lines As Integer)
      Const EM_LINESCROLL = WM_USER + 6
      Dim scroll As Long
      scroll = chars * 65536 + lines
      ctl.SetFocus
      dummy& = SendMessage(GetFocus(), EM_LINESCROLL, 0, ByVal scroll)
   End Sub
```

This is a relative scroll: if you use the value 2 the text box will scroll down by two lines; if you use the value -65536 the text box will scroll left by one character.

Another feature that Visual Basic does not directly support is a way to restrict the number of characters that can be entered in a text box. You can do this by responding to the various Key events, but there is an easier way: send the EM_LIMITTEXT message to the text box:

```
   Sub Text1_GotFocus()
      Const EM_LIMITTEXT = WM_USER+21
      dummy& = SendMessage(GetFocus(), EM_LIMITTEXT, numChars, ByVal 0&)
   End Sub
```

Here the third argument specifies the maximum number of characters the text box will accept. If you want to set it back to normal, send EM_LIMITTEXT with that argument set to zero. You can also restrict the number of characters accepted by a combo box by sending the combo box the message CB_LIMITTEXT:

```
   Const CB_LIMITTEXT = WM_USER+1
```

One last trick: turn a text box into a password control. Windows provides automatic support for text boxes that display asterisks (or some other character) instead of the actual characters the user types. To take advantage of this support, set a style bit in the text box. Normally you set style bits when you create a control, but you can't do that because Visual Basic creates the control for you.

Fortunately, Windows allows you to set that bit after the control is created (this is one of the few style bits that you can change after a control is created). The functions that get and set the style information for a window are as follows:

```
' Enter each of the following Declare statements as one, single line:
Declare Function GetWindowLong Lib "User"
   (ByVal hWnd%, ByVal nIndex%) As Long
Declare Function SetWindowLong Lib "User"
   (ByVal hWnd%, ByVal nIndex%, ByVal NewLong&) As Long
```

To set the password style bit, call GetWindowLong to get the style information, use the Or operator to set the bit, and then call SetWindowLong to store the new style. Once again, do this in the GotFocus event so you don't have to worry about using SetFocus to get the hWnd:

```
Sub Text1_GotFocus ()
   Const ES_PASSWORD = &H20
   Const EM_SETPASSWORD = 1052
   Const GWL_STYLE = -16
   Const Asterisk = 42
   Dim TxthWnd As Integer, WindowLong As Long
   TxthWnd = GetFocus()
   WindowLong = GetWindowLong(TxthWnd, GWL_STYLE)
   WindowLong = WindowLong Or ES_PASSWORD
   WindowLong = SetWindowLong(TxthWnd, GWL_STYLE, WindowLong)
   WindowLong = SendMessage(TxthWnd, EM_SETPASSWORD, Asterisk, ByVal 0&)
End Sub
```

You can define the character you want displayed in place of the actual characters the user types by sending the EM_SETPASSWORD message to the control. This example sets the password character to asterisks. If you want to use a different character, supply a different ANSI character code when you send the EM_SETPASSWORD message.

There are some limitations to this password functionality. For one thing, the characters in the text box are not stored as asterisks; they are just displayed that way. This is good, because it allows your code to easily check what the user typed. But it is also bad, because any user can select the contents of the text box, copy it, and paste it somewhere else and see the actual characters that were typed in the text box. Whether you consider this a flaw depends on whether you expect the text box containing a password to sit around on a screen where some malicious users can copy it. Usually, this is not a problem. But it is something to keep in mind.

Exploring on Your Own

The Windows API is like an enormous hidden world that is just waiting to be explored. The API documentation is the treasure map, and like the maps in all good adventure stories, it requires some translation to be useful. And you aren't limited to just the Windows API. Almost any DLL contains functions that you might find useful. For example, many spell checkers are implemented as DLLs; if you know how to declare and call the functions in one of these DLLs, you can add spell-checking to your Visual Basic programs (assuming that you obey the copyright restrictions for the DLL, of course).

Finding out how to declare and call the DLL functions can be tough sometimes, however. You'll learn to keep your eyes open for anything that looks like API documentation. Who knows what treasure you'll discover inside some obscure DLL? And as new versions of Windows appear, the

treasure will only increase. The Multimedia Extensions for Microsoft Windows are just a collection of DLLs, after all. With the right hardware, think of how much fun you'll have calling those from your Visual Basic programs.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgOther RefsProd

How to Create a Read-Only Text Box Using SendMessage API

Article ID: Q110403

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Visual Basic does not have a Read-Only property for a text box. But you can create a read-only text box by calling the Windows API SendMessage function with the EM_SETREADONLY message.

MORE INFORMATION

Setting the text box state to read-only allows the user to scroll and highlight the text in the text box, but does not allow them to edit it. The program can still modify the text by changing the text property.

To create a read-only text box, call the Windows API SendMessage function, using the EM_SETREADONLY message constant as the second parameter. The SendMessage function requires the following parameters:

```
ret& = SendMessage(hWnd%, uMsg%, wParam%, lParam&)
```

where:

ret&	holds the return value of the function call.
hWnd%	identifies the window handle that is to receive the message.
uMsg%	the message to be sent (EM_SETREADONLY).
wParam%	specifies whether to set or remove the read-only state of the edit control. A value of TRUE sets the state to read-only; a value of FALSE sets the state to read/write.
lParam&	not used for this message, set its value to 0&.

The return value of this function is nonzero if the function was successful, and it is zero if an error occurred.

Step-by-Step Example

The following example loads a file into a text box, and then sets the text box state to read-only:

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a text box (Text1) to Form1. Select the text box and press the F4 key to display the Properties window. Set the MultiLine property of Text1 to True and set the Scrollbars property to 3 - Both.

3. Add the following constants and declarations to the module level or to

the global section of your code:

```
Const WM_USER = &H400
Const EM_SETREADONLY = (WM_USER + 31)

' Enter the following Declare statement as one, single line:
Declare Function SendMessage Lib "User" (ByVal hWnd As Integer,
   ByVal wMsg As Integer, ByVal wParam As Integer,
   ByVal lParam As Any) As Long
```

4. Add the following code to the Form_Load Sub procedure:

```
Sub Form_Load()
   Dim TmpStr As String
   Dim ret as Long

   Open "C:\AUTOEXEC.BAT" For Input As #1
   While Not Eof(1)
      ' Read a line of text in from the input file:
      Line Input #1, TmpStr
      ' Append it to the text box, adding carriage return and line feed:
      Text1.Text = Text1.Text & TmpStr & Chr$(13) & Chr$(10)
   Wend
   ' Set the text box to read-only mode:
   ret = SendMessage(Text1.hWnd, EM_SETREADONLY, True, 0&)

   If ret = 0 Then    ' Check the return value for error
      MsgBox "Could Not Set Text Box to Read-Only."
   End If
End Sub
```

5. Start the program, or press the F5 key. When the program loads, it will read the AUTOEXEC.BAT file into the text box, and then set the read-only state of the text box. Then the user can scroll and highlight the text in the text box but won't be able to edit it.

Additional reference words: 2.00 3.00
KBCategory: APrg
KBSubcategory: APrgOther

How to Add Items into Control Menu Box of Visual Basic Form

Article ID: Q110498

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

To add items into the Control-menu box of a Visual Basic Form, you can use the AppendMenu API (application programming interface) function in Windows. However, Visual Basic cannot directly detect any events for the added menu item. To catch the message for the added menu item, you can use a subclass control. You can write subclass controls using Microsoft C, but not using Visual Basic. Alternatively, you can obtain subclass controls from third-party programs such as SpyWorks from Desaware.

The Control-menu box, found in the upper-left corner of a Visual Basic form, is also known as the System-menu box in other products for Windows. The default Control-menu box contains the following nine entries including separators:

```
Restore
Move
Size
Minimize
Maximize
-----------------------
Close            Alt+F4
-----------------------
Switch to...   Ctrl+Esc
```

MORE INFORMATION

In Windows programming terms, subclassing is the process of creating a message handling procedure and intercepting messages for a given window, handling any messages you choose, and passing the rest to the window's original message handler.

The subclass procedure is a message filter that performs nondefault processing for a few key messages, and passes other messages to a default window procedure using the CallWindowProc API function. The CallWindowProc function passes a message to the Windows system, which in turns sends the message to the target window procedure. The target window procedure cannot be called directly by the subclass procedure because the target procedure (in this case a window procedure) is exported.

How to Contact Desaware

NOTE: Desaware products are manufactured independent of Microsoft.

Microsoft makes no warranty, implied or otherwise, regarding these products' performance or reliability.

Desaware
5 Town & Country Village #790
San Jose, CA 95128
Contact: Gabriel Appleman (213) 943-3305
          Dan Appleman (408) 377-4770
Fax: (408) 371-3530

The Desaware company offers the following products:

- Custom Control Factory -- An interactive development tool for creating custom controls including animated pushbuttons, multistate buttons, enhanced buttons, check boxes, and option button controls for Windows applications.

- CCF-Cursors -- Provides you with complete control over cursors (mouse pointers) in Visual Basic applications. Create your own cursors or convert icons to cursors, and much more. Includes over 50 cursors.

- SpyWorks-VB -- An advanced development tool for use with Visual Basic. SpyWorks contains subclass controls.

This information is subject to change.

Additional reference words: 3.00
KBCategory: APrg Refs
KBSubcategory: APrgOther RefsThird

How to Turn on Mouse Trails with Visual Basic

Article ID: Q110541

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic Programming System for Windows, versions 2.0 and 3.0

SUMMARY

This article demonstrates how to turn on mouse trails by using the Escape() Windows API function. This works on computers that have video drivers that support mouse trails -- not all video drivers do. The Escape() function returns a zero if the function is not supported by the video driver.

MORE INFORMATION

Step by Step to an Application That Turns on Mouse Trails

1. Start Visual Basic, or if Visual Basic is already running, choose New Project from the File menu (ALT, F, N). Form1 is created by default.

2. Add a Command Button (Command1) to Form1.

3. Add the following code to the General Declarations section of Form1:

```
' Enter the following Declare statement on one, single line:
Declare Function Escape Lib "GDI" (ByVal hDC As Integer,
   ByVal nEscape As Integer, ByVal nCount As Integer, lpInData As Any,
   lpOutData As Any) As Integer

Const MouseTrails = 39
Const SizeOfWord = 2
```

4. Add the following code to the Command1_Click event of Form1:

```
Dim x As Integer

x = 7  ' Set x to one of the following values:
       ' 1 to 7 : turns mouse trails on and shows 1 to 7 trailers
       '      0 : turns off mouse trails
       '     -1 : turns mouse trails on, reads info from WIN.INI
       '     -2 : disables mouse trails, doesnt update WIN.INI
       '     -3 : enables mouse trails, updates WIN.INI

result% = Escape(form1.hDC, MouseTrails, SizeOfWord, x, 0&)
```

5. Run the program. Click the Command1 button to turn on mouse trails.

For more information about the Escape() Windows API function and mouse trails, please see Windows version 3.1 SDK help file that ships with the

Professional Edition of Visual Basic.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgOther

How to Make Mouse Pointer (Cursor) Maintain Hourglass Shape

Article ID: Q110542

The information in this article applies to:

- Professional and Standard Editions of Microsoft Visual Basic Programming System for Windows, versions 2.0 and 3.0

SUMMARY

You can have a Visual Basic application program set the mouse pointer (cursor) to an hour glass shape and wait. However, if the user moves the mouse over another application's window, the cursor will return to a mouse pointer. To force the cursor to maintain the hourglass shape even while over other windows, make the window a system modal window by using the SetSysModalWindow Windows API function.

MORE INFORMATION

Step-by-Step Instructions for Making a System Modal Window

1. Start Visual Basic, or if Visual Basic is already running, choose New Project from the File menu (ALT, F, N). Form1 is created by default.

2. Add the following code to the General Declarations section of Form1:

```
' Enter the following Declare statement as one, single line:
Declare Function SetSysModalWindow% Lib "User" (ByVal hwnd%) As
   Const HourGlass = 11
```

3. Add the following code to the Form_Load event of Form1:

```
Sub Form1_Load ()
   Me.Show
   Screen.MousePointer = HourGlass
   ' Remove the following line to see how the mouse behaves without it
   result% = SetSysModalWindow%(form1.hWnd)
End Sub
```

4. Add the following code to the Form_Click event of Form1:

```
Sub Form1_Click ()
   Unload Form1
End Sub
```

5. Run the program. Click the form to end it. Notice that the cursor remains an hourglass even when you move the mouse pointer over other windows.

For more information about the SetSysModalWindow Windows API function, see the Windows version 3.1 SDK help file that ships with the Professional

Edition of Visual Basic for Windows.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgWindow

How to Right Justify Items in List Box w/ Tabs & SendMessage

Article ID: Q110958

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The sample program below shows how to right justify items in a list box.

MORE INFORMATION

This program calls the SendMessage Windows API function to set a tab stop at every character position in the list box. The program prefixes the appropriate number of tabs to right justify each string in the list box. You need to set the maximum allowed string length in the program.

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a large list box (List1) to Form1.

3. Add the following to the Form Load event code:

```
Sub Form_Load ()

    Const WM_USER = &H400
    Const LB_SETTABSTOPS = WM_USER + 19
    Const maxlen = 10 ' Maximum expected string length in list box.
    tabchar = Chr$(9) ' ASCII code for a tab
    ReDim a$(maxlen)  ' String array to right justify in list box.
    form1.Show        ' Must Show form in Load event before Print
                      ' will become visible.

    ' GetDialogBaseUnits() API function lets you calculate the average
    ' width of characters in the system font.
    bu& = GetDialogBaseUnits()
    hiword = bu& \ (2 ^ 16)    ' 16 pixels high in default system font.
    loword = bu& And &HFFFF&   ' 8 pixels wide in default system font.
    Print "System font width and height, in pixels:   " & loword, hiword

    'Assign the array of defined tab stops.
    Static tabs(1 To maxlen) As Integer
    For j = 1 To maxlen  ' Set tabs every 4 dialog units (one character):
       tabs(j) = (loword * j) / 2
       ' On most Windows systems, you need only this: tabs(j) = j * 4
    Next
```

```
      'Send message to the List1 control through the Windows message queue:
      retVal& = SendMessage(List1.hWnd, LB_SETTABSTOPS, maxlen, tabs(1))

      For j = 1 To maxlen
         a$(j) = String$(j, "a") ' Assign an arbitrary character string.
         ' Add the appropriate number of tabstops to right justify:
         tabstring = String$(maxlen + 1 - Len(a$(j)), Chr$(9))
         List1.AddItem tabstring & a$(j)
      Next

   End Sub
```

4. Add the following Windows API declarations to the General Declarations section:

```
   Declare Function GetDialogBaseUnits Lib "User" () As Long
   ' Enter the following Declare statement on one, single line:
   Declare Function SendMessage Lib "user" (ByVal hWnd As Integer,
      ByVal wMsg As Integer, ByVal wp As Integer, lp As Any) As Long
```

5. Start the program, or press the F5 key. All strings are right-justified in the list box. Close the form to end the program.

Additional reference words: 3.00 alignment right-align align
KBCategory:
KBSubcategory: APrgOther PrgCtrlsStd

How to Get a Window's Class Name and Other Window Attributes

Article ID: Q112649

The information in this article applies to:

- Standard and Professional Editions of Visual Basic for Windows, version 3.0

SUMMARY

You could use SPY.EXE, which comes with Microsoft Visual C/C++, to get information such as a window's class name. However, this article shows by example how you, the Visual Basic programmer, can create your own Visual Basic application that does the same thing -- displays a window's class name along with several other attributes. You can use this Visual Basic application to find a window's class name anytime you need it. For example, you might need a window's class name for use in a function in the Microsoft Windows Application Programming Interface (Windows API).

MORE INFORMATION

This example uses several functions from the Windows API to get information about the window the cursor is currently over. First, the routine calls GetCursorPos to get the current position of the cursor. Then it calls WindowFromPoint to get the handle to the window the cursor is currently over. Then it calls several other functions in the Windows API to get specific information pertaining to the window.

The example finds the following information about the window:

- Window Handle
- Window Text
- Window Class Name
- Window Style
- Window ID Number
- Parent Window Handle (if applicable)
- Parent Window Text (if applicable)
- Parent Window Class Name (if applicable)
- Module File Name

The example can be easily expanded to get other window attributes by calling appropriate functions in the Windows API.

Step-by-Step Example

This example creates a Visual Basic program that produces results similar to those produced by SPY.EXE.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the following declarations to the general declarations section of

Form1:

```
' Enter each of the following Declare statements on one, single line:
Declare Sub GetCursorPos Lib "User" (lpPoint As Long)
Declare Function WindowFromPoint Lib "User" (ByVal ptScreen As Any)
   As Integer
Declare Function GetModuleFileName Lib "Kernel"
   (ByVal hModule As Integer, ByVal lpFilename As String,
   ByVal nSize As Integer) As Integer
Declare Function GetWindowWord Lib "User" (ByVal hWnd As Integer,
   ByVal nIndex As Integer) As Integer
Declare Function GetWindowLong Lib "User" (ByVal hWnd As Integer,
   ByVal nIndex As Integer) As Long
Declare Function GetParent Lib "User" (ByVal hWnd As Integer) As Integer
Declare Function GetClassName Lib "User" (ByVal hWnd As Integer,
   ByVal lpClassName As String, ByVal nMaxCount As Integer) As Integer
Declare Function GetWindowText Lib "User" (ByVal hWnd As Integer,
   ByVal lpString As String, ByVal aint As Integer) As Integer

Const GWW_HINSTANCE = (-6)
Const GWW_ID = (-12)
Const GWL_STYLE = (-16)
```

3. Add a timer control (Timer1) to the form.

4. Set the interval property of the timer to 200.

5. Add the following code to timer's timer event:

```
Sub Timer1_Timer()
   Dim ptCursor As Long
   Dim sWindowText As String * 100
   Dim sClassName As String * 100
   Dim hWndOver As Integer
   Dim hWndParent As Integer
   Dim sParentClassName As String * 100
   Dim wID As Integer
   Dim lWindowStyle As Long
   Dim hInstance As Integer
   Dim sParentWindowText As String * 100
   Dim sModuleFileName As String * 100
   Static hWndLast As Integer

   Call GetCursorPos(ptCursor)               ' Get cursor position
   hWndOver = WindowFromPoint(ptCursor)      ' Get window cursor is over

   If hWndOver <> hWndLast Then              ' If changed update display
      hWndLast = hWndOver                    ' Save change
      Cls                                      ' Clear the form
      Print "Window Handle: &H"; Hex(hWndOver) ' Display window handle

      r = GetWindowText(hWndOver, sWindowText, 100)      ' Window text
      Print "Window Text: " & Left(sWindowText, r)

      r = GetClassName(hWndOver, sClassName, 100)         ' Window Class
      Print "Window Class Name: "; Left(sClassName, r)
```

```
         lWindowStyle = GetWindowLong(hWndOver, GWL_STYLE)   ' Window Style
         Print "Window Style: &H"; Hex(lWindowStyle)
         ' Get handle of parent window:
         hWndParent = GetParent(hWndOver)
         ' If there is a parent get more info:
         If hWndParent <> 0 Then
            ' Get ID of window:
            wID = GetWindowWord(hWndOver, GWW_ID)
            Print "Window ID Number: &H"; Hex(wID)
            Print "Parent Window Handle: &H"; Hex(hWndParent)
            ' Get the text of the Parent window:
            r = GetWindowText(hWndParent, sParentWindowText, 100)
            Print "Parent Window Text: " & Left(sParentWindowText, r)
            ' Get the class name of the parent window:
            r = GetClassName(hWndParent, sParentClassName, 100)
            Print "Parent Window Class Name: "; Left(sParentClassName, r)
         Else
            ' Update fields when no parent:
            Print "Window ID Number: N/A"
            Print "Parent Window Handle: N/A"
            Print "Parent Window Text : N/A"
            Print "Parent Window Class Name: N/A"
         End If
         ' Get window instance:
         hInstance = GetWindowWord(hWndOver, GWW_HINSTANCE)
         ' Get module file name:
         r = GetModuleFileName(hInstance, sModuleFileName, 100)
         Print "Module: "; Left(sModuleFileName, r)
      End If
   End Sub
```

6. Save the project files.

7. Run the program, and move the mouse over different windows. You will see the field values change as you move the mouse over different windows.

REFERENCES
==========

- Microsoft Windows Software Development Kit (SDK)
- Microsoft Visual Basic for Windows SDK Help file

Additional reference words: ClassName FindWindow 3.00
KBCategory:
KBSubcategory: APrgWindow

How To Add a Scalable Font to Windows From Visual Basic

Article ID: Q112672

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article describes how to add a scalable font resource to Windows from Visual Basic for Windows by calling four Windows Application Programming Interface (Windows API) functions:

- CreateScalableFontResource
- AddFontResource
- WriteProfileString
- SendMessage

MORE INFORMATION

To make changes to the Windows font table, call CreateScalableFontResource to create a font resource file. Then call AddFontResource to add the resource you just created to the Windows font table. Next, make it permanent by calling WriteProfileString to make a change to the WIN.INI file. Finally, call SendMessage to tell all applications currently running that a change has been made to the file.

Step-by-Step Example

This example shows how to add a scalable font resource to Windows from Visual Basic.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the following to the general declarations section of Form1:

```
' Enter each of the following Declare statements on one, single line:
Declare Function CreateScalableFontResource% Lib "GDI"
  (ByVal fHidden%, ByVal lpszResourceFile$,
   ByVal lpszFontFile$, ByVal lpszCurrentPath$)
Declare Function AddFontResource Lib "GDI"
  (ByVal lpFilename As Any) As Integer
Declare Function WriteProfileString Lib "Kernel"
  (ByVal lpApplicationName As String, ByVal lpKeyName As String,
   ByVal lpString As String) As Integer
Declare Function SendMessage Lib "User" (ByVal hWnd As Integer,
   ByVal wMsg As Integer, ByVal wParam As Integer, lParam As Any)
   As Long
```

3. Add a command button (Command1) to Form1.

4. Add the following code to the Command1_Click event:

```
Sub Command1_Click()
   ' Initialize variables for calls to APIs.
   ' Set name of font to show up in font list:
   keyname$ = "Bookman Old Style Bold (TrueType)"
   ' Set name of font resource file:
   font$ = "C:\WINDOWS\SYSTEM\BOOKOSB.FOT"
   TTF_Font$ = "bookosb.ttf"
   ResPath$ = "c:\WINDOWS\SYSTEM"
   ' Initialize variables for SendMessage call:
   HWND_BROADCAST = &HFFFF
   WM_FONTCHANGE = &H1D
   ' Create the font resource file:
   result& = CreateScalableFontResource%(0, font$, TTF_Font$, ResPath$)
   If result& Then
      ' Add resource to Windows font table:
      result& = AddFontResource(font$)
      If result& Then
         ' Make changes to WIN.INI to reflect new font:
         result& = WriteProfileString("Fonts", keyname$, font$)
         If result& Then
            ' Let other applications know of the change:
            result& = SendMessage(HWND_BROADCAST, WM_FONTCHANGE, 0, 0&)
         Else
            ' Report error:
            MsgBox "Error Adding Entry to Win.Ini: " + Format$(result&)
         End If
      Else
         ' Report error:
         MsgBox "Error Adding Font: " + Format$(result&)
      End If
   Else
      ' Report error:
      MsgBox "Error Creating Scalable font: " + Format$(result&)
   End If

End Sub
```

5. Run the program. Click the Command1 button to end the program. If no errors occured, you should now be able to see the font "Bookman Old Style Bold (TrueType)" listed in the font list for Windows. To look at this list, choose the Fonts Panel in the Windows Control Panel. You should see the name listed in the list of installed fonts.

   If you receive this error:

      Error Creating Scalable font:0

   it's because the font is already loaded. Go into the Font List in the Control Panel and remove the font "Bookman Old Style Bold (TrueType)." Then run the program again.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgWindow

How to Pass & Return Unsigned Integers to DLLs from VB

Article ID: Q112673

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

Visual Basic supports signed integers, not unsigned integers. Therefore, the valid range of values, for an integer variable, is from -32767 to +32767.

The C language supports unsigned integers, which have a range from 0 to 65536. To pass a value within the range 32767 to 65536, you need to do a conversion in code to see the correct results. This article shows you how.

MORE INFORMATION

Visual Basic stores its integer variables in an 8-bit data field, as does C. Visual Basic uses signed integers only, so it reserves one of the bits as a sign bit. In C, you have the choice of an unsigned integer (the variable ranges from 0 to 65536) or a signed integer (the variable ranges from -32767 to +32767 as do Visual Basic integer variables).

Step-by-Step Example

Follow a process similar to the following to pass a value greater than 32767 as an integer from Visual Basic to a dynamic link library (DLL) that is expecting an unsigned integer or to return an integer value that is outside the range of valid Visual Basic integers:

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the following code to the general declarations section of Form1. Note that you must actually have a DLL that takes an unsigned integer as a parameter.

```
' MyLong is a function in a DLL that takes an unsigned integer as a
' parameter and returns the same value passed in. To run this sample you
' will have to create the MYLONG function. Enter the following Declare
' statement as one, single line:
Declare Function MyLong Lib "MyLong.DLL" (ByVal iInt AS Integer)
   As Integer
```

3. Add a command button (Comamnd1) to Form1.

4. Add the following code to the Command1_Click event:

```
Sub Command1_Click()
```

```
   Dim lValue As Long
   Dim i As Integer, w As Integer
   ' Initialize lvalue:
   lValue = 40000
   If lValue > 32767 Then
      w = lValue - 65536
   Else
      w = lValue     ' Just pass it on
   End If
   ' Call a DLL that is expecting an unsigned integer.
   ' For this example, the MyLong function will return
   ' the same value passed in.
   i = MyLong(w)

   ' Convert returned value:
   If i < 0 Then
      lValue = 65536 + i
   Else
      lValue = i
   End If
   ' Display the results:
   Print Str(lValue)
End Sub
```

5. Run the program.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgOther

How to use functions in VER.DLL -- a Sample Application

Article ID: Q112731

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

This article contains code and instructions that show you how to create a sample application that uses some of the functions in VER.DLL to retrieve file information embedded into a file with the resource compiler (RC.EXE).

This Visual Basic application was modeled after the Verstamp sample program included in the Microsoft Windows Software Development Kit (SDK).

Additional information can be found in the following documents:

- Microsoft Windows 3.1 Programmer's Reference, Vols. 2 & 3
- Microsoft SDK Reference Help File (WIN31WH.HLP) from Windows 3.1 SDK or the Professional Edition of Visual Basic version 2.0 or 3.0.

MORE INFORMATION

Instead of offering this article in a number of steps, we have modified our usual format to make it easier for you to create and use this Visual Basic application. Therefore, the three files you need (VERINFO.BAS, VERINFO1.FRM, VERINFO2.FRM) are listed below so you can easily copy them into a text editor and save them as separate files. Instructions for how to use the files are embedded in the files as comments.

VERINFO.BAS

```
' Place the following code in a single text file called VERINFO.BAS
'
' The Global constants below are defined in the VER.H header file, also
' included in Microsoft C/C++ 7.0, and the Windows SDK.
'
' NOTE: After copying this into a file in a text editor, modify each
' Declare statements so that each one uses only one, single line.

Type VS_VERSION
   wLength            As Integer
   wValueLength       As Integer
   szKey              As String * 16 ' "VS_VERSION_INFO"
   dwSignature        As Long        ' VS_FIXEDFILEINFO struct
   dwStrucVersion     As Long
   dwFileVersionMS    As Long
   dwFileVersionLS    As Long
```

```
   dwProductVersionMS As Long
   dwProductVersionLS As Long
   dwFileFlagsMasks   As Long
   dwFileFlags        As Long
   dwFileOS           As Long
   dwFileType         As Long
   dwFileSubType      As Long
   dwFileDateMS       As Long
   dwFileDateLS       As Long
End Type

Declare Function GetFileVersionInfo% Lib "Ver.dll" (ByVal Filename$,
   ByVal dwhandle&, ByVal cbBuff&, ByVal lpvData$)
Declare Function GetFileVersionInfoSize& Lib "Ver.dll"
   (ByVal Filename$, dwhandle&)
Declare Sub hmemcpy Lib "kernel" (hpvDest As Any, hpvSrc As Any,
   ByVal cbBytes&)
Declare Function GetSystemDirectory% Lib "kernel"
   (ByVal Path$, ByVal cbBytes%)

' **** VS_VERSION.dwFileFlags ****
Global Const VS_FF_DEBUG = &H1&
Global Const VS_FF_PRERELEASE = &H2&
Global Const VS_FF_PATCHED = &H4&
Global Const VS_FF_PRIVATEBUILD = &H8&
Global Const VS_FF_INFOINFERRED = &H10&
Global Const VS_FF_SPECIALBUILD = &H20&

' **** VS_VERSION.dwFileOS ****
Global Const VOS_UNKNOWN = &H0&
Global Const VOS_DOS = &H10000
Global Const VOS_OS216 = &H20000
Global Const VOS_OS232 = &H30000
Global Const VOS_NT = &H40000

Global Const VOS__BASE = &H0&
Global Const VOS__WINDOWS16 = &H1&
Global Const VOS__PM16 = &H2&
Global Const VOS__PM32 = &H3&
Global Const VOS__WINDOWS32 = &H4&

Global Const VOS_DOS_WINDOWS16 = &H10001
Global Const VOS_DOS_WINDOWS32 = &H10004
Global Const VOS_OS216_PM16 = &H20002
Global Const VOS_OS232_PM32 = &H30003
Global Const VOS_NT_WINDOWS32 = &H40004

' **** VS_VERSION.dwFileType ****
Global Const VFT_UNKNOWN = &H0&
Global Const VFT_APP = &H1&
Global Const VFT_DLL = &H2&
Global Const VFT_DRV = &H3&
Global Const VFT_FONT = &H4&
Global Const VFT_VXD = &H5&
Global Const VFT_STATIC_LIB = &H7&

' **** VS_VERSION.dwFileSubtype for VFT_WINDOWS_DRV ****
```

```
Global Const VFT2_UNKNOWN = &H0&
Global Const VFT2_DRV_PRINTER = &H1&
Global Const VFT2_DRV_KEYBOARD = &H2&
Global Const VFT2_DRV_LANGUAGE = &H3&
Global Const VFT2_DRV_DISPLAY = &H4&
Global Const VFT2_DRV_MOUSE = &H5&
Global Const VFT2_DRV_NETWORK = &H6&
Global Const VFT2_DRV_SYSTEM = &H7&
Global Const VFT2_DRV_INSTALLABLE = &H8&
Global Const VFT2_DRV_SOUND = &H9&
Global Const VFT2_DRV_COMM = &HA&

' **** VS_VERSION.dwFileSubtype for VFT_WINDOWS_FONT ****
Global Const VFT2_FONT_RASTER = &H1&
Global Const VFT2_FONT_VECTOR = &H2&
Global Const VFT2_FONT_TRUETYPE = &H3&

' **** Global variables used in both forms ****
Global Filename$
Global Directory$
Global FileVer$
Global ProdVer$
Global FileFlags$
Global FileOS$
Global FileType$
Global FileSubType$
```

VERINFO1.FRM

```
' The following is a text dump of the form VERINFO1. It includes the form
' and control description as well as necessary Function and Sub procedures.
' Save the code in a single TEXT file called VERINFO1.FRM and you will
' be able to load it as a form in Visual Basic.
'
' NOTE: To make the code fit in this article, some of the lines are listed
' using multiple lines. After copying the code into a file in a text editor
' modify it to ensure that all lines of code exist as one, single line
' in the file. Otherwise, you will receive errors when loading the form in
' Visual Basic.

Begin Form Form1
   BorderStyle     =   1  'Fixed Single
   Caption         =   "VerInfo Demo"
   ClientHeight    =   4290
   ClientLeft      =   2340
   ClientTop       =   2160
   ClientWidth     =   3855
   Height          =   4695
   Left            =   2280
   LinkMode        =   1  'Source
   LinkTopic       =   "Form1"
   ScaleHeight     =   17.875
   ScaleMode       =   4  'Character
   ScaleWidth      =   32.125
   Top             =   1815
   Width           =   3975
```

```
Begin DriveListBox Drive1
   Height          =   288
   Left            =   1836
   TabIndex        =   7
   Top             =   3792
   Width           =   1908
End
Begin DirListBox Dir1
   Height          =   1884
   Left            =   1830
   TabIndex        =   5
   Top             =   1428
   Width           =   1896
End
Begin FileListBox File1
   Height          =   2955
   Left            =   120
   TabIndex        =   3
   Top             =   984
   Width           =   1575
End
Begin TextBox Text1
   Height          =   288
   Left            =   1092
   TabIndex        =   1
   Text            =   "*.*"
   Top             =   204
   Width           =   2544
End
Begin Label Label1
   Caption         =   "Dri&ves:"
   Height          =   216
   Index           =   4
   Left            =   1830
   TabIndex        =   6
   Top             =   3480
   Width           =   660
End
Begin Label Label1
   Caption         =   "&Directories:"
   Height          =   192
   Index           =   3
   Left            =   1830
   TabIndex        =   4
   Top             =   1104
   Width           =   1236
End
Begin Label Label1
   Caption         =   "c:\"
   Height          =   204
   Index           =   2
   Left            =   1830
   TabIndex        =   8
   Top             =   648
   Width           =   1884
End
Begin Label Label1
```

```
      Caption         =   "&Files:"
      Height          =   204
      Index           =   0
      Left            =   120
      TabIndex        =   2
      Top             =   648
      Width           =   612
   End
   Begin Label Label1
      Caption         =   "File&Name:"
      Height          =   204
      Index           =   1
      Left            =   120
      TabIndex        =   0
      Top             =   252
      Width           =   936
   End
End

Sub Dir1_Change ()
   File1.Path = Dir1.Path
   Label1(2).Caption = File1.Path
End Sub

Sub DisplayVerInfo ()
   Dim X As VS_VERSION

   '*** Get Version Info ****
   FileVer$ = "": ProdVer$ = "": FileFlags$ = ""
   FileOS$ = "": FileType$ = "": FileSubType$ = ""
   FileName$ = File1.List(File1.ListIndex)
   Directory$ = Label1(2).Caption
   FullFileName$ = Label1(2).Caption + "\" + FileName$
   BufSize& = GetFileVersionInfoSize(FullFileName$, dwHandle&)
   If BufSize& = 0 Then
      MsgBox "No Version Info available!"
      Exit Sub
   End If
   lpvData$ = Space$(BufSize&)
   r% = GetFileVersionInfo(FullFileName$, dwHandle&, BufSize&, lpvData$)
   hmemcpy X, ByVal lpvData$, Len(X)

   '**** Determine File Version number ****
   FileVer$ = LTrim$(Str$(HIWORD(X.dwFileVersionMS))) + "."
   FileVer$ = FileVer$ + LTrim$(Str$(LOWORD(X.dwFileVersionMS))) + "."
   FileVer$ = FileVer$ + LTrim$(Str$(HIWORD(X.dwFileVersionLS))) + "."
   FileVer$ = FileVer$ + LTrim$(Str$(LOWORD(X.dwFileVersionLS)))

   '**** Determine Product Version number ****
   ProdVer$ = LTrim$(Str$(HIWORD(X.dwFileVersionMS))) + "."
   ProdVer$ = ProdVer$ + LTrim$(Str$(LOWORD(X.dwProductVersionMS))) + "."
   ProdVer$ = ProdVer$ + LTrim$(Str$(HIWORD(X.dwProductVersionLS))) + "."
   ProdVer$ = ProdVer$ + LTrim$(Str$(LOWORD(X.dwProductVersionLS)))

   '**** Determine Boolean attributes of File ****
   If X.dwFileFlags And VS_FF_DEBUG Then FileFlags$ = "DeBug"
   If X.dwFileFlags And VS_FF_PRERELEASE Then FileFlags$ =
```

```
   FileFlags$ + "PreRel"
If X.dwFileFlags And VS_FF_PATCHED Then FileFlags$ =
   FileFlags$ + "Patched"
If X.dwFileFlags And VS_FF_PRIVATEBUILD Then FileFlags$ =
   FileFlags$ +  "Private"
If X.dwFileFlags And VS_FF_INFOINFERRED Then FileFlags$ =
   FileFlags$ + "Info"
If X.dwFileFlags And VS_FF_DEBUG Then FileFlags$ =
   FileFlags$ + "Special"

If X.dwFileFlags And &HFFFFFF00 Then FileFlags$ = FileFlags$ + "Unknown"

'**** Determine OS for which file was designed ****
Select Case X.dwFileOS
   Case VOS_DOS_WINDOWS16
      FileOS$ = "DOS-Win16"
   Case VOS_DOS_WINDOWS32
      FileOS$ = "DOS=Win32"
   Case VOS_OS216_PM16
      FileOS$ = "OS/2-16 PM-16"
   Case VOS_OS232_PM32
      FileOS$ = "OS/2-32 PM-32"
   Case VOS_NT_WINDOWS32
      FileOS$ = "NT-Win32"
   Case Else
      FileOS$ = "Unknown"
End Select


'**** Determine Type and SubType of File ****
Select Case X.dwFileType
   Case VFT_APP
      FileType$ = "App"
   Case VFT_DLL
      FileType$ = "DLL"
   Case VFT_DRV
      FileType$ = "Driver"
      Select Case X.dwFileSubType
         Case VFT2_DRV_PRINTER
            FileSubType$ = "Printer drv"
         Case VFT2_DRV_KEYBOARD
            FileSubType$ = "Keyboard drv"
         Case VFT2_DRV_LANGUAGE
            FileSubType$ = "Language drv"
         Case VFT2_DRV_DISPLAY
            FileSubType$ = "Display drv"
         Case VFT2_DRV_MOUSE
            FileSubType$ = "Mouse drv"
         Case VFT2_DRV_NETWORK
            FileSubType$ = "Network drv"
         Case VFT2_DRV_SYSTEM
            FileSubType$ = "System drv"
         Case VFT2_DRV_INSTALLABLE
            FileSubType$ = "Installable"
         Case VFT2_DRV_SOUND
            FileSubType$ = "Sound drv"
         Case VFT2_DRV_COMM
```

```
               FileSubType$ = "Comm drv"
            Case VFT2_UNKNOWN
               FileSubType$ = "Unknown"
         End Select
      Case VFT_FONT
         FileType$ = "Font"
         Select Case X.dwFileSubType
            Case VFT_FONT_RASTER
               FileSubType$ = "Raster Font"
            Case VFT_FONT_VECTOR
               FileSubType$ = "Vector Font"
            Case VFT_FONT_TRUETYPE
               FileSubType$ = "TrueType Font"
         End Select
      Case VFT_VXD
         FileType$ = "VxD"
      Case VFT_STATIC_LIB
         FileType$ = "Lib"
      Case Else
         FileType$ = "Unknown"
   End Select

   Form2.Show 1
End Sub

Sub Drive1_Change ()
   Dir1.Path = Drive1.Drive
   File1.Path = Dir1.Path
   Label1(2).Caption = File1.Path
End Sub

Sub File1_Click ()
   Text1.Text = File1.List(File1.ListIndex)
End Sub

Sub File1_DblClick ()
   DisplayVerInfo
End Sub

Sub File1_PathChange ()
   Text1.Text = "*.*"
   File1.Pattern = "*.*"
End Sub

Sub Form_Load ()
   Dim Buffer$

   ' **** Set Default Dir to Windows System Subdirectory ****
   Buffer$ = Space$(256)
   r% = GetSystemDirectory(Buffer$, Len(Buffer$))
   Dir1.Path = Buffer$
   File1.Path = Buffer$
   Drive1.Drive = Left$(Buffer$, 1)
End Sub

Function HIWORD (X As Long) As Integer
   HIWORD = X \ &HFFFF&
```

```
End Function

Function LOWORD (X As Long) As Integer
   LOWORD = X And &HFFFF&
End Function

Sub Text1_KeyPress (KeyAscii As Integer)
   If KeyAscii = 13 Then
      File1.Pattern = Text1.Text
      KeyAscii = 0
      If File1.ListCount = 1 Then DisplayVerInfo
      If File1.ListCount = 0 Then
         MsgBox "Invalid Filename"
         File1.Pattern = "*.*"
         Text1.Text = "*.*"
      End If
      File1.SetFocus
   End If
End Sub
```

VERINFO2.FRM

```
' The following is a text dump of the form VERINFO2. It includes the form
' and control description as well as necessary Function and Sub procedures.
' Save the code in a single text file called VERINFO2.FRM and you will
' be able to load it as a form in Visual Basic.
'
VERSION 2.00
Begin Form Form2
   BorderStyle     =   1  'Fixed Single
   Caption         =   "File Version Info"
   ClientHeight    =   3345
   ClientLeft      =   6630
   ClientTop       =   2175
   ClientWidth     =   4500
   FontBold        =   0   'False
   FontItalic      =   0   'False
   FontName        =   "MS Sans Serif"
   FontSize        =   8.25
   FontStrikethru  =   0   'False
   FontUnderline   =   0   'False
   Height          =   3750
   Left            =   6570
   LinkMode        =   1  'Source
   LinkTopic       =   "Form3"
   MaxButton       =   0   'False
   MinButton       =   0   'False
   ScaleHeight     =   13.938
   ScaleMode       =   4  'Character
   ScaleWidth      =   37.5
   Top             =   1830
   Width           =   4620
   Begin CommandButton Command1
      Caption         =   "OK"
      Height          =   372
```

```
      Left            =   1680
      TabIndex        =   0
      Top             =   2880
      Width           =   1452
   End
End

Sub Command1_Click ()
   Form2.Hide
End Sub

Sub Command1_GotFocus ()
  Form_Paint
End Sub

Sub Form_Paint ()
   Form2.CurrentX = 2
   Form2.CurrentY = 1
   Form2.Print "FileName:"
   Form2.CurrentX = 2
   Form2.Print "Directory:"
   Form2.CurrentX = 2
   Form2.Print "File Version:"
   Form2.CurrentX = 2
   Form2.Print "Product Version:"
   Form2.CurrentX = 2
   Form2.Print "File Flags:"
   Form2.CurrentX = 2
   Form2.Print "File OS:"
   Form2.CurrentX = 2
   Form2.Print "File Type:"
   Form2.CurrentX = 2
   Form2.Print "File Sub-type:"

   Form2.CurrentX = 18
   Form2.CurrentY = 1
   Form2.Print FileName$
   Form2.CurrentX = 18
   Form2.Print Directory$
   Form2.CurrentX = 18
   Form2.Print FileVer$
   Form2.CurrentX = 18
   Form2.Print ProdVer$
   Form2.CurrentX = 18
   Form2.Print FileFlags$
   Form2.CurrentX = 18
   Form2.Print FileOS$
   Form2.CurrentX = 18
   Form2.Print FileType$
   Form2.CurrentX = 18
   Form2.Print FileSubType$
End Sub
```

How to Create and Run the Program
---

1. Start Visual Basic. Form1 is created by default.

2. From the File menu, choose Remove File to remove Form1.

3. From the File menu, choose Add File... to and add VERINFO.BAS

4. Repeat step 3 to add VERINFO1.FRM and VERINFO2.FRM to the project.

5. From the Options menu, choose Project... and set Start Up Form
   to Form1.

6. Run the application.

Additional reference words: 2.00 3.00 codesmpl
KBCategory: APrg
KBSubCategory: APrgWindow

ODBC Setup Program Gives Error: Could not open file...

Article ID: Q95736

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 2.0

SUMMARY
=======

Running the Data Access Setup program from Visual Basic version 2.0 for Windows into the ODBC directory that Visual Basic Setup created results in this error message:

> Could not open the file named: C:\VB\ODBCADM.HLP. It is in use by another application.

C:\VB is the path where Visual Basic exists. At this point you must quit the Setup program.

The ODBC Setup program is trying to copy the ODBCADM.HLP file on top of itself. The ODBC setup files already reside in the directory C:\VB\ODBC.

To work around the problem, choose the default directory (C:\ODBC) or any other subdirectory. Then the Setup program works correctly. After installing ODBC, you can move the contents of the directory to any other directory.

This is not a problem with Visual Basic, but rather a limitation of the ODBC Setup program.

More Information

The following steps reproduce the problem:

1. Set up Visual Basic version 2.0 for Windows in C:\VB.

2. Double-click the Data Access Setup icon to start the Setup program.

3. Choose continue.

4. Select Install ODBC Administration Utility.

5. On the choice entitled "The Microsoft ODBC administration utility will be copied into the following directory on your hard disk:" Change the default path from C:\ODBC to C:\VB\ODBC

You should receive the error message "Could not open the file named: 'C:\VB\ODBCADM.HLP' followed by a dialog asking if you want to quit setup. You will need to select Yes to this dialog in order to terminate the ODBC setup program.

Additional reference words: 2.00 setup ODBC
KBCategory:

KBSubcategory: APrgDataODBC

How to Keep the Current Record the Same After Using Refresh

Article ID: Q97181

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

In Visual Basic version 3.0 for Windows, when the Refresh method updates the recordset for a data control, it recreates the recordset and resets the current record. This invalidates all existing bookmarks for that recordset. This behavior is by design. It is not a Visual Basic bug but rather a design feature of the data control.

However, this behavior may be undesirable if you want to refresh the recordset and maintain the current record. This article explains how to restore the current record after executing the Refresh method.

MORE INFORMATION

This information is included with the Help file provided with Microsoft Visual Basic version 3.0 for Windows.

Although there is no simple way to retain the current record after executing the Refresh method, you can restore the current record. To do so, store unique field data for the current record. Then use the stored field data to execute the Refresh method followed by the FindFirst method. The FindFirst method uses the stored field data to restore the current record.

The following steps demonstrate how to restore the current record after executing the Refresh method:

1. Start Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Put a data control (Data1) on Form1.

3. Set the DatabaseName property for Data1 to <path name>BIBLIO.MDB where <path name> represents the full path to the Visual Basic BIBLIO.MDB sample database.

4. Set the RecordSource property of Data1 to Authors, which is the name of the table in the BIBLIO.MDB database.

5. Put a Text box (Text1) on Form1

6. Set the DataSource property of Text1 to Data1

7. Set the DataField property of Text1 to Author, which is the name of the field (column) in the Authors table.

8. Put a command button (Command1) on Form1

9. Change the Caption property of Command1 to Refresh.

10. Add the following code to the Command1_Click event

```
Sub Command1_Click ()

   Dim CurrRec As Variant

   'Hide the text box and emulate it by drawing a border
   text1.Visible = False
   Line (text1.Left, text1.Top)-(text1.Left + text1.Width,
            text1.Top + text1.Height), , B

   'Store the value of a unique field for the current record
   CurrRec = Data1.RecordSet!Au_ID

   'Update the RecordSet
   Data1.Refresh

   'Restore the current record by using the stored field value
   'to find
   Data1.RecordSet.FindFirst "Au_ID = " & CurrRec

   text1.Visible = True

End Sub
```

11. From the Run menu, choose Start (ALT, R, S) or press the F5 key to run the program.

12. Using the data control, move to the next record. You should see "Atre, Shaku" displayed in the text box

13. Using the data control, move further into the file. To do this, click the right arrow or click the rightmost button -- the one with the arrow and bar -- to move to the end of the file.

14. Click the Refresh button. The name of the first author in the recordset is displayed in Text1 for an instant. Then the current author is redisplayed.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

How to Copy Current Database Record into a Record Variable

Article ID: Q97413

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

Although Visual Basic version 3.0 for Windows does not provide a direct way to assign the current database record to a record variable, this article gives you a generic routine. Using this generic routine, you can assign the current record, containing any number of fields, to a record variable that represents the structure of the current database record.

This generic routine is useful if you have existing database code that uses record variables to represent database records. For example, using this routine, you can use the Visual Basic data access features without making major changes to how you read and handle records. After you assign the contents of the current record to a record variable of the appropriate type, your code can manipulate the record as before, independent of the underlying database.

The routine demonstrated below requires Windows version 3.1 or later because it uses the Windows API function hmemcpy(), which was introduced in Windows version 3.1. An error will result on the call to hmemcpy() if you attempt to run the sample using Windows version 3.0.

MORE INFORMATION

This information is included with the Help file provided with Microsoft Visual Basic version 3.0 for Windows.

Follow these general steps to assign the current database record to a record variable:

1. Define a Type ... End Type structure that represents the record structure of the database table that you are going to use. This requires that the number and data types of the fields in the table be known in advance.

   To determine the structure of the table quickly, run the Data Manager tool provided with Visual Basic. From the Data Manager File menu, choose Open to open the database. Select a Table from the list displayed in the Database window, and choose the Design button to see the table's field names, data types, and field lengths.

2. Dimension a variable of the user-defined type structure created in step 1.

3. Create a generic routine using the Windows API hmemcpy() function to copy each field of the current database record into a string. To do this, step through all of the fields in the Fields collection and accumulate the fields together into a single string.

4. Use the hmemcpy() function to copy the contents of the string created in step 3 to the record variable created in step 2.

Perform the following steps to create an example application that demonstrates how to copy the current database record into a user-defined structure. This example shows you how to use the Data control to copy a record from the BIBLIO.MDB sample database provided with Visual Basic.

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add two text boxes (Text1 and Text2) to Form1

3. Add a data control (Data1) to Form1

4. Add a command button (Command1) to Form1.

5. Using the following table as a guide, set the properties of the controls you added in steps 2, 3 and 4.

Control Name	Property	New Value	Comment
Command1	Caption	"Copy"	
Data1	DatabaseName	BIBLIO.MDB	You will also need to provide the full path to this file, which should be in your Visual Basic directory C:\VB
Data1	RecordSource	Authors	
Text1	DataSource	Data1	
Text1	DataField	AU_ID	
Text2	DataSource	Data1	
Text2	DataField	Author	

6. From the File menu, choose New Module (ALT, F, M). Module1 is created.

7. Add the following code to the general declarations section of Module1:

```
Type typeAuthor
   AU_ID As Long
   Author As String * 255
End Type
' Enter the following Declare on a single line:
Declare Sub hmemcpy Lib "KERNEL" (dest As Any, src As Any, ByVal
                                  Size As Long)
```

8. Add the following code to Module1:

```
Function GetCurrRec (ds As Dynaset) As String

   Dim i As Integer
   Static FieldStr As String
```

```
Static recStr As String

recStr = ""

'Step through each field in the current record and accumulate
'the contents of each field into a string
For i = 0 To ds.Fields.Count - 1

   'Pad out to the right size
   FieldStr = Space(ds.Fields(i).Size)

   Select Case ds.Fields(i).Type

      'Copy the binary representation of the field to a
      'string (FieldStr)

      Case 1, 2       'Bytes
         hmemcpy ByVal FieldStr, CInt(ds.Fields(i).Value),
                                                 ds.Fields(i).Size

      Case 3          'Integers
         hmemcpy ByVal FieldStr, CInt(ds.Fields(i).Value),
                                                 ds.Fields(i).Size

      Case 4          'Long integers
         hmemcpy ByVal FieldStr, CLng(ds.Fields(i).Value),
                                                 ds.Fields(i).Size

      Case 5          'Currency
         hmemcpy ByVal FieldStr, CCur(ds.Fields(i).Value),
                                                 ds.Fields(i).Size

      Case 6          'Singles
         hmemcpy ByVal FieldStr, CSng(ds.Fields(i).Value),
                                                 ds.Fields(i).Size

      Case 7, 8       'Doubles
         hmemcpy ByVal FieldStr, CDbl(ds.Fields(i).Value),
                                                 ds.Fields(i).Size

      Case 9, 10      'String types
         hmemcpy ByVal FieldStr, ByVal CStr(ds.Fields(i).Value),
                                          Len(ds.Fields(i).Value)

      Case 11, 12     'Memo and long binary
        FieldStr = ds.Fields(i).GetChunk(0, ds.Fields(i).FieldSize())

   End Select

   'Accumulate the field string into a record string
   recStr = recStr & FieldStr

Next

'Return the accumulated string containing the contents of all
'fields in the current record
GetCurrRec = recStr
```

```
   End Function
```

9. Add the following code to the Command1_Click event in Form1:

```
   Sub Command1_Click ()

      Dim recAuthor As typeAuthor
      Dim strCurrRec As String
      Dim strVerify As String

      'Copy the current record in the Authors table to a string
      strCurrRec = GetCurrRec(Data1.RecordSet)

      'Copy the string to the record variable that has a structure
      'matching the struture of the current record in the Authors table
      hmemcpy recAuthor, ByVal strCurrRec, Len(recAuthor)

      'Verify that the correct results were returned by displaying
      'the contents of the record variable
      strVerify = "AU_ID: " & Format$(recAuthor.AU_ID) & Chr$(13)
      strVerify = strVerify & "Author: " & Trim(recAuthor.Author)
      MsgBox strVerify

   End Sub
```

10. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program.

Click the scroll bar of the Data control to select an author. The Text1 box displays the author ID, and the Text2 box displays the author's name. Click the "Copy" button to copy the current author's information to the record variable and see contents of the record variable displayed in a MsgBox.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

How to Use Data Control to Scroll Up and Down in a Recordset

Article ID: Q97414

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The data control provided with Visual Basic does not provide an automatic way to scroll up or down in a recordset by groups (pages) of records. This article shows you how to use the MoveNext and MovePrevious methods to scroll up or down in a recordset by groups (pages) of records without displaying all the records.

MORE INFORMATION

This information is included with the Help file provided with Microsoft Visual Basic version 3.0 for Windows.

Usually, when you use the MoveNext and MovePrevious methods to scroll up or down by a specified number of records, all the records are displayed as you move through them. This is undesirable behavior if you want a way to scroll through the recordset by pages.

In order to display only the record you have scrolled to, without displaying all the records in between, you need to use the Clone method to clone the data control's recordset.

Once you clone the recordset, you can use the MoveNext and MovePrevious methods to move to the desired record within the cloned recordset. Then set the Bookmark property of the original recordset to the Bookmark property of the cloned recordset. This makes the desired record the current record in the original recordset and causes the fields of this record to be displayed in the bound data controls.

Perform the following steps to create an example program that demonstrates how to scroll up and down by pages in a data control's recordset:

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a data control (Data1) to Form1.

3. Add two text boxes (Text1 and Text2) to Form1.

4. Add two command buttons (Command1 and Command2) to Form1.

5. Using the following table as a guide, set the properties of the controls you added in steps 2, 3, and 4.

Control Name	Property	New Value	Comment
Command1	Caption	"Page Up"	
Command2	Caption	"Page Down"	
Data1	DatabaseName	BIBLIO.MDB	Provide the full path to to this file, which should be in the Visual Basic directory -- C:\VB
Data1	RecordSource	Authors	
Text1	DataSource	Data1	
Text1	DataField	AU_ID	
Text2	DataSource	Data1	
Text2	DataField	Author	

6. Add the following code to the general declarations section of Form1:

```
Const PAGEUP = 1
Const PAGEDOWN = 2
Const Records_per_Page = 10
```

7. Add the following procedure to Form1:

```
Sub Page (RecSet As Dynaset, ByVal iDirection As Integer, ByVal
                                                  Records As Integer)

   Dim dsClone As Dynaset
   Dim i As Integer

   'Copy the visible recordset. This is necessary so that you can
   'move through the clone recordset without displaying each record.
   Set dsClone = RecSet.Clone()

   'Set the current record of the cloned recordset to the current
   'record of the visible recordset.
   dsClone.Bookmark = RecSet.Bookmark

   'Scroll up or down N number of records in the cloned recordset.
   i = 1
   Do While i <= Records And Not dsClone.EOF And Not dsClone.BOF

      If iDirection = PAGEUP Then
         dsClone.MovePrevious
      Else
         dsClone.MoveNext
      End If

      i = i + 1
   Loop

   'If the above loop caused a BOF or EOF condition, move to the
   'beginning or end of the recordset as appropriate.
   If dsClone.BOF And iDirection = PAGEUP Then
      dsClone.MoveFirst
   ElseIf dsClone.EOF And iDirection = PAGEDOWN Then
      dsClone.MoveLast
   End If
```

```
      'Change the bookmark of the visible record set to the bookmark
      'of the desired record. This makes the current record of the
      'visible recordset match the record moved to in the cloned
      'dynaset. The fields of the record are displayed in the data
      'bound controls without displaying any intervening records.
      RecSet.Bookmark = dsClone.Bookmark

   End Sub
```

8. Add the following code to the Command1_Click event for Form1:

```
    Sub Command1_Click ()

       'Scroll up 10 records in the recordset associated with Data1
       Page Data1.RecordSet, PAGEUP, Records_per_Page

    End Sub
```

9. Add the following code to the Command2_Click event for Form1:

```
   Sub Command2_Click ()

      'Scroll down 10 records in the recordset associated with Data1
       Page Data1.RecordSet, PAGEDOWN, Records_per_Page

   End Sub
```

10. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program.

Click the "Page Up" or "Page Down" button to scroll up or down in 10-record increments. Change the value of Records_per_Page to modify the pagesize.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

ODBC Setup & Connection Issues for Visual Basic Version 3.0

Article ID: Q97415

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY
=======

There are four possible problem areas that can contribute to a failure to connect to a database server when using ODBC and Visual Basic:

- Having correct .INI file settings.
- Having the correct DLLs in the right place.
- Having the server information needed to connect to a server correctly.
- Meeting the needs of Microsoft and Sybase SQL Servers.

MORE INFORMATION
================

The following describes each of the four areas, giving possible errors and problems that can arise if things are not set up correctly.

INI file settings

There are two .INI files (ODBCINST.INI and ODBC.INI) that must reside in the Windows directory and must contain correct information about the installed ODBC drivers and servers.

ODBCINST.INI contains the ODBC driver information needed to register new servers using the RegisterDataBase() statement in Visual Basic. Here is an example .INI file for the SQL Server driver that ships with Visual Basic:

```
[ODBC Drivers]
SQL Server=Installed

[SQL Server]
Driver=C:\WINDOWS\SYSTEM\sqlsrvr.dll
Setup=C:\WINDOWS\SYSTEM\sqlsrvr.dll
```

The [ODBC Drivers] section tells the driver manager the names of the installed drivers. The [SQL Server] section tells the ODBC driver manager the names of the dynamic link libraries (DLLs) to use to access data from a server set up as a SQL Server. The order of the two sections and their entries is arbitrary.

ODBC.INI contains the data for each installed driver. The driver manager uses this information to determine which DLL to use to access data from a particular database backend. Here is an example of a file containing three data sources all using the SQL Server driver:

```
[ODBC Data Sources]
MySQL=SQL Server
```

```
   CorpSQL=SQL Server

   [MySQL]
   Driver=C:\WINDOWS\SYSTEM\sqlsrvr.dll
   Description=SQL Server on server MySQL
   OemToAnsi=No
   Network=dbnmp3
   Address=\\mysql\pipe\sql\query

   [CorpSQL]
   Driver=C:\WINDOWS\SYSTEM\sqlsrvr.dll
   Description=SQL Server on server CorpSQL
   OemToAnsi=No
   Network=dbnmp3
   Address=\\corpsql\pipe\sql\query
```

The first section tells the driver manager which sections appearing below it define the data source. As you can see, each entry has a value (in this case, SQL Server) that matches a value from the ODBCINST.INI file.

If the information on a data source is incorrect or missing, you may get the following error:

```
   ODBC - SQLConnect failure 'IM002[Microsoft][ODBC DLL] Data source
          not found and no default driver specified'
```

If the DLL listed on the Driver=... line cannot be found or is corrupt, the following error may occur:

```
   ODBC - SQLConnect failure 'IM003[Microsoft][ODBC DLL] Driver
          specified by data source could not be loaded'
```

ODBC and Driver DLLs

The following DLLs must be on the path or in the Windows system directory in order for ODBC to be accessible from Visual Basic:

```
   ODBC.DLL     - driver manager
   ODBCINST.DLL - driver setup manager
   VBDB300.DLL  - Visual Basic programming layer
```

If VBDB300.DLL is missing or corrupt, you see the following error in Visual Basic when you try to run the application:

```
   ODBC Objects require VBDB300.DLL
```

If either the ODBC.DLL or ODBCINST.DLL file is missing or corrupt, you see the following error in Visual Basic when you try to run the application:

```
  Cannot Find ODBC.DLL, File not Found
```

The SQL Server driver requires the following files:

```
   SQLSRVR.DLL  - actual driver
   SQLSETUP.DLL - driver setup routines
   DBNMP3.DLL   - named pipe routines needed by SQL server
```

If the SQLSRVR.DLL is missing or corrupt, you see the following error when calling the OpenDataBase() function with a SQL Server data source:

```
ODBC - SQLConnect failure 'IM003[Microsoft][ODBC DLL] Driver
       specified by data source could not be loaded'
```

If the SQLSETUP.DLL is missing or corrupt, you see the following error when calling the RegisterDataBase statement with SQL Server as the driver name:

```
The configuration DLL (C:\WINDOWS\SYSTEM\SQLSETUP.DLL) for the ODBC
SQL server driver could not be loaded.
```

Server Information Needed to Connect to a Data Source

Certain information is needed to connect to a data source using the OpenDataBase() function. This information is obtainable from the server administrator in the case of SQL Server. The following is an example of a call to the OpenDataBase() function to connect to a SQL Server called CorpSQL as a user named Guest with password set to taco:

```
Dim db As DataBase
Set db = OpenDataBase( "corpsql", False, False, "UID=guest;PWD=taco")
```

If any of this information is missing, an ODBC dialog box appears to give a user a chance to supply the needed data. If the information is incorrect, the following error occurs:

```
ODBC - SQLConnect failure '28000[Microsoft][ODBC SQL Server Driver]
       [SQL Server]  Login failed'
```

Information Specific to Microsoft and Sybase SQL Servers

For Microsoft and Sybase SQL Servers, you need to add stored procedures to the server itself by running a batch file of SQL statements to make a Microsoft or Sybase SQL Server ODBC-aware. In other words, before you can run a Visual Basic ODBC application using the SQL Server driver, you must first update the ODBC catalog of stored procedures. These procedures are provided in the INSTCAT.SQL file. The system administrator for the SQL Server should install the procedures by using the SQL Server Interactive SQL (ISQL) utility.

If the INSTCAT.SQL file is not processed on the server, the following error occurs:

```
ODBC - SQL Connect Failure
"08001" [Microsoft ODBC SQL Server Driver]
'unable to connect to data source'number: 606'
```

To install the catalog stored procedures by using the INSTCAT.SQL file, run INSTCAT.SQL from the command line using ISQL. Do not use the SAF utility provided with SQL Server. Microsoft SAF for MS-DOS and OS/2 is limited to 511 lines of code in a SQL script, and INSTCAT.SQL contains more than 511 lines of code.

Run ISQL from the OS/2 command line using the following syntax. Enter the two lines as one, single line, and do not include the angle braces <>.

```
   ISQL /U <sa login name > /n /P <password> /S <SQL server name> /i
      <drive: \path\INSTCAT.SQL > /o <drive:\path\output file name>

   /U  The login name for the system administrator.
   /n  Eliminates line numbering and prompting for user input.
   /P  Password used for the system administrator. This switch is case
       sensitive.
   /S  The name of the server to set up.
   /i  Provides the drive and fully qualified path for the location of
       INSTCAT.SQL
   /o  Provides ISQL with an output file destination for results including
       error listings.

Here's an example. Enter the following as one, single line:

   ISQL /U sa /n /P squeeze /S BLUEDWARF /i C: \SQL\INSTCAT.SQL /o
        C: \SQL OUTPUT.TXT

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataODBC
```

How to Implement the DLookup Function in Visual Basic

Article ID: Q99704

The information in this article applies to:

- Microsoft Visual Basic Programming System for Windows, version 3.0

SUMMARY

Microsoft Access provides a set of domain, or record set, functions that are useful in getting the value of one field based on criteria involving another field. The DLookup domain function is particularly useful.

Although Visual Basic does not contain the DLookup function, you can write the equivalent using Visual Basic code. This article describes how to implement the DLookup domain function in Visual Basic.

MORE INFORMATION

In Microsoft Access, the DLookup domain function returns the value of a field for a given set of criteria. The syntax for the DLookup function is as follows:

```
DLookup(expr, domain , criteria)
```

Argument	Description
expr	String expression identifying the field that contains the data you want to return. Operands in expr can include the name of a table field.
domain	String expression identifying the records that constitute the record set. It can be a table name, query name, or SQL expression that returns data.
criteria	Optional string expression used to restrict the range of data on which DLookup is performed. For example, criteria could be the SQL expression's WHERE clause without the word WHERE. If criteria is omitted, DLookup evaluates expr against the entire record set.

Step-by-Step to a Custom Visual Basic DLookup Function

The following steps show by example how to create a Visual Basic custom DLookup function.

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add the following controls with the associated properties to Form1:

Control	Name	Property Settings

```
-------------------------------------------------------------
Command Button   Command1   Caption = "Lookup"
Label            Label2
Data             Data1      Databasename = "BIBLIO.MDB"
                            RecordSource = "Authors"
Label            Label1     DataSource = Data1
                            DataField = Author
```

3. Add the following code to the general declarations section of Form1:

```
Dim gDefaultDatabase As Database
```

4. Add the following code to the general section of Form1:

```
'Enter the following two lines as one, single line:
Function DLookup (ByVal FieldName As String, ByVal RecSource
   As String, ByVal Criteria As String) As Variant

   Dim dsResult As Dynaset
   Dim ReturnValue As Variant

   On Local Error GoTo Error_DLookup:

   'Create a dynaset based on the record source or SQL string provided
   Set dsResult = gDefaultDatabase.CreateDynaset(RecSource)

   'Find the first record that meets the criteria provided
   dsResult.FindFirst Criteria

   'See if we found any records
   If Not dsResult.NoMatch Then

      'Return the value of the field
      DLookup = dsResult(FieldName).Value

   Else

       DLookup = Null

   End If

DLookup_Exit:
Exit Function

Error_DLookup:
    'Display the error and get out
    MsgBox "Error (" & Err & "): " & Error(Err) & " in DLookup", 64
    Resume DLookup_Exit:

End Function
```

5. Add the following code to the Command1_Click event procedure:

```
Sub Command1_Click ()

   'Get the first book title for the current author.
   'Enter the following two lines as one, single line:
```

```
      Label2.Caption = DLookup("Title", "Titles", "Au_ID = " &
         Format(data1.Recordset("Au_ID")))
   End Sub
```

6. Add the following code to the Form_Load event procedure of Form1:

```
   'Cause the records to be read from the database. This is
   'needed to initialize the Database property.
   data1.Refresh

   'Keep the default database in a global variable to be used
   'by the DLookup function
   Set gDefaultDatabase = data1.Database
```

7. From the Run menu, choose Start (ALT, R, S) or press F5 to run the program.

8. Click the directional arrows on the Data control to display different author names in Label1.

9. Click the Lookup button and title to display one of the author's books in Label2.

As demonstrated in this example program, you can use DLookup to return a field value such as book title based on the value of another field such as author ID.

Examples Showing How to Use DLookup

Below are some more examples showing how you can use the DLookup function.

In the following example, from the Authors table in the Visual Basic BIBLIO.MDB sample database, DLookup uses the Au_ID field to return the corresponding author name for the author whose ID is 17. Assume that the variable AuthorName is a string.

```
   AuthorName = DLookup("Author", "Authors", "Au_ID = 17")
```

If the criteria argument contains non-numeric text other than field names, you must enclose the text in single quotation marks. In the following example from the Titles table of the BIBLIO.MDB database, ISBN is the name of a field, and 0895886448 is a string literal.

```
   BookTitle1 = DLookup("Title", "Titles", "ISBN = '0895886448'")
   BookTitle2 = DLookup("Title", "Titles", "Au_Id = 17")
```

Even if more than one record satisfies criteria, DLookup returns only one field. If no record satisfies criteria, or if the domain contains no records, DLookup returns a Null.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

PRB: Can't Use ActiveForm to Reference Data Control in VB 3.0

Article ID: Q101252

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SYMPTOMS

Using the ActiveForm Property of the Screen control or an MDI Parent form to reference a Data control causes a "Type Mismatch" error in Visual Basic.

CAUSE

This behavior is by design. This is not a bug in Visual Basic. The Visual Basic environment does not know in advance that the Active form will actually contain a Data control, so it generates a "Type mismatch" error.

WORKAROUND

To avoid the error message, use global objects to reference the local controls. The "More Information" section below demonstrates one method for doing this.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Correct Problem

This example shows how to correct the problem. First, create the problem by following the steps listed in "Steps to Reproduce Problem." Then correct the problem with these steps:

1. Add the following code to the Form_Activate Event:

```
Sub Form_Activate ()
   Set CurrentDS = Data1.Recordset
End Sub
```

2. Change two lines of code into comments by adding a single quotation mark to the beginning of the line. Change the Set CurrentDS statement in the Set_CurrentDS Sub in Module1 to a comment, and do the same to the Call Set_CurrentDS statement in the Form_Click event of Form1.

Steps to Reproduce Problem
---

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a data control (Data1) to Form1.

3. Set the DatabaseName Property of Data1 to BIBLIO.MDB.

4. From the File menu, choose New Module (ALT, F, M). Module1 is created.

5. Add the following code to the General section of Module1:

```
Global CurrentDS As DynaSet
```

6. Add the following code to Module1:

```
Sub Set_CurrentDS ()
   Set CurrentDS = Screen.ActiveForm.Data1.Recordset
End Sub
```

7. Add the following code to the Form_Click event procedure of Form1:

```
Sub Form_Click ()
   Call Set_CurrentDS
End Sub
```

8. From the Run menu, choose start (ALT, R, S) or press the F5 key.

A "Type mismatch" error will occur on the Set statement.

Additional reference words: 3.00 errmsg
KBCategory:
KBSubcategory: APrgDataIISAM PrgCtrlsStd

PRB: Visual Basic 3.0 ODBC Does Not Support OpenTable Method

Article ID: Q101254

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

The OpenTable method is not supported in Visual Basic version 3.0 for ODBC data sources.

CAUSE

Visual Basic version 3.0 introduced a new layer of database management, the Microsoft Access engine, that lies between Visual Basic itself and the ODBC drivers. This new layer allows version 3.0 to work with Microsoft Access, FoxPro, Paradox, and dBASE databases. However, the Microsoft Access engine does not support using OpenTable on ODBC data sources, or any table that is not part of a Microsoft Access database (.MDB). Therefore, when you attempt to use OpenTable on tables that are not Microsoft Access tables or that come from on an ODBC data source, Visual Basic version 3.0 generates the error.

WORKAROUND

You can use CreateDynaset on any table that uses an ISAM or ODBC (attached tables).

STATUS

This behavior is by design. It is documented on page 149 of the Visual Basic version 3.0 "Professional Features Book 2" manual.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataODBC

Transactions on ODBC Data Sources in Visual Basic Version 3.0

Article ID: Q101518

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

Under the ExecuteSQL method, transactions are unsupported against ODBC data sources. Even though a transaction may be started by a Visual Basic application and this transaction space is global to all database objects, the transaction space on database objects does not include actions by the ExecuteSQL method against remote ODBC data sources such as SQL Server or Oracle.

MORE INFORMATION

When you use the ExecuteSQL method or the passthrough property on dynasets, Visual Basic version 3.0 dispatches the SQL code directly to the ODBC data source through the ODBC driver. Therefore, it does not offer any transaction support in terms of CommitTrans or Rollback even though the ODBC driver for that data source might support transactions. This behavior is by design.

The following example illustrates the behavior in Visual Basic version 3.0. The code in the example uses the ExecuteSQL method to delete all rows from the table even though a rollback is issued.

```
Dim D as Database
Set D = Opendatabase ' an ODBC data source such as Oracle
Begintrans
D.Executesql("delete from sometable")
Rollback
```

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataODBC

How to Open dBASE Table with Nonstandard File Extension

Article ID: Q101742

The information in this article applies to:

- Microsoft Visual Basic for Windows, version 3.0

SUMMARY

To open a dBASE table file that has a non-standard file extension, specify the table name as <filename>#<extension>.

MORE INFORMATION

The standard file extension used by dBASE for tables is .DBF. In Visual Basic version 3.0 using the dBASE installable ISAMs, you can open a table by specifying the file name without this extension because the dBASE installable ISAM assumes the extension to be .DBF by default. If you specify the extension <filename>.<extension>, the dBASE installable ISAM will not recognize it and will give you the following error message:

```
<filename>.<extension> isn't a valid name.
```

To open a dBASE table file that has a non-standard file extension, specify the table name as <filename>#<extension>. The dBASE installable ISAM interprets the pound sign (#) in the table name as a period and opens the dBASE table.

Example

The following code example demonstrates how to open a dBASE table file that has a non-standard file extension (AUTHORS.OLD) and print the first field of
all records in the table to the form. The following example assumes that you
have a dBASE III table with a file name of AUTHORS.OLD located in the C:\DBASEIII\OLDBOOKS directory. You may need to modify the example and create a dBASE III database with a table called AUTHORS.OLD in order for it to work correctly.

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a Command Button (Command1) to Form1.

3. Add the following code to the Click event of Command1:

```
Sub Command1_Click()
   Dim db As Database
   Dim OldAuthors As Table
```

```
      Connect$ = "dBASE III"          ' Specify database type
      dbName$ = "C:\DBASEIII\OLDBOOKS"      ' Specify database directory

      Set db = OpenDatabase(dbName$, False, False, Connect$)
      Set OldAuthors = db.OpenTable("Authors#Old") ' Open table
      While Not OldAuthors.EOF
         Print OldAuthors(0)           ' Print field(0) to the form
         OldAuthors.MoveNext           ' for all records.
      Wend

      OldAuthors.Close
      db.Close
   End Sub
```

4. Run the example.

5. Click the Command1 button.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataIISAM

PRB: Error When Updating Fields in Dynaset That Has 2+ Tables

Article ID: Q102681

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

When trying to edit or update fields in a dynaset that was created by a SQL select statement that joined two or more tables, the following errors may occur. In these messages 'item' is a field in a table to be changed.

```
Can't perform operation; it is illegal. (3219)
Can't update 'item'; field not updatable. (3113)
```

CAUSE

These errors occur if the Microsoft Access engine cannot insure that referential integrity of the table entries will be maintained as a result of the operation.

MORE INFORMATION

For a multiple table dynaset to be updatable, the following must be true:

- The dynaset needs to have been created with a SQL 'join' clause between tables that have a one-to-many relationship.

- There must be a unique index (or primary key) on the one-side of the query.

Reproducing the Behavior

These examples use the BIBLIO.MDB database that shipped as a sample database with Visual Basic version 3.0 for Windows. In BIBLIO.MDB, the Authors table has a unique (primary) index set on AU_ID, and the Titles table has an index set on AU_ID but it is not unique or primary. The following code causes the errors:

```
Dim db As database
Dim ds As dynaset
' The following SQL$ code is correct. It will not generate an error.
' Enter it as one, single line:
' SQL$ = "Select * from AUTHORS,TITLES, Titles INNER JOIN
'    authors on Titles.AU_ID = Authors.AU_ID"
' The following line will cause the error, but it won't be generated
' until the last line of the same code -- which is expected:
SQL$ = "Select * from AUTHORS,TITLES where Titles.AU_ID = Authors.AU_ID"
Set db = OpenDatabase("C:\vb3\biblio.mdb")
```

```
   Set ds = db.CreateDynaset(SQL$)
   ds.Edit
```

This is a classic example of a SQL inner join statement. It chooses all fields from both tables where the book titles match up with the author who wrote them. The unique index is the ID number of the author. This means one author can have many titles but books by a single author will have only one author in the Authors table.

If this query did not have a one-to-many relationship, the error, "Can't perform operation; it is illegal"(3219) would occur on the line "ds.Edit." The error is telling you that either there is not a unique index in the multiple-table dynaset, or there is no unambiguous one-side to the query. Checking the updatable property of the dynaset before invoking edit mode avoids the error from attempting to edit a non-updatable dynaset.

After the query is successfully created and the copy buffer is opened by issuing the Edit statement, you can proceed with updating records.

```
   ds.Fields("Title") = "Some new book title"
   ds.Update
```

This works because "Title" is on the non-unique or many-side of the initial query. All the records in the Titles table are editable whereas none of the records in Authors table are editable. The error "Can't update 'item'; field not updatable." (3113) occurs with an attempt to edit any item in the Authors table.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

How to Build Access DB & Load Data from Btrieve for Windows DB

Article ID: Q103440

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

The example in this article demonstrates how to build a Microsoft Access database without having a database or database template already built. The example uses a Btrieve for Windows database file to supply the data to be placed into the newly created Microsoft Access database.

MORE INFORMATION

NOTE: You will need to have a Btrieve for Windows database file already built to test this example. The Btrieve for Windows database file tested with this example can be sent upon request.

Steps to demonstrate the example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add three command buttons and two grid controls using GRID.VBX to Form1. Using the following table as a guide, set the properties of the controls you added in step 2.

Control	Property	New Value
Command1	Caption	"Press to Load Btrieve File and Display in Grid"
Command2	Caption	"Press to Transfer Data and Build New DB"
Command3	Caption	"Press to Display Data from the New Database"
Grid1	Cols	4
Grid1	Rows	15
Grid2	Cols	4
Grid2	Rows	15

3. Review the following brief outline of the table from the Btrieve for Windows database:

   Table Name: Big_Tab

Field Names	Field Type	Field Size
PrimaryKey	Long Integer	
MyMoney	Currency	
MyString	Text	154

Index Names	Index Fields	Unique	Primary

```
   ---------------------------------------------------
   tabindex       +PrimaryKey       Yes      No
```

4. Add the following variables and constants to the (general) section of Form1:

```
   Dim PrimaryKeys(30) As Long
   Dim Money(30) As Currency
   Dim Strings(30) As String * 154
   Const DB_LONG = 4
   Const DB_TEXT = 10
   Const DB_CURRENCY = 5
   Const DB_LANG_GENERAL = ";LANGID=0x0809;CP=1252;COUNTRY=0"
```

5. Add the following code to the Form Load event procedure:

```
   Sub Form_Load ()
      Show
      grid1.ColWidth(1) = 1000      'For PK ID
      grid1.ColWidth(2) = 2000      'For Money
      grid1.ColWidth(3) = 5000      'For Story
      grid1.Col = 1
      grid1.Row = 0
      grid1.Text = "Primary Keys"   'Header for PK ID
      grid1.Col = 2
      grid1.Row = 0
      grid1.Text = "Money"          'Header for Money
      grid1.Col = 3
      grid1.Row = 0
      grid1.Text = "Big String"     'Header for Story
      grid2.ColWidth(1) = 1000      'For PK ID
      grid2.ColWidth(2) = 2000      'For Money
      grid2.ColWidth(3) = 5000      'For Story
      grid2.Col = 1
      grid2.Row = 0
      grid2.Text = "Prime's"        'Header for PK ID
      grid2.Col = 2
      grid2.Row = 0
      grid2.Text = "Your Money"     'Header for Money
      grid2.Col = 3
      grid2.Row = 0
      grid2.Text = "Your Story"     'Header for Story
   End Sub
```

6. Add the following code to the Command1 Click event procedure:

```
   Sub Command1_Click ()
      Dim db As Database
      Dim conn$
      Dim dt As Table
      conn$ = "Btrieve;"
      ' Enter the following Set as one, single line:
      Set db = OpenDatabase("C:\articles\btrvwin\file.ddf", False, False,
         conn$)
      Set dt = db.OpenTable("Big_Tab")
      ' Counter for loading the grid
      For i% = 1 To 10       'Grab the first ten for a test
```

```
      grid1.Col = 1
      grid1.Row = i%
      grid1.Text = dt(0)       'Load the grid
      PrimaryKeys(i%) = dt(0)  'Load the temporary array
      grid1.Col = 2
      grid1.Row = i%
      grid1.Text = dt(1)       'Load the grid
      Money(i%) = dt(1)        'Load the temporary array
      grid1.Col = 3
      grid1.Row = i%
      grid1.Text = dt(2)       'Load the grid
      Strings(i%) = dt(2)      'Load the temporary array
      dt.MoveNext
   Next i%
End Sub
```

7. Add the following code to the Command2 Click event procedure:

```
Sub Command2_Click ()
   Dim newdb As Database
   Dim newtb As Table
   Dim newtd As New tabledef
   Dim newidx As New Index
   Dim field1 As New field     'For PK IDs
   Dim field2 As New field     'For Money
   Dim field3 As New field     'For Story's
   screen.MousePointer = 11    'To display the time to build
   Set newdb = CreateDatabase("NEWBTWDB.MDB", DB_LANG_GENERAL)
   newtd.Name = "Money_Table"  '* New table name
   field1.Name = "PK_ID"       '* Holds PK ID
   field1.Type = DB_LONG
   newtd.Fields.Append field1
   field2.Name = "Money"       '* Holds Money
   field2.Type = DB_CURRENCY
   newtd.Fields.Append field2
   field3.Name = "Story"       '* Holds Story
   field3.Type = DB_TEXT
   field3.Size = 154
   newtd.Fields.Append field3
   newidx.Name = "PK_ID_IDX"   '* You have to have an index
   newidx.Fields = "PK_ID"
   newidx.Primary = True
   newtd.Indexes.Append newidx
   newdb.TableDefs.Append newtd
   Set newtb = newdb.OpenTable("Money_Table")
   For i% = 1 To 10
      newtb.AddNew
      newtb("PK_ID") = PrimaryKeys(i%)      'place in field1
      newtb("Money") = Money(i%)            'place in field3
      newtb("Story") = Trim$(Strings(i%))   'place in field4
      newtb.Update                          'Saving to table
   Next i%
   newtb.Close                 '* Close DB's table
   newdb.Close                 '* Close DB
   screen.MousePointer = 0     'Set back to show done
End Sub
```

8. Add the following code to the Command3 Click event procedure:

```
Sub Command3_Click ()
   Dim db As Database
   Dim t As Table
   Dim counter%
   Set db = OpenDatabase("NEWBTWDB.MDB")
   Set t = db.OpenTable("Money_Table")
   counter% = 1           'Start counter at Row=1
   Do Until t.EOF
      grid2.Col = 1
      grid2.Row = counter%
      grid2.Text = t(0)         'Load the PK ID
      grid2.Col = 2
      grid2.Row = counter%
      grid2.Text = t(1)         'Load the Money
      grid2.Col = 3
      grid2.Row = counter%
      grid2.Text = t(2)         'Load the Story
      counter% = counter% + 1
      t.MoveNext
   Loop
   t.Close
   db.Close
End Sub
```

9. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program. First, click the Command1 button. Next, click the Command2 button. Then click the Command3 button. Compare the results.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataIISAM

How to Make Access DB & Transfer Data from Btrieve for MS-DOS

Article ID: Q103441

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

The example in this article demonstrates how to build a Microsoft Access database without having a database or database template already built. The example uses a Btrieve for MS-DOS database file to supply the data to be placed into the newly created Microsoft Access database.

MORE INFORMATION

NOTE: You will need to have a Btrieve for MS-DOS database file already built to test this example. The Btrieve for MS-DOS database file tested with this example can be sent upon request.

Steps to Demonstrate Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add three command buttons and two grid controls using GRID.VBX to Form1. Using the following table as a guide, set the properties of the controls you added in step 2.

Control	Property	New Value	Comment
Command1	Caption	"Press to Load Btrieve File and Display in Grid"	
Command2	Caption	"Press to Transfer Data and Build New DB"	
Command3	Caption	"Press to Display the Data of the New Database"	
Grid1	Cols	6	
Grid1	Rows	35	
Grid2	Cols	6	
Grid2	Rows	35	

3. The following is an brief outline of the table from the Btrieve for MS-DOS database:

Table Name: Customers

Field Names	Field Type	Field Size
Cust_ID	Long	
First_Name	Text	15
Last_Name	Text	15
Cust_Addr	Text	30
Cust_Phone	Text	20

```
Index Names      Index Fields      Unique   Primary
-------------------------------------------------
Cust_ID_IDX      +Cust_ID          Yes      No
```

4. Add the following variables and constants to the (general) section of Form1:

```
Dim cust_ids(30) As Integer
Dim first_names(30) As String * 15
Dim last_names(30) As String * 15
Dim cust_addr(30) As String * 30
Dim cust_phones(30) As String * 20
Const DB_LONG = 4
Const DB_TEXT = 10
Const DB_LANG_GENERAL = ";LANGID=0x0809;CP=1252;COUNTRY=0"
```

5. Add the following code to the Form1 Load event procedure:

```
Sub Form_Load ()
   Show
   grid1.ColWidth(1) = 1000       'For Cust ID
   grid1.ColWidth(2) = 2000       'For First Name
   grid1.ColWidth(3) = 2000       'For Last Name
   grid1.ColWidth(4) = 3000       'For Cust Addr
   grid1.ColWidth(5) = 2000       'For Cust Phone
   grid1.Col = 1
   grid1.Row = 0
   grid1.Text = "Cust ID"         'Header for Cust ID
   grid1.Col = 2
   grid1.Row = 0
   grid1.Text = "First Name"      'Header for First Name
   grid1.Col = 3
   grid1.Row = 0
   grid1.Text = "Last Name"       'Header for Last Name
   grid1.Col = 4
   grid1.Row = 0
   grid1.Text = "Cust Addr"       'Header for Cust Addr
   grid1.Col = 5
   grid1.Row = 0
   grid1.Text = "Cust Phone"      'Header for Cust Phone

   grid2.ColWidth(1) = 1000       'For Cust ID
   grid2.ColWidth(2) = 2000       'For First Name
   grid2.ColWidth(3) = 2000       'For Last Name
   grid2.ColWidth(4) = 3000       'For Cust Addr
   grid2.ColWidth(5) = 2000       'For Cust Phone
   grid2.Col = 1
   grid2.Row = 0
   grid2.Text = "Customer ID"     'Header for Cust ID
   grid2.Col = 2
   grid2.Row = 0
   grid2.Text = "Cust First Name" 'Header for First Name
   grid2.Col = 3
   grid2.Row = 0
   grid2.Text = "Cust Last Name"  'Header for Last Name
   grid2.Col = 4
   grid2.Row = 0
```

```
   grid2.Text = "Customer Addr"   'Header for Cust Addr
   grid2.Col = 5
   grid2.Row = 0
   grid2.Text = "Customer Phone"  'Header for Cust Phone
End Sub
```

6. Add the following code to the Command1 Click event procedure:

```
Sub Command1_Click ()
   Dim db As database
   Dim conn$
   Dim dt As table
   conn$ = "Btrieve;"
   ' Enter the following Set as one, single line:
   Set db = OpenDatabase("C:\articles\btrvdos\file.ddf", False,
      False, conn$)
   Set dt = db.OpenTable("Customers")
   i% = 1            '* counter for loading the grid
   Do Until (dt.EOF = True)
      grid1.Col = 1
      grid1.Row = i%
      grid1.Text = dt(0)          'Load the grid
      cust_ids(i%) = dt(0)        'Load the temporary array
      grid1.Col = 2
      grid1.Row = i%
      grid1.Text = dt(1)          'Load the grid
      first_names(i%) = dt(1)     'Load the temporary array
      grid1.Col = 3
      grid1.Row = i%
      grid1.Text = dt(2)          'Load the grid
      last_names(i%) = dt(2)      'Load the temporary array
      grid1.Col = 4
      grid1.Row = i%
      grid1.Text = dt(3)          'Load the grid
      cust_addr(i%) = dt(3)       'Load the temporary array
      grid1.Col = 5
      grid1.Row = i%
      grid1.Text = dt(4)          'Load the grid
      cust_phones(i%) = dt(1)     'Load the temporary array
      dt.MoveNext
      i% = i% + 1
   Loop
End Sub
```

7. Add the following code to the Command2 Click event procedure:

```
Sub Command2_Click ()
   Dim newdb As Database
   Dim newtb As Table
   Dim newtd As New tabledef
   Dim newidx As New Index
   Dim field1 As New field        'For Emp nums
   Dim field2 As New field        'For Emp names
   Dim field3 As New field        'For Emp addresses
   Dim field4 As New field        'For Emp ss_nums
   screen.MousePointer = 11       'To display the time to build
   Set newdb = CreateDatabase("NEWBTRDB.MDB", DB_LANG_GENERAL)
```

```
   newtd.Name = "Cust_Table"      '* New table name
   field1.Name = "Cust_ID"        '* Holds Cust ID nums()
   field1.Type = DB_LONG
   newtd.Fields.Append field1
   field2.Name = "First_Name"     '* Holds First names()
   field2.Type = DB_TEXT
   field2.Size = 15
   newtd.Fields.Append field2
   field3.Name = "Last_Name"      '* Holds Last names()
   field3.Type = DB_TEXT
   field3.Size = 15
   newtd.Fields.Append field3
   field4.Name = "Cust_Addr"      '* Holds cust Addr()
   field4.Type = DB_TEXT
   field4.Size = 30
   newtd.Fields.Append field4
   field5.Name = "Cust_Phone"     '* Holds cust phones()
   field5.Type = DB_TEXT
   field5.Size = 20
   newtd.Fields.Append field5
   newidx.Name = "Cust_ID_IDX"    '* You must have to have an index
   newidx.Fields = "Cust_ID"
   newidx.Primary = True
   newtd.Indexes.Append newidx
   newdb.TableDefs.Append newtd
   Set newtb = newdb.OpenTable("Cust_Table")
   For i%=1 to 10                 'There are only ten entries
      newtb.AddNew
      newtb("Cust_ID") = cust_ids(i%)              'place in field1
      newtb("First_Name") = Trim$(first)names(i%)) 'place in field2
      newtb("Last_Name") = Trim$(last_names(i%))   'place in field3
      newtb("Cust_Addr") = Trim$(Cust_addr(i%))    'place in field4
      newtb("Cust_Phone") = Trim$(Cust_phones(i%)) 'place in field5
      newtb.Update                                 'Saving to table
   Next i%
   newtb.Close                    'Close DB's table
   newdb.Close                    'Close DB
   screen.MousePointer = 0        'Set back to show finished
End Sub
```

8. Add the following code to the Command3 Click event procedure:

```
Sub Command3_Click ()
   Dim db As Database
   Dim t As Table
   Dim counter%
   Set db = OpenDatabase("NEWBTRDB.MDB")
   Set t = db.OpenTable("Cust_Table")
   counter% = 1                   'Start counter at Row=1
   Do Until t.EOF
      grid2.Col = 1
      grid2.Row = counter%
      grid2.Text = t(0)           'Load the Cust ID
      grid2.Col = 2
      grid2.Row = counter%
      grid2.Text = t(1)           'Load the First Name
      grid2.Col = 3
```

```
        grid2.Row = counter%
        grid2.Text = t(2)           'Load the Last Name
        grid2.Col = 4
        grid2.Row = counter%
        grid2.Text = t(3)           'Load the Cust Addr
        grid2.Col = 5
        grid2.Row = counter%
        grid2.Text = t(4)           'Load the Cust Phone
        counter% = counter% + 1
        t.MoveNext
      Loop
      t.Close
      db.Close
   End Sub
```

9. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program. First, click the Command1 button. Next, click the Command2 button. Then click the Command3 button, and compare the results.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataIISAM

Differences Between the Object Variables in VB Version 3.0

Article ID: Q103442

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY
=======

This article contains two references:

- A revised version of the table outlined on the back of the "Professional Features Book 2" manual for Microsoft Visual Basic version 3.0 Programming System for Windows. This table outlines the differences in the properties and methods of the three main data access objects (table, dynaset, and snapshot) in Visual Basic version 3.0.

- A brief list of the differences between table, database, dynaset, querydef, and snapshot objects.

MORE INFORMATION
================

Revised Table for the Back of "Professional Features Book 2"

The following table lists most of the properties and methods that apply to each of the database objects.

- Yes means the object does contain the property or method in both the Standard and Professional Editions of Visual Basic version 3.0 for Windows.

- No means the object does not contain the property or method in either the Standard or Professional Edition of Visual Basic version 3.0 for Windows.

- Yes/PRO means the object contains the property or method only in the Professional Edition, not the Standard Edition, of Visual Basic version 3.0 for Windows.

- (docerr) highlights that information as a correction to the information given in the original table shown on the back of the "Professional Features Book 2."

Properties	Table	Dynaset	Snapshot
BOF	Yes/PRO	Yes	Yes/PRO
BookMark	Yes/PRO	Yes	Yes/PRO
BookMarkable	Yes/PRO	Yes	Yes/PRO
DateCreated	Yes/PRO	No	No
EOF	Yes/PRO	Yes	Yes/PRO

Filter	No	Yes/PRO	Yes/PRO
Index	Yes/PRO	No	No
LastModified	Yes/PRO	Yes	No
LastUpdated	Yes/PRO	No	No
LockEdits	Yes/PRO	Yes	No
Name	Yes/PRO	Yes	Yes/PRO
NoMatch	Yes/PRO	Yes	Yes/PRO
RecordCount	Yes/PRO	Yes	Yes/PRO
Sort	No	Yes/PRO	Yes/PRO
Transactions	Yes/PRO	Yes	No
Updatable	Yes/PRO	Yes	No

Methods	Table	Dynaset	Snapshot
AddNew	Yes/PRO	Yes	No
Clone	Yes/PRO	Yes/PRO	Yes/PRO
Close	Yes/PRO	Yes	Yes/PRO
CreateDynaset	Yes/PRO	Yes/PRO(docerr)	No
CreateSnapshot	Yes/PRO	No	Yes/PRO(docerr)
Delete	Yes/PRO	Yes	No
Edit	Yes/PRO	Yes	No
FindFirst	No(docerr)	Yes	Yes/PRO
FindLast	No(docerr)	Yes	Yes/PRO
FindNext	No(docerr)	Yes	Yes/PRO
FindPrevious	No(docerr)	Yes	Yes/PRO
ListFields	Yes/PRO	Yes/PRO	Yes/PRO
ListIndexes	Yes/PRO	Yes/PRO	Yes/PRO
MoveFirst	Yes/PRO	Yes	Yes/PRO
MoveLast	Yes/PRO	Yes	Yes/PRO
MoveNext	Yes/PRO	Yes	Yes/PRO
MovePrevious	Yes/PRO	Yes	Yes/PRO
Seek	Yes/PRO	No	No
Update	Yes/PRO	Yes	No

List of Differences Between Data Access Objects

Below, object by object, is a list of differences, recommendations, and suggestions for each of the various data access objects. The page numbers refer to pages in the "Professional Features Book 2." Article Q numbers refer to other Microsoft Knowledge Base articles which give provide additional information.

Snapshot Objects

- Snapshots return all of the selected data and Dynasets return only a set of keys that indirectly reference the database's records (page 57). Therefore when retrieving a small number of records in a recordset, you may want to use a dynaset instead of a snapshot unless this is the first time you are using a newly created snapshot or dynaset.

- When either a snapshot or a dynaset is first created -- prior to any movelast operation -- both the snapshot and the dynaset return one page (2048 bytes) of data. The dynaset also fetches the keyset of the dynaset. This means that on first creation, snapshots, as the name implies, return faster. However, if you were to proceed record by record

sequentially through the entire recordset, you'd find that the dynaset navigates faster -- approximately two times faster. This is because navigating by keyset instead of by local pointers is more efficient.

- Snapshots return all the selected data when movelast is executed or when the entire recordset is completely navigated. Therefore, in these two cases, trying to retrieve a large amount of data (a large number of records) could take some time. It may take less time to use Dynasets instead of Snapshots in this scenario (page 57).

- Snapshots can become outdated (the data is no longer current) quickly in a multiuser environment (page 57).

- Snapshots cannot use the Transaction statements (BeginTrans, CommitTrans, and RollBack).

- Snapshots or dynasets cannot use the Seek method because Seek applies only to table objects. However, snapshots or dynasets can use the Find method instead of the Seek method.

- Snapshots cannot use Edit, AddNew, Delete, or Update properties that pertain to data changes made in records. Snapshot objects are a read-only type of dynaset.

- Snapshot objects may be good for taking summary reports, since they contain a fixed copy of the data as it existed when the snapshot was created. If data is changed, a snapshot will not show the change until the snapshot is rebuilt (page 57).

- Snapshots can be created from an existing dynaset or snapshot, but you cannot create a dynaset from an existing snapshot (page 56).

- Snapshots can contain table name(s), attached tables, querydef objects or SQL statements(pg. 56).

- Snapshot object membership is fixed (page 48).

Dynasets Objects

- Dynaset and snapshot objects can use the Sort property, but the table object and the data control cannot use the Sort property. To sort data with a data control, use the ORDER BY clause of an SQL statement or query. To sort a table object, set an Index property on a field that already has a Index specified (example shown on pages 50 and 75).

- Dynasets are the most flexible of the three objects listed in the table above (page 51).

- Dynasets are a dynamic (not fixed) subset of records. Dynasets can contain attached tables, table name(s), querydef object name or SQL query (page 51).

- Filters are used to screen records to be brought back in dynasets or snapshots (page 53). Table objects cannot use filters.

- Dynasets can be locked with a page-locking scheme with a page containing a maximum of 2K of data (page 54). Page 54 also mentions pessimistic and

optimistic locking methods.

- Dynasets that are formed because of a query or SQL string are suspended until the query or SQL string returns the first record (page 51).

- Dynaset or snapshot objects can be filtered using the Filter property or sorted using the Sort property even further by using a second dynaset or snapshot object(page 53).

- Dynaset or snapshot objects are used with querydef objects. Also, the ListParameters method returns a snapshot with one record for each parameter used by the query (pages 93 and 97).

- Dynaset objects do not reflect changes made by others until you recreate the Dynaset variable or execute the CreateDynaset method with no arguments (page 55).

- Dynaset object membership is fixed, you can add, change, and delete records, and a result is returned by a query (page 48).

- Dynaset objects can create an inconsistent dynaset with the DB_INCONSISTENT flag. But it may be harder to keep referential integrity when this flag is specified (pages 58, 59, and 85).

- To improve performance, you may want to add the option DB_READONLY if you are not writing to or allowing the users to make changes to database records (pages 58 and 59).

Table objects

- Table objects have direct access to the data records (page 49). The data in a table object variable always reflects all current changes, including the additions of new records and the deletions of existing records (page 50).

- Table objects cannot be created from attached tables (page 50).

- Table object membership can change. You can add, change, and delete records, but there is no result returned by a query (page 48).

- Table objects cannot use the Find method (page 72).

- Table, database and dynaset objects can be locked, but a snapshot object cannot be locked (pages 88 and 89).

- Table objects provide the most up-to-date view of your data because the data in a table variable always reflects all current changes (page 50).

- Table objects can be ordered on a Indexed field, the Index property does apply. But the Index does not apply to data controls, snapshots or dynasets (see example on page 75).

- When looking for a single, specific record, you may want to use the Seek method with a table object because it is the fasted way to retrieve a single record (page 74).

QueryDef Objects

- querydef objects may be more efficient. For example, use a stored query of an SQL string as an argument to the recordset of querydef to produce a filtered dynaset or snapshot instead of creating a dynaset or snapshot and then filtering it (page 67).

- querydef objects do not store data. They store the definition of a query used to retrieve data (page 91).

- querydefs can be created only on a Microsoft Access or Visual Basic database (page 92).

- querydefs require a name. You must supply a name for the query when you create it (page 92).

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

How to Convert a Text File into a New Access Database

Article ID: Q103807

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

This article shows by example how to build a Microsoft Access database from scratch without having a database or database template already built. Then it shows how to load that database from data supplied by a standard ASCII text file.

MORE INFORMATION

Step-by-Step Example

1. If you don't have one already, build an ASCII text file to use in this example. If you already have the text file built, you can ignore most of step 5 -- except for loading Grid1 with data from your text file.

2. Add three command buttons and two grid controls (GRID.VBX) to Form1.

3. Using the following table as a guide, set the properties of the controls you added in step 2.

Control	Property	New Value
Command1	Caption	"Press to Build Text File and Display in Grid"
Command2	Caption	"Press to Transfer Data and Build New DB"
Command3	Caption	"Press to Display the Data of the New Database"
Grid1	Cols	5
Grid1	Rows	35
Grid2	Cols	5
Grid2	Rows	35

4. Add the following code to the (general) section of Form1:

```
Dim nums(30) As Long
Dim names(30) As String * 20
Dim addresses(30) As String * 25
Dim ss_nums(30) As String * 12
Const DB_LONG = 4
Const DB_TEXT = 10
Const DB_LANG_GENERAL = ";LANGID=0x0809;CP=1252;COUNTRY=0"
```

5. Add the following code to the Form load event procedure:

```
Sub Form_Load ()
   Show
```

```
    grid1.ColWidth(1) = 1000       'For Emp ID
    grid1.ColWidth(2) = 2000       'For Emp Name
    grid1.ColWidth(3) = 3000       'For Emp Addr
    grid1.ColWidth(4) = 2000       'For Emp SSN
    grid1.Col = 1
    grid1.Row = 0
    grid1.Text = "Emp ID"          'Header for Emp ID from text file
    grid1.Col = 2
    grid1.Row = 0
    grid1.Text = "Emp Name"        'Header for Emp Name from text file
    grid1.Col = 3
    grid1.Row = 0
    grid1.Text = "Emp Addr"        'Header for Emp Addr from text file
    grid1.Col = 4
    grid1.Row = 0
    grid1.Text = "Emp SSN"         'Header for Emp SSN from text file

    grid2.ColWidth(1) = 1000       'For Emp ID
    grid2.ColWidth(2) = 2000       'For Emp Name
    grid2.ColWidth(3) = 3000       'For Emp Addr
    grid2.ColWidth(4) = 2000       'For Emp SSN
    grid2.Col = 1
    grid2.Row = 0
    grid2.Text = "Employee ID"     'Header for Emp ID from DB
    grid2.Col = 2
    grid2.Row = 0
    grid2.Text = "Employee Name"   'Header for Emp Name from DB
    grid2.Col = 3
    grid2.Row = 0
    grid2.Text = "Employee Addr"   'Header for Emp ID from DB
    grid2.Col = 4
    grid2.Row = 0
    grid2.Text = "Employee SSN"    'Header for Emp Name from DB
End Sub
```

6. Add the following code to the Command1 click event procedure:

```
Sub Command1_Click ()
   For i% = 1 To 30
      nums(i%) = i%
      names(i%) = "John Doe # " + Str$(i%)
      addresses(i%) = Str$(i%) + " Mocking Bird Lane"
      If i% < 9 Then
         '* Enter the following four lines as one, single line:
         ss_nums(i%) = Trim$(Str$(i%) + Trim$(Str$(i%))
            + Trim$(Str$(i%)) + "-" + Trim$(Str$(i% + 1))
            + Trim$(Str$(i% + 1)) + "-" + Trim$(Str$(i%))
            + Trim$(Str$(i%)) + Trim$(Str$(i%)) + Trim$(Str$(i%)))
      Else
         '* Enter the following two lines as one, single line:
         ss_nums(i%) = Trim$(Trim$(Str$(999)) + "-" + Trim$(Str$(88))
            + "-" + Trim$(Str$(7777)))
      End If
   Next i%
   Open "Testdata.DAT" For Output As #1
   For j% = 1 To 30
      Print #1, nums(j%)
```

```
      Print #1, names(j%)
      Print #1, addresses(j%)
      Print #1, ss_nums(j%)
   Next j%
   Close #1
   For i% = 1 To 30               'Display results from text file
      grid1.Col = 1
      grid1.Row = i%
      grid1.Text = nums(i%)       'Load Emp IDs
      grid1.Col = 2
      grid1.Row = i%
      grid1.Text = names(i%)      'Load Emp Names
      grid1.Col = 3
      grid1.Row = i%
      grid1.Text = addresses(i%)  'Load Emp Addrs
      grid1.Col = 4
      grid1.Row = i%
      grid1.Text = ss_nums(i%)    'Load Emp SSNs
   Next i%
End Sub
```

7. Add the following code to the Command2 click event procedure:

```
Sub Command2_Click ()
   Dim newdb As Database
   Dim newtb As Table
   Dim newtd As New tabledef
   Dim newidx As New Index
   Dim field1 As New field        'For Emp nums
   Dim field2 As New field        'For Emp names
   Dim field3 As New field        'For Emp addresses
   Dim field4 As New field        'For Emp ss_nums
   screen.MousePointer = 11       'Display the time to build
   Set newdb = CreateDatabase("NEWDB.MDB", DB_LANG_GENERAL)
   newtd.Name = "Emp_Table"       '* New table name
   field1.Name = "Emp_ID"         '* Holds Employee ID nums()
   field1.Type = DB_LONG
   newtd.Fields.Append field1
   field2.Name = "Emp_Name"       '* Holds Emp names()
   field2.Type = DB_TEXT
   field2.Size = 20
   newtd.Fields.Append field2
   field3.Name = "Emp_Addr"       '* Holds Employee addr()
   field3.Type = DB_TEXT
   field3.Size = 25
   newtd.Fields.Append field3
   field4.Name = "Emp_SSN"        '* Holds emp ss_nums()
   field4.Type = DB_TEXT
   field4.Size = 12
   newtd.Fields.Append field4
   newidx.Name = "Emp_ID_IDX"     '* You have to have an index
   newidx.Fields = "Emp_ID"
   newidx.Primary = True
   newtd.Indexes.Append newidx
   newdb.TableDefs.Append newtd
   Set newtb = newdb.OpenTable("Emp_Table")
   Open "Testdata.dat" For Input As #1
```

```
   BeginTrans
   Do While Not (EOF(1))
      newtb.AddNew
      Line Input #1, tmp1$                'Retrieve empl_id
      Line Input #1, tmp2$                'Retrieve empl_name
      Line Input #1, tmp3$                'Retrieve empl_addr
      Line Input #1, tmp4$
      newtb("Emp_ID") = Trim$(tmp1$)      'Place in field1
      newtb("Emp_Name") = Trim$(tmp2$)    'Place in field2
      newtb("Emp_Addr") = Trim$(tmp3$)    'Place in field3
      newtb("Emp_SSN") = Trim$(tmp4$)     'Place in field4
      newtb.Update                        'Save to table
   Loop
   CommitTrans
   Close #1                               'Close text file
   newtb.Close                            'Close DB's table
   newdb.Close                            'Close DB
   screen.MousePointer = 0                'Set back to show done
End Sub
```

8. Add the following code to the Command3 click event procedure:

```
Sub Command3_Click ()
   Dim db As Database
   Dim t As Table
   Dim counter%
   Set db = OpenDatabase("NEWDB.MDB")
   Set t = db.OpenTable("Emp_Table")
   counter% = 1                    'Start counter at Row=1
   Do Until t.EOF
      grid2.Col = 1
      grid2.Row = counter%
      grid2.Text = t(0)            'Load Emp ID
      grid2.Col = 2
      grid2.Row = counter%
      grid2.Text = t(1)            'Load Emp Name
      grid2.Col = 3
      grid2.Row = counter%
      grid2.Text = t(2)            'Load Emp Addr
      grid2.Col = 4
      grid2.Row = counter%
      grid2.Text = t(3)            'Load Emp SSN
      counter% = counter% + 1
      t.MoveNext
   Loop
   t.Close
   db.Close
End Sub
```

9. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program. First click the Command1 button first. Then click the Command2 button, and then click the Command3 button to compare the results.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

Limitations of the Data Control in Visual Basic Version 3.0

Article ID: Q103808

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY
=======

Chapter 20 in the "Programmer's Guide" explains how to use a data control in Microsoft Visual Basic version 3.0 for Windows. You may want to use an object variable such as Snapshot instead of using the data control. Chapter 20 does not explain the limitations of using the data control, so this article lists those limitations for you.

MORE INFORMATION
================

Because a data control is a special type of Dynaset, its limitations are similar to those of Dynasets. Here are the limitations of data controls:

- You cannot use a QueryDef requiring a Parameter in the RecordSource property of the data control.

- Using a data control along with other bound controls uses System Resources (memory). When you build larger programs, you may want to look at other programming methods (Database objects don't require controls, therefore you don't use System Resources) to display your database data.

- Not every method and property specific to the Table object can be performed by the data control. Here are two such cases:

  - You cannot take advantage of the Index property of the Table object to display your database data in a specific indexed order with the data control. This technique, described in the example shown in the Help file topic "Index Property (Data Access)," works only with the Table object, not the data control. As an alternative, you can use an ORDER BY clause in an SQL statement, as in this example:

    ```
    Data1.RecordSource = "Select * From Publishers Order By PubID"
    ```

    The ORDER BY clause technique is also more flexible than the Index property technique. Using the ORDER BY clause, you can sort on any field, and no specified index is required.

  - You cannot use a Seek method on your database data for a specific record with the data control. The Seek method can only be used by the Table object. You can, however, perform a FindFirst method with the data control.

- You cannot use the Sort property on a specific database record with the data control. The Sort property technique is specific to a Dynaset or

Snapshot object. The following example proves this limitation:

```
Data1.Recordset.Sort = "City DESC"   '** No error occurs
Data1.Refresh                        '** No change in order occurs
```

If you try to sort the Publishers table by City, nothing happens. But if you use an ORDER BY clause in an SQL statement, as in the following example, you will see the database data displayed in descending order by the City names:

```
Data1.RecordSource = "Select * From Publishers Order By City DESC"
Data1.Refresh
```

- A data control is bound to one, single form -- the form on which it resides. Therefore, when the form that contains the data control is not loaded, you cannot refer to the data control from another form.

- You cannot perform a FileCopy statement on a database while a form that contains a data control is loaded. A "Permission Denied" error occurs if you try to use the FileCopy statement to make a backup of your database while a form containing a data control is loaded in memory. To prevent this error, first close or unload the form that contains the data control. Then run the FileCopy statement to make a database backup.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

How to Create an Access DB & Transfer Data from dBASE III DB

Article ID: Q104013

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

This example demonstrates how to build a new Microsoft Access database and load it with data coming from a dBASE III database file.

MORE INFORMATION

To use this example, you will need a dBASE III database file. The dBASE III database file that was tested with this example can be sent upon request.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add three command buttons and use GRID.VBX to add two grid controls to Form1. Then using following table as a guide, set the properties of the controls:

Control Name	Property	New Value
Command1	Caption	"Press to Load dBASE III DB File and Display in Grid"
Command2	Caption	"Press to Transfer Data and Build New DB"
Command3	Caption	"Press to Display the Data of the New Database"
Grid1	Cols	7
Grid1	Rows	15
Grid2	Cols	7
Grid2	Rows	15

3. The following is an brief outline of the table from the dBASE III database:

Table Name: CHECKS

Field Name	Field Type	Field Size
CHKNO	Double	
PAYTO	Text	30
AMT	Double	
DATE	Date/Time	
MEMO	Text	25
NAME5	Double	

```
Index Name   Index Field   Unique   Primary
-----------------------------------------
nm5          +NAME5        Yes      No
```

4. Add the following variables and constants to the (general) section of Form1:

```
Dim CK_nums(20) As Double
Dim paytos(20) As String * 30
Dim amts(20) As Double
Dim dates(20) As Variant
Dim memos(20) As String * 25
Dim indexs(20) As Double
Dim counter%
Const DB_DATE = 8
Const DB_DOUBLE = 7
Const DB_TEXT = 10
Const DB_LANG_GENERAL = ";LANGID=0x0809;CP=1252;COUNTRY=0"
```

5. Add the following lines to the Form load event procedure:

```
Sub Form_Load ()
   Show
   grid1.ColWidth(1) = 1000      'For Chk nums
   grid1.ColWidth(2) = 2000      'For Paid to
   grid1.ColWidth(3) = 1500      'For Amt for
   grid1.ColWidth(4) = 2000      'For Date written
   grid1.ColWidth(5) = 3000      'For Memo
   grid1.ColWidth(6) = 1000      'For index
   grid1.Col = 1
   grid1.Row = 0
   grid1.Text = "Check No."
   grid1.Col = 2
   grid1.Row = 0
   grid1.Text = "Party Paid"
   grid1.Col = 3
   grid1.Row = 0
   grid1.Text = "Amount"
   grid1.Col = 4
   grid1.Row = 0
   grid1.Text = "Date Written"
   grid1.Col = 5
   grid1.Row = 0
   grid1.Text = "Memo about"
   grid1.Col = 6
   grid1.Row = 0
   grid1.Text = "Index"
   grid2.ColWidth(1) = 1000      'For Chk nums
   grid2.ColWidth(2) = 2000      'For Paid to
   grid2.ColWidth(3) = 1500      'For Amt for
   grid2.ColWidth(4) = 2000      'For Date written
   grid2.ColWidth(5) = 3000      'For Memo
   grid2.ColWidth(6) = 1000      'For index
   grid2.Col = 1
   grid2.Row = 0
   grid2.Text = "Check No."
```

```
   grid2.Col = 2
   grid2.Row = 0
   grid2.Text = "Party Paid"
   grid2.Col = 3
   grid2.Row = 0
   grid2.Text = "Amount"
   grid2.Col = 4
   grid2.Row = 0
   grid2.Text = "Date Written"
   grid2.Col = 5
   grid2.Row = 0
   grid2.Text = "Memo about"
   grid2.Col = 6
   grid2.Row = 0
   grid2.Text = "Index"
End Sub
```

6. Add the following code to the Command1 click event procedure:

```
Sub Command1_Click ()
   Dim db As Database
   Dim conn$
   Dim dt As Table
   conn$ = "dBASE III;"

   ' Enter the following two lines as one, single line:
   Set db = OpenDatabase("c:\articles\db3\dbaseiii", False,
      False, conn$)

   Set dt = db.OpenTable("CHECKS")
   screen.MousePointer = 11
   counter% = 1
   Do Until (dt.EOF = True)
      grid1.Col = 1
      grid1.Row = counter%
      grid1.Text = dt(0)
      CK_nums(counter%) = Val(grid1.Text)
      grid1.Col = 2
      grid1.Row = counter%
      grid1.Text = dt(1)
      paytos(counter%) = grid1.Text
      grid1.Col = 3
      grid1.Row = counter%
      grid1.Text = dt(2)
      amts(counter%) = Val(grid1.Text)
      grid1.Col = 4
      grid1.Row = counter%
      If IsNull(dt(4)) Then 'In case there is no date entered
         grid1.Text = ""
      Else
         grid1.Text = dt(4)
      End If
      dates(counter%) = grid1.Text
      grid1.Col = 5
      grid1.Row = counter%
      grid1.Text = dt(5)
      memos(counter%) = grid1.Text
```

```
      grid1.Col = 6
      grid1.Row = counter%
      grid1.Text = dt(8)
      indexs(counter%) = Val(grid1.Text)
      counter% = counter% + 1
      dt.MoveNext
   Loop
   screen.MousePointer = 0
End Sub
```

7. Add the following code to the Command2 click event procedure:

```
Sub Command2_Click ()
   Dim newdb As Database
   Dim newtb As Table
   Dim newtd As New tabledef
   Dim newidx As New Index
   Dim field1 As New field       'For chknum
   Dim field2 As New field       'For party paid to
   Dim field3 As New field       'For amount
   Dim field4 As New field       'For date written
   Dim field5 As New field       'For memo field
   Dim field6 As New field       'For in index
   screen.MousePointer = 11
   Set newdb = CreateDatabase("DBASE3.MDB", DB_LANG_GENERAL)
   newtd.Name = "Checks_Table"   'New table name
   field1.Name = "Check_nums"
   field1.Type = DB_DOUBLE
   newtd.Fields.Append field1
   field2.Name = "Paid_to"
   field2.Type = DB_TEXT
   field2.Size = 30
   newtd.Fields.Append field2
   field3.Name = "Check_amt"
   field3.Type = DB_DOUBLE
   newtd.Fields.Append field3
   field4.Name = "Date_wrt"
   field4.Type = DB_DATE
   newtd.Fields.Append field4
   field5.Name = "Check_memo"
   field5.Type = DB_TEXT
   field5.Size = 25
   newtd.Fields.Append field5
   field6.Name = "Check_indx"
   field6.Type = DB_DOUBLE
   newtd.Fields.Append field6
   newidx.Name = "Check_nums_IDX"
   newidx.Fields = "Check_indx"
   newidx.Primary = True
   newtd.Indexes.Append newidx
   newdb.TableDefs.Append newtd
   Set newtb = newdb.OpenTable("Checks_Table")
   For j% = 1 To counter% - 1
      newtb.AddNew
      newtb("Check_nums") = CK_nums(j%)  'from dBASE III file
      newtb("Paid_to") = paytos(j%)      'from dBASE III file
      newtb("Check_amt") = amts(j%)      'from dBASE III file
```

```
         newtb("Date_wrt") = dates(j%)      'from dBASE III file
         newtb("Check_memo") = memos(j%)    'from dBASE III file
         newtb("Check_indx") = indexs(j%)   'from dBASE III file
         newtb.Update                       'Saving to table
      Next j%
      newtb.Close
      newdb.Close
      screen.MousePointer = 0
   End Sub
```

8. Add the following code to the Command3 click event procedure:

```
   Sub Command3_Click ()
      Dim db As Database
      Dim t As Table
      Dim cntr%
      Set db = OpenDatabase("DBASE3.MDB")
      Set t = db.OpenTable("Checks_Table")
      cntr% = 1           'Start counter at Row=1
      Do Until t.EOF
         grid2.Col = 1
         grid2.Row = cntr%
         grid2.Text = t(0)
         grid2.Col = 2
         grid2.Row = cntr%
         grid2.Text = t(1)
         grid2.Col = 3
         grid2.Row = cntr%
         grid2.Text = t(2)
         grid2.Col = 4
         grid2.Row = cntr%
         If IsNull(t(3)) Then 'In case there is no date entered
            grid2.Text = ""
         Else
            grid2.Text = t(3)
         End If
         grid2.Col = 5
         grid2.Row = cntr%
         grid2.Text = t(4)
         grid2.Col = 6
         grid2.Row = cntr%
         grid2.Text = t(5)
         cntr% = cntr% + 1
         t.MoveNext
      Loop
      t.Close
      db.Close
   End Sub
```

9. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program. Click the Command1 button first. Then click the Command2 button. Then click the Command3 button, and compare the results.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataIISAM

Examples Show How to Query BIBLIO.MDB Database

Article ID: Q104155

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

Most of the examples in the Visual Basic Help menu for SQL statements do not show how to work with the BIBLIO.MDB Microsoft Access database that comes with Microsoft Visual Basic version 3.0 for Windows. Therefore this article shows by example how to use SQL statements with the BIBLIO.MDB database.

MORE INFORMATION

The following example gives 16 different SQL statements to test on the BIBLIO.MDB database. If you try one of the query statements on your own database and the result set is not what you had expected, try the Query By Example routine that comes with Microsoft Access to test your query. Note that if you try these examples on a computer that does not have SHARE.EXE loaded in memory, you will see this error:

```
Object Variable not Set, number 91
```

SHARE.EXE must be loaded for the Microsoft Access database to work.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add one list box, two command buttons, and one text box to Form1.

3. Using the following table as a guide, set the properties of the controls you added in step 2:

Control Name	Property	New Value
Command1	Caption	"Select Query from List box"
Command2	Caption	"Press to Clear Text Box"
Text1	Multiline	True
Text1	Scrollbars	Vertical
Text1	Text	" "

4. Add the following code to the (general) (declarations) section of Form1:

```
Dim query_array(0 To 15) As String
```

5. Add the following code to the form load event procedure:

```
Sub Form_Load ()
   '*** Note that each statement, including those shown on more than one
   '*** line, must be entered as one, single line.

   'Load query array with some example queries:
   query_array(0) = "Select all * from publishers"           'Select All
   query_array(1) = "Select all * from publishers"          'From clause
   query_array(2) = "Select publishers.name from publishers
      where publisher s.name in ('ETN Corporation', 'ACM')"    'Where In
   query_array(3) = "Select publishers.name from publishers
      order by publishers.city"                                'Order By
   query_array(4) = "Select publishers.name from publishers,
      [publisher comments] where [publisher comments].publisher =
      publishers.name group by publishers.name"               'Group By
   query_array(5) = "Select publishers.name from publishers
      where publisher s.name between 'ETN Corporation' and
      'ACM'"                                              'Where Between
   query_array(6) = "Select Distinct publishers.name from
      publishers, [publisher comments] where
      [publisher comments].publisher = publishers.name
      group by publishers.name"                                'Distinct
   query_array(7) = "Select publishers.name from publishers
      In biblio.mdb"                                          'In clause
   query_array(8) = "Select Distinctrow publishers.name
      from publishers, [publisher comments] where
      [publisher comments].publisher = publishers.name
      group by publishers.name"                             'Distinctrow
   query_array(9) = "Select all * from publishers order
      by Publishers.name WITH OWNERACCESS OPTION"    'Owneraccess Option
   query_array(10) = "Select publishers.name from
      publishers group by publishers.name having
      publishers.name like 'A*'"                          'Having clause
   query_array(11) = "Select publishers.name from
      publishers, [publisher comments], [publisher comments]
      left join publishers on [publisher comments].pubid =
      publishers.pubid"                                       'Left Join
   query_array(12) = "Select publishers.name from
      publishers, [publisher comments], [publisher comments]
      right join publishers on [publisher comments].pubid =
      publishers.pubid"                                      'Right Join
   query_array(13) = "Select publishers.name from
      publishers, [publisher comments], [publisher comments]
      inner join publishers on [publisher comments].pubid =
      publishers.pubid"                                      'Inner Join
   query_array(14) = "Select publishers.name from
      publishers order by publishers.name ASC"               'ASC order
   query_array(15) = "Select publishers.name from
      publishers order by publishers.name DESC"             'DESC order
   list1.AddItem "Example of:  'Select All' Query"
   list1.AddItem "Example of:  'From clause' Query"
   list1.AddItem "Example of:  'Where In' Query"
   list1.AddItem "Example of:  'Order By' Query"
   list1.AddItem "Example of:  'Group By' Query"
   list1.AddItem "Example of:  'Where Between' Query"
   list1.AddItem "Example of:  'Distinct' Query"
   list1.AddItem "Example of:  'In clause' Query"
```

```
   list1.AddItem "Example of:  'Distinctrow' Query"
   list1.AddItem "Example of:  'Owneraccess Option' Query"
   list1.AddItem "Example of:  'Having clause' Query"
   list1.AddItem "Example of:  'Left Join' Query"
   list1.AddItem "Example of:  'Right Join' Query"
   list1.AddItem "Example of:  'Inner Join' Query"
   list1.AddItem "Example of:  'ASC order' Query"
   list1.AddItem "Example of:  'DESC order' Query"
End Sub
```

6. Add the following code to the list1 click event procedure:

```
Sub List1_Click ()
   idx% = list1.ListIndex
   Select Case idx%
      Case 0: command1.Caption = "Press for 'Select All'"
      Case 1: command1.Caption = "Press for 'From clause'"
      Case 2: command1.Caption = "Press for 'Where In'"
      Case 3: command1.Caption = "Press for 'Order By'"
      Case 4: command1.Caption = "Press for 'Group By'"
      Case 5: command1.Caption = "Press for 'Where Between'"
      Case 6: command1.Caption = "Press from 'Distinct'"
      Case 7: command1.Caption = "Press from 'In clause'"
      Case 8: command1.Caption = "Press from 'Distinctrow'"
      Case 9: command1.Caption = "Press from 'Owneraccess Option'"
      Case 10: command1.Caption = "Press from 'Having clause'"
      Case 11: command1.Caption = "Press from 'Left Join'"
      Case 12: command1.Caption = "Press from 'Right Join'"
      Case 13: command1.Caption = "Press from 'Inner Join'"
      Case 14: command1.Caption = "Press from 'ASC order'"
      Case 15: command1.Caption = "Press from 'DESC order'"
      Case Else: command1.Caption = "Select Query from List box"
   End Select
End Sub
```

7. Add the following code to the text1 keypress event procedure:

```
Sub Text1_KeyPress (keyascii As Integer)
   If keyascii > 0 Then  '** this routine makes it a read-only text box
      keyascii = 0
   End If
End Sub
```

8. Add the following code to the command1 click event procedure:

```
Sub Command1_Click ()
   Dim db As database
   Dim ds As dynaset
   On Error GoTo type_error
   idx% = list1.ListIndex
   tmp$ = query_array(idx%)
   Set db = OpenDatabase("C:\vb3\biblio.mdb")
   Set ds = db.CreateDynaset(tmp$)
   Do Until ds.EOF = True
      If IsNull(ds(0)) Then
         text1.Text = text1.Text + " " + Chr$(13) + Chr$(10)
      Else
```

```
            text1.Text = text1.Text + ds(0) + Chr$(13) + Chr$(10)
         End If
         ds.MoveNext
      Loop
      ds.Close
      db.Close
      command2.SetFocus
      type_error:
      If Err = 13 Then     '*** Type Mismatch error
         Do Until ds.EOF = True
            If IsNull((ds(1))) Then
               text1.Text = text1.Text + " " + Chr$(13) + Chr$(10)
            Else
               text1.Text = text1.Text + ds(1) + Chr$(13) + Chr$(10)
            End If
            ds.MoveNext
         Loop
         ds.Close
         db.Close
         command2.SetFocus
         Exit Sub
      Else
         command2.SetFocus
         Resume Next
      End If
   End Sub
```

9. Add the following code to the command2 click event procedure:

```
   Sub Command2_Click ()
      text1.Text = ""
      command1.Caption = "Select Query from List box"
   End Sub
```

10. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program. Select a query from the list box. Press the command button to have the result set added to the text box.

    To clear the contents of the text box, press the second command button.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

PRB: TableDefs Not Updated When SQL Statement Creates Table

Article ID: Q104339

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

If you create a table with an SQL action statement such as SELECT INTO, the new table is not immediately reflected in the TableDefs collection of the database object or property. This may result in the error "Name not found in this collection" (error 3265).

WORKAROUND

- Perform TableDefs.Refresh immediately after creating the new table. The Refresh method should be but is not documented for the TableDefs collection in the Visual Basic manuals or Help menu.

- Create the table using database.Tabldefs.Append instead of using an SQL statement.

STATUS

This behavior is by design.

MORE INFORMATION

A table created with an SQL statement is correctly reflected in the Tabledefs
collection after the database is closed then reopened.

Steps to Reproduce Problem

The following program demonstrates this problem. It results in the error "Name not found in this collection" on the Tabledefs.Delete statement.

```
Dim db As Database
Print "TableDefs.Count before:"; db.TableDefs.Count
Set db = OpenDatabase("biblio.mdb")
' Create new table using SQL action statement:
db.Execute "select distinctrow * into NewTable from Authors"
' Remove apostrophe from the following statement to work around problem:
' db.TableDefs.Refresh
Print "TableDefs.Count after: "; db.TableDefs.Count
' Now attempt to delete the new table just created:
db.TableDefs.Delete "NewTable"
db.Close
```

Additional reference words: 3.00 docerr
KBCategory:
KBSubcategory: APrgDataOther

PRB: Error 3219 When Updating Record Set Created w/ Distinct

Article ID: Q104459

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SYMPTOMS

Trying to perform an update on a result set created with the Distinct predicate results in error 3219 "Can't perform operation, it is illegal."

CAUSE

An object variable or a data control result set created with the Distinct predicate is not updatable according to Help. The following statement is in the Distinct keyword Help topic:

> The output of a query that uses Distinct is not updatable and doesn't reflect subsequent changes made by other users. Therefore, when you use the Distinct predicate in a query, you are prevented from trying to update your records.

WORKAROUND

The only workaround at this time is to not use the DISTINCT predicate to build the results set. Note that you may have to handle the duplicates by some other coding means.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Reproduce Problem

1. Start Visual Basic or from the File menu, choose Open Project(ALT, F, O) if Visual Basic is already running. Form1 is created by default.

2. Add a data control, two command buttons, and one text box to Form1.

3. Using the following table as a guide, set the properties of the controls you added in step 2.

Control Name	Property	New Value	Comment
Command1	Caption	"Set Up Distinct Predicate"	

Command2	Caption	"Press for Update"	
Data1	DatabaseName	BIBLIO.MDB	Provide the full path to this file, which should be in C:\VB
Data1	RecordSource	Authors	
Text1	DataSource	Data1	
Text1	DataField	Author	

4. Add the following code to Command1 click event procedure:

```
Sub Command1_Click ()
   '* Enter the following two lines of code as one, single line:
   data1.RecordSource = "Select DISTINCT Author From authors
      where author > 'a'"
   data1.Refresh
End Sub
```

5. Add the following code to Command2 click event procedure:

```
Sub Command2_Click ()
   data1.Recordset.Update
End Sub
```

6. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program. Click the Command1 button to set up the Distinct predicate. Delete the zero in "Arnson, Robert, 1970." Then click the Command2 button. This should result in the 3219 error "Can't perform operation, it is illegal."

Additional reference words: 3.00
KBCategory: IAP
KBSubcategory: APrgDataAcc

How to Encrypt a Microsoft Access Database in Visual Basic

Article ID: Q104875

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

Database encryption has nothing to do with security. However, you can use database encryption to prevent someone from using a file or disk editor to read and write data in a Microsoft Access .MDB file. This article shows by example how to encrypt a Microsoft Access database file in Microsoft Visual Basic version 3.0 for Windows.

MORE INFORMATION

Microsoft Access reads and writes all data a page at a time. Each page is always 2K in size. Encryption is done at the page level, not at the data level. This means the encryption process has no knowledge of what is on the page, only that there is 2K of data that needs to be encrypted and written. or read and decrypted.

Everything in a Microsoft Access .MDB database file is encrypted, including tables, queries, forms, indexes, and so on. Microsoft Access uses the RSA company algorithm for database encryption.

The overhead involved in encrypting and decrypting causes is a performance degradation of approximately 10-15% in encrypted databases. Encrypted files cannot be compressed using tools such as PKZip, Stacker, MS-DOS version 6 DoubleSpace, and so on.

Encryption in Visual Basic

Use the CompactDatabase statement in Microsoft Visual Basic version 3.0 for Windows to encrypt a Microsoft Access database file. For more information on the CompactDatabase statement, review pages 90-92 in the Visual Basic version 3.0 "Language Reference" manual.

Step-by-Step Encryption Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the Windows menu, choose Data Manager.

3. In Data Manager, choose New Database from the File menu. Then select either Microsoft Access 1.0 or Microsoft Access 1.1.

4. Enter the name TESTING.MDB for the Microsoft Access file name that you

are about to create.

5. Click the New button and enter Table1 for the table name.

6. Click the Add button and enter First Name as the Field Name. Then select Text for the Field Type and enter 15 as the Field Size.

7. Click the Add button for Indexes, and enter First Name Index as the Index Name. Then select Unique, Primary and click Done.

8. Click the Open button, then the Add button. Next enter a name (Bob, for example) into the First Name field. Then click the Add button.

9. Close the Data Manager and add a Command button to Form1.

10. Add the following code to the Command1 Click event procedure:

```
Sub Command1_Click ()
   Const DB_ENCRYPT = 2
   Const DB_LANG_GENERAL = ";LANGID=0x0809;CP=1252;COUNTRY=0"

   '** Enter the following two lines as one, single line:
   CompactDatabase "C:\VB\TESTING.MDB", "C:\VB\NEWTEST.MDB",
      DB_LANG_GENERAL, DB_ENCRYPT
End Sub
```

11. From the Run menu, choose Start (ALT, R, S) to run the program. Click the Command1 button to encrypt the TESTING.MDB database file. To check the new NEWTEST.MDB file, choose Data Manager from the Window menu in Visual Basic version 3.0 for Windows. In the Data Manager, choose Open Database from the File menu. Then select the NEWTEST.MDB file.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

Differences Among the Installable ISAMs

Article ID: Q104918

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article describes the differences in the way the Installable ISAMs (IISAMs) handle databases, tables, indexes, and data types compared to the way Microsoft Access databases handle these structures.

The ISAMs discussed are:

- Btrieve
- dBASE III/IV
- FoxPro version 2.0/2.5
- Paradox version 3.x

This article does not include differences in ODBC data sources.

MORE INFORMATION

General Differences

Unsupported commands or functions

- CreateDatabase, CompactDatabase, RepairDatabase, SetDefaultWorkspace, ListTables, CreateQueryDef, OpenQueryDef, DeleteQueryDef, and ListParameters are Microsoft Access-only functions or statements. CreateDatabase is not supported because the ISAMs are all single-table databases where the database can be thought of as the directory in which the tables reside. Functions similar to CompactDatabase and RepairDatabase on an ISAM database should be done by using the native database tools.

- As a general rule, Microsoft Access database searches are not case sensitive but searches on the ISAM databases are case sensitive. The following are exceptions to this:

  - If a search is made across two different database types, that search is not case sensitive.

  - Some of the international settings cause a difference in case sensitivity. For example, searches are not case sensitive if in the [Paradox ISAM] section, CollatingSequence= is set to any of the

following: International, Noregian-Danish, or Swedish-Finnish.

For more information about case sensitivity of the ISAMs, please see the following article in the Microsoft Knowledge Base:

ARTICLE_ID: Q100921
TITLE : PRB: Case Sensitivity is Different with Attached Tables

- New field columns cannot be added to the ISAM database tables once there are records present. This is not the case with Microsoft Access.

- MS-DOS does not recognize the ANSI character set. This means extended ANSI characters will be converted by the OemToAnsi and AnsiToOem Windows API calls. This is not a one to one conversion, so some characters may be lost or changed in the process -- ANSI characters 147-159 specifically. Examples:

  - Saving Chr$(148) to a dBASE database, returns Asc(34) from the database, but saving Chr$(148) to a native Microsoft Access database correctly returns Asc(148).

==========
Btrieve ISAM
==========

For more information, please read BTRIEVE.TXT located in the Visual Basic directory.

Databases

- The DatabaseName string property used on opening a Btrieve database needs to be:

  driveletter\directory\FILE.DDF

  If just the directory name is listed, an error will occur.

  Code Sample

  ```
  Dim db As Database
  Set db = OpenDatabase("c:\btrieve\FILE.DDF",0,0,"btrieve;")
  ```

  This code opens the database located in the c:\btrieve directory. If a FILE.DDF is not located in the specified subdirectory, Visual Basic will create one. Note that a filename is needed for Btrieve ISAM, but the filename is ignored. It will always look for, or create a FILE.DDF and other supporting files.

- Databases are a set of *.DDF files. You can think of the directory where the files exist as the database. There can be only one FILE.DDF in a subdirectory.

Tables

- Table data is stored in *.DAT files. FILE.DDF contains the table name

and path to data files. FIELD.DDF contains the information about the columns. Visual Basic cannot change the directory where these data files are located. It stores them by default in the databasename subdirectory. Visual Basic can however read Btrieve databases that have data files in separate directories, as long as the same directory structure exists from where it was created.

For more information about table data in Btrieve databases, please see the following articles in the Microsoft Knowledge Base:

ARTICLE-ID: Q93685
TITLE : PRB: 'Couldn't find object <tablename>' Error with Btrieve

ARTICLE-ID: Q94828
TITLE : PRB: Empty Table List When Attaching Btrieve Table

Indexes

- Some compound indexes created by applications other than Visual Basic may not be viewable by Visual Basic. Btrieve permits index keys to be defined as specific byte ranges in a record, If a specified byte range is not aligned on the column boundaries of the fields in a table then Visual Basic will not be able to use that index.

Data Types

- The following table shows how data types are converted to Microsoft Access when reading an existing table.

Btrieve	Microsoft Access
String	DB_TEXT
Integer	DB_INTEGER or DB_LONG
Float	DB_SINGLE or DB_DOUBLE
Date	DB_DATE
Time	DB_DATE
Decimal	DB_DOUBLE
Money	DB_CURRENCY
Logical	DB_BOOLEAN
Numeric	DB_DOUBLE
Bfloat	DB_SINGLE or DB_DOUBLE
Lstring	DB_TEXT
Zstring	DB_TEXT

- The following table shows how data types are converted when creating a new table in Visual Basic.

Data Field	Result
DB_BOOLEAN	DB_BOOLEAN
DB_BYTE	DB_BYTE
DB_INTEGER	DB_INTEGER
DB_LONG	DB_LONG
DB_CURRENCY	DB_CURRENCY

```
   DB_SINGLE         DB_SINGLE
   DB_DOUBLE         DB_DOUBLE
   DB_DATE           DB_DATE
   DB_TEXT           DB_TEXT
   DB_LONGBINARY     DB_MEMO
   DB_MEMO           DB_MEMO
```

- There can be only one memo field or one long binary field per Btrieve table. Having more generates Error 3054 "Too many memo or long binary fields." For more information about Btrieve memo fields, please see the following article in the Microsoft Knowledge Base:

  ARTICLE-ID: Q103186
  TITLE : PRB: Error Message: Too Many Memo or OLE Fields

dBASE III / IV ISAM

Databases

- Databases are directories. On a data control or OpenDatabase statement, the exclusive property is ignored. The database name is the path to a directory.

Tables

- The following shows by example how to create a dBASE database and table. The code sample demonstrates the steps necessary to create a table for a dBASE database. Think of the database as the C:\DBASE directory. In Microsoft Access, databases are created using the CreateDatabase function.

```
Sub Command1_Click ()
   Const DB_TEXT = 10
   Dim db As database
   Dim tb As New tabledef
   Dim fd As Field
   Set db = OpenDatabase("c:\dBASE", False, False, "dBASE iii;")
   tb.Name = "MyTable"
   Set fd = New Field
   fd.Name = "f1"
   fd.Type = DB_TEXT
   fd.Size = 15         'Creates a text field length 15 characters
   tb.Fields.Append fd
   db.TableDefs.Append tb
End Sub
```

  The code sample creates a table that has one field and places it in the C:\DBASE subdirectory. If that subdirectory does not exist, the following error occurs:

```
'MyTable' isn't a valid path
```

  Tables are *.DBF files in the database directory. If the code sample is

successful, a file called MYTABLE.DBF is created.

Numeric Fields

- When you use Visual Basic to create a numeric field in a DBASE III/IV Database, Visual Basic creates a numeric field with five decimal places. This is by design.

- dBASE III/IV numeric fields can have up to 19 decimal places. If you want a dBASE III/IV numeric field with more than five decimal places, you have to use dBASE III/IV to modify the structure. Then Visual Basic will display and modify the value with all the decimal places and save it to the database correctly.

Indexes

- Indexes are separate files. They are placed in the database subdirectory as they are created. *.INF files list the indexes on a table. dBASE III indexes are *.NDX files and dBASE IV indexes are *.MDX files.

- A FoxPro or dBASE complex index can only be made from string type fields. Internally, both FoxPro and dBASE provide functions to convert and manipulate fields into strings so that they can be combined into a complex index across several fields of different types. Visual Basic does not have the ability to manipulate these functions so all complex indexes must be made up of DB_TEXT (string) types.

- dBASE allows duplicates in the Primary Key field. This is by design of the dBASE structure, the concept of Primary Keys does not exist.

Data Types

- The following table shows how data types are converted to Microsoft Access when reading an existing table.

dBASE	Microsoft Access
Character	DB_TEXT
Numeric	DB_DOUBLE
Date	DB_DATE
Logical	DB_BOOLEAN
Memo	DB_MEMO

- The following table shows how data types are converted when creating a new table in Visual Basic.

Data Field	Result
DB_BOOLEAN	DB_BOOLEAN
DB_BYTE	DB_DOUBLE
DB_INTEGER	DB_DOUBLE
DB_LONG	DB_DOUBLE
DB_CURRENCY	DB_DOUBLE
DB_SINGLE	DB_DOUBLE

DB_DOUBLE	DB_DOUBLE
DB_DATE	DB_DATE
DB_TEXT	DB_TEXT
DB_LONGBINARY	DB_MEMO
DB_MEMO	DB_MEMO

- Memo fields in dBASE and FoxPro are for text only. This is not the case for a Microsoft Access memo field, which can contain text or binary data.

- Viewing dBASE Memo fields that were created in dBASE IV may result in strange vertical line characters every 65th characters. This is by design; that is, it is the way dBASE displays its memo fields. For more information about problems viewing dBASE Memo fields, please see the following article in the Microsoft Knowledge Base:

  ARTICLE-ID: Q88647
  TITLE : PRB: Irregular Characters in Attached dBASE IV Memo Field

FoxPro 2.0/2.5 ISAM

Databases

- Databases are directories. On a data control or OpenDatabase statement, the exclusive property is ignored. The database name is the path to a directory.

Tables

- Tables are *.DBF files in the database directory.

Indexes

- Index information is stored in a file (tablename.CDX). This file contains the information about all the indexes on a table. This file must exist in the database directory.

- A FoxPro or dBASE complex index can only be made from string type fields. Internally FoxPro and dBASE provide functions to convert and manipulate fields into strings so that they can be combined into a complex index across several fields of different types. Visual Basic does not have the ability to manipulate these functions so all complex indexes must be made up of DB_TEXT (string) types.

Data Types

- The following table shows how data types are converted to Microsoft Access when reading an existing table.

FoxPro	Microsoft Access

Character	DB_TEXT
Numeric	DB_DOUBLE
Float	DB_DOUBLE
Date	DB_DATE
Logical	DB_BOOLEAN
Memo	DB_MEMO
General	DB_MEMO

- The following table shows how data types are converted when creating a new table in Visual Basic.

Data Field	Result
DB_BOOLEAN	DB_BOOLEAN
DB_BYTE	DB_DOUBLE
DB_INTEGER	DB_DOUBLE
DB_LONG	DB_DOUBLE
DB_CURRENCY	DB_DOUBLE
DB_SINGLE	DB_DOUBLE
DB_DOUBLE	DB_DOUBLE
DB_DATE	DB_DATE
DB_TEXT	DB_TEXT
DB_LONGBINARY	DB_LONGBINARY
DB_MEMO	DB_MEMO

- There can be only one Memo or LongBinary field per FoxPro Table. It is stored in the database directory as a tablename.FPT file.

- Memo fields in dBASE and FoxPro are for text only. This is not the case for a Microsoft Access memo field, which can contain text or binary data.

Paradox 3.X ISAM

Databases

- Visual Basic version 3.0 is not compatible with Paradox 4.0 or Paradox for Windows. Paradox 4.0 and Paradox for Windows added some new data types that are not compatible with the Paradox ISAM driver in Visual Basic For Windows.

  For more information about compatibility of Paradox 4.0 or Paradox for Windows with Visual Basic, please see the following article in the Microsoft Knowledge Base:

  ARTICLE-ID: Q93699
  TITLE : INF: Access Cannot Attach or Import Paradox 4.0 Tables

- Databases are directories. On a data control or OpenDatabase statement, the exclusive property is ignored. The database name is the path to a directory.

Tables

- In Paradox, the data in a table is ordered physically according to the Primary Key. This is by design in the Paradox database.

Indexes

- The first index created on a Paradox database must be a primary unique index. This is by design in the Paradox database.

- Only the primary index can contain multiple fields, and they must be in the sequential order that they were created. For example, if a table was created with three fields in it, paradox keeps track of the order in which these fields were created. To create a complex index, you must set it on the first n fields of the table. This is by design in the Paradox Database.

- Primary indexes will create a file in the database directory called tablename.PX. If you set the Name property upon creation of a primary index, it will be ignored. After it has been created, the Name property of a primary index will return tablename#px. This is by design in Paradox; it is the way it names the primary index.

- Primary indexes on a Paradox table can not be deleted even if the table is empty.

- Secondary indexes will be named after the field that they are an indexed on. Setting the Name property will be ignored. The Name property after the index has been created will return the name of the field.

- Descending indexes are not supported.

- Records cannot be added without a primary index.

- A Paradox table without a primary key (no *.PX file) can only be opened once because it isn't possible for the Paradox ISAM to keep track of updates without a primary key.

Data Types

- The following table shows how data types are converted to Microsoft Access when reading an existing table.

Paradox	Microsoft Access
Alphanumeric	DB_TEXT
Currency	DB_DOUBLE
Date	DB_DATE
Number	DB_DOUBLE
Short number	DB_INTEGER

- The following table shows how data types are converted when creating a new table in Visual Basic.

Data Field	Result
DB_BOOLEAN	DB_INTEGER
DB_BYTE	DB_INTEGER
DB_INTEGER	DB_INTEGER
DB_LONG	DB_DOUBLE
DB_CURRENCY	DB_DOUBLE
DB_SINGLE	DB_DOUBLE
DB_DOUBLE	DB_DOUBLE
DB_DATE	DB_DATE
DB_TEXT	DB_TEXT
DB_LONGBINARY	error
DB_MEMO	error

- You cannot create a field of Type LongBinary or Memo on a Paradox table.

REFERENCES
==========

BTRIEVE.TXT, EXTERNAL.TXT, and Appendix C of Visual Basic version 3.0 "Professional Features Book 2."

Btrieve is Manufactured by Novell, dBASE III, dBASE IV, and Paradox 3.X are manufactured by Borland International Inc., both vendors are independent of Microsoft. We make no warranty implied or otherwise, regarding the performance or reliability of these products.

Additional reference words: 3.00
KBCategory: APrg
KBSubCategory: APrgDataIISAM

Referential Integrity Enforced for DBs Created in Access

Article ID: Q104983

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY
=======

Microsoft Visual Basic version 3.0 for Windows has no built-in features or properties that provide Referential Integrity. To make Visual Basic version 3.0 for Windows enforce referential integrity rules on a Microsoft Access database, build the database in Microsoft Access. To do this, open the Database window in Microsoft Access, and choose Relationships... from the Edit menu. Then in the Relationships window, select the Enforce Referential Integrity option.

MORE INFORMATION
================

For more information on the way Visual Basic version 3.0 handles and enforces referential integrity, please refer to page 85 of "Professional Features Book 2."

Visual Basic version 3.0 can enforce referential integrity between tables as long as the Enforce Referential Integrity option was selected in Microsoft Access. Visual Basic enforces these rules by providing certain error codes when a database built in Microsoft Access has violated the referential integrity rules in Visual Basic code. These are trappable errors in Visual Basic, so you as the programmer have the option to handle these violations as you wish.

Below are the possible errors you could get that refer to referential integrity:

- Couldn't initialize data access because file 'SYSTEM.MDA' couldn't be opened.

  Error 3028

  In order to ensure referential integrity in databases created by Microsoft Access, Visual Basic must read the Access SYSTEM.MDA file. Make sure the file is in the location specified in the SystemDB entry in the [Options] section in the .INI file.

- Can't delete or change record. Because related records exist in table 'Item', referential integrity rules would be violated.

  Error 3200

  You tried to perform an operation that would have violated referential integrity rules for related tables. For example, this error occurs if you try to delete or change a record in the "one" table in a one-to-many

relationship when there are related records in the "many" table. If you want to delete or change the record, first delete the related records from the "many" table.

- Can't add or change record. Referential integrity rules require a related record in table 'Item'.

  Error 3201

  You tried to perform an operation that would have violated referential integrity rules for related tables. For example, this error occurs if you try to change or insert a record in the "many" table in a one-to-many relationship, and that record doesn't have a related record in the table on the "one" side. If you want to add or change the record, first add a record to the "one" table that contains the same value for the matching field.

There is more information in the Visual Basic version 3.0 "Professional Features Book 2" manual. Referential integrity implementation differences between Microsoft Access and Visual Basic are described on page 119. System table differences are explained on page 21, and using multiple tables is described on page 85.

Dynaset objects can create an inconsistent dynaset with the DB_INCONSISTENT flag. But it may be harder to keep referential integrity when this flag is specified. See pages 58, 59, and 85 of "Professional Features Book 2."

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

How to Query for Dates Using a SQL Statement in VB 3.0

Article ID: Q105173

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

When you query for Dates in a Microsoft Access database, you may receive an incorrect result or a 'Type Mismatch' error message. To query for a date in a SQL statement in Visual Basic version 3.0 for Windows, enclose the date in pound signs (#).

NOTE: This article shows dates written in American format (MM/DD/YY). For example, 12/31/60 means December 31, 1960.

MORE INFORMATION

The following example code selects every field from the Employees table in the NWIND.MDB sample database where the field Birth Date is greater than 12/31/60. NWIND.MDB is the Microsoft Access sample database provided with Microsoft Access versions 1.0 and 1.1.

```
   ' Data1 is a data control.
   Data1.DataBase = "C:\ACCESS\NWIND.MDB"
   ' Enter the following two lines as one, single line:
   Data1.RecordSource = "SELECT * FROM Employees
      WHERE [Birth Date] > #12/31/60#"

   ' The following example uses FindFirst with the same Data Control:
   Data1.RecordSet.FindFirst "[Hire Date] <= #9/21/92#"
```

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

How to Use VB Control Property or Variable in SQL Statement

Article ID: Q105539

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

You can have a Visual Basic application build a SQL query based on choices made by a person using the application. The application can then use the SQL query when creating a view into a database.

This article describes methods developers can use to create SQL queries that are based on control properties or names of variables. The information in this article applies to the following methods: FindFirst, FindLast, FindNext, FindPrevious, CreateDynaset, CreateSnapshot, Execute, and ExecuteSQL.

MORE INFORMATION

When building a SQL query, do not include the variable or control name inside the SQL string; instead, you should reference its value. Using the variable or control name inside the SQL string is a mistake. For example, the following code is incorrect:

```
Dim ds As Dynaset
   ds.FindFirst "NameField = Text1.Text"    'this code is incorrect
```

This code is trying to create a dynaset that finds the first occurrence of the contents of Text1 in a field called NameField. Although the code will not produce an error, it will not find the desired value. It will search for the first occurrence of the string "Text1.Text" not the value of the Text1.Text control property.

The criteria being sought is a string, so the programmer must use string concatenation rules to build the criteria string. The following gives the correct version of the code example:

```
Dim ds As Dynaset
   ds.FindFirst "NameField = '" & Text1.Text & "'"
```

The ampersand (&) operator concatenates the strings together correctly. Also, in SQL syntax, you need to enclose string data in single quotation marks to differentiate strings from variables.

If you think the corrected version looks confusing with all the single and double quotation marks, you can assign the criteria to a string. Then use Debug.Print to view the contents of the string. The following is the same example enhanced to take advantage of the debug window:

```
   Dim ds As Dynaset
   Dim SQL$ as String
      SQL$ = "NameField = '" & Text1.Text & "'"
      Debug.Print SQL$
      ds.FindFirst SQL$
```

If Text1 contains the string "Wilson," the Debug windows displays:

```
   NameField = 'Wilson'
```

If the data type of a field is a number instead of a string, don't enclose the value being sought in single quotation marks. For example, use the following code to create a dynaset that finds the first occurrence of a zip code in a field called ZipCodeField where the ZipCodeField data type is not a string:

```
   Dim ds As Dynaset
   Dim ZipCodeVar as Double
   Dim SQL$ as String
      ZipCodeVar = 98052
      SQL$ = "ZipCodeField = " & ZipCodeVar    'This line builds the string
      Debug.Print SQL$
      ds.FindFirst SQL$
```

Additional reference words: 3.00 pitfall RecordSource
KBCategory: PRG
KBSubcategory: APrgDataOther

PRB: Error or GP Fault When Pass Data Control as Control

Article ID: Q105540

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, version 3.0

SYMPTOMS

Trying to reference the properties of a Recordset or Database using a data control that was passed as Control to a Sub or Function results in this error message:

```
Invalid object reference
```

In addition, passing a data control into a Sub procedure as Control may result in a general protection (GP) fault, as it does in this example:

```
Sub subtest (crtname As Control)
   crtname.UpdateControls
   ' Simply having this line in the code causes the GP fault.
   ' If you turn the line into a comment, the GP fault does not occur.
End Sub
```

CAUSE

Recordset and Database specific properties are not available to objects passed as Control. You should replace the "As Control" with "As Data."

NOTE: the Data object type is not easy to find in the Visual Basic documentation. It is, however, listed with the other possible object data types in the "Object Variables" popup on the Help menu topic, "Database Objects." Object types include CheckBox, ComboBox, CommandButton, CommonDialog, Control, Data, DirListBox, DriveListBox, FileListBox, Form, Frame, Grid, HScrollBar, Image, Label, ListBox, MDIForm, OptionButton, PictureBox, TextBox, Timer, and VScrollBar. The Professional Edition adds the following additional object types: Database, Dynaset, Field, Fields, Index, Indexes, QueryDef, Snapshot, Table, TableDef, and TableDefs.

WORKAROUND

Pass the data control as Data. For example, use the following code instead of the code shown below in the MORE INFORMATION section:

```
Sub MySub(d As Data)
   Debug.Print d.Recordset.EOF
```

MORE INFORMATION

The following code example gives the error message:

```
   Sub MySub(d As Control)
      Debug.Print d.Recordset.EOF    ' an error appears on this line
                                '^^^     with EOF highlighted
```

Additional reference words: 3.00 gpf
KBCategory: APrg
KBSubCategory: APrgDataOther

PRB: Invalid Property Value When Binding Masked Edit Control

Article ID: Q105766

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

Binding a masked edit control (MSMASKED.VBX) to a data control can result in the error "Invalid property value" when the form is loaded. Also, the error "Type mismatch" can occur when you attempt to update a record. For example, this problem can occur when a masked edit control is bound to a date field and has the mask ##/##/##.

CAUSE

Visual Basic uses the Text property of a bound masked edit control to transfer data to and from the data control. This requires strict compatibility, character for character, between the Mask property and the format of the data in the database. For example, dates are stored in a database as a numeric value, not as a string of the format dd/mm/yy.

WORKAROUND

Bind an invisible text box to the data control instead of binding the masked edit control. Then transfer data between the text box and the masked edit control. This allows you precise control over the format of the data for the masked edit control making it appear as if the masked edit control is bound.

This alternative approach involves more lines of code and complexity, but it should prove much more flexible and forgiving. Below is an example showing how to use this technique with a date field.

STATUS

This behavior is by design.

MORE INFORMATION

When a bound masked edit control tries to pull data in from a data control, it behaves differently depending upon the PromptInclude property of the masked edit control.

- If PromptInclude is True (the default), the data coming from the data control must match the mask property exactly or an "Invalid property value" error occurs. If a user tries to change a valid value in a bound masked edit control and doesn't entirely fill up the masked edit

control, the error "Type mismatch" occurs upon writing values into a numeric data type field of a database.

- If PromptInclude is False, the masked edit control does not require or provide literal characters from the Mask property. In other words, the Text property operates like ClipText. For example, a Mask of ##/##/## receives the date 1/2/93 into the Text property as 12/93/__.

The only time it would be advisable to bind a masked edit control directly to a data source would be when you are binding the masked edit control to a fixed-length primary key field; that is, a field that holds a unique value for each record in a table, and that value is always a fixed number of characters in length -- for example, a serial number or product identification code)

Step-by-Step Example for the Workaround

This example shows how to use the Masked edit control with an Access database. This particular example demonstrates using the masked edit control on a field of data type Date/Time.

Before testing the example, either load DATAMGR.EXE (located in the \VB directory) or run VISDATA.MAK (located in the \VB\SAMPLES\VISDATA directory.) Open the BIBLIO.MDB sample database by selecting it after choosing Open Database from the File menu and selecting the Access database option.

Next, select the Authors table, and click the Design button. Select the Add button, enter 'Dates' for the field name, and select the Date/Time for the field type. Then choose OK to close the Add Field window, and then click the Open button. Add several dates to the 'Dates' field in an 'mm/dd/yy' format. You may leave some of the 'Dates' fields blank, but you should enter at least five different dates of five different records in the Authors table to test the example.

Now, you are ready to complete the example:

1. Start Visual Basic for Windows, or from the File menu, choose New Project (ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Add the following controls with the associated properties to Form1:

Control	Name	Property Settings
Data	Data1	DatabaseName = "BIBLIO.MDB" RecordSource = "Authors"
MaskedEdit	MaskedEdit1	Mask = "##/##/##" PromptInclude = False
TextBox	Text1	Visible = False DataSource = Data1 DataField = Dates '** this field was added to BIBLIO.MDB previously

3. Add the following lines of code to the (general)(declarations) section of Form1:

```
   Dim UpdFlag As Integer      'Flag to indicate updating the text box from
                               ' the data control or from the masked edit
                               ' control.

   Const MAXMASKLEN = 6        ' This constant is the maximum number of
                               ' characters the user can enter in this
                               ' particular MaskedEdit.

   Function IsValidDate% (MyMask As MaskEdBox, MaskFullLen As Integer)
   ' This function checks the validity of a date in a Masked edit control.
   ' It returns a zero if the FormattedText is not a valid date, a one
   ' if the field is empty, and a two if the FormattedText is valid.
   '
   ' Parameters:
   '    MyMask       - the Masked edit control being checked
   '    MaskFullLen  - max. number of chars the Masked edit control can
hold

      If MyMask.Text = "" Then
         IsValidDate% = 1
      ElseIf Len(MyMask.Text) = MaskFullLen Then
         If IsDate(MyMask.FormattedText) Then
            IsValidDate% = 2
         End If
      Else
        IsValidDate% = 0
      End If
   End Function
```

4. Add the following code to the Load event of Form1:

```
   Sub Form_Load ()

      UpdFlag = False

   End Sub
```

5. Add the following code to the Validate Event of the Data1 Control:

```
   Sub Data1_Validate (Action As Integer, Save As Integer)

      Const  DATA_ACTIONCANCEL   = 0

      If IsValidDate%(MaskedEdit1, MAXMASKLEN) = False Then
         MsgBox "Not a valid date!"
         Action = DATA_ACTIONCANCEL         'don't allow changes
         MaskedEdit1.SetFocus
         Exit Sub
      End If

   End Sub
```

6. Add the following code to the KeyPress event of MaskedEdit1:

```
   Sub MaskedEdit1_KeyPress (KeyAscii As Integer)

      UpdFlag = True

   End Sub
```

7. Add the following code to the Change event to MaskedEdit1:

```
   Sub MaskedEdit1_Change ()

      If UpdFlag = True Then
         Select Case IsValidDate%(MaskedEdit1, MAXMASKLEN)
            Case 1
               Text1.Text = ""
            Case 2
               Text1.Text = CVDate(MaskedEdit1.FormattedText)
         End Select
      End If

   End Sub
```

8. Add the following code to the Change event of Text1:

```
   Sub Text1_Change ()
      Const DATEFMT = "mmddyy"

      ' The invisible text box can get changed two ways: from the
      ' database because it is bound or from the MaskedEdit when pushing
      ' values back into the data control. This condition handles the
      ' situation when the data is coming from the database and the
      ' MaskedEdit needs to be updated.

      If Not UpdFlag Then
         If Text1.Text = "" Then            ' If NULL condition  then
            MaskedEdit1.Text = ""           ' Set the MaskedEdit to ""
         Else
            MaskedEdit1.Text = Format$(Text1.Text, DATEFMT) 'Format output.
         End If
      End If
      UpdFlag = False

   End Sub
```

9. Press the F5 key to run the program. The masked edit control should behave as if it was bound to the data control.

   NOTE: This example verifies that the dates are valid in the Validate event before actually placing the dates in the database.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

How Visual Basic Handles Security Set by Microsoft Access

Article ID: Q105990

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

Visual Basic version 3.0 includes the Microsoft Access database engine. Visual Basic contains the syntax to manipulate a Microsoft Access database in almost every way that Microsoft Access can. One major exception is in the area of security. Only Microsoft Access can set or modify security options (such as logon IDs and passwords for the system) and set or modify permissions on specific objects in a particular database.

Visual Basic version 3.0 does contain two statements (SetDataAccessOption and SetDefaultWorkspace) that allow a Visual Basic application to satisfy the security mechanism that Microsoft Access implements, and log on using Visual Basic code. By using these statements, you can gain the permissions granted to a particular user.

This article explains those mechanisms of Microsoft Access security that apply to Visual Basic version 3.0 and the Visual Basic programmer. The entire security capabilities of Microsoft Access are beyond the scope of this article.

MORE INFORMATION

Microsoft Access security is implemented in two parts:

- Each user and group has a unique security ID (SID) code.
- That SID code is stored in the database along with the associated permissions for that SID.

The next two sections give the details.

Each User and Group Has Unique Security ID (SID)

In Microsoft Access, each User and Group has a security ID (SID). The SID is a binary string that uniquely identifies the User or Group. When a user logs on, whether from the logon dialog in Microsoft Access or from code in Visual Basic (illustrated later in the article), the Microsoft Access engine reads from the MSysAccounts table of the SYSTEM.MDA database. This database is created only by Microsoft Access and a new (empty) one will be created if the original copy is deleted.

NOTE: If the original SYSTEM.MDA is accidentally deleted, all the unique SIDs are lost. Therefore, all ability to gain access to protected databases is also lost. Therefore, it is a good idea to back up both the database and

the SYSTEM.MDA file in place when the permissions were set on the database.

When logging on, the user supplies the user name (not case-sensitive) and the password (case-sensitive). If the user name and password are correct, the SID of the user is retrieved and saved in a structure internal to the engine. The password is used only to validate the user. From this point on, once the user becomes a validated user, the password has no effect on security.

NOTE: Here is a key point that pertains to Visual Basic's behavior. By default, the Microsoft Access engine attempts to validate the user and password of Admin and "" respectively. Visual Basic version 3.0 will, without any code, send this key combination to the Microsoft Access engine by default. This means that, even without the use of the Visual Basic security-related statements, the Visual Basic program will gain admission to the database, if the user "Admin" of Group Admins has not had its password changed from the default of none ("").

Once logged on, the user's SID is retrieved. This SID is used for all subsequent operations within the Microsoft Access engine.

The SID Is Stored in the SYSTEM.MDA Database

The SID is stored in the database itself. Therefore, all permissions granted to a particular User or Group are also stored in the database, associated with the unique SID.

This brings up another key point pertaining to Visual Basic's behavior, which is a possible source of confusion. The Visual Basic program will gain entrance to the database and have full permissions, seeming to ignore the Microsoft Access security mechanism if either of the following is true:

- The Visual Basic programmer has not taken the location of the SYSTEM.MDA database into account in the program code.

- The User "Admin" has not had its password altered from the default of none ("").

This occurs because of the default behavior of both the Microsoft Access engine and Visual Basic. The combined effect is to allow entry to the database and its objects by the Visual Basic code.

The list of the object types in Microsoft Access are: Table, Query, Form, Report, Macro, and Module. Of these, only the first two are accessible from Visual Basic code, so the others can be omitted from this explanation.

The following two sections explain each of the two Visual Basic security-related statements (SetDataAccessOption and SetDefaultWorkspace). The two statements are designed to provide a choice of SYSTEM.MDA files and logon entries to an Microsoft Access database, with security set by Microsoft Access. Following these two sections is a section that relates the two statements to the behavior of the Microsoft Access engine with regard to security.

SetDataAccessOption Statement -- Syntax and Behavior

SetDataAccessOption has the following parameters:

```
SetDataAccessOption option, value
```

where option is a numeric value with at present only one legal value (1). For example:

```
SetDataAccessOption 1, "e:\vbproj\My.ini"
```

In the DATACONS.TXT file supplied at the root of the \VB directory, a constant is defined for this value:

```
Global Const DB_OPTIONINIPATH = 1
```

SetDataAccessOption sets the name and path of your application's initialization (.INI) file. The application's .INI file takes effect only when SetDataAccessOption is used before the data access functionality is loaded and initialized. Once data access has been initialized, this setting cannot be changed without first exiting the application. The value is a string expression. For the DB_OPTIONINIPATH option, the value argument contains a string expression providing the path and name of your application's initialization (.INI) file. Initialization files are usually stored in a user's \WINDOWS directory, and have the same name as the executable file but with a .INI extension. Use this statement only if your application's initialization file has a different name or is in a directory other than the \WINDOWS directory.

The SetDataAccessOption statement is not needed when you run the Visual Basic project in the VB.EXE environment if the VB.INI file (in the \WINDOWS directory) contains the following lines:

```
[Options]
SystemDB=T:\ACCESS\SYSTEM.MDA
UtilityDB=T:\ACCESS\UTILITY.MDA
```

NOTE: the actual location of the SYSTEM.MDA is not significant provided both Microsoft Access and Visual Basic have an entry pointing to the SYSTEM.MDA they will share. The SetDataAccessOption statement is not required if the application .EXE file has its own .INI file in the \WINDOWS and the .EXE and .INI files share the same name.

SetDefaultWorkspace Statement -- Syntax and Behavior

SetDefaultWorkspace has the following parameters:

```
SetDefaultWorkspace username, password
```

If this statement is left out, Visual Basic will send the equivalent of the following line to the Microsoft Access engine included with Visual Basic:

```
SetDefaultWorkspace "Admin" , ""
```

This statement has the effect of obtaining a valid SID and gaining entry to all the Table and Query objects in the database.

Relationship Between Visual Basic and Microsoft Access Security

First, here is a detailed explanation of the Microsoft Access security mechanism for the benefit of the Visual Basic programmer who has not used Microsoft Access extensively:

There is a hierarchy of permissions in Microsoft Access. At the top level, there are Groups. Contained within a particular Group are Users. To grant permissions selectively to particular Users, all permissions must first be deselected or removed from the Users' Group. Then and only then, can permissions be granted or revoked for individual Users.

Permissions listed for an individual User are called Explicit permissions. Permissions set for the Group containing the User account are called Implicit permissions. Implicit permissions take priority over Explicit permissions.

You can use the Security menu to set permissions in Microsoft Access after a database has been opened and the user has logged on. From the Security menu, choose Permissions to assign permissions on each object in the database, which in Visual Basic means Table and Query objects only.

For example, if there was a Group in the Microsoft Access database named Analysts containing the Users Bob and Sue and you want to limit Bob to Read Data only and grant Sue Full Permissions, follow these steps:

1. Log on as a User in the Admins Group. For example, enter Admin or Fred.

2. From the Security menu, choose Permissions (ALT S P).

3. Table objects are the type selected by default so select the name of the table you want to set permissions on. For example, select TestTbl.

4. Set the option in the User/Group frame to Groups. Then click the combo box list down and click Analysts to select that Group.

5. Clear all check boxes to revoke all permissions for the entire Group.

6. Change the List option button back to Users and select Bob. Clear the check boxes for all of Bob's permissions.

7. Select Sue from the list, and check the Full Permissions check box.

8. Click the Assign button to apply the changes to the table.

At this point, assume you have a Visual Basic program containing the following code in the form load event:

```
Sub Form_Load ()
   Dim db As database
   Dim ds As dynaset

   ' Scenario one:
   ' SetDataAccessOption 1, "e:\vb.ini"    ' not in \WINDOWS directory
   ' SetDefaultWorkspace "bob", "leftout"
```

```
    ' Scenario two:
    ' SetDataAccessOption 1, "e:\vb.ini"    ' not in \WINDOWS directory
    ' SetDefaultWorkspace "bob", "leftout"                          ' point 0

    ' Scenario three:
    ' SetDataAccessOption 1, "e:\vb.ini"    ' not in \WINDOWS directory
    ' SetDefaultWorkspace "bob", "leftout"

    ' Scenario four
    ' SetDataAccessOption 1, "e:\vb.ini"    ' not in \WINDOWS directory
    ' SetDefaultWorkspace "bob", "leftout"

    Set db = OpenDatabase("e:\datacon\bases\access11\asample.mdb") ' point 1
    Set ds = db.CreateDynaset("TestTbl")                           ' point 2

    autoredraw = True   ' to make Print  statement persist on the form
    Print ds(0), ds(1)

End Sub
```

Assume that this code is run with the comment apostrophes removed from only one of the four scenarios and that each scenario is run in order.

SCENARIO ONE: In this case, there is no reference to the location of the SYSTEM.MDA file. Windows and the Microsoft Access engine are unable to find the .INI file with the [Options] section listed previously in this article. Therefore, the SYSTEM.MDA is ignored and Visual Basic defaults to its default user and password combination ("Admin", ""). However, previously, the default password for the User Admin was changed to something other than "". In addition, all permissions were revoked for the Group Admins and the User "Admin" in the Admins Group. Therefore, the following Visual Basic error occurs at point 2:

```
   Couldn't read; no read permission for table or query 'f))'
```

We have closed the back door to Visual Basic and any Visual Basic application attempting to bypass the logons in the SYSTEM.MDA file.

SCENARIO TWO: In this case, because we invoke the SetDefaultWorkspace statement without having any pointer to the SYSTEM.MDA file, the Visual Basic Microsoft Access engine hunts for the SYSTEM.MDA file and, not finding it, gives the following error at point 0 in the code:

```
   Couldn't find file 'SYSTEM.MDA'
```

NOTE: The errors that occur in both Scenarios one and two are the same as would occur if the SYSTEM.MDA file was moved, renamed, or deleted.

SCENARIO THREE: In this case, we tell the Visual Basic Microsoft Access engine where the SYSTEM.MDA file resides, but we don't supply a user and password combination, so again, Visual Basic supplies the only user and password combination it knows ("Admin", ""), which is no longer a valid combination because we added a password to the Admin User account. As a result, Visual Basic gives the following error at point 1 in the code:

```
   Not a valid account or password.
```

SCENARIO FOUR: In this case, we supply both parameters correctly. Therefore, because we gave Bob Read Data permission as well as Read Definitions to allow the Visual Basic Microsoft Access engine to read, the Visual Basic application prints the first two fields in the first record of the table named TestTbl.

If we repeated the four scenarios with the User Sue, all would be the same. However, Sue could go further and modify the table structure and the data as well. Remember that we first selected the Group analysts and revoked all permissions. Then we added back all permissions to Sue, but only Read Data and Read Definitions were added back to Bob.

NOTE: The Admins Group has special significance with regard to security. This applies to any User in that Group. The Admins group's SID is stored in the SYSTEM.MDA when a database is created. As a result, the Admins group will always have permission to change the permissions on all objects in that database. This permission cannot be taken away by anyone. This permission remains even when all permissions have been revoked from the Admins Group, and it is not displayed in the Permissions dialog. This is another reason to keep a backup and keep track of which SYSTEM.MDA was in use when the database was created.

One last point of possible confusion revolves around the use of the following phrase in a SQL query:

```
... With OwnerAccess Option
```

For example, look at this code:

```
Sub Form_Load ()
   Dim db As Database
   Dim qd As querydef

   Set db = OpenDatabase("c:\access\db1.mdb")

   ' Enter the following two lines of code as one, single line:

   Set qd = db.CreateQueryDef("myQD", "select * from [TableDetails]
      with owneraccess option ;")
   db.Close
End Sub
```

This code results in this error:

```
Invalid Database ID.
```

This is because OwnerAccess refers to the owner of the database. The owner is the creator of the database. In other words, OwnerAccess refers to the owner's user and password combination (unique SID) that is stored in the database (BD1.MDB in this case). However, the code does not contain the two statements needed to point to the SYSTEM.MDA file of a secured database. Actually, in this case, only the SetDefaultWorkspace statement is essential if the compiled .EXE file's .INI file containing a valid [Options] section, is in the \WINDOWS directory.

The code uses the backdoor. It has not supplied the unique SID of the

database owner to the engine, so the engine doesn't know the default name and password combination (Admin, "") of the user is the database owner. Even if it turns out that the User Admin is the database owner, without having read the SYSTEM.MDA file, the engine cannot verify that fact, so it gives the error.

Key Points to Remember

1. Only Microsoft Access can create and modify the SYSTEM.MDA file.

2. The SYSTEM.MDA file contains the unique SID used in a database with permissions to sort out who is who for the Microsoft Access engine to enforce those permissions. The SID is obtained by supplying the Microsoft Access engine with a valid user and password combination, from which it obtains the unique SID that the engine stores in memory to enforce security on an open database.

3. Both Microsoft Access and Visual Basic need to be pointed to the location off the SYSTEM.MDA file in order to gain entry to databases that have security and permissions implemented.

4. There is a back door available to the Visual Basic application program if the password for the default User in the Admins group (named Admin) is not changed from the default none ("").

5. If the phrase "With OwnerAccess Option" is used in the SQL query of a CreateQueryDef, CreateDynaset, or CreateSnapshot method, a pointer to the SYSTEM.MDA file must exist. Even if you are using the back door (the default user and password combination of Admin and "") and don't seem to need the SYSTEM.MDA, when you use "With OwnerAccess Option" in a SQL query, the engine must see the SYSTEM.MDA file to match the SID of the owner (creator) of the database to the user who logged on.

6. The valid logon user and password combinations are stored in the SYSTEM.MDA file but the permissions are stored in the database (.MDB file) itself. A unique key (the SID) is extracted from the SYSTEM.MDA by using a valid user and password combination, supplied to the Microsoft Access engine by the logon dialog in Microsoft Access or by the code in Visual Basic.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

PRB: Illegal to Use Find Methods w/ SQL PASSTHROUGH & ODBC DB

Article ID: Q106111

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

When you create a Dynaset or Snapshot using the SQL PASSTHROUGH option with an ODBC database, the FindFirst, FindNext, FindLast, and FindPrevious methods give the error "Can't perform operation; it is illegal."

CAUSE

FindFirst, FindNext, FindLast, and FindPrevious work only on record sets opened by a query. Visual Basic version 3.0 doesn't use a query to open a record set when you use DB_SQLPASSTHROUGH, so these Find methods are not allowed, by design.

RESOLUTION

The SQL PASSTHROUGH option causes the query to be processed by an external database server, instead of by Visual Basic. Avoid using the SQL PASSTHROUGH option if you want to use the FindFirst, FindNext, FindLast, or FindPrevious methods with an ODBC database.

Also, you can avoid the problem by creating a copy of the dynaset or snapshot. This will allow the Microsoft Access Engine to perform the FindFirst rather than allowing the ODBC server to do it. However, the the dynaset copy must not use the SQL PASSTHROUGH option. Here is an example:

```
Dim db as database
Dim ds as dynaset
Dim newds as dynaset

Set db = OpenDatabase("", 0, 0, "ODBC;DSN=texas")
Set ds = db.createdynaset("Select * from Authors", 64) ' SQL PASSTHROUGH
Set newds = ds.createdynaset()  ' No SQL PASSTHROUGH
newds.FindFirst ""  ' Now FindFirst works on newds
```

STATUS

This behavior is by design.

REFERENCE

Pages 58-60, Visual Basic Professional Edition, Version 3.0, "Professional

Features Book 2."

MORE INFORMATION
==========

Steps to Reproduce Behavior

The following code results in the "Can't perform operation; it is illegal" error:

```
   Dim db As database
   Set db = OpenDatabase("", 0, 0, "ODBC;DSN=texas")
   Dim ds As Dynaset   ' Creates a dynaset.
   ' The DB_SQLPASSTHROUGH option is 64:
   Set ds = db.CreateDynaset("select * from authors", 64)
   ds.FindFirst ""     ' FindFirst causes error message.
```

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataODBC

PRB: Illegal to Use Find Method with Table Object Variable

Article ID: Q106270

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

Using the FindFirst, FindNext, FindLast, or FindPrevious method on an object variable of type Table results in this error:

```
Can't perform operation; it is illegal.
```

Pressing the F1 key on this error dialog gives the following description from the Visual Basic Help:

```
Error 3219.
You tried to use a method or property with or on a recordset,
and it isn't valid for that object.
```

CAUSE

The FindFirst, FindNext, FindLast, and FindPrevious methods can be used only with a Dynaset or Snapshot. These Find methods cannot be used with a Table object variable.

WORKAROUND

To move between the records of a Table, use the Seek method or the Move methods (MoveFirst, MoveLast, MoveNext, and MovePrevious).

Also, you can create a Dynaset or Snapshot variable on the whole Table by using the CreateDynaset or CreateSnapshot method. Then you can use the FindFirst, FindNext, FindLast, and FindPrevious methods on that Dynaset or Snapshot. The Find methods move between records that meet specific conditions.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Reproduce Behavior

The following code demonstrates the error using the FindFirst method on a Table object variable:

```
   Dim db As database
   Dim recordset As table  ' Correction: Dim recordset As dynaset
   Set db = OpenDatabase("c:\vb3\biblio.mdb")
   Set recordset = db.OpenTable("authors") ' Instead: db.CreateDynaset
   ' The following line gives "can't perform operation; it is illegal":
   recordset.FindFirst "Author like 'a*'"
   Debug.Print recordset.Fields("Author")
```

The following code works around this behavior by first creating a Dynaset from the Table, and then using FindFirst on the Dynaset:

```
   Dim db As database
      ' Dim recordset As table   ' Gives problem.
   Dim recordset As dynaset      ' Workaround.
   Set db = OpenDatabase("c:\vb3\biblio.mdb")
      ' Set recordset = db.OpenTable("authors")   ' Gives problem.
   Set recordset = db.CreateDynaset("Authors")    ' Workaround.
   recordset.FindFirst "Author like 'a*'"
   Debug.Print recordset.Fields("Author")
```

REFERENCES
==========

See "Positioning the Current Record in a Recordset" on Pages 68-77 of the Visual Basic Professional Edition, version 3.0, "Professional Features Book 2" manual. Page 72 states, "The Find methods cannot be used on Table objects."

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Call SQL Stored Procedures from Visual Basic

Article ID: Q106492

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article describes how to call Microsoft SQL stored procedures from Visual Basic. A stored procedure is a precompiled collection of SQL statements, often including control-of-flow language.

MORE INFORMATION

The method of calling depends on whether the SQL stored procedure returns records or not:

1. Stored procedures that don't return records (or rows) can be executed from Visual Basic with the ExecuteSQL method as follows:

   ```
   i% = MyDb.ExecuteSQL("sp_name")
   ```

   This executes the stored procedure sp_name and returns the affected number of rows in i%. The ExecuteSQL method is strictly for action queries such as:

   ```
   Delete Authors where name like "fred%"
   ```

   The ExecuteSQL method is valid only for SQL statements that do not return records (or rows). An SQL statement that uses "SELECT..." returns records, while an SQL statement that uses "DELETE..." does not. Neither Execute nor ExecuteSQL return a recordset, so using ExecuteSQL on a query that selects records produces an error.

2. Stored procedures that return records (or rows) require a Dynaset or Snapshot to capture the values. Here are two examples:

   Example Using a Data Control on a Visual Basic Form:

   ```
   DB_SQLPassThrough = 64
   Data1.Options = DB_SQLPassThrough
   Data1.Recordsource = "sp_name"  ' name of the stored procedure
   Data1.Refresh   ' Refresh the data control
   ```

   When you use the SqlPassThrough bit, Visual Basic's Microsoft Access database engine will ignore the syntax used and will pass the command through to the SQL server.

   Alternative Example Using Object Variables:

   ```
   Dim Ds as Dynaset
   ```

```
      Set MyDB = OpenDatabase(... ' Open your desired database here.
      Set Ds = MyDB.CreateDynaset("sp_name",Db_SQLPassThrough)
      ' You can also Dim as Snapshot and use MyDb.CreateSnapshot above.
```

How to Pass Parameters to a Stored Procedure

To pass parameters, include them after the name of the stored procedure in a string, for example:

```
   SQLx = "My_StorProc parm1, parm2, parm3"  ' String specifying SQL
                                             ' command.
   ...
   i = MyDB.ExecuteSQL(SQLx)  ' For stored procedure that
                              ' doesn't return records.
   ...
   set Ds = MyDB.CreateDynaset(SQLx,64) ' For stored procedure that
                                        ' returns records.
```

The object variable (Ds) will contain the first set of results from the stored procedure (My_StorProc).

Another Example

Here's more example code showing both methods:

```
   Dim db as Database; l as long; Ss as Snapshot

   ' Enter the following two lines as one, single line:
   Set Db = OpenDatabase
      ("",false,false, "ODBC;dsn=yourdsn;uid=youruid;pwd=yourpwd:")

   l=ExecuteSQL("YourSP_Name")        ' for SPs that don't return rows
   Set Ss = Db.CreateSnapshot("YourSP_Name", 64) ' for SPs that return rows
   Col1.text = Ss(0) ' Column one
   Col2.text = Ss!ColumnName
   Col3.Text=Ss("ColumnName")
```

REFERENCES

More information about calling stored procedures is documented in the following Microsoft SQL manual which covers the Visual Basic Library for SQL Server:

- Microsoft SQL Server Programmer's Reference for Visual Basic

See the functions SqlRpcInit% (pages 200-201), SqlRpcParam%, and SqlRpcSend%. These functions call stored procedures more quickly than do the methods described above.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

PRB: Invalid Database Object after Rollback without BeginTrans

Article ID: Q106493

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

After your program gets this error:

```
Commit or Rollback without BeginTrans (Err = 3034)
```

clicking a data control or executing a database method may result in this error:

```
Invalid Database Object
```

CAUSE

You tried to commit or roll back a transaction that you didn't start with a BeginTrans statement.

WORKAROUND

To avoid the problem entirely, always do a BeginTrans before attempting a Rollback.

You can work around the "Invalid Database Object" error by using the Refresh method on the data control. For example, add the statement Data1.Refresh after the Rollback, at the bottom of the Command1_Click procedure shown in the Steps to Reproduce Behavior section below.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Reproduce Behavior Using Database Object Variables

1. Start Visual Basic or begin a New Project. Form1 is created by default.

2. Add a command button (Command1) to Form1. Enter the following code:

```
Sub Command1_Click ()
   Dim db As database
```

```
      Dim ds As dynaset
      Set db = OpenDatabase("c:\vb3\biblio.mdb")
      Set ds = db.CreateDynaset("authors")
      On Error Resume Next
      ' WORKAROUND: Add the following statement here:   BeginTrans
      Rollback
      Print Error$
      On Error GoTo 0
   End Sub
```

Steps to Reproduce Behavior using Text Control Bound to Data Control
---

1. Start Visual Basic or begin a New Project. Form1 is created by default.

2. Add a data control (Data1) to Form1, and give Data1 these properties:

   ```
   DatabaseName = C:\VB\BIBLIO.MDB   ' This database shipped with VB
   RecordSource = Authors            ' Use the Authors Table.
   ```

3. Add a text box (Text1) to Form1, and give Text1 these properties:

   ```
   DataSource = Data1
   DataField = Au_ID
   ```

4. Add a command button (Command1) to Form1. Enter the following code:

   ```
   Sub Command1_Click ()
      On Error Resume Next
      Rollback
      Print Error$
      ' WORKAROUND is to add the following statement here:  Data1.Refresh
   End Sub
   ```

5. Start the program by pressing the F5 key. Click the Command1 button. The following correct error is trapped by the error handler:

   ```
   Commit or Rollback without BeginTrans
   ```

6. Now click the data control arrow to move to the next record. This causes the following error:

   ```
   Invalid Database Object
   ```

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

PRB: No Current Record Error In VB When Database is Empty

Article ID: Q106494

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

If a data control and text box are both bound to an empty table in a database, clicking the data control arrows gives this error:

   No Current Record

This error also occurs if you enter text into the text box, and execute the AddNew method or the Edit method while the database table is empty.

The program has difficulty recovering because of the recurring "No Current Record" errors. The On Error statement fails to trap this error.

CAUSE

The program does not know the table is empty until the automatic record update occurs. That automatic record update occurs when you click the data control arrow or you enter text in the text box and then execute an AddNew or Edit method.

When the table is not empty the automatic update is a nice feature, but when the table is empty the automatic update causes the no current record error. The error message occurs because the underlying recordset contains no records.

You must execute AddNew to create a current record before doing anything that causes an automatic record update.

WORKAROUND

To work around this behavior, execute the AddNew method on an empty database table before allowing the user to click the data control or enter text into the bound text box control. For example, set the Enabled property for the text control and data control to False at the beginning of the program. Then you can force the user to click a command button that executes the AddNew method before enabling the text and data controls.

STATUS

This behavior is by design. This design is under review and will be considered for enhancement in a future release.

More Information

How to Create an Empty Microsoft Access Database

Before using Example 1 or 2 shown below, create an empty database by following these steps in Visual Basic:

1. Open the Data Manager program by choosing it from the Window menu in Visual Basic or by running DATAMGR.EXE from the Windows File Manager.

2. In the Data Manager, choose New Database from the File menu and select Access 1.1.

3. Click the New button and enter tbl1 for the table name.

4. Click the Design button. Then click the Add button. Enter fld1 for the Field Name and select Integer for the Field Type.

5. Save this Microsoft Access database with the name TEST1.MDB. Close the Data Manager.

Example One: Steps to Reproduce Behavior using a Data Control

1. Start a new project in Visual Basic. Form1 is created by default.

2. Create the empty Microsoft Access database (TEST1.MDB) as described above.

3. Add the following controls and set the following properties:

Control Name	Property	New Value	NOTE
Data1	DatabaseName	C:\VB\TEST1.MDB	Empty MDB created above.
Data1	RecordSource	tbl1	Table name.
Text1	DataSource	Data1	Name of data control.
Text1	DataField	Fld1	Field name.
Command1	Caption	"Press for AddNew"	

4. Add the following code in the Command1 Click event procedure:

```
Sub Command1_Click ()
   data1.Recordset.AddNew
   text1.SetFocus
End Sub
```

5. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program.

6. Either of the following will give the no current record error:

   - Click any of the arrow buttons on the data control.

   - Enter text into the text box, and then click the command button.

Notice that if you rerun the program, press the Command1 button first, and then enter some text into the Text1 text box, you can then click any of the arrow buttons on the data control without getting the error.

Example One Workaround

To work around this behavior, set the Enabled property for the text and data controls to False at design time or in the Load procedure for the form. Then add the following to the top of the Command1_Click procedure before the AddNew:

```
Data1.enabled = True
Text1.enabled = True
```

This prevents the user from automatically updating an empty database, thus avoiding the no current record error.

Example Two: Steps to Reproduce Behavior Using Object Variables
----

1. Start a new project in Visual Basic. Form1 is created by default.

2. Create the empty Microsoft Access database (TEST1.MDB) as described above.

3. Add a command button. Enter "Press to Test" as its Caption property.

4. Add the following code in the general Declarations section of Form1:

```
Dim db As database
Dim ds As dynaset
```

5. Add the following code in the Command1 Click event procedure:

```
Sub Command1_Click ()
   Set db = OpenDatabase("TEST1.MDB")
   Set ds = db.CreateDynaset("tbl1")
   ' Execute the following line to work around problem:
   ' ds.AddNew
   If IsNull(ds(0)) Then  'The No Current Record error occurs here
      Print "No entry"
   Else
      Print ds(0)
   End If
End Sub
```

6. From the Run menu, choose Start (ALT, R, S), or press the F5 key to run the program.

To correct the error, add the line ds.AddNew shown in a comment above.

REFERENCES
==========

Additional information can be found in the Visual Basic Help menu. The no current record error is described in the Visual Basic Help topic "Data

Access error messages." Here is that description:

> No current record. Error 3021.
>
> This error occurs following the unsuccessful application of one of the FindFirst, FindLast, FindNext, FindPrevious Methods or the Seek Method, or when the underlying recordset contains no records. Move to or select a record, and then try the operation again.

The following is paraphrased from the Help topic for AddNew in Visual Basic:

> The AddNew method clears the copy buffer in preparation for creating a new record in a Table or Dynaset. AddNew sets all fields in the copy buffer to Null and makes it the current record. After putting data in the record, you can use the Update method to add the record to the recordset. Update is automatically invoked with a data control if an Edit or AddNew operation is pending when you use one of the Find or Move methods.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

How VB Can Determine If Table Is Locked By Other Processes

Article ID: Q106535

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article describes how to detect if a database table has any records locked by other users or processes. If you open the table with the options to deny read and write access, a trappable error will indicate that other users or processes are using the table. This information is useful for managing tables in a multiuser or network system.

MORE INFORMATION

Step-by-Step Example

The following example uses one program (PROGLOC1.EXE) to optionally lock a record in a table. A separate, concurrent program (PROGLOC2) checks to see if any records in the table are currently locked.

1. Make PROGLOC1.EXE by following these steps in Visual Basic:

   a. Start a new project in Visual Basic. Form1 is created by default.

   b. Add a large command button to Form1.

   c. Enter the following code for the Command1_Click event procedure:

```
Sub Command1_Click ()
   Dim db As database
   Set db = OpenDatabase("biblio.mdb")
   Dim ds As dynaset
   Set ds = db.CreateDynaset("authors")
   ds.Edit     ' Locks the first record in the dynaset.
   MsgBox "First record in dynaset is locked. Press OK to unlock."
   command1.Caption = "record now unlocked"
End Sub
```

   d. Choose Save File As from the File menu, and save as PROGLOC1.FRM. Choose Save Project As from the File menu, and save as PROGLOC1.MAK.

   e. Choose Make EXE File from the File menu to create PROGLOC1.EXE.

2. Make PROGLOC2 by following these steps in Visual Basic:

   a. Start a new project in Visual Basic. Form1 is created by default.

b. Add a large command button to Form1.

c. Enter the following code for the Command1_Click event procedure:

```
Sub Command1_Click ()

   Dim db As database
   Set db = OpenDatabase("biblio.mdb")
   Dim tb As table
   ' See if table has locks by opening and denying others Read/Write:
   On Error Resume Next
   Set tb = db.OpenTable("authors", 3) ' 3 = Deny Read & Write (2+1)
   If Err = 0 Then
      command1.Caption = "not locked"
   Else
      command1.Caption = "locked due to err=" & err
   End If
   tb.Close
   ' If no error here you could reopen table without denying access.

End Sub
```

d. Optional steps to save this sample program:
   Choose Save File As from the File menu, and save as PROGLOC2.FRM.
   Choose Save Project As from the File menu, and save as PROGLOC2.MAK.

e. Run PROGLOC1.EXE from Windows File Manager and click the command button to lock a record. Leave the following message box up without pressing OK:

   First record in dynaset is now locked. Press OK to unlock.

f. Start PROGLOC2 from Visual Basic by pressing the F5 key. Click the command button to report whether or not a record is locked.

Additional reference words: 3.00 row locking
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Change Read-Only Access of a Data Control at Run Time

Article ID: Q107074

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article describes and demonstrates techniques you can use to change (toggle on or off) read-only access of a data control or database at run time. All behavior discussed in this article is by design.

MORE INFORMATION

Notes to Remember When Changing Read-Only Access

For database object variables, you must close all database variables opened for a given database before you can change its read-only access. Use the Close method to close a database object stored in a variable. Use the OpenDatabase function with a parameter to specify the read-only access for the opened database.

For a data control, you must use the Refresh method to reset the data control after you change its read-only access by changing the ReadOnly property. Another more complicated way to reset the data control is to unload the form, which closes the data control. Then use Load and Show to reload the display form. You must set the data control's new ReadOnly property value in the Form Load event procedure, or else ReadOnly will use its design-time setting.

The read-only parameter of the OpenDatabase function will be ignored at run time when you open a database that is currently bound to a data control. To change the read-only access of a bound database at run time, you must set the ReadOnly property for the data control and use the Refresh method.

The ReadOnly property of a data control will retain the value that you set, but the new ReadOnly property won't take effect until you use the Refresh method. For example, if you change the ReadOnly property from True to False and fail to use the Refresh method, the database bound to the data control will remain read-only even though the ReadOnly property does contain the new value of False. You must execute the Refresh method on the data control to tell the bound database that you changed the ReadOnly property.

The following examples demonstrate how to change the read-only access of a database at run time.

Close Database Before Changing Read-Only Access

The following two examples demonstrate that you must close a database before changing the read-only access (False = 0, True = -1).

Example One -- Using Database Object Variables:

Open database A and set the read-only parameter to True. Without closing database A, open database A a second time and set read-only access to False. The second open uses read-only access, ignoring your request for read-only to be False. This behavior is by design. The following code example demonstrates this behavior:

```
Dim d1 As database, d2 As database
Set d1 = OpenDatabase("biblio.mdb", 0, -1) 'set read-only option True
debug.print d1.Updatable   ' Updatable prints False, 0, as expected.
Set d2 = OpenDatabase("biblio.mdb", 0, 0)  'set read-only option False
debug.print d2.Updatable   ' Updatable still prints False, 0
```

By definition, the Updatable property returns False when the read-only access is True, and vice versa. In the above example, the Updatable property of database variable d2 remains False from the first open, despite your request to turn off read-only access.

To reset the read-only access, close the database first. Then change the read-only access. For example:

```
Dim d1 As database, d2  As database
Set d1 = OpenDatabase("biblio.mdb", 0, -1) ' set read-only option True
debug.print d1.Updatable  ' Updatable prints False, 0, as desired.
d1.Close
Set d2 = OpenDatabase("biblio.mdb", 0, 0)  ' set read-only option False
debug.print d2.Updatable  ' Updatable prints True, -1, as desired.
```

Database variable d1 is erased by the Close. The Updatable property of the second database variable (d2) will be False, as desired. Updatable returns False when the database was opened with read-only access.

Example Two -- Using a Text Control Bound to a Data Control:

A data control does implicit OpenDatabase and CreateDynaset function calls at load time using design-time properties such as DatabaseName, RecordSource, and ReadOnly.

If you change the ReadOnly property of a data control at run time, you must use the Refresh method to reset the database. The following steps show how to do it:

1. Start Visual Basic or begin a New Project. This creates Form1 by default.

2. Add a data control (Data1) to Form1, and give Data1 these properties:

```
DatabaseName:  C:\VB\BIBLIO.MDB   [This database ships with VB.]
RecordSource:  Authors            [Uses Authors Table in BIBLIO.MDB.]
ReadOnly:      True               [Makes ReadOnly=True at start.]
```

3. Add a text box (Text1) to Form1. Give Text1 the following properties:

```
   DataSource:    Data1
   DataField:     Authors
```

4. Add two command buttons (Command1 and Command2) to Form1. Change the Caption of Command1 to say "Enable Database Writing." Change the Caption of Command2 to say "Make Database Read-only." Enter the following code:

```
   Sub Command1_Click ()
      debug.Print Data1.ReadOnly   ' DATA1.ReadOnly starts True, -1
      Data1.ReadOnly = False       ' Change to False
      Data1.Refresh   ' Refresh required to change ReadOnly access.
      debug.Print Data1.ReadOnly   ' Prints False, 0
   End Sub

   Sub Command2_Click ()
      debug.Print Data1.ReadOnly   ' Prints False, 0
      Data1.ReadOnly = True        ' Change to True
      Data1.Refresh   ' Refresh required to change ReadOnly access.
      debug.Print DATA1.ReadOnly   ' Prints True, -1
   End Sub
```

5. Start the program by pressing the F5 key or choosing Start from the run menu.

6. Add some characters to the record in the text box. This record is the Authors field of the C:\VB\BIBLIO.MDB database. Click the right arrow on the data control to move to the next record. This automatically updates the first record if you have write access. Then click the left arrow on the data control to view your change to the first record. When ReadOnly is True, such as at the beginning of the program, no change will occur in the record shown in Text1.

7. Now click the Command1 button, "Enable Database Writing." Then repeat step 6 to update a record. Because ReadOnly is False now, the Text1 box will now reflect your change in the record.

8. Now click the Command2 button, "Make Database Read-only." Then repeat step 6 to update a record. Because ReadOnly is True now, no change will occur in Text1.

9. Close the form to end the program.

You Can Open Different Databases with Different Read-only Access

Each different database you open can have a different read-only setting. In the following example, d1.Updatable will be True and d2.Updatable will be False, because they refer to different databases:

```
   Dim d1 As database, d2 As database
   Set d1 = OpenDatabase("biblio.mdb", 0, -1)  ' Set read-only option True
   Set d2 = OpenDatabase("mydb.mdb", 0, 0)     ' Set read-only option False
   debug.print d1.Updatable, d2.Updatable      ' Prints -1 and 0
```

Additional reference words: 3.00 R/W status lock locking how-to
KBCategory: APrg

KBSubcategory: APrgDataOther

Possible Reasons for Couldn't Find Installable ISAM Error

Article ID: Q107672

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

There are a number of reasons that can cause the 'Couldn't Find Installable ISAM' error message. Below is a list of the possible reasons for the error.

MORE INFORMATION

The Data Access Guide in "Professional Features Book 2" has information in the sections: 'General Tips for Using External Tables' and 'Initialization File Details' that can be used as an additional reference. These sections begin on pages 134 and 148.

The path for the installable ISAM driver should be listed in the file VB.INI in the \WINDOWS directory. Here is an example showing what the Installable ISAM section in VB.INI should look like:

```
[Installable ISAMs]
Btrieve=C:\WINDOWS\SYSTEM\btrv110.dll
FoxPro 2.0=C:\WINDOWS\SYSTEM\xbs110.dll
FoxPro 2.5=C:\WINDOWS\SYSTEM\xbs110.dll
dBASE III=C:\WINDOWS\SYSTEM\xbs110.dll
dBASE IV=C:\WINDOWS\SYSTEM\xbs110.dll
Paradox 3.X=C:\WINDOWS\SYSTEM\pdx110.dll
```

List of Possible Reasons for Error (Couldn't find Installable ISAM)

---

1. An entry in the [Installable ISAM] section in VB.INI or <APPNAME>.INI is incorrect. For example, this error occurs if you're accessing a Paradox external table, and the Paradox entry of the APP.INI file points to a nonexistent directory.

   To correct this problem, exit Visual Basic. Then make necessary corrections in VB.INI or <APPNAME>.INI by using Microsoft Windows Notepad or another text editor. Then restart Visual Basic, and try the operation again.

2. One of the entries in the [Installable ISAM] section in VB.INI points to a network drive, and that drive isn't connected. Check to make sure the network is available and the correct drive letter is established. Then try the operation again.

3. The APP.INI file isn't in the Windows directory.

4. The APP.INI file doesn't have the same name as the .EXE file.

5. The path to the IISAM specified in the .INI file is incorrect.

6. There are extraneous spaces in the APP.INI file.

7. The APP.INI file doesn't contain the same IISAM syntax as in the VB.INI for each database that uses the drivers.

8. The connect string should be 'Paradox 3.X;' NOT 'Paradox;' this is a documentation error in the manual.

9. Another possibility is that if you installed Microsoft Access after installing Visual Basic, an older driver may have replaced a newer Visual Basic driver. Check to ensure that the following files are all dated 4-28-93 (the Visual Basic version 3.0 file date stamp):

   MSAES110.DLL
   MSAJT110.DLL
   VBDB300.DLL

10. The error can also occur when the Connect property of Tabledef is not filled with the exact characters needed.

    The Connect property is analogous to the connect portion of the OpenDatabase method, so the parser looks for semicolons after every entry -- other than the null entry in the case of a native Microsoft Access database. In the case of an attached table, Visual Basic syntax requires an entry in the Connect property. There needs to be a leading semicolon before any optional parameters in the Connect property, if the source database of the attached table is Microsoft Access. If it is omitted, the error occurs.

    In other words, the leading semicolon is needed when attaching a native Microsoft Access table. If it is omitted, the first parameter placed in the Connect property is seen as the database type. The error occurs because the parser tries to find an entry in the .INI file that corresponds to the first entry in the Connect property, but it doesn't exist. In the first example, the parser looks for a match for "database=" and fails to find it.

11. Providing an extra space between dBASE and IV, as in dBASE IV, prevents the Microsoft Access engine from finding the entry in the .INI file, so the error occurs.

12. Another possible reason for this error message is that the ISAM driver can't be loaded because the file is corrupt or not decompressed. You can verify this by attempting to force the DLL to load using the WPS.EXE program (available in the Professional Edition only), which you will find in the CDK subdirectory. If WPS can't load the DLL, you should reinstall and expand the DLL by using Setup /Z.

13. If you create a program that uses the ISAM drivers, your APPNAME.INI file should contain the entire [Installable ISAMs] section. If you don't do this the follow scenario could occur: A customer may have a program that uses the paradox driver and it works fine with the

following APPNAME.INI:

```
[Installable ISAMs]
Paradox 3.X=C:\WINDOWS\SYSTEM\pdx110.dll
```

Then a second program that uses a Btrieve driver works fine with the following APPNAME.INI:

```
[Installable ISAMs]
Btrieve=C:\WINDOWS\SYSTEM\btrv110.dll
```

But if you run the two programs together, the second will have the error "Can't Find Installable ISAM."

NOTE: the order of execution doesn't play a factor.

To prevent this problem, include the entire [Installable ISAMs] section in your APPNAME.INI files. For example, it might be similar to this:

```
[Installable ISAMs]
Btrieve=C:\WINDOWS\SYSTEM\btrv110.dll
FoxPro 2.0=C:\WINDOWS\SYSTEM\xbs110.dll
FoxPro 2.5=C:\WINDOWS\SYSTEM\xbs110.dll
dBASE III=C:\WINDOWS\SYSTEM\xbs110.dll
dBASE IV=C:\WINDOWS\SYSTEM\xbs110.dll
Paradox 3.X=C:\WINDOWS\SYSTEM\pdx110.dll
```

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataIISAM

How to Create a Parameter Query in Visual Basic for Windows

Article ID: Q107748

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article explains how to create and use a parameter query. A parameter query is a type of QueryDef specific to Visual Basic and Microsoft Access. Parameter queries enable you to automate the process of changing query criteria. With a parameter query, you can set new values for the parameters each time you run the query.

MORE INFORMATION

A parameter query is created in a program by using the CreateQueryDef() function. Here is the syntax for the CreateQueryDef() function:

```
Set querydef = database.CreateQueryDef(name, sqltext)

querydef - a QueryDef object
database - a Database object
name     - string containing query name
sqltext  - string containing the SQL query text
```

The sqltext string is optional or it can be defined by using the .SQL property of the QueryDef. To create a parameter query, place the PARAMETERS statement in the sqltext string. Here is the syntax for the PARAMETERS statement:

```
PARAMETERS parametertext datatype

parametertext - name of the parameter
datatype      - type of the parameter
```

The following table lists the appropriate Microsoft Access SQL datatype that should be used with the PARAMETERS statement as well as the corresponding Microsoft Access field type, Visual Basic variable type, and constant value from the DATACONS.TXT file.

Microsoft Access SQL	Microsoft Access Field	Visual Basic Type	DATACONS.TXT Constant
Bit	Yes/No	Integer	DB_BOOLEAN = 1
Byte	Byte	Integer	DB_BYTE = 2
Short	Integer	Integer	DB_INTEGER = 3
Long	Long Integer	Long	DB_LONG = 4
Currency	Currency	Double	DB_CURRENCY = 5
IEEESingle	Single	Single	DB_SINGLE = 6

IEEEDouble	Double	Double	DB_DOUBLE = 7
DateTime	Date/Time	Variant	DB_DATE = 8
Binary	Binary	String	
Text	Text	String	DB_TEXT = 10
LongBinary	OLE Object	String	DB_LONGBINARY = 11
LongText	Memo	String	DB_MEMO = 12

Following the PARAMETERS statement in the sqltext string, place the query. The query can refer to the parameter (parametertext) named in the PARAMETERS statement. Wherever the query refers to a parameter the current value will be substituted when the query is executed.

For example, if the query text is:

```
PARAMETERS i SHORT; SELECT fld FROM tbl WHERE fld=i
```

and the parameter i was set to 42 in the program. The parameter i would be substituted and the resulting query would be equivalent to:

```
SELECT fld FROM tbl WHERE fld=42
```

Multiple Parameters in a PARAMETERS statement

It is also possible to have multiple parameters in a PARAMETERS statement. To do this, use commas to separate the parameters as follows:

```
PARAMETERS parametertext datatype, parametertext datatype, ...
```

Prior to executing the query, set the parameters using this syntax:

```
querydef!parametertext = value
```

```
querydef      - a QueryDef object
parametertext - the name of the parameter in the PARAMETERS statement
value         - the value the parameter will have
```

In the previous example, you would use QD!i=42 before executing the query.

Once the parameters are set, you are ready to execute the query. There are three methods (Execute, CreateDynaset, and CreateSnapshot) supported by a QueryDef that will cause the query to be executed.

More information on parameter queries is available in the Visual Basic, version 3.0, "Professional Features Book 2."

Example Parameter Query

The following example illustrates the use of a short parameter in a query. The example has two parts. The first part creates a new QueryDef for BIBLIO.MDB (the sample Microsoft Access database that ships with Visual Basic) and should be executed only once. The second part uses the QueryDef to create a snapshot, which is then displayed. To test the example, place each of the following code segments in a command button Click event procedure:

```
   'Create QueryDef "by date"
   Dim Db As Database
   Dim Qd As QueryDef
   Set Db = OpenDatabase("C:\VB\BIBLIO.MDB")
   Set Qd = Db.CreateQueryDef("By date") 'Create the query "By date"
   QdText = "PARAMETERS dp Short; "
   QdText = QdText & "SELECT * from Titles WHERE [Year Published] = dp"
   Qd.SQL = QdText
   Print Qd.SQL
   Qd.Close
   Db.Close

   ' Create Snapshot from QueryDef
   Dim Db As Database
   Dim Qd As QueryDef
   Dim Sn As Snapshot
   Set Db = OpenDatabase("C:\VB\BIBLIO.MDB")
   Set Qd = Db.OpenQueryDef("By Date")  'Open the "By date" query
   Qd!dp = 1991                         'Set the value of the dp parameter
   Set Sn = Qd.CreateSnapshot()         'Create a snapshot from the query
   Sn.MoveFirst
   Do Until Sn.EOF
      For i = 1 To Sn.Fields.Count - 1
         Print Sn(i);                   'Display results of query
      Next
      Print
      Sn.MoveNext
   Loop
   Sn.Close
   Qd.Close
   Db.Close
```

Additional reference words: 3.00 parameterized querydefs
KBCategory: APrg
KBSubcategory: APrgDataOther

LONG: PERFORM.TXT - Performance Tuning Tips for VB and Access

Article ID: Q107751

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article contains the complete text of the PERFORM.TXT file distributed with the Standard and Professional Editions of Visual Basic version 3.0 for Windows.

MORE INFORMATION

```
                      PERFORM.TXT

  Release Notes for Microsoft (R) Visual Basic (R) Professional Edition

                      Version 3.00

            (C) Copyright Microsoft Corporation, 1993

This document contains performance tuning tips for Microsoft Visual Basic
for Windows version 3.0 and Microsoft Access (TM) Relational Database
Systems for Windows version 1.1.

------------------------
How to Use This Document
------------------------

To view PERFORM.TXT on screen in Windows Notepad, maximize the Notepad
window.

To print PERFORM.TXT, open it in Windows Write, Microsoft Word, or
another word processor. Then select the entire document and format the
text in 10-point Courier before printing.

========
Contents
========

Part    Description
----    -----------
 1      Running Multiple Data Access Applications
 2      Manipulating Secured Microsoft Access Databases
 3      Tuning [ISAM] Entries in VB.INI or <APPNAME>.INI
 4      Using Transactions to Maximize Data Throughput
 5      Minimizing Keyset Overhead When Working with Large Recordsets
 6      Performance Tips for Visual Basic Data Access
```

Part 1: Running Multiple Data Access Applications

Visual Basic and Microsoft Access both use the same database engine to perform their database-related operations. When these applications are run at the same time on the same machine, it is important to coordinate use of the database engine, since it is only initialized by the first program accessing a data access operation. Re-initialization does not take place until all programs using the database engine are ended and another started.

Proper database initialization is especially important when accessing external databases like dBASE, FoxPro, Paradox, or Btrieve. All of these require special notations in the initialization file.

Note that data access applications can take the form of one or more instances of:

- Microsoft Access
- A Microsoft Access application
- Visual Basic at design time
- Visual Basic at run time
- A Visual Basic application

For example, you might have a copy of Microsoft Access running with an .EXE file created with Visual Basic. On the other hand, you might have two or more dissimilar applications running -- both of which need to access the database engine. You will need to make sure that the initialization files are set up to deal with each of these situations.

To ensure that all applications using the database engine function as intended, you must ensure that all initialization parameters pertaining to external databases are identified in various .INI files under section headings (such as [Installable ISAM], [Paradox Isam], [Btrieve ISAM]) and are copied into each of the .INI files pertaining to each data access application that can potentially be running at the same time. The list below shows where each program looks for its initialization information:

Program	.INI file
Microsoft Access	MSACCESS.INI
Microsoft Access application	MSACCESS.INI
Visual Basic at design time	VB.INI
Visual Basic at run time	VB.INI
Visual Basic .EXE application	<APPNAME>.INI

All of these .INI files are located in your Windows directory. During development, your Visual Basic application defaults to VB.INI unless your application uses the SetDataAccessOption statement to indicate a specific .INI file location. Once you create an executable program with Visual Basic, the initialization file will default to <APPNAME>.INI unless you use SetDataAccessOption.

If you want to coordinate operations between an instance of a Visual Basic application (or .EXE), you will want to force Visual Basic to indicate the same initialization file that a second instance of your

program or an instance of Microsoft Access will use. This way, regardless of which application starts (and initializes) the database engine, both applications will be using the same initialization parameters.

Part 2: Manipulating Secured Microsoft Access Databases

For Visual Basic to manipulate secured Microsoft Access databases, you must provide Visual Basic with the location of the SYSTEM.MDA file associated with that Microsoft Access database, a valid user name, and password. This can be accomplished in three steps:

1) Use the SetDataAccessOptions statement to point to a valid .INI file.
2) Include a path to the SystemDB with a valid VB.INI or <appname>.INI entry to locate the file. For example:

```
[Options]
SystemDB=C:\ACCESS\SYSTEM.MDA
```

3) Set the user name and password (if other than "admin" with no password) with the SetDefaultWorkspace statement.

Part 3: Tuning [ISAM] Entries in VB.INI or <APPNAME>.INI

You can enhance the database access performance of Visual Basic by:

- Adding or changing entries in the VB.INI or <APPNAME>.INI initialization file
- Using transactions

Visual Basic automatically provides default internal settings for most common database operations. However, advanced users may want to tune these settings to provide maximum performance for a particular system configuration or application.

Setup automatically installs <APPNAME>.INI in your Windows directory. These [ISAM] entries determine the sizes of data page and read-ahead caches in memory, the amount of time data is held in a page cache, and the number of times Visual Basic will retry a lock operation.

WARNING

Determining the best settings for your system configuration or application can be time-consuming and difficult, usually involving much trial and error. In addition, settings that seem optimal for one situation may not be optimal for others. Casual users should not try to edit these entries.

Visual Basic automatically includes a PageTimeout entry in the [ISAM] section of the VB.INI or <APPNAME>.INI file. This entry sets the amount of time Visual Basic holds a data page in memory. For additional performance tuning, you can add MaxBufferSize, ReadAheadPages, LockRetry, and CommitLockRetry entries to this section.

Visual Basic reads these initialization settings at startup time. They can

be changed while Visual Basic is running, but the changes won't take effect until you restart Visual Basic or your application. As with all .INI settings that affect the database engine, these settings are fixed once the engine is initialized just before the first data access operation.

PageTimeout Entry (Shared Data Only)

The PageTimeout entry sets the amount of time, in tenths of a second, that Visual Basic holds a data page in a memory "page cache" if the database has been opened for shared access. Visual Basic reads data in 2K blocks of records, or "pages."

For example, when Visual Basic reads a data page, it places the data in the page cache. If Visual Basic receives another read request for the same data page during the timeout period, it reads the data directly from the page cache rather than from disk.

Note

Access Basic ignores the PageTimeout setting unless your code allows background processing by periodically calling the DoEvents statement or function.

PageTimeOut Settings

Maximum: 2147483647 (max Long integer)
Minimum: 0
Default: 300

If you remove the PageTimeout entry, Visual Basic uses a default PageTimeout setting of 5. For example:

```
PageTimeout=20     ; This example sets PageTimeout to 2 seconds.
```

MaxBufferSize Entry

The MaxBufferSize entry sets the amount of memory, in kilobytes, reserved for use as a page cache. Visual Basic reads data in 2K pages, placing the data in the page cache. Once the data is placed in the cache, Visual Basic can use it wherever it is needed -- in tables, queries, forms, or reports.

When Visual Basic receives a read request, it first checks the data pages in the page cache. If the page isn't in the cache, Visual Basic reads the data page from disk and then places it in the page cache. Visual Basic uses physical memory and if necessary virtual memory to create the cache. All pages stay in the cache until it is full and pages need to be flushed to make room for new reads.

MaxBufferSize Settings

Maximum: 4096
Minimum: 18
Default: If there is no MaxBufferSize entry in your VB.INI or <APPNAME>.INI file, Visual Basic uses a default setting of 512.

Note

Because Visual Basic reads data in 2-kilobyte pages, it always uses an even MaxBufferSize setting. If you type an odd number, Visual Basic uses a MaxBufferSize setting of one less than the number. For example:

```
MaxBufferSize=4096  ; This example sets MaxBufferSize to 4 MB.
```

ReadAheadPages Entry

The ReadAheadPages entry sets the size, in data pages, of a "read-ahead" cache used by Visual Basic for sequential page reads. A sequential page read occurs when Visual Basic detects that data in a current read request is on a data page adjacent on physical disk to the data page of the previous request. Visual Basic uses the "read-ahead" cache only when it detects that a sequential read is taking place.

- If Visual Basic detects a sequential page read, it reads the requested page plus the next (N-1) pages in that direction, where N is the ReadAheadPages setting, placing the data pages in the read-ahead cache.
- If Visual Basic then detects a sequential read, it can make the next N reads directly from the read-ahead cache.

The read-ahead cache increases the speed of sequential reads, especially for reading data stored on a network. It increases record updates per second (throughput) on a network by sending a few large packets rather than many small packets over the network. If possible, Visual Basic places the read-ahead cache in the first 640K of memory in order to benefit from the ability of Windows to read from and write to conventional memory. If the read-ahead cache can't be placed in conventional memory, Visual Basic places it in high memory. Placing the cache in high memory is less efficient than placing it in conventional memory because Windows must copy all reads and writes to its own buffer before completing the memory operation.

ReadAheadPages Settings

Maximum: 31
Minimum: 0
Default: If there is no ReadAheadPages entry in your VB.INI or <APPNAME>.INI
file, Visual Basic uses a default setting of 8. For example:

```
ReadAheadPages = 16
```

Note

Visual Basic creates a separate read-ahead cache for each database open on your computer. Each library database has its own read-ahead cache.

LockRetry Entry

The LockRetry entry sets the number of times Visual Basic retries a page-locking operation before it reports an error. For example, if a user tries to lock a data page that is already locked by another user, the attempt will fail. Visual Basic will try to lock the page N more times, where N is the LockRetry setting. If the attempt to lock the page still fails on the Nth retry, Visual Basic reports an error.

LockRetry Settings

Maximum: 2147483647 (max Long integer)
Minimum: 0
Default: If there is no LockRetry entry in your VB.INI or <APPNAME>.INI
file, Visual Basic uses a default setting of 20. For example:

```
LockRetry = 6
```

CommitLockRetry Entry

The CommitLockRetry entry is used with the LockRetry entry to set the number of retries that Visual Basic attempts when a user tries to lock a record on a data page already locked by a transaction. If a user tries to lock a data page that is already locked by a transaction, Visual Basic will try to lock the page N more times, where N is the product of the LockRetry setting and the CommitLockRetry setting. For example, if the LockRetry setting is 5 and the CommitLockRetry is 6, Visual Basic will try to lock the page 30 more times.

CommitLockRetry Settings

Maximum: 2147483647 (max Long integer)
Minimum: 0
Default: If there is no CommitLockRetry entry in your VB.INI or
<APPNAME>.INI file, Visual Basic uses a default setting of 20.
For example:

```
CommitLockRetry = 6   ; Assuming a LockRetry setting of 6,
                      ; this example causes Visual Basic to
                      ; retry a page locked by a transaction 36
                      ; times.
```

==========
Part 4: Using Transactions to Maximize Data Throughput
==========

In a multiuser environment, you can further tune the performance of Visual Basic by using transactions for operations that update data. A transaction is a series of operations that must execute as a whole or not at all. You mark the beginning of a transaction with the BeginTrans statement. You use the Rollback or CommitTrans statement to end a transaction.

You can usually increase the record updates per second (throughput) of an application by placing operations that update data within an Access Basic transaction.

Tip

Because Visual Basic locks data pages used in a transaction until the transaction ends, using transactions will prevent access to those data pages while the transaction is pending. If you use transactions, try to find a balance between data throughput and data access.

==========
Part 5: Minimizing Keyset Overhead When Working with Large Recordsets
==========

When a query selects a large number of records from the database, Visual Basic only fetches the first row of that Dynaset or Snapshot and places the key to refetch that row in memory. Once a record is fetched or visited, it becomes a member of the recordset. As you "visit" additional rows of the recordset, the keys are stored in workstation memory (in a temporary table), and in the case of Snapshots, so is the data. If you move back to previously fetched rows, Visual Basic refetches the rows using the old key fetched from the temporary key table.

- If the database record is no longer there, you get a trappable error.
- If the record has changed, the new information is fetched from the database.

As you move further and further into the recordset, more and more memory is taken up storing the keys. Eventually, Visual Basic will begin saving the keyset on disk. If this happens, space is used on disk in the directory specified by your \TEMP environment variable. Generally, you won't see a performance degradation until Visual Basic has to swap the keyset temporary table to disk. If you run out of disk space because Visual Basic has exhausted the space in your \TEMP directory, you will get a trappable error.

Moving to the end of the Dynaset or Snapshot does at least two things: First, it forces Visual Basic to visit all of the records in your recordset. Hence, all keys will be saved on the workstation. If this is a few hundred rows, this may not take long or take up more space than the workstation can handle.

However, for larger recordsets, a MoveLast operation may be far more than the workstation can save. When working with Snapshots, not only are the keys fetched, but the data for all records is also brought into local memory. Generally, you should avoid operations that fetch more rows than your user or workstation can deal with. Operations that must touch each record in a recordset may best be performed with an action query that consumes less system resources. In any case, your performance will not be severely degraded, either as you move forward until you have to swap, or hardly at all if you move backwards in the recordset -- even to the first record.

Note

The Dynaset or Snapshot membership is not set until the record is actually fetched for the first time. Since this can take from seconds to days depending on how fast you fetch the records (moving down through the recordset with MoveNext or with MoveLast), no Dynaset or Snapshot is really a frozen subset of the data at a point in time. The only way to ensure that no changes are made while the recordset is built is to get exclusive access to the table or database before fetching -- which essentially locks out all other users until the recordset (or database) is closed.

==========
Part 6: Performance tips for Visual Basic Data Access
==========

The following tips are suggested for operations involving more than just a few records to increase the overall performance of your system.

1) When working with large recordsets (Dynasets or Snapshots), do not use the MoveLast method unless absolutely necessary.

   Moving to the end of a recordset requires Visual Basic to load all keys for the recordset into memory. In the case of Snapshots, not only are keys loaded into memory, but the data is also brought into workstation memory. If temporary memory space is exhausted, Visual Basic may be forced to swap this temporary cache to disk. In this case, Visual Basic will use space as addressed by the \TEMP environment variable. Once this space is exhausted, your application will trigger a trappable error.

2) When you want to access external tables fast, attach the table to your database instead of using the IN clause in a SQL statement or addressing the table directly.

   When Visual Basic needs to access your external table, all linkage information is resolved when the database is opened and does not have to be re-established and initialized each time the data is accessed (for example, with non-attached tables.

3) For reasonably small recordsets, especially where you do not intend to write to the recordset, use Snapshots instead of Dynasets.

   If possible, set the READONLY flag on the data control or DB_READONLY option when opening databases. This will permit Visual Basic to bypass significant logic to handle multi-user read-write access to your tables.

4) In cases where you are working with external ODBC databases, you will achieve maximum possible speed if you use SQL Passthrough instead of attaching or direct access that involves the Visual Basic database engine.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataAcc

Microsoft Access Database RAM Cache Is Faster Data File Method

Article ID: Q107871

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

Before Microsoft Access was available, many Basic programs read disk files into string arrays. Then wrote them back to disk. You may be able to improve the speed and flexibility of file operations by instead using a Microsoft Access database with a RAM cache.

MORE INFORMATION

Visual Basic limits variable-length string arrays to 64K bytes. However, fixed-length string arrays are limited only by memory, and a Microsoft Access database can contain many megabytes of data.

The Microsoft Access database engine used by Visual Basic uses a robust, RAM-based caching scheme for speedy data access.

You can open a Microsoft Access database using a dynaset, which is a set of pointers to the original data. When you update the dynaset, you actually update the underlying physical table on disk. Dynasets are live; they are not just copies of the data. You can specify a large Microsoft Access data buffer, or cache, in RAM to obtain speeds faster than older Basic file input/output methods.

Please read the PERFORM.TXT file for performance tuning tips for data access in Visual Basic version 3.0 and Microsoft Access version 1.1 for Windows. The PERFORM.TXT file is in your Visual Basic directory. A related PERFORM.TXT file is installed with Microsoft Access.

A copy of the PERFORM.TXT file can also be found by searching for the following words in this knowledge base:

   PERFORM.TXT and Visual and Basic

Using a table object variable usually gives faster access to a database than using a data control bound to a database.

Using a Microsoft Access Data Buffer in RAM

You can specify the size of the Microsoft Access buffer, or cache, in RAM by adding an [ISAM] section to your initialization file. Specify the [ISAM] section in the VB.INI file for Visual Basic, or in the <appname>.INI file for your Visual Basic application. For example, use the following cache if your computer has at least eight megabytes of RAM installed:

```
   MAXBUFFERSIZE = 4096
```

This gives the Microsoft Access engine a four-megabyte dedicated cache.

You can also specify ReadAheadPages and other parameters. Please read the PERFORM.TXT file for more information.

Using the above Microsoft Access caching scheme can be faster than using Basic file input/output statements with a disk-caching product such as the SMARTDRV.SYS driver that ships with most MS-DOS versions 4.x and later.

In one test on a PC with a 486 chip and 50-megahertz clock speed, a Seek method used on a cached table took less than .6 milliseconds. FindFirst methods with simple criteria took only .7 to .9 milliseconds. Performance is about the same whether you use the primary key or any other index that is either defined as unique or resolves to a single record pointer.

For sample data manager source code, search for a separate article in this knowledge base using the following words:

```
   Data and Manager and Source and Code and CompuServe
```

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

VISDATA Example of Every Data Access Function in VB Prof 3.0

Article ID: Q108145

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The VISDATA.MAK file installed in the VB3\SAMPLES\VISDATA directory loads extensive examples of data access. The VISDATA sample program uses every data access function in Visual Basic. You can refer to the VISDATA source code for examples of how to use each data access function.

MORE INFORMATION

The following description of the VISDATA sample program is taken from the VB3\SAMPLES\SAMPLES.TXT file:

VISDATA EXCERPT FROM SAMPLES.TXT

```
...
========
6:  ODBC
========

Program example using ODBC and the VT (Virtual Table) object layer.

NOTE:  To access ODBC data sources with this sample you must
       first install ODBC using the ODBC setup program provided
       with Visual Basic Professional Edition.

BRIEF DESCRIPTION:

This sample program illustrates various programming techniques
used to access data through the VT layer built into Visual Basic
Professional. It behaves like a general purpose database utility
capable of the following functions:

   1. Table Creation
   2. Table Modification (adding and deleting fields and indexes)
   3. Data Browsing/Modifying one record at a time using a
      dynaset or a table
   4. Data Browsing via the Grid control (non-updatable)
   5. Data Browsing/Modifying via the new Data Control
   6. Data Export to Tab Delimited text file
   7. Direct SQL statement execution for any SQL supported
      functions such  as Insert, Update, Delete, Drop, Create,
      and Dump
   8. AdHoc Query tool that helps users unfamiliar with SQL
      create complex queries with where clauses, joins, order
```

by and group by expressions while limiting output to selected columns
9. Transaction Processing
10. Copying table structures and data to same or different server
11. Support of Microsoft Access, dBASE 3, dBASE 4, FoxPro 2.0, Paradox 3.x, Btrieve, SQL and Oracle data, both DDL and DML

The code contains comments to help explain the use of the various methods in the data access layer. Code and forms may be copied from this application to other applications with minimal modification.

ODBC BACKGROUND:

ODBC (Open DataBase Connectivity) is a standard adopted by multiple vendors designed to enable users to connect to any data source with a single application. This is achieved through a layered approach including:

1. Programming Layer -- embedded functions in the development tool, which in this case is Visual Basic Professional Edition.
2. Driver Manager -- the basic ODBC library that routes calls to the appropriate driver.
3. Data Driver -- the library of functions that acts upon a specific database backend such as SQL Server, Xbase, Excel, etc. (note that SQL Server is the first of many drivers to become available for ODBC)

These layers work together to enable data access from any source for which an ODBC driver exists. The sample application will work, without modification, on any new, level-one ODBC driver that becomes available. With multiple drivers, connections may be made to different data sources from the same application at the same time enabling seamless data access from disparate data sources.

FILES:

```
ABOUTBOX.FRM......Standard "About box" for the application.
ABOUTBOX.FRX......Icon for the "About Box".
ADDFIELD.FRM......Form to add fields to Tables.
CPYSTRU.FRM.......Form to copy Table structures.
DATABOX.FRM.......General purpose list form.
DYNAGRID.FRM......Form used to display data in a Grid control.
DYNAGRID.FRX......Icon for DYNAGRID.FRM.
DYNASET.FRM.......Form to display data in single record mode.
DYNASET.FRX.......Icon for DYNASET.FRM
FIND.FRM..........Form used to find records in a Dynaset.
INDEXADD.FRM......Form used to add indexes to Tables.
JOIN.FRM..........Form used to add Joins to the Query Builder.
OPENDB.FRM........Form used to open a database.
QUERY.FRM.........Form used to build Queries.
QUERY.FRX.........Icon for QUERY.FRM.
REPLACE.FRM.......Form to perform global replaces on a Table.
REPLACE.FRX.......Icon for REPLACE.FRM.
SEEK.FRM..........Form used to get input for Seek function on
                  Table form.
SQL.FRM...........Form to enter and execute SQL statements.
```

```
   SQL.FRX...........Icon for SQL.FRM.
   TABLES.FRM........Form used to display table lists.
   TABLES.FRX........Icon for TABLES.FRM.
   TABLEOBJ.FRM......Form used to display data in Table object
   TABLEOBJ.FRX......Icon for TABLEOBJ.FRM
   TBLSTRU.FRM.......Form to display and modify table structures.
   VDMDI.FRM.........Main MDI form for the application.
   VDMDI.FRX.........Icon for VDMDI.FRM.
   VISDATA.BAS.......Support functions for the application.
   VISDATA.ICO.......Icon for the applicaiton.
   VISDATA.MAK.......Make file for applicaiton.
   ZOOM.FRM..........Form to zoom in on character data in the
                     dynaset forms.
```

TO RUN:

After starting the Visual Basic environment (VB.EXE), you can load files in this sample program by choosing Open Project from the File menu and selecting the VISDATA.MAK file in the SAMPLES\VISDATA directory.

If you want to open a local database, you need to choose the type of database and a file open common dialog will be provided with the file type set to the requested data file type.

If you choose ODBC from the File/Open menu, the next dialog you will see is the Open DataBase form. Because you probably have no servers entered, you will need to enter a name for an existing SQL server on your network. If you already know the user ID and password, you can add them as well. The Database name is optional. Once you have entered this data, choose Okay. Now, you should be able to log on to the server. You may get some more dialogs in the process. Answer any questions you can and ask the SQL administrator for help if you run into problems or don't know some of the parameters.

Once a database is open, double-click a table name to open the table in the selected mode (Single Record or Table View). Use the Query Builder to create dynasets with selected data from one or more tables at a time. If you choose Use Data Control or Use Grid, a dynaset will be created. However, the following objects will be created under the following circumstances when you choose No Data Control:

Data Type	Feature Chosen	Object Type Created
MS Access	Table Open	Table
MS Access	Query Open	Dynaset
MS Access	Execute SQL	Dynaset
ISAM	Table Open	Table
ISAM	Execute SQL	Dynaset
ODBC	Any	Dynaset

The table is always updatable and the dynaset will be updatable in most cases except on ODBC with no unique index, certain multiple table joins, and other SQL select statements such as count(*) or max().

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Create a Microsoft Access Database using VB Prof 3.0

Article ID: Q108146

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

Below is an example showing how to use the CreateDatabase function and data definition language (DDL) statements to create an empty Microsoft Access database. This example defines the structure of the Microsoft Access database, and doesn't add any data.

MORE INFORMATION

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Double-click the form to open its code window. Add the following code to the Form Load event:

```
Sub Form_Load ()

   Const DB_LANG_GENERAL = ";LANGID=0x0809;CP=1252;COUNTRY=0"
   Dim db As Database
   Dim tdEmployee As New TableDef
   Dim tdStore As New TableDef

   Dim fEmp_ID As New Field
   Dim fEmp_FirstName As New Field
   Dim fEmp_LastName As New Field
   Dim fEmp_Address As New Field

   Dim fStore_ID As New Field
   Dim fStore_Location As New Field

   Dim indxEmployee As New Index
   Dim indxStore As New Index

   Set db = CreateDatabase("COMPANY.MDB", DB_LANG_GENERAL)

   tdEmployee.Name = "Employee"
   tdStore.Name = "Store"

   'Define all Employee fields
   fEmp_ID.Name = "Emp_ID"
   fEmp_ID.Type = 4                 'Long integer
```

```
    fEmp_FirstName.Name = "Emp_FirstName"
    fEmp_FirstName.Type = 10          'Text (32)
    fEmp_FirstName.Size = 32

    fEmp_LastName.Name = "Emp_LastName"
    fEmp_LastName.Type = 10           'Text (32)
    fEmp_LastName.Size = 32

    fEmp_Address.Name = "Address"
    fEmp_Address.Type = 10            'Text (256)
    fEmp_Address.Size = 255

    'Define all Store fields
    fStore_ID.Name = "Store_ID"
    fStore_ID.Type = 4                'Long integer

    fStore_Location.Name = "Store_Location"
    fStore_Location.Type = 10   'Text (256)
    fStore_Location.Size = 255

    'Add employee fields to Fields collection
    tdEmployee.Fields.Append fEmp_ID
    tdEmployee.Fields.Append fEmp_FirstName
    tdEmployee.Fields.Append fEmp_LastName
    tdEmployee.Fields.Append fEmp_Address

    'Add store fields to Fields collection
    tdStore.Fields.Append fStore_ID
    tdStore.Fields.Append fStore_Location

    'Define employee table index
    indxEmployee.Name = "INDEX_EMPLOYEE"
    indxEmployee.Fields = "Emp_ID"
    indxEmployee.Unique = True
    indxEmployee.Primary = True

    'Define store table index
    indxStore.Name = "INDEX_STORE"
    indxStore.Fields = "Store_ID"
    indxStore.Unique = True
    indxStore.Primary = True

    'Append the employee and store indexes
    'to the respective Indexes collection
    tdEmployee.Indexes.Append indxEmployee
    tdStore.Indexes.Append indxStore

    'Append employee and store TableDefs
    'to TableDefs collection
    db.TableDefs.Append tdEmployee
    db.TableDefs.Append tdStore

End Sub
```

3. Start the program or press the F5 key. This creates a Microsoft Access database called COMPANY.MDB. End the program by closing the form.

4. You can check that COMPANY.MDB was created correctly by opening it with Microsoft Access or with the Data Manager provided with Visual Basic. You can run the Data Manager program from the Window menu in Visual Basic, or from the Windows File Manager run DATAMGR.EXE in the Visual Basic directory.

REFERENCES
==========

The VISDATA.MAK file installed in the VB3\SAMPLES\VISDATA directory loads extensive examples of data access. The VISDATA sample program uses every data access function in Visual Basic. You can refer to the VISDATA source code for examples of how to use each data access function.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Copy Table from One Database to Another in VB Prof 3.0

Article ID: Q108147

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

Below is an example of how to copy a table from one database to another using the Professional Edition of Visual Basic version 3.0.

MORE INFORMATION

Sample Program

The following sample code contains two functions taken almost unchanged from the VISDATA sample project, from the code module VISDATA.BAS. The Command1_Click procedure shown below invokes these two functions, CopyStruct and CopyData. NOTE: The VISDATA.MAK project file is installed in the Visual Basic SAMPLES\VISDATA directory.

This example assumes that the databases have Microsoft Access format. The same techniques apply to the other supported database types.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a command button to Form1. Add the following code to the Command1 Click event:

```
Sub Command1_Click ()
   Dim dbsource As database
   Dim dbdest As database
   ' The following hard-coded database names could be changed to
   ' selections from a text box, list box, or combo box to make the
   ' program more generic:
   Set dbsource = OpenDatabase("c:\vb3\biblio.mdb", True, True)
   Set dbdest = OpenDatabase("c:\vb3\test1.mdb", True, False)
   Print CopyStruct(dbsource, dbdest, "titles", "ctitles", True)
   Print CopyData(dbsource, dbdest, "titles", "ctitles")
   dbsource.Close
   dbdest.Close
End Sub
```

3. Add the following code to the General Declarations section of Form1:

```
'Place the following Function statement on one, single line:
Function CopyStruct (from_db As Database, to_db As Database,
   from_nm As String, to_nm As String, create_ind As Integer) As Integer
```

```
On Error GoTo CSErr

Dim i As Integer
Dim tbl As New Tabledef    'table object
Dim fld As Field           'field object
Dim ind As Index           'index object

'Search to see if the table exists:
namesearch:
For i = 0 To to_db.TableDefs.Count - 1
   If UCase(to_db.TableDefs(i).Name) = UCase(to_nm) Then
      If MsgBox(to_nm + " already exists, delete it?", 4) = YES
      Then
         to_db.TableDefs.Delete to_db.TableDefs(to_nm)
      Else
         to_nm = InputBox("Enter New Table Name:")
         If to_nm = "" Then
            Exit Function
         Else
            GoTo namesearch
         End If
      End If
      Exit For
   End If
Next

'Strip off owner if necessary:
If InStr(to_nm, ".") <> 0 Then
   to_nm = Mid(to_nm, InStr(to_nm, ".") + 1, Len(to_nm))
End If
tbl.Name = to_nm

'Create the fields:
For i = 0 To from_db.TableDefs(from_nm).Fields.Count - 1
   Set fld = New Field
   fld.Name = from_db.TableDefs(from_nm).Fields(i).Name
   fld.Type = from_db.TableDefs(from_nm).Fields(i).Type
   fld.Size = from_db.TableDefs(from_nm).Fields(i).Size
   fld.Attributes = from_db.TableDefs(from_nm).Fields(i).Attributes
   tbl.Fields.Append fld
Next

'Create the indexes:
If create_ind <> False Then
   For i = 0 To from_db.TableDefs(from_nm).Indexes.Count - 1
   Set ind = New Index
   ind.Name = from_db.TableDefs(from_nm).Indexes(i).Name
   ind.Fields = from_db.TableDefs(from_nm).Indexes(i).Fields
   ind.Unique = from_db.TableDefs(from_nm).Indexes(i).Unique
   If gstDataType <> "ODBC" Then
      ind.Primary = from_db.TableDefs(from_nm).Indexes(i).Primary
   End If
   tbl.Indexes.Append ind
   Next
End If

'Append the new table:
```

```
      to_db.TableDefs.Append tbl

      CopyStruct = True
      GoTo CSEnd

      CSErr:
      CopyStruct = False
      Resume CSEnd

      CSEnd:
   End Function

   'Place the following Function statement on one, single line:
   Function CopyData (from_db As Database, to_db As Database,
      from_nm As String, to_nm As String) As Integer

      On Error GoTo CopyErr
      Dim ds1 As Dynaset, ds2 As Dynaset
      Dim i As Integer
      Set ds1 = from_db.CreateDynaset(from_nm)
      Set ds2 = to_db.CreateDynaset(to_nm)
      While ds1.EOF = False
         ds2.AddNew
         For i = 0 To ds1.Fields.Count - 1
            ds2(i) = ds1(i)
         Next
         ds2.Update
         ds1.MoveNext
      Wend
      CopyData = True
      GoTo CopyEnd
      CopyErr:
      CopyData = False
      Resume CopyEnd
      CopyEnd:
   End Function
```

4. Start the program or press the F5 key.

5. You can check to see if the table was copied correctly to the TEST1.MDB database by opening TEST1.MDB with Microsoft Access or with the Data Manager provided with Visual Basic. You can run the Data Manager program from the Window menu in Visual Basic or from the Windows File Manager run DATAMGR.EXE in the Visual Basic directory.

REFERENCE
=========

The VISDATA.MAK file installed in the VB3\SAMPLES\VISDATA directory loads extensive examples of data access. The VISDATA sample program uses every data access function in Visual Basic. Refer to the VISDATA source code for examples that show how to use each data access function.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Delete a Field from a Populated Table

Article ID: Q108148

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article shows by code example how to delete one or more fields from an existing table.

Field definitions cannot be removed from the TableDef of the table, but once the Field has been appended to the Fields collection, you can create a new TableDef, minus the unwanted fields, and then copy the data from the old table into the new table. An intermediate step uses a Microsoft Access database as a staging area. This will work for databases other than Microsoft Access because the Microsoft Access database is used and deleted, with the data never having been affected by the intermediate stage.

MORE INFORMATION

Step-by-Step Instructions for Creating the Program

To create a Visual Basic utility program that allows selective field deletions, follow these steps:

1. Start a new project in Visual Basic. This creates Form1 by default.

2. From the File menu, choose New Module, or click on the Toolbar icon second from the left.

3. From the File menu, choose Add File, and add the CMDIALOG.VBX custom control to your project.

4. On the form, create the following controls, and set the design-time properties shown:

Control	Name	Property
Common Dialog	Cmdialog1	(defaults)
Command Button	Pickdb	Caption="Which Database?"
Command Button	Command1	Caption="Copy table minus fields"
Text Box	Text1	(defaults)
List Box	List1	(defaults)
List Box	List2	(defaults)
Label	Label1	Caption="Tables in Database"
Label	Label2	Caption="Select Field(s) to Remove"
Label	Label3	Caption=""

5. Position the textbox in the vicinity of the Pickdb button, so it will display the path and filename of the database selected.

6. Position the Label1 label over the List1 list box, and position Label2 under List2.

7. Position Label3 over List2.

8. Add the following code to the form load event:

```
Sub Form_Load ()
   ' set gtempdir to an appropriate directory in the global .BAS module
   On Error Resume Next
   Kill gtempdir & "tempDB.mdb"
   Set gdb1 = CreateDatabase(gtempdir & "tempDB.mdb", DB_LANG_GENERAL)
   command1.Enabled = False
End Sub
```

9. Add the following code to the Command1_Click event:

```
Sub Command1_Click ()

   Dim dbsource As database
   Dim dbdest As database

   Set dbsource = gdb2  ' the database with table to be modified
   Set dbdest = gdb1    ' the temp base

   ' Indexes can be compound (defined to include several fields) and
   ' one or more of the fields int he compound index may be deleted.
   ' Therefore, to simplify the copy process, no indexes are copied
   ' to the new table. You must make note of the indexes on the old
   ' table and recreate them based on the new fields by using Data
   ' Manager, the VISDATA sample application, or code.

   Cls
   currentx = 0: currenty = 0
   ' Place the following Print statement on one, single line:
   Print DCopyStruct(dbsource, dbdest, (label3), "tempctable",
      gdelfield_arr(), gdelfields_count)
   Print DCopyData(dbsource, dbdest, (label3), "tempctable")

   ' Reset storage arrays and counters for next operation:
   ReDim gdelfield_arr(1 To 1)
   ReDim gfieldorder_arr(1 To 1)
   gdelfields_count = 0
   gfieldorder_count = 0

   ' Copy back from temp after deleting old table:
   Set dbsource = gdb1  ' the temp base
   Set dbdest = gdb2    ' the database with table to be modified

   ' NOTE: If the table was defined in Microsoft Access to be in a
   ' relationship (using primary/foreign keys) to other tables, you will
   ' not be able to Delete it without undoing those relationships first.
   ' In that case, use something like the following to create the new
   ' table, and place it all on one, single line:
```

```
      response = MsgBox("Delete old table from database?", 3,
         "Decision    Time!")

      Select Case response
      Case 6
         ' If Okay, delete the old table:
         gdb2.TableDefs.Delete label3
         ' Place the following Print statement on one, single line:
         Print DCopyStruct(dbsource, dbdest, "tempctable", (label3),
            gdelfield_arr(), gdelfields_count)
         Print DCopyData(dbsource, dbdest, "tempctable", (label3))
      Case 7
         ' Copy the new table with "new" appended to its name:
         ' Place the following Print statement on one, single line:
         Print DCopyStruct(dbsource, dbdest, "tempctable",
            (label3) & "new", gdelfield_arr(), gdelfields_count)
         Print DCopyData(dbsource, dbdest, "tempctable", (label3) & "new")
      Case 2
         ' Place the following MsgBox statement on one, single line:
         MsgBox "Cancelling copy of the new table back to the database.",
            0, "Decision Made"
      End Select

      Set dbsource = Nothing
      Set dbdest = Nothing

      gdb2.Close

      command1.Enabled = False
      list1.Clear
      list2.Clear
   End Sub
```

10. Add the following code to the Pickdb_Click event:

```
   Sub Pickdb_Click ()
      ' Reset global storage arrays and counters for next operation:
      ReDim gdelfield_arr(1 To 1)
      ReDim gfieldorder_arr(1 To 1)
      gdelfields_count = 0
      gfieldorder_count = 0

      ' Enter the following two lines as one, single line:
      cmdialog1.Filter = "Access (*.MDB)|*.mdb|Btrieve (*.DDF)|*.ddf|dBase
         (*.DBF)|*.dbf|FoxPro (*.DBF)|*.dbf|Paradox (*.DB)|*.db"
      cmdialog1.Action = 1
      text1 = cmdialog1.Filename ' Display the choice
      prompt$ = "Type the database connect string. For Access, press ENTER"
      title$ = "Connect string for OpenDatabase"
      connect$ = InputBox$(prompt$, title$, "Access")

      Select Case connect$
      Case ""
         Exit Sub

      Case "Btrieve"
         dbname$ = text1
```

```
      Case "Access"
         dbname$ = text1
         connect$ = ""

      Case Else
         dbname$ = StripFileName((text1))
         Debug.Print "else!"
      End Select

      ' Open the database with Exclusive set to True:
      Set gdb2 = OpenDatabase(dbname$, True, False, connect$)
      Set gtabledefs = gdb2.TableDefs
      ' List the tables in list1
      For i = 0 To gdb2.TableDefs.Count - 1
         If (gdb2.TableDefs(i).Attributes And DB_SYSTEMOBJECT) = 0 Then
            list1.AddItem gdb2.TableDefs(i)
         End If
      Next i

      command1.Enabled = True

   End Sub
```

11. Add the following code to the Form_QueryUnload event:

```
   Sub Form_QueryUnload (Cancel As Integer, UnloadMode As Integer)
      Debug.Print "Query unload"
      gdb1.Close
      ' Make sure the original database is explicitly closed:
      On Error Resume Next
   gdb2.Close

   Kill gtempdir & "tempDB.mdb"

   End Sub
```

12. Add the following code to the List1_DblClick event:

```
   Sub List1_DblClick ()

      list2.Clear

      ' Place the following two lines on one, single line:
      For i = 0 To
         gdb2.TableDefs(list1.List(list1.ListIndex)).Fields.Count - 1

         ' Place the following two lines on one, single line:
         list2.AddItem
            gdb2.TableDefs(list1.List(list1.ListIndex)).Fields(i).Name

         ' Display the table name of the table that has its fields
         ' displayed in List2:
         label3 = gdb2.TableDefs(list1.List(list1.ListIndex))
      Next i

   End Sub
```

13. Add the following code to the List2_DblClick event:

```
Sub list2_DblClick ()

   ' Increment the global counter of the fields to be deleted:
   gdelfields_count = gdelfields_count + 1

   ' Increase the size of the global array holding the name of the field
   ' to be deleted:
   ReDim Preserve gdelfield_arr(1 To gdelfields_count) As String

   ' Store the field name to be deleted:
   gdelfield_arr(gdelfields_count) = list2.List(list2.ListIndex)

   ' Remove it from the list:
   list2.RemoveItem list2.ListIndex

End Sub
```

14. Add the following code to the code module's General Declarations and merge it with the DATACONS.TXT file. Give the code module's code window the focus, choose Load Text from the File menu. Then browse for DATACONS.TXT at the root of the Visual Basic directory, and choose Merge.

```
Global gdb1 As Database
Global gdb2 As Database
Global gtable1 As table
Global gtable2 As table
Global gtabledefs As TableDefs
Global gdelfield_arr() As String
Global gdelfields_count As Integer

Global gfieldorder_arr() As Integer
Global gfieldorder_count As Integer

' Set the following to an appropriate directory:
Global Const gtempdir = "C:\temp\"
```

15. Add the following code to the code module:

```
' Place the following Function statement on one, single line:
Function DCopyData (from_db As Database, to_db As Database, from_nm As
   String, to_nm As String) As Integer

On Error GoTo CopyErr
Dim ds1 As Dynaset, ds2 As Dynaset
Dim i As Integer, skip  As Integer
Set ds1 = from_db.CreateDynaset(from_nm)
Set ds2 = to_db.CreateDynaset(to_nm)
While ds1.EOF = False
   skip = False
   ds2.AddNew
   For i = 0 To ds1.Fields.Count - 1

      For n = 1 To gfieldorder_count
```

```
            If gfieldorder_arr(n) = i Then
            skip = True
            Exit For
            End If
         Next n

         If Not skip Then ds2(i) = ds1(i)
      Next
      ds2.Update
      ds1.MoveNext
   Wend

   DCopyData = True
   GoTo CopyEnd

   CopyErr:
   ShowError
   CopyData = False
   Resume CopyEnd

   CopyEnd:

   End Function
```

16. Add the following code to the code module:

```
   ' Place the following Function statement on one, single line:
   Function DCopyStruct (from_db As Database, to_db As Database,
      from_nm As String, to_nm As String, delarray() As String,
      delfields As Integer) As Integer

   On Error GoTo CSErr
   Dim i As Integer, skip As Integer
   Dim tbl As New Tabledef    'table object
   Dim fld As Field           'field object
   Dim ind As Index           'index object

   ' Search to see if the table exists:
   namesearch:
   For i = 0 To to_db.TableDefs.Count - 1
      If UCase(to_db.TableDefs(i).Name) = UCase(to_nm) Then
         ' Place the following two lines on one, single line:
         If MsgBox(to_nm+" already exists, delete it?",
            4," DCopyStruct ")=YES Then

            to_db.TableDefs.Delete to_db.TableDefs(to_nm)
         Else
            to_nm = InputBox("Enter New Table Name:")
            If to_nm = "" Then
               Exit Function
            Else
               GoTo namesearch
            End If
         End If
         Exit For
      End If
   Next
```

```
  ' Strip off owner if needed
  If InStr(to_nm, ".") <> 0 Then
     to_nm = Mid(to_nm, InStr(to_nm, ".") + 1, Len(to_nm))
  End If
  tbl.Name = to_nm

  'create the fields
  For i = 0 To from_db.TableDefs(from_nm).Fields.Count - 1
     Set fld = New Field
     skip = False

     For n = 1 To delfields

        If from_db.TableDefs(from_nm).Fields(i).Name = delarray(n) Then
           ' Track the field ordinal position for the DCopyData call:
           gfieldorder_count = gfieldorder_count + 1
           ReDim Preserve gfieldorder_arr(1 To gfieldorder_count)
           gfieldorder_arr(gfieldorder_count) = i - 1
           skip = True
           Exit For
        End If
     Next n
     If Not skip Then
        fld.Name = from_db.TableDefs(from_nm).Fields(i).Name
        fld.Type = from_db.TableDefs(from_nm).Fields(i).Type
        fld.Size = from_db.TableDefs(from_nm).Fields(i).Size
        fld.Attributes = from_db.TableDefs(from_nm).Fields(i).Attributes
        tbl.Fields.Append fld
     End If
  Next

  ' Append the new table:
  to_db.TableDefs.Append tbl

  DCopyStruct = True
  GoTo CSEnd

  CSErr:
  ShowError
  DCopyStruct = False
  Resume CSEnd

  CSEnd:

  End Function
```

17. Add the following code to the code module:

```
  Sub ShowError ()
     Dim s As String
     Dim crlf As String
     crlf = Chr(13) + Chr(10)
     s = "The following Error occurred:" + crlf + crlf
     ' Add the error string:
     s = s + Error$ + crlf
     ' Add the error number:
```

```
      s = s + "Number: " + CStr(Err)
      ' Beep and show the error:
      Beep
      MsgBox (s)
   End Sub
```

18. Add the following code to the code module:

```
   Function StripFileName (fname As String) As String
      On Error Resume Next
      Dim i As Integer
      For i = Len(fname) To 1 Step -1
         If Mid(fname, i, 1) = "\" Then
           Exit For
         End If
      Next
      StripFileName = Mid(fname, 1, i - 1)
   End Function
```

19. Save the project.

Step-by-Step Instructions for Using the Program
---

1. Press the F5 key and click the Which Database? button. Then browse for the database to modify.

2. Click OK to the dialog. Then type in the correct connect string in the Input box that follows. Press Cancel on the Input box if you don't want to open the database.

3. Double-click the table names displayed in List1 to get the field names displayed in List2.

4. Double-clicking on the fields will remove them from the list and build a list of fields to be deleted. This will not actually affect the table's fields.

5. When you have selected all the fields to be deleted, Click the button labeled "Copy table minus fields." This will cause a new table to be created minus the fields in a temporary database.

6. When prompted to delete the old table, you can choose to delete, not delete, or cancel.

7. If you choose not to delete the old table, a new table will be created in the original database with "new" appended to the end.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

Comparison of Seek Versus Find Methods, for VB Data Access

Article ID: Q108149

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article compares the Seek method to the find methods (FindFirst, FindLast, FindNext, and FindPrevious) for data access in Visual Basic.

MORE INFORMATION

The Seek and find methods differ in performance and in the type of recordsets to which they apply:

- The find methods (FindFirst, FindLast, FindNext, and FindPrevious) apply to Dynasets and Snapshots but not to Table objects. Conversely, the Seek method is available only on the Table object.

- The Seek method is significantly faster than the find methods. It is also more flexible because you can change the Index property of the Table object to change the order of the Seek. For intensive searches, you may want to create a Table object so that you can use the Seek method along with the find methods on the open Dynasets.

FindFirst, FindLast, FindNext, FindPrevious Methods

The FindFirst, FindLast, FindNext, and FindPrevious methods locate the first, last, next, or previous record, respectively, that satisfies the specified criteria and makes that record the current record. These methods are referred to as the find methods and have the following syntax:

```
<recordset>.FindFirst <criteria>
```

where <recordset> and <criteria> are defined as follows:

<recordset> is the Recordset property of a data control or an object variable identifying a Dynaset or Snapshot.

<criteria> is a string expression specifying the records that you want. The string is the WHERE clause in an SQL string without the word WHERE.

If the recordset contains more than one record that matches the criteria, FindFirst locates the first occurrence, FindNext locates the next occurrence, and so on. You can follow a find method with a move method, such as MoveNext, which moves to the next record regardless of whether it matches any criteria. If no matching records are found, the NoMatch

property is True and the current record remains the same as it was before the find method was used.

NOTE: With a data control, if an Edit or AddNew operation is pending when you use one of the find or move methods, the Update method is automatically invoked if not intercepted during the Validate event.

CAUTION: In the Professional Edition of Visual Basic, if you are not using a data control and use one of the find or move methods while an Edit or AddNew method is pending, any existing changes will be lost and no error will occur. An Edit or AddNew will be pending until an Update occurs. For more information, see the Update method in the Help menu.

Example Code for FindFirst Method

The following example creates a Dynaset, and then uses FindFirst to locate the first record satisfying the title condition:

```
Sub Form_Load ()
   Dim MyCriteria As String, MyDB As Database, MySet As Dynaset
   MyCriteria = "State = 'NY'"    ' Define search criteria.
   Set MyDB = OpenDatabase("BIBLIO.MDB")
   ' Create a Dynaset based on the Publishers table:
   Set MySet = MyDB.CreateDynaset("Publishers")

   ' Find first matching record:
   MySet.FindFirst MyCriteria
   If Not MySet.NoMatch Then
      MsgBox "match was found"
   Else
      MsgBox "match was not found"
   End If  ' For a data control, you can use Data1.Recordset.NoMatch

   ' Find next matching record:
   MySet.FindNext MyCriteria
   If Not MySet.NoMatch Then
      MsgBox "match was found"
   Else
      MsgBox "match was not found"
   End If

End Sub
```

Seek Method

The Seek method locates a record that meets the specified criteria for the current index in an indexed table and makes it the current record. The Seek method has the following syntax:

```
<table>.Seek <comparison>, <key1>, <key2>, ...
```

where the arguments are defined as follows:

<comparison>	is one of the following string expressions: <, <=, =, >=, >, or <>

<key1>, <key2>, ...	one value for each field in the table's current index.

To use the Seek method, you must first use the OpenTable method to create an object variable for the table.

Seek searches through the specified Table using the current index and locates the first record satisfying the criteria specified by comparison and the key values (key1, key2...) and makes it the current record.

You must set the current index with the Index property before you use Seek. If the index identifies a non-unique-key field, Seek locates the first record satisfying the criteria.

When the comparison is =, >=, >, or <>, Seek starts at the beginning of the index and searches forward. When the comparison is <= or <, Seek starts at the end of the index and searches backward.

If <table> doesn't refer to an open table, or if there is no current index, an error occurs.

Always inspect the value of the NoMatch property of the recordset to determine whether each Seek method has succeeded. If Seek fails, NoMatch is True and the current record is unchanged.

Example Code for Seek Method

The following example uses Seek to locate the first record in the Publishers table where the PubID field is 3, using the existing primary key index:

```
Sub Form_Load ()
   Dim MyDB As database, MyTable As table
   Set MyDB = OpenDatabase("BIBLIO.MDB")      ' Open a database.
   Set MyTable = MyDB.OpenTable("Publishers") ' Open a table.
   MyTable.Index = "PrimaryKey"               ' Define current index.
   MyTable.Seek "=", 3                        ' Seek record.
   If MyTable.NoMatch Then
      MsgBox "match was not found"
   Else
      MsgBox "match was found"
   End If
End Sub
```

REFERENCES
==========

For more information, please read the Visual Basic online Help for the Seek, FindFirst, FindLast, FindNext, and FindPrevious Methods.

You can study the database design of a database file such as BIBLIO.MDB by opening it with Microsoft Access, or with the Data Manager or VISDATA provided with Visual Basic.

You can run the Data Manager program from the Window menu in Visual Basic,

or from the Windows File Manager run DATAMGR.EXE in the Visual Basic directory.

The VISDATA.MAK file installed in the VB3\SAMPLES\VISDATA directory loads extensive examples of data access. The VISDATA sample program uses every data access function in Visual Basic. You can refer to the VISDATA source code for examples of how to use each data access function.

Additional reference words: 3.00 MoveFirst MoveLast MoveNext MovePrevious
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Count Rows Affected Before Query in VB Prof ver 3.0

Article ID: Q108150

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article shows by example how to count the number of rows affected by a query before executing the query. You can execute a Select query using the same Where clause that your action query will use. Then you can examine the return value. This is an excellent practice before using the SQL Delete or Update methods.

MORE INFORMATION

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Double-click the form to open its code window. Add the following code to the Form Load event:

```
Sub Form_Load ()
   Dim db As database
   Dim ds As dynaset
   Set db = OpenDatabase("c:\vb3\biblio.mdb")
   ' Place the following Set statement on one, single line:
   Set ds = db.CreateDynaset("SELECT COUNT(*) as alias
      FROM Authors where AU_ID > 10")
   Debug.Print ds(0)
End Sub
```

3. Start the program or press the F5 key. The Debug window will display a count of 36. To end the program, close the form.

You can check the contents of the BIBLIO.MDB file by opening it with Microsoft Access or with the Data Manager provided with Visual Basic. You can run the Data Manager program from the Window menu in Visual Basic, or you can select DATAMGR.EXE in the Visual Basic directory and run it from the Windows File Manager.

REFERENCES

The VISDATA.MAK file installed in the VB3\SAMPLES\VISDATA directory loads extensive examples of data access. The VISDATA sample program uses every data access function in Visual Basic. You can refer to the VISDATA source code for examples of how to use each data access function.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

DOC: Revised Index Property (Data Access)

Article ID: Q108235

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

Page 279 of the "Language Reference" discusses the Index Property (Data Access). It has some incorrect and misleading information. This article is a revised and corrected version of the entire Index Property (Data Access) section. Please replace page 279 with this article.

MORE INFORMATION

Index Property (Data Access)

Applies To

Table object

Description

With data access, the Index property determines which existing index is the current index used to sort records in a Table and in recordsets created from that Table. The default is blank. The Index property is not available at design time and is available read/write at run time.

Syntax

```
table.Index [ = indexname ]
```

Remarks

The order of the data in a table is determined by the order in which the data is added to the table. To alter the order of records fetched from the table when using a Table object, set the Index property to the name of an index in the Indexes collection of the Table's TableDef object. For example, to set the index to be used on a Seek against the Titles table:

```
Dim Tb as Table, Db as Database
Set Db = OpenDatabase("Biblio.MDB")
Set Tb = Db.OpenTable("Titles")
Tb.Index = "PubID"
Tb.Seek "=", 3
```

The specified index must already be defined. If you set the Index property to an index that doesn't exist, or if the index isn't set when you use the Seek method, an error occurs.

In the Professional Edition, you can create a new Index in a Table by creating a new Index object, setting its properties, and then appending it to the Indexes collection of the Table's TableDef.

The records in a Table can be ordered only according to the indexes defined for it. To sort the Table records in some other order, create a new Index for the table and append it to the Table's Index Collection, or create a Dynaset or Snapshot that has a different sort order. To specify the sort order for Dynasets and Snapshots, use the Sort property after the Dynaset or Snapshot has been created. You can also set the order of a Dynaset or Snapshot by including an Order By clause in a SQL statement used to define the Dynaset or Snapshot.

The Index property of a control array element is not the same as the Index property of a data access object.

Data Type
String

Additional reference words: 3.00 docerr
KBCategory: APrg
KBSubCategory: APrgDataOther

LONG: Overview of Data Access in Visual Basic Version 3.0

Article ID: Q108379

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY
=======

This article provides an overview of data access in Visual Basic version 3.0. It contains following sections:

- Data Access In Visual Basic Version 3.0 Versus 2.0
- Relational Database Theory
- Visual Basic Can Use SQL
- Visual Basic Uses the Microsoft Access Database Engine
- Data Access Object Hierarchy

MORE INFORMATION
================

DATA ACCESS IN VISUAL BASIC VERSION 3.0 VERSUS 2.0
==

Historically, database management systems (DBMS) represent some of the most mission-critical and most complex programming challenges in the field of computer science.

Visual Basic version 3.0 ships with full-featured, multi-faceted data access capability. It has a full range of connectivity, flexibility, and support for Open Database Connectivity (ODBC) on the Microsoft Windows platform.

Database features were introduced to Visual Basic in version 2.0. The upgrade to Visual Basic version 3.0 offers much easier and much more powerful database management.

Microsoft added new data methods, objects, and properties in Visual Basic version 3.0. Version 2.0 was not able to create an updatable Dynaset on an entire table, and could not navigate the Dynaset with methods other than MoveNext. Visual Basic version 3.0 can do both. In version 3.0, Dynasets created using SQL statements are updatable, and there are three new move methods (MovePrevious, MoveLast and MoveFirst).

Programmers are much more likely to face network architecture issues in Visual Basic version 3.0 than with version 2.0. This is because of the fuller implementation of ODBC and the inclusion of the Microsoft Access engine and its ISAM database connectivity.

Some programmers use Visual Basic version 3.0 as the center of mission-critical database management systems connected to proprietary databases. For example, a Visual Basic version 3.0 application could be written as a front end for an enterprise-wide information system containing data in

formats for SQL Server, Oracle, dBASE and FoxPro. The Visual Basic application could integrate data from all these sources and from multiple servers. The performance of a system such as this depends significantly on the network behavior and back-end systems outside of Visual Basic.

RELATIONAL DATABASE THEORY

Database Models

The field of computer science has evolved four models for databases, in the following order of progressing theory and technology:

1. Flat File Database
2. Hierarchical Database
3. Network Database
4. Relational Database

The relational model is a major step forward for database programmers. With the relational model, none of the physical and logical pointers between records are exposed to the programmer. The relational database handles all low-level structure. A relational database management system (RDBMS) makes database programming much easier and more flexible than earlier database systems.

Relational Database Model

Visual Basic uses a relational database model. The relational database model offers the following benefits:

- Organizes data in a collection of tables making the design easy to understand.

- Provides a relationally complete language for data definition, retrieval, and update. It is non-procedural and criteria-based.

- Provides data integrity rules that define consistent states of the database to improve data reliability.

A relational database management system (RDBMS) is software that allows you to represent your data according to the relational model. Both the programmer and the user think in terms of groups of tables comprising the database, with tables composed of rows and columns. The data in those rows and columns relate to each other according to a consistent theory and practice.

Relational databases support a standard language called Structured Query Language (SQL). SQL has evolved into a comprehensive language for controlling and interacting with a database management system (DBMS). SQL is now a standard approved by the American National Standards Institute (ANSI).

SQL provides three important functions:

1. Data Definition -- to define the tables that hold the data.

2. Data Manipulation -- to insert, update, or delete information stored in tables.

3. Data Control -- to prevent access to private data in the database.

Dr. Codd, considered the father of relational database theory, has defined twelve conditions that a database must obey to be considered fully relational, and he defined three criteria for a minimally relational DBMS:

1. Information is represented as values in tables.

2. Internal data structures and pointers are not visible to the user.

3. The DBMS language supports at least the following syntax: SELECTION, PROJECTION, and JOIN.

These three criteria are necessary and sufficient for a minimally relational definition because of the following:

1. The relational operations only work on tables, therefore all the data must be in tables.

2. If internal data structures and pointers were visible to the user, the data would not appear to be in a table. It would appear to be in some DBMS dependent structure.

3. Without SELECTION, the DBMS could not perform operations on subsets of the table. It would be forced to operate on the entire table. In effect, it would be just a file handler.

   Without PROJECTION, the DBMS could only perform operations on an entire row. Therefore, it would be just a unit record handler.

   Without JOIN, data could not be correlated across tables. It would not be a related database, just a collection of unrelated tables.

The following additional terms are associated with relational database theory:

   primary key
   foreign key
   null values
   duplicate values
   updatable values
   derivative data
   constraints
   referential integrity

For more information on relational database theory, refer to any of the books listed in the BIBLIO.MDB database in Visual Basic version 3.0.

VISUAL BASIC CAN USE SQL

The Microsoft Access engine included with Visual Basic version 3.0 uses a dialect of Structured Query Language (SQL). This dialect is based on the

ANSI 1986 standard and differs from that of Microsoft's SQL Server in certain syntax. For that syntactical reference, please refer to Appendix B of "Microsoft Visual Basic 3.0: Professional Features Book 2: Data Access Guide."

The SQL parsing capability of the Microsoft Access engine adds considerable power and flexibility to Visual Basic. SQL gives database programmers and users more leverage and a standardized approach to querying databases.

VISUAL BASIC USES THE MICROSOFT ACCESS DATABASE ENGINE

Visual Basic version 3.0 uses the database engine from Microsoft Access version 1.1. This engine provides data access to many database formats, including Microsoft Access, FoxPro, dBASE, Paradox, Btrieve, SQL Server, Oracle, and other formats that support the ODBC specification.

The Microsoft Access database engine in Visual Basic version 3.0 provides the following:

- Provides a query engine
- Supports multi-user applications
- Allows for transaction processing
- Offers choice of optimistic or pessimistic locking
- Supports rich data types such as sound, video, OLE objects, and pictures
- Parses SQL
- Performs distributed joins, such as joining a FoxPro table with an Oracle table
- Performs updatable queries and query optimization
- Supports international collating orders.

In Visual Basic, you can harness the database engine in two different ways:

1. By writing code using the data definition language (DDL) and data manipulation language (DML). This involves dimensioning and using database object variables.

2. By using the data control and bound controls. Bound controls include the text box, label, check box, image control, and picture control in the Standard Edition of Visual Basic, plus the masked edit, 3DPanel, and 3DCheckBox in the Professional Edition. You can enable data access without code by setting design-time properties or by setting properties in run-time code.

Programmers can handle database objects easily in Visual Basic code. The object layer provides a uniform system catalog, independent of whether the database is a Microsoft Access database or an external database such as an ODBC or ISAM database. You can gain access to the hierarchical structure of the system catalog by using the TableDef objects in the TableDefs collection of each database.

Component Model of Data Access in Visual Basic

The architecture of the database components is the same for Microsoft Access version 1.1 and Visual Basic version 3.0.

You can access three types of databases from Visual Basic:

1. Microsoft Access databases, which are native to Visual Basic's database engine. Visual Basic can use Microsoft Access databases directly.

2. Indexed sequential access method (ISAM) databases, such as dBASE, Paradox, and Btrieve databases. Visual Basic reaches these databases through user-installable drivers that link Visual Basic to the specific databases.

3. Open Database Connectivity (ODBC) accessible databases. These include client-server database management systems (DBMS), such as Microsoft SQL Server and ORACLE. Visual Basic reaches these databases through the appropriate ODBC drivers.

Various gateways are also available to connect to databases on mainframe computers. This is usually implemented through an ODBC driver.

DATA ACCESS OBJECT HIERARCHY

At the top of the database object hierarchy is the Database object, not to be confused with the Database property of the data control. One of the properties of the Database object is the TableDefs collection, which is also an object. The TableDefs collection represents all the individual TableDef objects associated with the Table objects. Please read further about objects in the NOTE sections in the sample program below.

Step-by-Step Example Shows How to Use Database Objects

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add four list boxes to the form.

3. Add the following code to the Form Load event:

```
Sub Form_Load ()
   form1.Show
   Dim MyDb As Database
   Dim MySingleTableDef As TableDef
   Dim AllTableDefs As TableDefs
   Set MyDb = OpenDatabase("BIBLIO.MDB", True, False)
   Set AllTableDefs = MyDb.TableDefs
   For i = 0 To AllTableDefs.Count - 1
      ' Only Count property is applicable to top-level Tabledefs object
      list1.AddItem AllTableDefs(i).Name ' Get each table name in MyDb
      list2.AddItem AllTableDefs(i).DateCreated
      list3.AddItem AllTableDefs(i).Updatable
      list4.AddItem AllTableDefs(i).Attributes
      ' Value property is only valid if part of a recordset:
      ' list5.AddItem AllTableDefs(i).Value
   Next i
End Sub
```

4. Start the program or press the F5 key. Examine the contents of the list boxes. Close the form to end the program.

NOTE: Using the values of the Name property of the TableDefs object (the top-level collection), you can examine the properties of the TableDef object of the individual tables as shown below. You can walk through the Fields collection of the TableDef object of the individual tables using the Count property. The Count property is the only property of the collection objects. The collection objects are Fields, TableDefs, and Indexes.

5. Add four more list boxes to the form, numbered 5 through 8.

6. Append the following code to the existing code in the form load procedure:

```
' Get information on the first table listed on list box 1:
Set MySingleTableDef = MyDb(list1.List(0))
For i = 0 To MySingleTableDef.Fields.Count - 1
   list5.AddItem MySingleTableDef.Fields(i).Name
   ' or you can use: list5.AddItem MySingleTableDef(i).Name
   ' because Fields are the default collection.
   list6.AddItem MySingleTableDef.Fields(i).Size
   list7.AddItem MySingleTableDef.Fields(i).Type
   If i <= MySingleTableDef.Indexes.Count - 1 Then
      list8.AddItem MySingleTableDef.Indexes(i).Name
   End If

   ' The Value property is only valid if part of a recordset:
   ' MySingleTableDef.Fields(i).Value
   ' The other 5 properties are valid for a field of a TableDef object:
   ' MySingleTableDef.Fields(i).OrdinalPosition
   ' MySingleTableDef.Fields(i).CollatingOrder
   ' MySingleTableDef.Fields(i).Attributes
   ' MySingleTableDef.Fields(i).SourceField
   ' MySingleTableDef.Fields(i).SourceTable
Next i
```

7. Start the program or press the F5 key. Examine the contents of the list boxes. Close the form to end the program.

NOTE: The Field and Index objects are contained in the Field and Index collections of the Table and TableDefs objects. The following code shows this.

8. Append the following code to the existing code in the form load procedure:

```
msgbox "Next, show indexes for the " & MySingleTableDef.Name & " Table"
list5.Clear
list6.Clear
list7.Clear
list8.Clear
For i = 0 To MySingleTableDef.Indexes.Count - 1
   list5.AddItem MySingleTableDef.Indexes(i).Name
   list6.AddItem MySingleTableDef.Indexes(i).Primary
   list7.AddItem MySingleTableDef.Indexes(i).Unique
   list8.AddItem MySingleTableDef.Indexes(i).Fields
   ' property of Index object: indicates simple/composite keys
```

```
      ' Determines which TableDef fields are key fields in an index.
      ' Read-only when the Index is a member of a collection.
      ' Read/write only in the Professional Edition
      ' with a new object not yet appended to an Indexes collection.
      ' An Index object has field(s) representing key values
      ' for each record. Field names are separated by semicolons.
   Next i
```

REFERENCES
==========

- "Microsoft Visual Basic 3.0: Professional Features Book 2: Data Access Guide."

- The VISDATA.MAK file installed in the VB3\SAMPLES\VISDATA directory loads extensive examples of data access. The VISDATA sample program uses every data access function in Visual Basic. You can refer to the VISDATA source code for examples of how to use each data access function.

Additional reference words: 3.00 1.10 2.00
KBCategory: APrg
KBSubcategory: APrgDataOther

PRB: Find Methods Don't Use Indexes to Speed Up VB Data Access

Article ID: Q108380

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

The find methods take longer to find farther records than nearer records. For example, the FindFirst method takes longer to find records that are farther into the Dynaset or Snapshot. Using FindFirst, FindNext, or FindLast on indexed fields isn't any faster than on non-indexed fields.

CAUSE

Creating a Dynaset or Snapshot is relatively fast because of indexes. Indexes are used when you create the Dynaset or Snapshot, such as when you specify a selection criteria or an ORDER BY clause in the CreateDynaset method. However, after the Dynaset or Snapshot is created, it does not use the indexes for searches.

The find and move methods always search sequentially within the current Dynaset or Snapshot. The find and move methods don't use indexes. For example, the FindLast and MoveLast methods require reading the entire recordset before processing continues.

The find methods are: FindFirst, FindLast, FindNext, and FindPrevious. The move methods are: MoveFirst, MoveLast, MoveNext, and MovePrevious.

WORKAROUND

Repositioning the record pointer to a location far into a large Dynaset or Snapshot may be faster if you recreate the Dynaset or Snapshot instead of searching the existing one.

For example, you could change the RecordSource property of a data control at run time as follows:

```
txtSearchString = "target string to find"
' Place the following two lines on one, single line:
data1.Recordset = "select * from mytable where myfield = '" &
   txtSearchString & "' order by myotherfield"
data1.Refresh   ' Update the data control with the new Recordset
```

You can also open a Table object in addition to the Dynaset or Snapshot. You can use the Seek method on the Table to find a specific record keyed by an index.

STATUS

This behavior is by design.

REFERENCES
==========

- "Microsoft Visual Basic 3.0: Professional Features Book 2: Data Access Guide," Chapter 3. See the section "Positioning the Current Record in a Recordset."

- Help menu in Visual Basic.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Attach an External Database Table to a VB 3.0 Database

Article ID: Q108423

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY
=======

An attached table is a table from an external database linked at run time to a Microsoft Access database. You can gain access to the data in the attached table by using a data control, a Dynaset, or a Snapshot. The database format native to Visual Basic is the Microsoft Access format.

Using the data control, you can open a Dynaset on an external table. Specify the external database type in the Connect property. Specify an appropriate directory or file name in the DatabaseName property of the data control.

Using object variables, you can attach a table from any supported external database to a Microsoft Access database as shown in the examples below.

NOTE: You can use queries and the move and find methods on an attached table. An attached table cannot be opened with the OpenTable method. Therefore you cannot use the Seek method on an attached table.

MORE INFORMATION
================

The following steps describe how to attach a table to an existing Visual Basic database or a Microsoft Access database:

1. Create a variable for the Database object you are going to modify:

   ```
   Dim Db As Database
   ```

2. Use the OpenDatabase function to open the existing Visual Basic or Microsoft Access database:

   ```
   Set Db = OpenDatabase("BIBLIO.MDB")
   ```

3. Dimension a new TableDef object for the table from the external database.

4. Set the following properties of the TableDef object to prepare for attaching the external table:

   a. Name property: A new name for the table to be used in Visual Basic.

   b. SourceTableName property: The original name of the external table or file name.

   c. Connect property: The database type and other parameters. If a password is required but not provided in the Connect property, a Login dialog box appears each time the table is accessed.

5. Repeat steps 3 and 4 for each external table.

6. Use the Append method to add the TableDef object(s) to the TableDefs collection of the Microsoft Access database. This step actually creates the object links in the Microsoft Access database file.

Example One

Both databases shown below are Microsoft Access databases. But the table to be attached to the Microsoft Access database could be from any of the other database formats that Visual Basic version 3.0 supports.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Double-click the form to open its code window. Add the following code to the Form Load event:

```
Sub Form_Load ()
   Dim db As database
   Dim td As New Tabledef
   Dim ds As dynaset
   Set db = OpenDatabase("BIBLIO.MDB")
   td.Name = "MyNewCustomersTable"   ' New Table name for use in VB.
   td.SourceTableName = "Customers"  ' Table name in source database.
   td.Connect = ";DATABASE=c:\access\nwind.mdb;"   ' Source database.
   db.TableDefs.Append td   ' Append Customers Table to BIBLIO.MDB.
   Set ds = db.CreateDynaset("MyNewCustomersTable")  ' Create dynaset.
   Debug.Print ds.Fields(0)     ' Proves the Table is attached.
   Debug.Print ds.Fields(1)     ' Proves the Table is attached.
   Debug.Print ds.Fields(2)     ' Proves the Table is attached.
   ' The following statement deletes the appended Table, if desired:
   db.TableDefs.Delete "MyNewCustomersTable"
End Sub
```

3. Start the program or press the F5 key. To end program, close the form.

Example Two
----

1. Start a new project in Visual Basic. Form1 is created by default.

2. Double-click the form to open its code window. Add the following code to the Form Load event:

```
Sub Form_Load ()

   Dim db1 As database, db2 As database
   Dim td As New Tabledef
   Dim tb As Table
   Dim ds As dynaset
   Dim f1 As New field, f2 As New field
```

```
      Const DB_LANG_GENERAL = ";LANGID=0x0809;CP=1252;COUNTRY=0"
      Const DB_VERSION10 = 1
      Const file1 = "test1.mdb"  'contains Table food1"
      Const file2 = "test2.mdb"  'contains Table food2

      Set db1 = OpenDatabase(file1)
      Set db2 = OpenDatabase(file2)
      ' db2.TableDefs.Delete "new_food1" ' Deletes Table if desired.

      td.Name = "new_food1"
      td.SourceTableName = "food1"
      td.Connect = ";database=" & file1 & ";"

      ' NOTE: For an ODBC database, the connect string would be similar to:
      ' td.Connect = "ODBC;UID=sa;PWD=;DSN=texas;DATABASE=pubs;"
      ' td.Attributes = DB_ATTACHEDTABLE
      ' or, if password protected: td.Attributes = &H20000
      ' or, if exclusive: td.Attributes = tbl.Attributes + &H10000

      db2.TableDefs.Append td   ' Attaches the external Table.

      ' NOTE: The OpenTable method is illegal for attached Tables:
      '  Set tb = db2.OpenTable("new_food1")   ' Gives an error.

      Set ds = db2.CreateDynaset("new_food1")
      Print ds.Fields(0)     ' Proves the Table is attached.
      ds.Close
      db1.Close
      db2.Close
   End Sub
```

3. Modify the code to use your existing database and table names. Start the program or press the F5 key. To end the program, close the form.

REFERENCES
==========

- See the "attached tables" topic in the Help menu.

- See the EXTERNAL.TXT file provided with Visual Basic.

- The VISDATA.MAK file installed in the VB3\SAMPLES\VISDATA directory loads extensive examples of data access. The VISDATA sample program uses every data access function in Visual Basic. You can refer to the VISDATA source code for examples of how to use each data access function.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Request Exclusive Use of a Table in VB Prof 3.0

Article ID: Q108467

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY
=======

Below is an example of how to request exclusive use of a table and deny access to other users.

MORE INFORMATION
================

If you do not want any other users to access a table while you have it open, open the table exclusively. Set the options argument of the OpenTable method to a value equal to DB_DENYREAD plus DB_DENYWRITE to give your program exclusive access.

The values of all constants for data access, such as DB_DENYREAD and DB_DENYWRITE, are defined in the DATACONS.TXT text file installed in your Visual Basic directory.

Step-by-Step Example
----

1. Start a new project in Visual Basic. Form1 is created by default.

2. Double-click the form to open its code window. Add the following code to the Form Load event:

```
Sub Form_Load ()
   Dim i As Integer
   Dim db As Database
   Dim tb As table

   On Error GoTo ExclusiveErr
   Const DB_DENYWRITE = &H1   ' Constant defined in DATACONS.TXT file
   Const DB_DENYREAD = &H2    ' Constant defined in DATACONS.TXT file
   Set db = OpenDatabase("C:\VB3\BIBLIO.MDB")
   Set tb = db.OpenTable("authors", DB_DENYREAD + DB_DENYWRITE)
   ' To open shared access, use: Set tb = db.OpenTable("authors", 0)
   MsgBox "Database & table opened successfully, denying read & write."
   'PlaceDataInControls
   EndExclusiveOpen:
Exit Sub

ExclusiveErr:
Select Case Err
Case 3262
   MsgBox "Table is locked. You cannot open it exclusively. Try shared."
   'optExclusive.Value = False
```

```
      Resume EndExclusiveOpen
   Case 3261
      MsgBox "Table exclusively locked by another user -- cannot open."
      End
   Case Else
      MsgBox Err & " " & Error$
   End Select

   End Sub
```

3. From the File menu, choose Make EXE File. Name the executable file TEST.EXE.

4. Run one copy of TEST.EXE from the Windows File Manager or Program Manager. Leave the following message on the screen without choosing OK, in order to leave the database open:

   ```
   Database & table opened successfully, denying read & write.
   ```

5. Start the program in Visual Basic by pressing the F5 key. This second instance of the program displays the following message:

   ```
   Table is locked. You cannot open it exclusively. Try shared.
   ```

6. Close the form to end the program session in Visual Basic. Change DB_DENYREAD + DB_DENYWRITE to 0 in the OpenTable method as follows:

   ```
   Set tb = db.OpenTable("authors", 0)
   ```

   This opens the table with shared access, the default.

7. Start the program in Visual Basic by pressing the F5 key. This second instance of the program now displays the following message:

   ```
   Table exclusively locked by another user. You cannot open it.
   ```

8. You can end the first instance of the program, TEST.EXE, by clicking OK and closing the form.

REFERENCES

- Microsoft Visual Basic 3.0: Professional Features Book 2: Data Access Guide, page 58.

- DATACONS.TXT text file installed in your Visual Basic directory.

- The VISDATA.MAK file installed in the VB3\SAMPLES\VISDATA directory loads extensive examples of data access. The VISDATA sample program uses every data access function in Visual Basic. You can refer to the VISDATA source code for examples of how to use each data access function.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

Category Keywords for All Visual Basic KB Articles

Article ID: Q108753

The information in this article applies to:

- Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Each article in the Visual Basic for Windows collection contains at least one keyword (called a KBSubcategory keyword) that places the article in an appropriate category. This article lists all the KBSubcategory keywords.

MORE INFORMATION

Category & Subcategory Description	KBSubcategory Keyword
Setup / Installation (Setins)	Setins
Environment-specific Issues (Envt)	
VB Design Environment	EnvtDes
Run-Time Environment	EnvtRun
Programming (Prg)	
Visual Basic Forms and Controls	
Standard Controls / Forms	PrgCtrlsStd
Custom Controls	PrgCtrlsCus
Third-Party Controls	PrgCtrlsThird
Optimization	
Memory Management	PrgOptMemMgt
General Optimization Tips	PrgOptTips
General VB Programming	PrgOther
Advanced programming (APrg)	
Network	APrgNet
Windows Programming (APIs / DLLs)	
Printing	APrgPrint
Graphics	APrgGrap
Windowing	APrgWindow
INI Files	APrgINI
Other API / DLL Programming	APrgOther
Data Access	
ODBC	APrgDataODBC
IISAM	APrgDataIISAM
Access	APrgDataAcc
General Database Programming	APrgDataOther
3rd Party DLL's	APrgThirdDLL

Inter-Application Programmability (IAP)

OLE	IAPOLE
DDE	IAPDDE
3rd Party Interoperability	IAPThird

Tools (Tls)

Setup Toolkit / Wizard	TlsSetWiz
Control Development Kit (CDK)	TlsCDK
Help Compiler (HC)	TlsHC

References (Refs)

Documentation / Help File Fixes	RefsDoc
Product Information	RefsProd
Third-Party Information	RefsThird
PSS-Only Information	RefsPSS

Using Keywords to Query the KB

At Microsoft, we use the subcategory keywords to organize the articles for Help files and for the FastTips Catalog. You can use them to query the Microsoft Knowledge Base for Visual Basic articles that apply to that category or subcategory. For example, you can find all the general database programming articles by querying on the following words in the Microsoft Knowledge Base:

```
visual and basic and APrgDataOther
```

Use the asterisk (*) wildcard to find articles that fall into the general categories or into an intermediate subcategory. The first element in each keyword is the category. For example, to find all the articles that apply to Visual Basic Forms and Controls regardless of whether they are standard, custom, or third-party controls, use the following words to query the Microsoft Knowledge Base:

```
visual and basic and PrgCtrls*
```

To find all advanced programming articles, query on these words:

```
visual and basic and APrg*
```

Add KBSubcategory Keyword to Each Article

When contributing an article to the Visual Basic Knowledge Base, add the appropriate KBSubcategory keyword to the bottom of the article on the KBSubcategory line. Each article in the Visual Basic for Windows collection contains the following section at the bottom of the article:

```
Additional reference words:
KBCategory:
KBSubcategory: <keyword>
```

An article usually has only one subcategory keyword, but it may have more.

If you are interested in contributing, please obtain the guidelines by

querying on the following words in the Microsoft Knowledge Base:

   visual and basic and kbguide and kbartwrite

Additional reference words: 3.00 dskbguide subcatkey
KBCategory:
KBSubcategory: RefsPSS

Basic Cannot Get Description Shown in Access Table Design View

Article ID: Q109136

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0
- Microsoft Access versions 1.0 and 1.1

When you View a database in Table Design mode in Microsoft Access, a Description property is shown. You can describe a table and each of its fields in the Description property using up to 255 characters. That Description property is not visible when you choose the Datasheet view. The Description property is only visible in design mode in Microsoft Access.

The Description property cannot be read by Access Basic or Visual Basic at run time. The Description property is encrypted in one of the MSys* system tables within each database. Visual Basic cannot access the Description property, even if you use the ListFields or ListTables method. The Description property is designed for use only at database design time.

Additional reference words: 3.00 1.00 1.10
KBCategory: APrg
KBSubcategory: APrgDataOther

How to List the Fields in a Table & the Tables in a Database

Article ID: Q109219

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

Below are two examples showing how to list all the fields in a table and all the tables in a database.

- Example One shows how to list all the fields in a database table by using the Fields collection of a table's TableDef object. It also shows how to list the names of all tables in a database.

- Example Two shows how to list all the fields in a database table using the ListFields method. The ListFields method creates a Snapshot with one record for each field in a specified recordset.

The technique used in Example One is more efficient than Example Two.

MORE INFORMATION

Example One: How to List the Fields in a Table Using the Fields Collection

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add two list boxes to Form1.

3. Double-click the form to open the code window. Add the following code to the Form Load event:

```
Sub Form_Load ()

   Set MyDb = OpenDatabase("BIBLIO.MDB")
   ' Set AllTableDefs to definitions of all tables in the database:
   Set AllTableDefs = MyDb.TableDefs
   ' Display names of all tables in database:
   For j = 0 To AllTableDefs.Count - 1
      List1.AddItem AllTableDefs(j).Name
   Next

End Sub
```

4. Double-click the List1 list box and enter the following code in its Click event:

```
Sub List1_Click ()

   ' Delete any existing entries in List2 box:
```

```
        Do While list2.ListCount > 0
           list2.RemoveItem 0
        Loop

       ' Get the definition of the single table currently selected in List1:
       Set SingleTableDef = MyDb(List1.List(List1.ListIndex))
       ' Display the properties of each field in the table:
       For j = 0 To SingleTableDef.Fields.Count - 1
          list2.AddItem "Field item number " & Val(j) & ":"

          ' Display the name of the field in the table selected in List1:
          list2.AddItem SingleTableDef.Fields(j).Name
          ' or use the following since Fields are the default collection:
          ' List2.AddItem SingleTableDef(j).Name

          list2.AddItem SingleTableDef.Fields(j).Size  ' Size of field.
          list2.AddItem SingleTableDef.Fields(j).Type  ' Type of field.
          ' If field is an index, list the name of the index:
          If j <= SingleTableDef.Indexes.Count - 1 Then
             list2.AddItem "Index name: " & SingleTableDef.Indexes(j).Name
          End If

          ' The Value property is only valid if part of a recordset:
          ' list2.AddItem SingleTableDef.Fields(i).Value

          ' The other 5 properties are valid for a field of TableDef object:
          list2.AddItem SingleTableDef.Fields(j).OrdinalPosition
          list2.AddItem SingleTableDef.Fields(j).CollatingOrder
          list2.AddItem SingleTableDef.Fields(j).Attributes
          list2.AddItem SingleTableDef.Fields(j).SourceField
          list2.AddItem SingleTableDef.Fields(j).SourceTable
          list2.AddItem " "
       Next

    End Sub
```

5. From the File menu, choose New Module. Then enter the following code in the General Declarations section:

```
    Global MyDb As Database
    Global SingleTableDef As TableDef
    Global AllTableDefs As TableDefs
```

6. Start the program. Click any table name in the first list box. In the second list box, the program displays all the fields and field properties for that table. Close the form to end the program.
   NOTE: Some MSys* tables (such as MSysACEs) have no fields.

Example Two: How to List the Fields in a Table Using the ListFields Method
---

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a list box to Form1. Size the list box to fill all of Form1.

3. Double-click the form to open the code window. Add the following code to the Form Load event:

```
Sub Form_Load ()
Dim ListSet As Snapshot, MyDB As database, MyTable As table
Set MyDB = OpenDatabase("BIBLIO.MDB")
Set MyTable = MyDB.OpenTable("Publishers")  ' Open Table.
Set ListSet = MyTable.ListFields()  ' Put field info in ListSet.
MyTable.Close           ' Close Table.
Do While Not ListSet.EOF
   list1.AddItem "Name: " & ListSet("Name")
   list1.AddItem "type: " & ListSet("Type")
   list1.AddItem "size: " & ListSet("Size")
   list1.AddItem "Attributes: " & ListSet("Attributes")
   list1.AddItem "SourceTable: " & ListSet("SourceTable")
   list1.AddItem "SourceField: " & ListSet("SourceField")
   list1.AddItem " "
   ListSet.MoveNext
Loop
End Sub
```

The above program uses the BIBLIO.MDB database that ships with Visual Basic version 3.0

4. Start the program (or press the F5 key). Close the form to end the program.

The above program lists the following field structure for the Publishers table in the BIBLIO.MDB database:

```
Name: PubID
type: 4
size: 4
Attributes: 33
SourceTable: Publishers
SourceField: PubID

Name: Name
type: 10
size: 50
Attributes: 32
SourceTable: Publishers
SourceField: Name

Name: Company Name
type: 10
size: 255
Attributes: 32
SourceTable: Publishers
SourceField: Company Name

Name: Address
type: 10
size: 50
Attributes: 32
SourceTable: Publishers
SourceField: Address

Name: City
```

type: 10
size: 20
Attributes: 32
SourceTable: Publishers
SourceField: City

Name: State
type: 10
size: 10
Attributes: 32
SourceTable: Publishers
SourceField: State

Name: Zip
type: 10
size: 15
Attributes: 32
SourceTable: Publishers
SourceField: Zip

Name: Telephone
type: 10
size: 15
Attributes: 32
SourceTable: Publishers
SourceField: Telephone

Name: Fax
type: 10
size: 15
Attributes: 32
SourceTable: Publishers
SourceField: Fax

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Use SQL Outer Join to Find All Table B Records Not in A

Article ID: Q109563

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

The following SQL query retrieves all of the records from table B in which there is no matching record in table A:

```
Select B.* From A,B, B Left Join A On B.Key=A.Key Where A.Key Is Null;
```

This type of outer join query is the functional opposite of an inner join.

You can use this SQL query to create a Dynaset of records with the CreateDynaset statement.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Set VB Data Control to External ODBC Database Dynaset

Article ID: Q109800

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The following sample code fragment sets a data control (Data1) to a Dynaset based upon an external ODBC database:

```
   ' Set DatabaseName property to a valid Data Source Name (DSN) that is
   ' registered in the ODBC.INI file in your Windows directory:
   Data1.DatabaseName = "datasourcename"
   Data1.Connect = "odbc;uid=user;pwd=password;database=pubs"
   Data1.RecordSource = "tablename"
   ' Or you can set the Data1.RecordSource property to an SQL query:
   ' Data1.RecordSource = "select * from tablename where ..."
   Data1.Refresh   ' Must update the data control with new Dynaset.
```

NOTE: You cannot set a data control directly to a Dynaset variable, but you can set a Dynaset variable to the data control's recordset:

```
   Dim ds as Dynaset
   ...
   Set Data1.Recordset = ds   'This statement is not supported.
   Set ds = Data1.Recordset   'This statement is supported.
```

A data control requires additional information that is not available in a Dynaset object.

MORE INFORMATION

Before you can edit an external ODBC table, the table must contain a unique index. If you get the following error message, you might not have a unique index on the table:

```
   Can't perform operation; it is illegal.
```

You could also receive this error if the Data1.Recordset.Updatable flag is not set to True. Also, if you set the Data1.Options property to 64 (SQL_PASSTHROUGH), the data control will not be updatable.

REFERENCES

- "Microsoft Visual Basic for Windows Professional Features Book 2: Data Access Guide," pages 14-15 and Appendix C.

Additional reference words: 3.00

KBCategory: APrg
KBSubcategory: APrgDataODBC

How to Speed Up Data Access by Using BeginTrans & CommitTrans

Article ID: Q109830

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

You can speed up database operations by many times in a Microsoft Access database by using transactions. A transaction starts with a BeginTrans statement and ends with a CommitTrans or Rollback statement.

The sample program below is more than 17 times faster when using BeginTrans/CommitTrans. Performance may vary on different computers.

MORE INFORMATION

You can tune the performance of Visual Basic by using transactions for operations that update data. A transaction is a series of operations that must execute as a whole or not at all. You mark the beginning of a transaction with the BeginTrans statement. You use the Rollback or CommitTrans statement to end a transaction.

You can usually increase the record updates per second (throughput) of an application by placing operations that update data within an Access Basic transaction.

Because Visual Basic locks data pages used in a transaction until the transaction ends, using transactions will prevent access to those data pages by other users while the transaction is pending. If you use transactions in a multi-user environment, try to find a balance between data throughput and data access.

If database operations are not within a transaction, every Update method causes a disk write.

Transactions are very fast because they are written to a buffer in memory instead of to disk. CommitTrans writes the changes in the transaction buffer to disk. The size of the transaction buffer can be set in your MSACCESS.INI file, found in your Windows directory. See the PERFORM.TXT file in your Visual Basic directory for more information. Robust error trapping is important when using transactions to avoid losing writes if the program gets an error in the middle of a transaction.

For more performance tuning tips for data access in Microsoft Visual Basic version 3.0, see the PERFORM.TXT file.

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the following to the Form Load event code:

```
Sub Form_Load ()
   Dim Starttime, Endtime
   Dim db As Database
   Dim t As Table
   Dim i As Integer
   Dim tempName As String
   Dim temphone As String
   Set db = OpenDatabase("c:\BIBLIO.MDB")  ' Uses a copy of BIBLIO.MDB
   Set t = db.OpenTable("Publishers")
   Starttime = Now
   'BeginTrans   ' Add this and CommitTrans (below) for greater speed.
   For i = 1 To 100
      tempName = "testname" & Str$(i)  'Make an arbitrary unique string.
      tempPhone = Str$(i)              'Make arbitrary number.
      t.AddNew  'AddNew clears copy buffer to prepare for new record.
      t!PubID = 30 + i  ' Set primary key to unique value.
      t!Name = tempName  ' Set Name field to unique value.
      t!Telephone = tempPhone  ' Set Telephone field to unique value.
      t.Update   ' Write the record to disk or to transaction buffer.
   Next i
   'CommitTrans   ' Add this and BeginTrans (above) for greater speed.
   Endtime = Now
   MsgBox "Time required= " & Format(Endtime - Starttime, "hh:mm:ss")
   t.Close
   db.Close
   End
End Sub
```

   The above code adds 100 new records to the BIBLIO.MDB database file. Add the records to a copy of BIBLIO.MDB instead of to the original.

3. Start the program (or press the F5 key). A message box reports the time required to add 100 new records. Close the form to end the program.

If you don't use the BeginTrans and CommitTrans statements, this program reports 17 seconds to add 100 records on a 486/66 PC. When you add BeginTrans and CommitTrans as shown in the program comments above, the program takes less than 1 second on that computer. Performance may vary on different computers.

REFERENCES
==========

- "Microsoft Developer Network News" newspaper, January 1994, Volume 3, Number 1, published by Microsoft Corporation.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataAcc PrgOptTips

PRB: Dynaset Loses Contents After Transaction Rollback

Article ID: Q109995

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 3.0

SYMPTOMS

If a Dynaset is used in a transaction, records in the Dynaset appear to be lost when the transaction is rolled back.

CAUSE

This is by design. A Dynaset is populated one record at a time, so the RollBack operation removes all the records added during the transaction. After the RollBack operation, the Dynaset contains the single record that was there before BeginTrans began the transaction.

RESOLUTION

Create the complete Dynaset before starting the transaction. For example, replace the code shown in step 3 of the More Information section with this code:

```
Sub Command1_Click ()
   Dim db As database
   Dim ds As Dynaset
   Set db = OpenDatabase("Biblio.mdb")
   Set ds = db.CreateDynaset("authors")

   ' Create the complete Dynaset before starting the transaction.
   ds.MoveLast
   ds.MoveFirst

   ' Populate the listbox with the contents of the Dynaset.
   BeginTrans
      While Not ds.EOF
         list1.AddItem ds(0)
         ds.MoveNext
      Wend
   Rollback

   ds.MoveFirst
   While Not ds.EOF
      list2.AddItem ds(0)
      ds.MoveNext
   Wend
End Sub
```

STATUS
======

This behaviour is by design.

MORE INFORMATION
==================

Steps to Reproduce Behavior
----

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add two list boxes (List1 and List2) and a command button (Command1)
   to Form1.

3. Add the following code to the Command1_Click event:

```
   Sub Command1_Click ()
      Dim db As database
      Dim ds As Dynaset
      Set db = OpenDatabase("Biblio.mdb")
      Set ds = db.CreateDynaset("authors")

      'This code populates the listbox with the contents of the Dynaset.
      BeginTrans
         While Not ds.EOF
            list1.AddItem ds(0)
            ds.MoveNext
         Wend
      Rollback

      'This code reports only one record in the dynaset.
      ds.MoveFirst
      While Not ds.EOF
         list2.AddItem ds(0)
         ds.MoveNext
      Wend
   End Sub
```

4. Run the application or press the F5 key. Click the Command1 button. The
   first list box is populated correctly but the second contains only a
   single record.

Additional reference words: 3.00
KBCategory:
KBSubCategory: APrgDataOther

How to Use Wildcards in SQL Query to Make Dynasets & Snapshots

Article ID: Q110069

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

You can build a Dynaset or Snapshot based upon wildcard field-search characters in an SQL query. The find methods (FindFirst, FindLast, FindNext, and FindPrevious) can also search a Dynaset or Snapshot using wildcard search characters in an SQL query.

MORE INFORMATION

By using the Like statement in the SQL query language, you can search for database field values using the asterisk (*) and question mark (?) characters as wildcards. The * and ? wildcards let you find a wider set of field values beginning or ending with any desired root. For example, the following SQL syntax selects the records from a table where the Authorfield field values begin with the letter b:

```
Select * from XTable Where Authorfield Like 'b*'
```

NOTE: The Seek method, which only applies to Table object variables, cannot use SQL queries or wildcard search characters. The Seek method is limited to finding a single record using the comparison operators: >, >=, <=, <, =, and <>.

Asterisk (*) Wildcard Usage

In the SQL syntax for the Like statement, the asterisk (*) acts as a wildcard place holder for any number of characters, from zero up to the field length. A search for b* finds any field value beginning with the letter b. A search for *b finds any field value ending with b. A search for *xxxx* finds any field value that contains the xxxx substring. A search for * by itself matches all field values.

Question Mark (?) Wildcard Usage

In the SQL syntax for the Like statement, the question mark (?) acts as a wildcard place holder for a single character. A search for ??b* finds any field value that has b in the third character. A query for *b?? finds any field value with b as the third from the last character.

Speed Considerations

Of the following two techniques, 1 is faster than 2:

1. For greater speed, invoke the SQL wildcard field search only once to build the Dynaset or Snapshot of records that match your search criteria. Then use the fast move methods (MoveFirst, MoveLast, MoveNext, and MovePrevious) or click the data control to quickly navigate between all the records that match the specified search criteria. For example:

```
Dim MyDS As Dynaset, MyDB As database, SQLx As String
SQLx = "Select * from Authors Where Author Like 'b*' "
Set MyDB = OpenDatabase("BIBLIO.MDB")   'Open a database.
Set MyDS = MyDB.CreateDynaset(SQLx)     'Create Dynaset using SQLx.
While Not MyDS.Eof
   Print MyDS!author
   MyDS.MoveNext
Wend
```

   The Eof property is True after MoveNext moves past the last record.

   Visual Basic creates a Dynaset or Snapshot very quickly when using indexes. Subsequent find methods are relatively slow and sequential, as shown in technique 2 below.

2. A slower technique is to create a Dynaset composed of the entire table and then to use multiple find methods. Each FindNext would re-invoke the SQL wildcard field search to find the next matching record. This adds query time overhead. After finding a certain number of records, the total time taken would be slower than with technique 1 described above.

```
Dim MyDS As Dynaset, MyDB As Database, SQLx As String
SQLx = "author Like 'b*'"
Set MyDB = OpenDatabase("BIBLIO.MDB")     'Open a database.
Set MyDS = MyDB.CreateDynaset("Authors")  'Create Dynaset with table.
MyDS.FindFirst SQLx      'Find first record matching criteria.
Do Until MyDS.NoMatch
   Print MyDS!author
   MyDS.FindNext SQLx    'Find next record matching criteria.
Loop
```

   You can invoke the FindNext method until Nomatch = True, as shown.

Example Using SQL Wildcard Search with a Data Control

The Text1 box in the following program shows individual records of the Author field of the BIBLIO.MDB database. When you click the Command1 button, the program automatically appends and prefixes the * wildcard search character to any search string that you enter in the Text2 text box. That widens the resulting recordset shown in Text1. You can browse the recordset shown in Text1 by clicking the data control.

1. Start Visual Basic or begin a New Project. Form1 is created by default.

2. Double-click the form. Add the following to the Form Load event code:

```
Sub Form_Load ()
```

```
   text1.Text = "Enter ar* in Text2 and click Command1. Also try *z* "
   text2.Text = "*"  'A lone asterisk finds all records.
End Sub
```

3. Add a data control (Data1) to Form1.

4. Add a text box (Text1) to Form1. Give Text1 the following properties in order to bind it to the data control and to the Author field in the database table:

   ```
   DataSource = Data1
   DataField = Author
   ```

5. Add a second text box (Text2) without setting any properties. You can change the wildcard criteria for database queries in Text2 at run time.

6. Add a command button (Command1) to Form1. Add the following code to its Click event:

   ```
   Sub Command1_Click ()

      Dim db As database, ds As dynaset, MyTable As table
      Dim SQLX As String, SearchText As String
      Set db = OpenDatabase("biblio.mdb")

      'Optional: In Text2, append & prefix the * wildcard to widen search:
      If Right$(text2.Text, 1) <> "*" Then text2.Text = text2.Text & "*"
      If Left$(text2.Text, 1) <> "*" Then text2.Text = "*" + text2.Text
      'Remove the above 2 lines if you want the user to enter the asterisk
      SearchText = text2.Text

      ' The following SQL syntax selects all records from the Authors table
      ' where the Author field matches the SearchText string, using any *
      ' or ? wildcard characters. The result is ordered by the Au_id field:
      SQLx = "Select * From Authors Where Author Like '" & SearchText
      SQLx = SQLx & "' Order By Au_id"
      Data1.DatabaseName = "biblio.mdb" ' Tells Data1 the database name.
      Data1.RecordSource = SQLx ' Data1 control will use SQLx query string.
      Data1.Refresh ' Update the data control with results of SQL query.

   End Sub
   ```

7. Start the program by pressing the F5 key. When a lone asterisk (*) is in the Text2 box, clicking the Command1 button finds all the records.

   Enter ar in Text2 and click Command1. The program changes the query to *ar*. That finds all Author field values that contain the letters ar.

   Enter z or *z* and click Command1 to find all Author field values that contain the letter z anywhere in the field.

   Close the form to end the program.

To change the way the program automatically adds the * wildcard, you can modify or remove the If Left$... and If Right$... statements.

The Seek Method Does Not Support Wildcard Searches

The Seek method, which works only with Table object variables, is very fast but doesn't support wildcard searches. Seek is mainly useful for finding a single record that matches a given criteria. The find and move methods are more practical than the Seek method for finding a group of records.

The Seek method feature that is closest to a wildcard search is a comparison operator: >, >=, <=, or <. For example, you could find the first record that is greater than or equal to your search key value as follows:

```
Dim MyDB As Database, MyTable As Table
Set MyDB = OpenDatabase("BIBLIO.MDB")      ' Open a database.
Set MyTable = MyDB.OpenTable("Publishers") ' Open a table.
MyTable.Index = "PrimaryKey"               ' Define current index.
MyTable.Seek ">=", 3         ' Seek a record with PrimaryKey >= 3.
If MyTable.NoMatch Then
   MsgBox "match was not found"
Else
   MsgBox "match was found"
End If
```

Additional reference words: 3.00 faster slow speedy quick quicker
KBCategory: APrg
KBSubcategory: APrgDataAcc

PRB: Closed ODBC Database Stays Open Until Time-Out or VB Ends

Article ID: Q110227

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

Using a Close method on a database opened with Open Database Connectivity (ODBC) drivers doesn't close the database. The ODBC database process keeps running. To close the connection successfully, you must end the Visual Basic application.

CAUSE

After Visual Basic executes the Close method on an ODBC connection, the Microsoft Access engine in Visual Basic maintains a persistent connection in case the user reopens the database later in the program. This makes the program more efficient.

RESOLUTION

If a Visual Basic program does not reopen the ODBC connection after doing a Close method, a time-out occurs and the connection closes automatically.

You can control the time-out period by placing the following line in your VB.INI or <vb_exe_app_name>.INI file, where x is the number of seconds:

```
[ODBC]
ConnectionTimeout=x
```

The default ConnectionTimeout is usually 600 seconds (10 minutes). The lowest supported ConnectionTimeout value is 1 second. A ConnectionTimeout value of 0 says to never cause a time-out.

To enforce the fastest possible time-out, you can set ConnectionTimeout to 1. In addition, you can add the following code after you close the database to make sure the connection is terminated:

```
db.Close      ' Close database, using database object variable (db).
Start = Timer
Do            ' This loop pauses a second to allow a time-out
   FreeLocks  ' Tell Microsoft Access engine that program is idle.
   DoEvents   ' Tell Windows to do any pending events.
While Timer <= Start + 1
```

This loop delays for a second after the db.Close. The FreeLocks statement tells the database engine that the user is idle. If you run the Visual Basic program with ConnectionTimeout set to 1 in your VB.INI or

<vb_exe_app_name>.INI file, the database engine will disconnect the one-second-old connection to the server.

STATUS
======

This behavior is by design for all ODBC database connections.

MORE INFORMATION
================

Reproducing the Behavior

For example, using a Close method on a database opened via an ODBC connection to a Sybase SQL server leaves the Sybase session open. You can confirm this by executing sp_who on the Sybase server.

As another example, assume you have a Visual Basic application on a timer that regularly checks SQL Server version 4.2a on OS/2. To avoid wasting an SQL user connection between the timed checks, you might want the Close method to release the user connection. However, the Close method doesn't release the user connection.

NOTE: You can create Dynaset or Snapshot objects against ODBC databases, but you cannot use the OpenTable method to directly open ODBC tables.

Additional VB.INI or <vb_exe_app_name>.INI Settings

The following additional settings for the VB.INI or <vb_exe_app_name>.INI initialization file are useful for handling ODBC databases and time-outs:

Entry for [Debug]

Section	Value	Effect
RmtTrace	0	Use asynchronous query execution if possible; no ODBC API tracing (default).
	8	Trace ODBC API calls in ODBCAPI.TXT in the Microsoft Access directory.
	16	Force synchronous query execution.
	24	Trace ODBC API calls; force asynchronous query execution.

Entries for [ODBC]

Section	Value	Effect
TraceSQLMode	0	No tracing of SQL queries (default).
	1	Trace SQL queries sent to ODBC in SQLOUT.TXT in the Microsoft Access directory.

QueryTimeout	S	Wait S seconds for queries sent to ODBC, and then stop trying to process the query results (for asynchronous queries only). (Default: 60 seconds).
LoginTimeout	S	Wait S seconds for ODBC login response, and then stop trying to connect to a server. (Default: 20 seconds).
ConnectionTimeout	S	Wait S seconds, and then close idle ODBC connections. (Default: 600 seconds).
AsyncRetryInterval	M	Retry asynchronous queries every M milliseconds. (Default: 500 milliseconds).
AttachCaseSensitive	0	Attach the first table whose name matches the specified string, regardless of case.
	1	Attach a table only if its name exactly matches the specified string.
AttachableObjects	string	A list of object types you can attach. (Default: 'TABLE', 'VIEW', 'SYSTEM TABLE', 'ALIAS', 'SYNONYM'.)
SnapshotOnly	0	Get index information when tables are attached so that dynasets are allowed (default).
	1	Ignore index information when tables are attached so that only snapshots are allowed.

REFERENCES
==========

- "Microsoft Visual Basic for Windows Professional Features Book 2: Data Access Guide," pages 149-154 in Appendix C.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataODBC

How to Use Seek and MoveNext to Find a Group/Range of Records

Article ID: Q110497

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 3.0

SUMMARY

The Seek method can search for a value only in an indexed field. Seek can find only one record at a time. The Seek method alone cannot find all duplicate field values.

After doing a Seek, to find a group of records that have indexed field values that are duplicates or within a given range, you must do a series of move methods (MoveNext or MovePrevious). After each move method in a loop, you must check the indexed field value until your criteria is exceeded. The indexed field values are automatically in alphabetical or numerical order.

The sample program below uses a Seek method, then uses MoveNext in a loop to roughly emulate the FindNext method.

NOTE: FindNext applies only to Dynasets or Snapshots. The Seek method applies only to Table object variables.

MORE INFORMATION

The Seek method is very fast, but doesn't support SQL or wildcard searches to find groups of articles. Seek is mainly useful for finding one, single record that matches or exceeds a given value.

You can use one of the following methods instead of the Seek method to find a group of records:

- CreateDynaset method.
- CreateSnapshot method.
- Find methods (FindFirst, FindLast, FindNext, and FindPrevious), which work only on a Dynaset or Snapshot.
- Move methods (MoveFirst, MoveLast, MoveNext, MovePrevious), which work on a Table object variable, Dynaset, or Snapshot.

The Seek method requires you to first set the current index with the Index property. This orders the records alphabetically or numerically.

Seek can use only the following comparison operators: >, >=, <=, <, =, and <>. When the comparison is =, >=, >, or <>, Seek starts at the beginning of the index and searches forward. When the comparison is <= or <, Seek starts at the end of the index and searches backward. Thus, if three or more records have duplicate values in the current index, the Seek method cannot locate the middle records. Seek can locate only the first or last of those records, depending upon the comparison operator used. A move method is

required to locate those middle records. A MoveNext always moves forward one record from the current record found by a Seek, independent of the comparison operator that Seek used. MovePrevious moves one record previous.

Example: How to Use Seek and MoveNext to Find a Group of Records
---

The following sample program finds all records for which the PubID field is 2 in the BIBLIO.MDB database (9 records). The program uses one Seek to find the first record for which PubID is 2. The NoMatch property is False if the first match is found. From there onwards, the program uses MoveNext and tests MyTable!PubID in a loop to find all remaining records where PubID is 2. You could also modify this program to find a range of PubID field values.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the following to the Form Load event code:

```
Sub Form_Load ()

form1.Show   ' In Load event, must Show form to make Print visible.
Dim MyDB As Database, MyTable As Table, testval As Long
' Several duplicates exist in BIBLIO.MDB for PubID = 2 in Titles table.
' testval is the key value for which you want to Seek all duplicates:
testval = 2
Set MyDB = OpenDatabase("BIBLIO.MDB")      ' Open a database.
Set MyTable = MyDB.OpenTable("Titles") ' Open a table.
' Sort the Titles table by the PubID indexed field, which is designed
' with duplicates OK:

MyTable.Index = "PubID"
MyTable.Seek "=", testval  ' Seek a record with PubID key = testval.
If MyTable.NoMatch Then
   MsgBox "Match for " & testval & " was not found"
Else
   Do
      Print MyTable!PubID & ": " & MyTable!Title
      x = MsgBox("Match was found. PubID = " & MyTable!PubID & ": ", 1)
      If x = 2 Then End  ' End if user clicks Cancel on message box.
      MyTable.MoveNext   ' Move to next record.
      If MyTable!PubID <> testval Then Exit Do  'Stop when past testval.
   Loop
End If

End Sub
```

3. Start the program (or press F5). Click OK multiple times to see all record titles where PubID is 2. Choose Cancel if you want to abort the MoveNext loop. Close the form to end the program.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Copy a Record from One Table to Another in VB

Article ID: Q110588

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

A physical table in a database has records (rows) and fields (columns). A single record contains a single row of field values.

To copy a record in one table to another table, you need to copy all the fields in the source record to the corresponding fields in the destination record. You can do this by using the Value property of the Fields collection, or by using an SQL statement.

MORE INFORMATION

How to Copy a Record Using SQL

You can use the SQL Insert Into statement to copy specified records from one table into another:

```
SELECT FromTableName.* INTO ToTableName FROM FromTableName
```

You can also add a WHERE clause at the end to add any selected records:

```
SELECT FromTableName.* Into ToTableName.* Where Key = 'Key'
```

How to Copy a Record Using the Fields Collection and Value Property

The following loop copies all the fields in the current record in table 1 to the corresponding fields in the current record in table 2:

```
Dim MyDB As Database, Tbl1 As Table, Tbl2 As Table
Set MyDB = OpenDatabase("BIBLIO.MDB")      ' Open Database.
Set Tbl1 = MyDB.OpenTable("Publishers")     ' Open Table.
Set Tbl2 = ...

For i = 0 to Tbl2.Fields.Count - 1
   Tbl1(Tbl2.Fields(i).Name).Value = Tbl2.Fields(i).Value
Next
```

The above loop assumes that the fields in table 2 are identical to those in table 1.

Additional reference words: 3.00
KBCategory: APrg

KBSubcategory: APrgDataOther

PRB: Couldn't Open PARADOX.NET When Opening Paradox 3.x Table

Article ID: Q110590

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

When you attempt to open a Paradox version 3.x table, the error message "Couldn't open PARADOX.NET" (trappable error number 3172) can occur under circumstances described below.

CAUSE

Your Visual Basic <program>.INI file may have an incorrect or missing path to the PARADOX.NET file. Paradox uses the PARADOX.NET network control file to list the network rights, locks, users, and the type of network you are using.

Paradox installs a PARADOX.NET file during normal installation. By default, PARADOX.NET will be located in the Paradox program directory. During a network installation, the location of PARADOX.NET is determined by the network administrator.

RESOLUTION

You usually only need the PARADOX.NET file if you are running Paradox on a network or accessing Paradox files on a network.

You can create or change the PARADOX.NET configuration by running NUPDATE.EXE, a utility that comes with Paradox. NUPDATE.EXE lets you specify the directory in which to store your PARADOX.NET file. Refer to the Paradox "Network Administrator's Guide" for more information on this topic.

Once you find or create the PARADOX.NET file, you need to update your Visual Basic VB.INI or <program>.INI file with the correct path to find PARADOX.NET. For example, if you store your PARADOX.NET path on Drive C in the root directory, your VB.INI or <program>.INI file would look like the following example:

```
[Installable ISAMs]
Paradox 3.x = c:\windows\system\pdx110.dll     ;Path of Paradox driver

[Paradox ISAM]
PageTimeout=600              ; 60-second default Page Timeout
ParadoxUserName=Joe User     ;Name displayed when lock conflicts occur
ParadoxNetPath=C:            ;Path to the PARADOX.NET file
CollatingSequence=Ascii      ;Collating sequence of your files
                             ;  (ASCII, International, Norwegian-Danish,
```

; or Swedish-Finnish)

If you are not opening the Paradox table on a network, you can correct other possible PARADOX.NET error messages as follows:

1. If you or someone on your PC previously defined a Network by running NUPDATE.EXE, run NUPDATE.EXE again to remove the Network settings.

2. If you have the ParadoxNetPath statement in your Visual Basic VB.INI or <program>.INI file, and you don't have a PARADOX.NET file, remove the offending ParadoxNetPath statement in the VB.INI or <program>.INI file.

Paradox products are manufactured independent of Microsoft. Microsoft makes no warranty, implied or otherwise, regarding Paradox product performance or reliability.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

PRB: Object Variable Not Set When Referencing Data Control

Article ID: Q110618

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 3.0

SYMPTOMS

If a Data control references a Data control in the Form_Load event or anytime BEFORE the Form_Activate event, you will receive the following error message (Error#91):

```
Object Variable Not Set
```

CAUSE

The RecordSet property of a Data Control is not resolved until after the form has been loaded -- that is, after the Form_Load event. This means that you cannot issue any method such as MoveNext or FindNext of the Data Control that uses its record set (its internal Dynaset).

RESOLUTION

To prevent the error, force the creation of the internal dynaset in the data control by issuing the Refresh method:

```
Data1.Refresh
```

This will validate the RecordSet property allowing you to use methods that require this.

STATUS

This behaviour is by design.

MORE INFORMATION

Steps to Reproduce Behavior

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a data control (Data1) to Form1.

3. In the Form_Load event of Form1, place the following code:

```
' Data1.Refresh
```

```
    Data1.MoveLast
```

4. Run the application. You should get Error 91 "Object Variable Not Set."

5. Remove the apostrophe from the Data1.Refresh line.

6. Run the application again (press the F5 key). Now, you won't get the error.

Additional reference words: 3.00
KBCategory:
KBSubCategory: APrgDataOther

How to Determine the Restored State of a Minimized Form

Article ID: Q110620

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

It is not possible within Visual Basic to determine programatically whether an iconized form will be restored to a maximized or normal state without using the Windows API (application programming interface) function GetWindowPlacement(). This article gives an example of how to call this function from a Visual Basic application.

MORE INFORMATION

The following program demonstrates how to determine whether a minimized form will be restored as maximized or normal. The program requires two forms with a Label and a Command control on Form 1.

1. Start a new project in Visual Basic. Form1 is created by default.

2. From the File menu, choose New Form to create Form2.

3. From the File menu, choose New Module to create a .BAS file and enter the following code in the (General) (Declarations) section:

```
Type RECT
   left As Integer
   top As Integer
   right As Integer
   bottom As Integer
End Type

Type POINTAPI
   x As Integer
   y As Integer
End Type

Type WINDOWPLACEMENT
   length As Integer
    flags As Integer
   showCmd As Integer
   ptMinPosition As POINTAPI
   ptMaxPosition As POINTAPI
   rcNormalPosition As RECT
End Type

Global Const WPF_RESTORETOMAXIMIZED = &H0002
```

```
' Enter the following Declare statement as one, single line:

Declare Function GetWindowPlacement Lib "User"
   (ByVal hWnd As Integer, lpwndpl As WINDOWPLACEMENT) As Integer

Function is_max (hWnd As Integer) As Integer
   Dim wp As WINDOWPLACEMENT
   Dim rtn As Integer
   wp.length = Len(wp) ' Initialize size
   rtn = GetWindowPlacement(hWnd, wp)
   If wp.flags = WPF_RESTORETOMAXIMIZED Then
      is_max = True
   Else
      is_max = False
   End If
End Function
```

NOTE: The value of wp.length must be initialized or the call to GetWindowPlacement will return an error. There is no mention of this in the Microsoft Windows "Programmers Reference," but it is described as a documentation error in the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q89569.
TITLE : DOCERR: GetWindowPlacement Function Always Returns an Error

4. Add a Label control and Command control to Form1.

5. Add the following code to the Command_Click procedure of Form1:

```
Sub Command1_Click ()
   If is_max((Form2.hWnd)) Then
      Label1.Caption = "Form 2 will be Maximized"
   Else
      Label1.Caption = "Form 2 will be Normalized"
   End If
End Sub
```

NOTE: Form2.hWnd cannot be passed directly to a function as a parameter. It must be enclosed in an extra set of parentheses or stored in a temporary variable.

6. From the File menu, choose Save Project to save the forms and project.

7. Run your application by choosing Start from the Run menu or by pressing the F5 function key. Minimize Form2, click the Command button on Form1, and observe the message displayed in the Label control. Restore Form2, maximize it, and then minimize it again. Click the Command button on Form1, and again observe the message displayed in the Label control.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataODBC

PRB: Commit or Rollback without BeginTrans Error and VB Forms

Article ID: Q110722

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

Under the conditions outlined below, you may get this error:

```
Commit or Rollback without BeginTrans
(trappable error, Err = 3034)
```

This error occurs when a program has two or more forms, and two of the forms each contain a data control that is connected to a different database and table. The first form invokes a BeginTrans statement. The program loads, then later unloads the second form, without explicitly invoking the Database.Close method for the second data control. When the first form invokes a CommitTrans or Rollback statement, you get the error message.

CAUSE

Transactions are global and not limited to only one database or recordset. If you include operations on more than one database or recordset within a transaction, Rollback restores all operations on all databases.

If your code does not explicitly invoke a Database.Close method for a data control on a form that is unloaded, Visual Basic automatically invokes a Rollback statement and a Database.Close method. That automatic Rollback cancels your previous BeginTrans statement. Then, invoking a CommitTrans or Rollback statement correctly gives the "Commit or Rollback without BeginTrans" error message.

WORKAROUND

In the second form's Unload event, add a Data1.Database.Close method to prevent the automatic Rollback.

STATUS

This behavior is by design.

MORE INFORMATION

BeginTrans, CommitTrans, and Rollback Statements

To perform database transactions in Visual Basic, you can use the BeginTrans, CommitTrans, Rollback statements. BeginTrans begins a new transaction. CommitTrans ends the current transaction. Rollback ends the current transaction and restores the database to the state it was in just before the current transaction began.

A transaction is a series of changes you make to a database that you want to treat as one complete unit. A transaction begins when you use the BeginTrans statement. Use Rollback to undo changes made during the current transaction, and CommitTrans to accept changes and end the current transaction. Both Rollback and CommitTrans end a transaction. Once you use CommitTrans, you can't undo changes made during that transaction.

You can have up to five levels of transactions open at once by using multiple BeginTrans statements. Typically, you use transactions to maintain the integrity of your data when records in two or more tables must be updated. For example, if you transfer money from one account to another, you might subtract a sum from one and add the sum to another. If either update fails, the accounts no longer balance. Use BeginTrans before updating the first record, and then if any subsequent update fails, you can use Rollback to undo all of the updates. Use CommitTrans after the last record has been successfully updated.

NOTE: Some databases, such as Paradox, may not support transactions, in which case the Transactions property of the Database object is False. Test the value of the Transactions property before using BeginTrans to make sure the Database supports transactions. If transactions are not supported, these statements are ignored and no error occurs.

If you use CommitTrans or Rollback statements without first using BeginTrans, an error occurs. If you use Rollback, you should use Refresh on any data control that refers to data that may have changed since the transaction began.

Steps to Reproduce Behavior
---

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a data control (Data1) to Form1.

3. Connect Data1 on Form1 to a table in a database as follows:

   Select the Data1 control and press the F4 key to display the Properties window. Set the DatabaseName property to C:\VB3\BIBLIO.MDB, and set the RecordSource property to the source table name Publishers.

4. From the File menu, choose New Form to create Form2.

5. Add a data control (Data1) to Form2.

6. Connect Data1 on Form2 to any table in any database (the same or different database than on Form1) as follows:

   Select the Data1 control and press the F4 key to display the Properties window. Set the DatabaseName property to C:\ACCESS\NWIND.MDB, and set the RecordSource property to the source table name Categories.

7. Add the following code to the Form1 Load event:

```
Sub Form_Load ()
   BeginTrans  ' Begin the transaction.
   Form1.Show
   Form2.Show  ' Show Form2 on top of Form1.
End Sub

Sub Form_Unload (Cancel As Integer)
   CommitTrans   ' This statement causes an error.
End Sub
```

8. Start the program, or press the F5 key.

9. Close Form2.

10. Close Form1. This results in the following error message:

    Commit or Rollback without BeginTrans (Err = 3034)

In this example, the error you get when unloading Form1 is actually caused by unloading Form2.

When Form1 loads, Visual Basic automatically invokes the Data1.Refresh method for the attached data control. That automatically opens the specified database and table. When Form2 loads, the same behavior occurs to open the second database and table.

As Form2 unloads, Form2 checks to see if the data control's database is still open. If the database is still open, Visual Basic automatically does a Rollback and closes the database in order to cancel any unsaved changes to the current record in the data control. This default behavior often saves you from writing extra code. That automatic Rollback cancels the BeginTrans that you invoked in the Form1 Load event. As Form1 unloads, the CommitTrans in the form's unload event has no transaction to commit, so you get the error message.

To work around this behavior, add a Data1.Database.Close method in the Unload event for Form2 to prevent the automatic Rollback.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

Documentation and Features for Visual Basic's Data Manager

Article ID: Q110723

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

You can run the DATAMGR.EXE Data Manager program as follows:

- Choose Data Manager from the Window menu in Visual Basic, or
- Run the DATAMGR.EXE program from the File Manager in Windows.

The complete documentation for using Data Manager is found in Data Manager's Help menu. You can also run the DATAMGR.HLP Help file from File Manager in Windows. Visual Basic provides no printed documentation for Data Manager. Instead, you can print the topics from the Help menu.

MORE INFORMATION

The DATAMGR.EXE Data Manager program provided with Visual Basic has the following features:

- Open an existing database that has one of the following formats:
  - Microsoft Access
  - FoxPro version 2.0 or 2.5
  - dBase III or dBase IV
  - Paradox version 3.x
  - BTRIEVE
- Create a new database in Microsoft Access format (.MDB)
- Create new tables in a database
- Create an index in a table
- Add fields to a table
- Modify table data
- Delete an existing table
- Compact a database
- Repair a database

The Data Manager gives you a subset of the features found in Microsoft Access for Windows. Data Manager, Visual Basic, and Microsoft Access all have the same native database format, an .MDB file. A single .MDB file contains the database structure as well as the data itself.

Introduction from Data Manager's Help Menu Topic

The Visual Basic Data Manager allows Visual Basic users to create new databases (.MDB format) and examine or map the structure of existing external databases in a variety of formats. Formats that you can either create or modify with the Data Manager are shown below:

Database Format	Create	Modify
Microsoft Access 1.0	Yes	Yes
Microsoft Access 1.1	Yes	Yes
Paradox 3.0 and 3.5	No	Yes
dBASE III and IV	No	Yes
FoxPro 2.0 and 2.5	No	Yes
Btrieve	No	Yes

Visual Basic shares its database engine with Microsoft Access, so databases created with Visual Basic or the Data Manager can be manipulated using Microsoft Access. In addition, databases created with Microsoft Access can be manipulated using Visual Basic and the Data Manager. Throughout the Visual Basic documentation, databases created with that engine are referred to as Visual Basic databases.

For additional information on using external databases, see the file EXTERNAL.TXT or BTRIEVE.TXT supplied with Visual Basic.

Many external databases exist as directories on your disk. To create databases in these formats, use the File Manager to create a directory that will become the database. Once the database directory is created, use the Data Manager to add tables and indexes, which become files in this directory.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

PRB: Error: Couldn't Lock File SHARE.EXE Hasn't Been Loaded

Article ID: Q110732

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 3.0

SYMPTOMS

Trying to use a Microsoft Access Database that is located on a Read-Only Share or Network Drive might generate the following error:

```
Couldn't lock file SHARE.EXE hasn't been loaded.
```

CAUSE

When Visual Basic tries to connect to a Microsoft Access database, an .LDB file is created or the existing .LDB is used in the Database directory. The .LDB file is used to assist in the management of the file locking mechnisim with multiple users. This file must be in the same directory as the database and the file or directory must have Read/Write access or the above error is generated.

This happens because the Microsoft Access engine in Visual Basic cannot create or write to the necessary .LDB file.

RESOLUTION

In order to prevent the use or creation of the .LDB file, open the database Exclusive use and Readonly (because of the readonly of the server) access. This tells the Microsoft Access Engine that the database will be opened for single user only and that the .LDB file will not be necessary. If the database is multiuser, users will have to have Read/Write access to the .LDB file.

STATUS

This behaviour is by design.

MORE INFORMATION

Steps to Reproduce Behavior

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a Data Control to Form1, and set its Read-Only property to True.

3. In the DatabaseName property of the Data Control, enter the name of a Microsoft Access Database that is located on a Read-Only network share.

3. Run the application, and you will recieve Error 91:

   Couldn't lock file SHARE.EXE hasn't been loaded.

Additional reference words: 3.00
KBCategory:
KBSubCategory: APrgDataOther

How to Use SQL SELECT Statement Without Field Syntax Error

Article ID: Q110752

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

The SQL command "SELECT FROM AuthorsTable" gives the following error message:

```
Syntax error in Select statement
```

CAUSE

The SELECT statement must be followed immediately by a valid field name, or by an asterisk (*) to indicate all fields in the table. This requirement applies to all forms of the SELECT statement, including the SELECT INTO, SELECT ALL, SELECT DISTINCT, and SELECT DISTINCTROW statements.

RESOLUTION

Indicate the fields you want immediately after the SELECT statement, for example:

```
SELECT * FROM AuthorsTable
```

STATUS

This behavior is by design.

MORE INFORMATION

SQL SELECT Statement Syntax

The SQL SELECT statement specifies which fields you want to retrieve. You use the FROM clause to indicate which tables contain those fields. You use the WHERE clause to indicate which records are to be retrieved.

SELECT is usually the first word in an SQL statement. If you include more than one field, separate the field names with commas. List the fields in the order you want them to be retrieved. If a field name appears in more than one table listed in the FROM clause, precede the field name with the table name and the . (dot) operator. In the following example, the AU_ID field is in both the Authors table and the Titles table. The SQL statement selects the Title field from the Titles table and the Author field from the

Authors table:

```
SELECT Titles.Title.Dept, Author
FROM Titles, Authors
WHERE Titles.AU_ID = Authors.AU_ID
```

Visual Basic requires the SQL command string (such as the one above) to be concatenated into one, single line in a string variable or quoted string.

You can use an asterisk (*) to select all fields in a table. The following example selects all of the fields in the Publishers table:

```
SELECT Publishers.* FROM Publishers
```

You can use the AS reserved word to create an alias for a field name. The following example uses the Year for the field name:

```
SELECT [Year Published] AS Year
FROM Titles
```

When you use a field name that contains a space or punctuation, surround the name with brackets:

```
SELECT [Year Published], Title
FROM Titles
```

For more information on SQL syntax, see the SQL topic in Visual Basic's Help menu.

Steps to Reproduce Behavior
---

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the following to the Form Load event code:

```
Sub Form_Load ()
   Dim db As database
   Dim ds As dynaset
   Set db = OpenDatabase("C:\VB3\BIBLIO.MDB")
   ' The following line gives "Syntax error in SELECT statement":
   Set ds = db.CreateDynaset("SELECT FROM Publishers")

   ' Replace the above line as follows to correct the syntax error:
   ' Set ds = db.CreateDynaset("SELECT * FROM Publishers")

   form1.Show
   Print ds![Company Name]   'Prints Company Name field from record 1.

End Sub
```

3. Start the program, or press the F5 key. The "Syntax error in SELECT statement" message displays. From Visual Basic's Run menu, choose End to clear that error and end Visual Basic's break mode.

To correct this syntax error, change the SELECT so that it is followed by a valid field name, or by an asterisk (*) to select all fields. For example,

the SQL command "SELECT * FROM Publishers" correctly selects all fields in the Publishers table.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

Create Database with Data Manager & View w/ Text/Data Control

Article ID: Q110753

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article gives step-by-step examples of the following:

- How to create a database with Visual Basic's DATAMGR.EXE Data Manager.
- How to view a database using a text box bound to a data control.

MORE INFORMATION

How to Create a Database Using Visual Basic's Data Manager

You can create a new database as follows:

1. Open the Data Manager program by choosing it from the Window menu in Visual Basic or by running DATAMGR.EXE from the Windows File Manager.

2. In the Data Manager, choose New Database from the File menu and select Access 1.1.

3. Click the New button and enter tbl1 for the table name.

4. Click the Design button. Then click the Add button. Enter fld1 for the Field Name and select Integer for the Field Type.

5. Save this Microsoft Access database with the name TEST1.MDB. Close the Data Manager when finished.

You can add data to a database with the Data Manager by selecting a table and choosing the Open button. You can click Add; then enter data into the field(s) of the new record. Click Update, or click Add again to enter the next record. You can also scroll to any existing record, modify the data in any field, and then click the Update button to write the record to the table.

How to View a Database By Binding a Text Control to a Data Control

1. Start a new project in Visual Basic. Form1 is created by default.

2. Create a Visual Basic database (or Microsoft Access database) as described above. Name this database TEST1.MDB.

3. Add the following controls to Form1, and set the following properties:

Control Name	Property	New Value	NOTE
Data1	DatabaseName	C:\VB\TEST1.MDB	Database created above.
Data1	RecordSource	tbl1	Valid table name.
Text1	DataSource	Data1	Name of data control.
Text1	DataField	Fld1	Valid field name.

4. To run the program, choose Start from the Run menu, or press the F5 key.

5. Scroll through one record at a time in the database using the data control. View the contents of the Fld1 field in the text box. Close the form to end the program.

Data Manager

By using the DATAMGR.EXE Data Manager program provided with Visual Basic, you can do all this:

- Create a database
- Create new tables
- Create an index
- Open an existing database
- Add fields to a table
- Modify table data
- Delete an existing table
- Compact a database
- Repair a database

The Data Manager gives you a subset of the features found in Microsoft Access for Windows. Data Manager, Visual Basic, and Microsoft Access all create the same database format, an .MDB file. An .MDB file contains the database structure as well as the data itself.

The documentation for Data Manager is found in Data Manager's Help menu.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Delete a Table from a Database Using Visual Basic

Article ID: Q110959

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article describes how to delete a table from a database using the Professional Edition of Visual Basic version 3.0 for Windows.

This technique works for any database that is in the native Microsoft Access database format. With slight modifications, it will also work with non-Microsoft Access databases.

MORE INFORMATION

To delete a table from a Microsoft Access database in Visual Basic, use any of the following methods:

- Open the database in the Visual Basic Data Manager, select the table, and choose the Delete button. You can run the Data Manager program from the Window menu in Visual Basic, or from the Windows File Manager (run DATAMGR.EXE in the Visual Basic directory). You can delete a table from any database type supported by Visual Basic.

- Use the sample Visual Basic program listed below to delete a table using database object variables.

- Open the database in Microsoft Access, select the table, and choose Delete from the Edit menu.

CAUTION: When you delete a table, all the data stored in that table is also deleted. If you want to preserve the data in the table you are going to delete, write a Visual Basic application to copy the data to a new table before deleting the existing table.

If you want to delete all the records in a table and still preserve the TableDef table definition, you can use the Execute method to do an SQL Delete command. For example:

```
Dim db as database
Set db=OpenDatabase("testing.mdb")
db.Execute "Delete From BadTable"
```

Sample Program

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a the following code to the Form Load event:

```
Sub Form_Load ()

   Dim db As database
   Dim tds As TableDefs
   form1.Show  ' Must Show form in Load event for Print to be visible.
   form1.WindowState = 2   ' Maximize Form1 to make room for table list.
   sourcedb = "c:\VB3\BIBLIO.MDB"  ' Original master database.
   destdb = "C:\TEST.MDB"     ' Path to database with table to delete.
   tabletodelete = "Authors"
   FileCopy sourcedb, destdb  ' Use copy of database; preserve original.
   Set db = OpenDatabase(destdb)
   Set tds = db.TableDefs     ' Open the TableDefs collection.

   ' Display names of all tables in database:
   For j = 0 To tds.Count - 1
      Print tds(j).Name
   Next
   Print

   ' Delete a table. (This deletes the TableDef and all records):
   tds.Delete tabletodelete
   ' or use:   db.TableDefs.Delete tabletodelete

   ' If you want to delete all records and still preserve the TableDef
   ' table definition, use the following instead of the above Delete:
   ' db.Execute "Delete From " & tabletodelete

   ' Display names of all tables in database:
   Print "List of tables after deleting one table:": Print
   For j = 0 To tds.Count - 1
      Print tds(j).Name
   Next

End Sub
```

3. Start the program or press the F5 key. The program lists all the tables in the database before and after deleting a table. Close the form to end the program.

You can also confirm that the table was deleted from the database by opening the TEST.MDB database with the Data Manager provided with Visual Basic or with Microsoft Access.

The Database Object Hierarchy

At the top of the database object hierarchy is the Database object, not to be confused with the Database property of the data control. One of the properties of the Database object is the TableDefs collection, which is also an object. The TableDefs collection represents all the individual TableDef objects associated with the Table objects, including any attached external tables. The TableDef objects each represent the structure or metadata of a table.

Each TableDef object consists of properties. For example, the Name property

gives you the name of the table. The Fields and Indexes properties of a TableDef object are collections of two additional data access objects, the Field object and the Index object. For more information, see the Visual Basic Help menu.

More Examples of Data Access

The VISDATA.MAK project, which is installed in the VB3\SAMPLES\VISDATA directory, gives extensive examples of data access. The VISDATA sample program uses every data access function in Visual Basic. Refer to the VISDATA source code for examples that show how to use each data access function.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

PRB: Novell Btrieve Unexpected Error from External DB Driver

Article ID: Q111286

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

The following error message number 3275 can occur from a Visual Basic program attempting to open a database on the Novell Netware network:

   Unexpected error from external database driver ().

CAUSE

This error message can occur when Btrieve is not started on the server.

RESOLUTION

Start Btrieve using the BStart command on the server.

NOTE: The products included here are manufactured by vendors independent of Microsoft; we make no warranty, implied or otherwise, regarding these products' performance or reliability.

STATUS

This behavior is by design.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

PRB: VB Record Too Large When Add or Update Record > 2K

Article ID: Q111304

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

The following error is the result when you Update or Add a Text field in a table and the total record size exceeds about 2000 bytes for all fields combined (not counting Memo fields):

```
Record is too large.
[Trappable Error number 3047.]
```

CAUSE

Records in a table in a Visual Basic or in a Microsoft Access database are limited to slightly under 2K, not counting Memo fields. The "Record is too large" error occurs when you enter data into such a record, not when you define the table structure.

RESOLUTION

Redefine the table by making some fields shorter or by removing unneeded fields.

You can also avoid this problem by using fields with the Memo type instead of the Text type. You can set a field's Type property to 12 to get a Memo type, instead of 10 to get a Text type. When a Memo field is greater than 250 bytes or whenever the 2K limit is reached on a record, Visual Basic automatically puts the Memo field on a separate page in the database file. If your Text fields contain related data, you could further improve space usage by concatenating the fields into one large Memo field.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Reproduce Behavior

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the following to the Form Load event code:

```
Sub Form_Load ()

   ' @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
   ' Create an empty database with the following design:

   Const DB_LANG_GENERAL = ";LANGID=0x0809;CP=1252;COUNTRY=0"
   Const numfields = 9  'Number of text fields to add to db, minus 1.
   Dim db As Database
   Dim MyDS As Dynaset
   Dim tdef As New TableDef
   Dim FieldInteger As New field
   Dim fieldname0 As New field
   Dim fieldname1 As New field
   Dim fieldname2 As New field
   Dim fieldname3 As New field
   Dim fieldname4 As New field
   Dim fieldname5 As New field
   Dim fieldname6 As New field
   Dim fieldname7 As New field
   Dim fieldname8 As New field
   Dim fieldname9 As New field
   Dim uniqindex As New Index

   form1.Show  ' Must Show form in Load event for Print to work.
   Kill "c:\tempx.MDB"
   Set db = CreateDatabase("c:\tempx.MDB", DB_LANG_GENERAL)
   tdef.Name = "Testtable"    ' Name of table to create.

   'Define the fields in the Testtable table:
   FieldInteger.Name = "fieldinteger"
   FieldInteger.Type = 4   'Long integer
   fieldname0.Name = "fieldname0"
   fieldname0.Type = 10   ' Type 10 = Text. Type 12 = Memo.
   fieldname0.Size = 255  ' Maximum size of field.
   fieldname1.Name = "fieldname1"
   fieldname1.Type = 10   ' Type 10 = Text. Type 12 = Memo.
   fieldname1.Size = 255  ' Maximum size of field.
   fieldname2.Name = "fieldname2"
   fieldname2.Type = 10   ' Type 10 = Text. Type 12 = Memo.
   fieldname2.Size = 255  ' Maximum size of field.
   fieldname3.Name = "fieldname3"
   fieldname3.Type = 10   ' Type 10 = Text. Type 12 = Memo.
   fieldname3.Size = 255  ' Maximum size of field.
   fieldname4.Name = "fieldname4"
   fieldname4.Type = 10   ' Type 10 = Text. Type 12 = Memo.
   fieldname4.Size = 255  ' Maximum size of field.
   fieldname5.Name = "fieldname5"
   fieldname5.Type = 10   ' Type 10 = Text. Type 12 = Memo.
   fieldname5.Size = 255  ' Maximum size of field.
   fieldname6.Name = "fieldname6"
   fieldname6.Type = 10   ' Type 10 = Text. Type 12 = Memo.
   fieldname6.Size = 255  ' Maximum size of field.
   fieldname7.Name = "fieldname7"
   fieldname7.Type = 10   ' Type 10 = Text. Type 12 = Memo.
   fieldname7.Size = 255  ' Maximum size of field.
   fieldname8.Name = "fieldname8"
   fieldname8.Type = 10   ' Type 10 = Text. Type 12 = Memo.
```

```
fieldname8.Size = 255  ' Maximum size of field.
fieldname9.Name = "fieldname9"
fieldname9.Type = 10   ' Type 10 = Text. Type 12 = Memo.
fieldname9.Size = 255  ' Maximum size of field.

'Add the fieldinteger and fieldnameN fields to the Fields collection:
tdef.Fields.Append FieldInteger
tdef.Fields.Append fieldname0
tdef.Fields.Append fieldname1
tdef.Fields.Append fieldname2
tdef.Fields.Append fieldname3
tdef.Fields.Append fieldname4
tdef.Fields.Append fieldname5
tdef.Fields.Append fieldname6
tdef.Fields.Append fieldname7
tdef.Fields.Append fieldname8
tdef.Fields.Append fieldname9

'Define fieldinteger_index, the unique primary-key index:
uniqindex.Name = "fieldinteger_index"
uniqindex.Fields = "fieldinteger"
uniqindex.Unique = True
uniqindex.Primary = True

'Append the fieldinteger_index index to the Indexes collection:
tdef.Indexes.Append uniqindex

'Append the tdef table definition (TableDef object) to the TableDefs
'collection:
db.TableDefs.Append tdef
db.Close
' The above code creates the empty database.
' @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Set db = OpenDatabase("c:\tempx.MDB")     ' Open the empty database.
Set MyDS = db.CreateDynaset("Testtable")  ' Make dynaset from table.

For i = 0 To 5  ' Add index field values for 5 new records:
   MyDS.AddNew
   MyDS!FieldInteger = i
   MyDS.Update
Next
MyDS.MoveFirst  ' Move to the first record.

' Add more than 2K of string data to the fields in the first record:
For j = 0 To numfields
   MyDS.Edit  ' Opens current record for editing, into copy buffer.
   f$ = "fieldname" & j
   Debug.Print f$
   ' The maximum allowed record size is a little less than 2K.
   ' Fields fieldname0 through fieldname6 are each assigned 255 bytes
   ' with no problem. However, when assigning fieldname7, following
   ' field assignment fails at run time with Error 3047:
   '     "Record is too large"
   MyDS(f$) = String$(255, "x") 'Assign 255 bytes to each text field.
   MyDS.Update  ' Saves the copy buffer to the table.
Next
```

```
      MyDS.Close
      db.Close

   End Sub
```

3. Start the program, or press the F5 key. After a few seconds, the program gives Error 3047, "Record is too large." Choose End from the Run menu to clear the error.

To correct this behavior, redefine the database using fields of type Memo instead of type Text. In the program listed above, replace all the

```
   fieldnamex.Type = 10
```

statements with:

```
   fieldnamex.Type = 12
```

where x = 0 to 9.

Additional reference words: 3.00 limitation specification larger smaller bigger
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Create Database with Memo Fields Up to 32000 Bytes

Article ID: Q111314

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY
=======

When you use a database Update or Add method at run time, the sum of all Memo field sizes in a record cannot exceed about 32000 or 33000 characters.

In the example program listed in the More Information section below, 33000 total characters write successfully to ten Memo fields. But writing 34000 characters causes the write to fail on the last Memo field. No error message is displayed.

For fields of type Text, the sum of the Text field sizes in a record cannot exceed about 2000 characters. You can use Memo fields instead of Text fields to get a capacity of up to about 32000 or 33000 characters in a record.

MORE INFORMATION
================

CAUTION: Large record sizes quickly consume a large amount of disk space as you write new records. If you have large records, consider redesigning the database to make use of related tables containing small record sizes.

The sample program below creates a new database (C:\TEMPZ.MDB) with 10 Memo fields and one long-integer field. This program demonstrates the 32000-byte or 33000-byte maximum size for the sum of all Memo fields.

Step-by-Step Example
----

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the following code to the Form Load event:

```
Sub Form_Load ()

    Const DB_LANG_GENERAL = ";LANGID=0x0809;CP=1252;COUNTRY=0"
    Const numfields = 9  'Number of Memo fields to add to db, minus 1.
    Dim db As Database
    Dim MyDS As Dynaset
    Dim tdef As New TableDef
    Dim FieldInteger As New field
    Dim fieldname0 As New field
    Dim fieldname1 As New field
    Dim fieldname2 As New field
    Dim fieldname3 As New field
    Dim fieldname4 As New field
```

```
Dim fieldname5 As New field
Dim fieldname6 As New field
Dim fieldname7 As New field
Dim fieldname8 As New field
Dim fieldname9 As New field
Dim uniqindex As New Index

form1.Show  ' Must Show form in Load event for Print to work.
On Error Resume Next  ' Ignore the error if file doesn't exist yet:
Kill "C:\TEMPZ.MDB"   ' Delete db if it exists from previous run.
On Error GoTo 0
Set db = CreateDatabase("C:\TEMPZ.MDB", DB_LANG_GENERAL)
tdef.Name = "Testtable"    ' Name of table to create.

'Define the fields in the Testtable table:
FieldInteger.Name = "fieldinteger"
FieldInteger.Type = 4   'Long integer
fieldname0.Name = "fieldname0"
fieldname0.Type = 12   ' Type 10 = Text, Type 12 = Memo
fieldname1.Name = "fieldname1"
fieldname1.Type = 12   ' Type 10 = Text, Type 12 = Memo
fieldname2.Name = "fieldname2"
fieldname2.Type = 12   ' Type 10 = Text, Type 12 = Memo
fieldname3.Name = "fieldname3"
fieldname3.Type = 12   ' Type 10 = Text, Type 12 = Memo
fieldname4.Name = "fieldname4"
fieldname4.Type = 12   ' Type 10 = Text, Type 12 = Memo
fieldname5.Name = "fieldname5"
fieldname5.Type = 12   ' Type 10 = Text, Type 12 = Memo
fieldname6.Name = "fieldname6"
fieldname6.Type = 12   ' Type 10 = Text, Type 12 = Memo
fieldname7.Name = "fieldname7"
fieldname7.Type = 12   ' Type 10 = Text, Type 12 = Memo
fieldname8.Name = "fieldname8"
fieldname8.Type = 12   ' Type 10 = Text, Type 12 = Memo
fieldname9.Name = "fieldname9"
fieldname9.Type = 12   ' Type 10 = Text, Type 12 = Memo

'Add the fieldinteger and fieldnameN fields to the Fields collection:
tdef.Fields.Append FieldInteger
tdef.Fields.Append fieldname0
tdef.Fields.Append fieldname1
tdef.Fields.Append fieldname2
tdef.Fields.Append fieldname3
tdef.Fields.Append fieldname4
tdef.Fields.Append fieldname5
tdef.Fields.Append fieldname6
tdef.Fields.Append fieldname7
tdef.Fields.Append fieldname8
tdef.Fields.Append fieldname9

'Define fieldinteger_index, the unique primary-key index:
uniqindex.Name = "fieldinteger_index"
uniqindex.Fields = "fieldinteger"
uniqindex.Unique = True
uniqindex.Primary = True
```

```
    'Append the fieldinteger_index index to the Indexes collection:
    tdef.Indexes.Append uniqindex

    'Append the tdef table definition (TableDef object) to the TableDefs
    'collection:
    db.TableDefs.Append tdef
    db.Close  ' Close and create the empty database.

    Set db = OpenDatabase("c:\TEMPZ.MDB")     ' Open the empty database.
    Set MyDS = db.CreateDynaset("Testtable")  ' Make dynaset from table.

    For i = 0 To 5  ' Add index field values for 5 new records:
       MyDS.AddNew
       MyDS!FieldInteger = i
       MyDS.Update
    Next
    MyDS.MoveFirst  ' Move to the first record.

    ' Add 32000 total bytes of string data to the fields in first record:
    For j = 0 To numfields
       MyDS.Edit  ' Opens current record for editing, into copy buffer.
       f$ = "fieldname" & j
       Debug.Print f$
       ' The total size of all Memo fields added together cannot exceed
       ' about 33000 bytes.
       ' Fields fieldname0 to fieldname9 are each assigned 3200 bytes:
       MyDS(f$) = String$(3200, "x")
       ' If you increase the string size to 3300, all fields write okay.
       ' But if you increase the string size to 3400, the program fails
       ' to write a string in the last field, the fieldname9 field. That
       ' demonstrates the maximum allowed size.
       MyDS.Update  ' Saves the copy buffer to the table.
    Next

    MyDS.Close
    db.Close
    MsgBox "done"
    End

End Sub
```

3. Start the program, or press the F5 key. After a few seconds, the program displays a message box saying "done."

4. Examine the new database C:\TEMPZ.MDB using Data Manager or Microsoft Access. You can run the Data Manager program from the Window menu in Visual Basic, or by using the Windows File Manager to run DATAMGR.EXE from the Visual Basic directory. To confirm that the number of characters in fieldname9 is 3200, copy it to the clipboard and paste it into a text editor. Close the database when finished; this will avoid a file-sharing conflict.

5. Change the String$(3200, "x") function in the above program so that it assigns 3400 characters to each of the ten Memo fields. Run the program again. Examine the new database using Data Manager or Microsoft Access. The last field, fieldname9, fails to receive any characters because the maximum record size was reached before 34000 characters.

Additional reference words: 3.00 limitation specification larger smaller bigger
KBCategory: APrg
KBSubcategory: APrgDataAcc

Hitchhiker's Guide to VBSQL -- VBSQL vs ODBC API Data Access

Article ID: Q111490

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY
=======

If you want to use ODBC API functions instead of VBSQL data object variables for data access in Visual Basic, consider the following issues:

- On average, the data access speed of the VBSQL API functions versus ODBC API functions is about equal. ODBC does some calls slightly faster, and VBSQL (DBLIB) does some other calls faster.

- The VBSQL API may be slightly easier to use than the ODBC API, but is not as generic as the ODBC API.

- If portability to C is important to you, the API calls for VBSQL are identical to the C calls -- except with C you need to bind each variable with each column, which can be tedious. The error handlers with VBSQL resemble the C language in that VBSQL implements real call-back handlers, but this is not an option with the ODBC API.

MORE INFORMATION
================

The VBSQL.VBX custom control file comes from the Microsoft SQL Server Programmer's Toolkit for Visual Basic.

The following guide offers further information on VBSQL:

"Hitchhiker's Guide to VBSQL: The developer's roadmap to the Visual Basic Library for SQL Server," by Bill Vaughn.

ISBN # 0-9640242-0-9
Available in London, England, at "The PC Bookshop Ltd."

In the rest of the world, purchase the book by sending $45 in U.S. funds (plus $3.84 tax if you live in Washington State) to:

Beta V
Book Order
16212 NE 113th Ct,
Redmond, WA 98052-2773
(206) 556-9205

C.O.D. orders to U.S. addresses are okay. Overnight C.O.D. orders are $63.50 in US funds. No credit card orders or purchase orders are accepted.

The Third Edition covers accessing the SQL Server (Microsoft and Sybase)

through VBSQL (DBLIB) and data access object variables in Visual Basic version 3.0. This is a definitive work on accessing the SQL Server from Visual Basic.

Open Database Connectivity (ODBC)

The greatest impact of Open Database Connectivity (ODBC) is on organizations where information is stored on a variety of dissimilar computers and databases. Under the ODBC scheme, a driver written for a specific database program, such as ORACLE, Ingres, or IBMs DB2, acts as the intermediary between the application and the database.

By using ODBC calls and the appropriate driver(s), the same application, be it a spreadsheet or an accounting package, can easily extract and manipulate information stored in a variety of databases.

ODBC-compliant applications include Microsoft Excel version 5.0 and Word version 6.0.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataODBC

How to Implement ToolTips Help in Visual Basic Applications

Article ID: Q111495

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

This article shows by example how to implement the toolbar "ToolTips" help in a Visual Basic application in a manner similar to the way "ToolTips" help system is implemented in Microsoft Word version 6.0 for Windows and Microsoft Excel version 5.0.

MORE INFORMATION

The following example implements a "ToolTips" help system by taking advantage of the GetCursorPos() and WindowFromPoint() Windows API functions. The GetCursorPos() function returns the current location of the mouse cursor and the WindowFromPoint() function returns the hWnd (window handle) of the control or form at a particular location on the screen.

Important Note: Modifications Required If Using Image Controls

IMPORTANT NOTE: Most toolbars in Visual Basic are made by using a container control (picture box) with individual image controls acting as buttons placed inside the container. The Image control, however, is a "light" control; it does not have an hWnd property. Because of this, the WindowFromPoint() Windows API function will not be able to detect which image control the cursor is over. It will give the hWnd of the image control's parent, in this case the container.

Therefore, if you are using image controls as your toolbar buttons (or some other control that doesn't have an hWnd property), you need to modify this program. Have it detect when the mouse cursor is over the parent container. Then calculate which image control the cursor is over by comparing the x and y coordinate returned by GetCursorPos, which is relative to the screen, to the coordinates of the image controls, which are relative to the parent container.

How to Use the GetCursorPos() and WindowFromPoint() Windows API functions

By using a timer, you can call the GetCursorPos() and WindowFromPoint() API functions to find out if the mouse cursor is located over a control for which you want to display a help window.

The help window is displayed by using an auto sizing label control on a separate form. By locating the form relative to the cursor, changing the label's caption, and resizing the form, you can display the ToolTips help.

Then you can use the ShowWindow() Windows API function to display the help form without giving it the focus. By using ShowWindow(), you can keep the main forms title bar from flashing when focus changes. In contrast, the Show method would cause the title bar to flash.

Step-by-Step Example
---

1. Start a new project in Visual Basic. Form1 is created by default.

2. Place the following controls on Form1:

   ```
   Timer (Timer1)
   Text Box (Text1)
   List Box (List1)
   Combo Box (Combo1)    'See NOTE below
   Check Box (Check1)
   Picture Box (Picture1)
   Option Button (Option1)
   Command Button (Command1)
   Vertical Scroll Bar (vScroll1)
   Horizontal Scroll Bar (hScroll1)
   ```

   NOTE: The combo box is a special case. The edit box portion of the combo box is a child window of the combo box, so it has a different window handle. Therefore, you must make call the GetWindow() Windows API function, with the GW_CHILD parameter, in order to get the hWnd of this edit box.

3. Add the following code to the Form_Load event of Form1:

   ```
   Sub Form_Load ()

      ' Set timer for 1 second, and enable it.
      Timer1.Interval = 1000
      Timer1.Enabled = True

   End Sub
   ```

4. Add the following code to the timer event of Timer1:

   ```
   Sub Timer1_Timer ()
      Dim curhWnd As Integer      'Current hWnd
      Dim p As POINTAPI
      Static LasthWnd As Integer  'Hold previous hWnd

      ' Make sure the program has the input focus:
      If GetActiveWindow() = Form1.hWnd Then
         ' Initialize point structure:
         Call GetCursorPos(p)
         ' Which window is the mouse cursor over?
         curhWnd = WindowFromPoint(p.y, p.x)

         ' Same as last window? If so, don't need to redraw:
         If curhWnd <> LasthWnd Then
            ' Store the current hWnd:
            LasthWnd = curhWnd
   ```

```
            ' Decrease timer interval to 5 ms (could choose 1 ms):
            Timer1.Interval = 5
            ' Which control is the cursor over?
            Select Case curhWnd
               Case Command1.hWnd
                  DisplayHelp "Command Button"
               Case Text1.hWnd
                  DisplayHelp "Text Box"
               Case List1.hWnd
                  DisplayHelp "List Box"
               Case Picture1.hWnd
                  DisplayHelp "Picture Box"
               Case Check1.hWnd
                  DisplayHelp "Check Box"
               Case Option1.hWnd
                  DisplayHelp "Option Box"
               Case Combo1.hWnd
                  DisplayHelp "Drop Down Combo Box"
               Case GetWindow(Combo1.hWnd, GW_CHILD) 'Edit box of combo box
                  DisplayHelp "Edit Box of Combo Box"
               Case hScroll1.hWnd
                  DisplayHelp "hScroll Bar"
               Case vScroll1.hWnd
                  DisplayHelp "vScroll Bar"
               Case frmHelp.hWnd
                  ' If it moves onto the help window, hide it:
                  frmHelp.Hide
               Case Else
                  ' Cursor is over the form or something else, so
                  ' change interval back to 1 sec delay:
                  DisplayHelp ""
                  Timer1.Interval = 1000
            End Select
         End If
      End If
   End Sub
```

5. Create a new module. From the File menu, choose New Module (ALT, F, M). Module1 is created by default.

6. Add the following declarations to the General Declarations section of Module1:

```
   Global Const SW_SHOWNOACTIVATE = 4
   Global Const GW_CHILD = 5         ' Needed for edit portion of combo box

   Type POINTAPI       ' Stores location of cursor
      x As Integer
      y As Integer
   End Type

   Declare Sub GetCursorPos Lib "User" (lpPoint As POINTAPI)
   Declare Function GetActiveWindow Lib "User" () As Integer
   ' Entr each of the following Declare statements on one, single line:
   Declare Function WindowFromPoint Lib "user" (ByVal lpPointY As Integer,
      ByVal lpPointX As Integer) As Integer
   Declare Function GetWindow Lib "User" (ByVal hWnd As Integer,
```

```
   ByVal wCmd As Integer) As Integer
Declare Function ShowWindow Lib "User" (ByVal hWnd As Integer,
   ByVal nCmdShow As Integer) As Integer
```

7. Add the following Sub procedure to Module1:

```
Sub DisplayHelp (Help$)
   Dim lpPoint As POINTAPI ' Cursor Point variable
   Dim ret As Integer      ' Return value of ShowWindow() API function

   Rem Display Help String
   Rem
   Rem This Function displays the Help$ if Help$ <> "".
   Rem if Help$ = "" then the Help String is removed.
   Rem
   Rem FUNCTION REQUIREMENTS:
   Rem     GetCursorPos()    Windows API function
   Rem     frmHelp           Name of the Help form
   Rem

   If Len(Help$) <> 0 Then  ' Double check help$

      ' Make sure help form is invisible:
      frmHelp.Hide

      ' Change caption of label:
      frmHelp.Label1.Caption = Help$

      ' Get the cursor position so you can calculate where to place the
      ' help form:
      Call GetCursorPos(lpPoint)

      ' Offset the form from the cursor by 18 and 2 pixels (values
      ' chosen to simulate the look of Microsoft Word version 6.0)
      frmHelp.Top = (lpPoint.y + 18) * Screen.TwipsPerPixelY
      frmHelp.Left = (lpPoint.x - 2) * Screen.TwipsPerPixelY

      ' Adjust width of form to label + 4  because 2 are needed for each
      ' pixel of the border and 2 are needed to center the label (the
      ' label is inset by 1 pixel on the form). Also, adjust height of
      ' form to height of label + 2 because 2 ar needed for each pixel
      ' of the border:
      frmHelp.Width = frmHelp.Label1.Width + (4 * Screen.TwipsPerPixelX)
      frmHelp.Height = frmHelp.Label1.Height + 2 * Screen.TwipsPerPixelY

      ' Make sure form is on top:
      frmHelp.ZOrder

      ' Show form without the focus:
      ret = ShowWindow(frmHelp.hWnd, SW_SHOWNOACTIVATE)
   Else
      ' Hide the form:
      frmHelp.Hide
   End If
End Sub
```

8. Create another form by choosing New Form from the File menu (ALT, F, F).

Form2 is created by default. Change the following properties of Form2:

```
Name = frmHelp
BorderStyle = 1 - Fixed Single
ControlBox = False
MinButton = False
MaxButton = False
```

9. Add a label control (Label1) to the frmHelp form and set AutoSize=True.

10. Add the following code the to Form_Load event of the frmHelp form:

```
Sub Form_Load ()

   ' Get rid of the forms Caption so title bar does not display:
   Me.Caption = ""

   ' Give the form and label a light yellow background:
   Me.BackColor = &H80FFFF
   Label1.BackColor = &H80FFFF

   ' Inset label by 1 pixel:
   Label1.Left = 1 * Screen.TwipsPerPixelX
   Label1.Top = 0

End Sub
```

11. Start the program by pressing the F5 key. Then place the cursor over one of the controls to display the help window.

Additional reference words: 3.00
KBCategory: APrg
KBSubcategory: APrgDataOther

How to Read Database Fields Into and Out of a List Box

Article ID: Q112195

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The list box that comes with Visual Basic is not bound, but you can simulate a bound list box in a Visual Basic program. Visual Basic can read records from a database placing the values from each individual field within the record into columns in a list box, which can then be extracted by the Visual Basic program.

MORE INFORMATION

By reading each field into the list box and separating each field from the next with a TAB character, you can create the illusion of columns.

NOTE: By using the SendMessage Windows API function and the LB_SETTABSTOPS constant, you can set the size of your tab stops within your listbox to create custom spacing between fields.

Here's an example:

```
  List1.AddItem Data1.Recordset(Field1) & Chr$(9) & Data1.Recordset(Field2)
```

This makes two columns in the list box. Field1 is separated from Field2 by the TAB character. You can use the TAB character to parse the columns back into separate fields. For example:

```
   Dim X As Integer
   X = InStr(List1.Text, Chr$(9))
   Text1 = Mid$(List1.Text, 1, X - 1) ' Will Contain Field1
   Text2 = Mid$(List1.Text, X + 1, (Len(List1.Text) - X)) ' Contains Field2
```

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Place two Text Boxes (Text1 and Text2), a List Box (List1), and a data control (Data1) on Form1.

3. Set the following properties of the Data Control:

Property	Setting	Comment
DatabaseName	BIBLIO.MDB	The sample db in the Visual Basic directory
Recordsource	Authors	The Authors table is in BIBLIO.MDB

4. Add the following code to the Form_Load event:

```
Sub Form_Load()
   Data1.Refresh
   ' Loop until you reach the last record:
   Do Until Data1.Recordset.EOF
      ' Load the list box with fields separated with a tab:
      ' Enter the following two lines as one, single line:
      List1.AddItem Data1.Recordset("Au_Id") & Chr$(9) &
         Data1.Recordset("Author")
      Data1.Recordset.MoveNext
   Loop
   ' Initialize list box and text boxes to first item:
   List1.ListIndex = 0
End Sub
```

5. Add the following code to the Click event of List1:

```
Sub List1_Click()
   Dim X As Integer
   ' Find first tab character:
   X = InStr(List1.Text, Chr$(9))
   ' Put all characters before tab into Text1:
   Text1 = Mid$(List1.Text, 1, X - 1)
   ' Put all characters after tab into Text2:
   Text2 = Mid$(List1.Text, X + 1, (Len(List1.Text) - X))
End Sub
```

6. Press the F5 key to run the program. Select an item in the List Box. The Author ID should be in Text1 and Author name should be in Text2.

Additional reference words: 3.00
KBCategory:
KBSubcategory: APrgDataAcc

PRB: Can't find Installable ISAM When Run Two DB Apps in VB

Article ID: Q112652

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

The following error message occurs under the conditions listed below:

   Can't Find Installable ISAM

A combination of the following conditions result in the error:

 - A Visual Basic application (.EXE) is executed that uses an IISAM driver.
 - The <appname>.INI file associated with the Visual Basic application includes a reference to only that IISAM driver.
 - A second Visual Basic application (.EXE) is executed that requires a different IISAM driver.

No matter what entries are in the second application's <appname>.INI file, the "Can't Find Installable ISAM" error occurs.

CAUSE

This behavior is by design in the Microsoft Access engine included in Visual Basic version 3.0 for Windows. The Microsoft Access engine is not currently designed to look for the new .INI file required by the second application. Instead, it is designed to use the established references that the first .INI file provided when the first Visual Basic application was executed. This design is under review for possible revision in future product versions.

WORKAROUND

To work around this limitation, place the entire Installable ISAMs section of the VB.INI file into the <appname>.INI file of each Visual Basic application that executes an IISAM driver.

The section in the <appname>.INI should look like this:

```
[Installable ISAMs]
Btrieve=C:\WINDOWS\SYSTEM\btrv110.dll
FoxPro 2.0=C:\WINDOWS\SYSTEM\xbs110.dll
FoxPro 2.5=C:\WINDOWS\SYSTEM\xbs110.dll
dBASE III=C:\WINDOWS\SYSTEM\xbs110.dll
dBASE IV=C:\WINDOWS\SYSTEM\xbs110.dll
Paradox 3.X=C:\WINDOWS\SYSTEM\pdx110.dll
```

MORE INFORMATION
==========

Steps to Reproduce Behavior

TestApp1
---

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a Text Box (Text1) and a Data control (Data1) to the form.

3. Set the Connect and DatabaseName properties of Data1 to connect to
   a Paradox 3.x database.

4. Make an .EXE file, and call it TESTAPP1.EXE

5. Using Notepad, create a text file called TESTAPP1.INI and in that
   file place the following:

   ```
   [Installable ISAMs]
   Paradox 3.X=C:\WINDOWS\SYSTEM\PDX110.DLL
   ```

6. Place the file TESTAPP1.INI in your \WINDOWS subdirectory.

TestApp2
---

7. Start a new project in Visual Basic. Form1 is created by default.

8. Add a Text Box (Text1) and a Data control (Data1) to the form.

9. Set the Connect and DatabaseName properties of Data1 to connect to
   a Btrieve database.

10. Make an .EXE file, and call it TESTAPP2.EXE

11. Using Notepad, create a text file called TESTAPP2.INI and in that
    file place the following:

    ```
    [Installable ISAMs]
    Btrieve=C:\WINDOWS\SYSTEM\BTRV110.DLL
    ```

12. Place the file TESTAPP2.INI in your \WINDOWS subdirectory

13. Run TESTAPP1. It should run with no problem.

14. Next Run TESTAPP2 while TESTAPP1 is still running. You will receive
    the following error:

       Can't Find Installable ISAM

Additional reference words: 3.00
KBCategory: APrg
KBSubCategory: APrgDataIISAM

How to Perform Microsoft Access Macro Action Via DDE from VB

Article ID: Q112767

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

From Visual Basic, you cannot directly use a form or report that was created by the Microsoft Access engine. This article shows by example how to use DDE to do it indirectly. The example prints one of the built-in reports from the NWIND.MDB sample database by using DDE and the OpenReport macro action in Microsoft Access.

MORE INFORMATION

For more information about Microsoft Access macro actions, please see the Microsoft Access documentation or Help menu. A Visual Basic application can call most of these actions by using DDE.

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a text box (Text1) and Command button (Command1) to Form1.

3. Place the following code in the Command1 button's click event:

```
' Note the time-out has to be long enough to allow for the print
' to complete or an error will occur.
Sub Command1_Click ()
   Text1.LinkTimeout = 600 'Set DDE Time-out for 60 Seconds
   Text1.LinkTopic = "MSACCESS|SYSTEM"
   Text1.LinkMode = 2  ' Establish manual DDE link to Microsoft Access.
   Text1.LinkExecute "[OPENREPORT Catalog]" 'Open and Print     Report
   Text1.LinkMode = 0  ' Terminate the DDE link to Microsoft Access
End Sub
```

4. Start Microsoft Access and open the NWIND.MDB sample database.

5. Run the Visual Basic program, and click the Command1 button.

Additional reference words: 3.00
KBCategory:
KBSubcategory: IAPDDE APrgDataAcc

Clicking Toolbox/Color Palette Menu Doesn't Leave Menu Open

Article ID: Q73418

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Windows, versions 3.0 and 3.1

SUMMARY

When you single click in the System menu (the Control menu in the upper left corner) of the Toolbox or the Color Palette in the Visual Basic environment, the menu flashes on the screen but does not stay visible like normal Windows System menus.

This behavior only occurs when running under Microsoft Windows, version 3.0. When running under Microsoft Windows, version 3.1, the menus visible as you would expect.

To keep the Toolbox or Color Palette system menu pulled down, when running under Microsoft Windows, version 3.0, you must hold down the mouse button.

MORE INFORMATION

On a normal Windows System menu, a single mouse click pulls down the menu and keeps the menu down without having to hold down the mouse button. Even though the System menus of the Toolbox and Color Palette may look like normal System menus, they do not act like them, and are not "normal" Windows System menus. They were not designed to stay open with a single mouse click; therefore, the menu will flash briefly on the screen on a single mouse click and then disappear, unless you keep the mouse button held down.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtDes

No Edit Menu Access for Property Entry; Use Edit Shortcut Keys

Article ID: Q73800

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Properties Bar entry window was not designed to handle the Edit menu commands (such as Cut, Copy, Paste, and Undo) when chosen with the mouse. The Edit menu commands affect the controls on the form, not the Properties Bar entry window (even if you first select text in the Properties Bar entry window before choosing the Edit menu with the mouse).

Instead of choosing Edit commands with the mouse, you can select the desired text in the Properties Bar entry window and use the Edit shortcut keys, as follows:

Command	Shortcut Keys
Cut	SHIFT+DELETE
Copy	CTRL+INSERT
Paste	SHIFT+INSERT
Undo	ALT+BACKSPACE

(NUM LOCK needs to be off if you select the DELETE or INSERT key from the numeric keypad.)

MORE INFORMATION

Example: Edit Menu Cannot Affect Properties Bar Entry Window

1. Start Visual Basic.

2. From the File menu, choose New Project.

3. Double-click a label box from the Toolbox (symbolized by a capital A in script). This should display a label box on the form.

4. With the mouse, select the text fragment "Lab" from the "Label1" Caption in the Properties Bar entry window, and choose Copy from the Edit menu.

   Note: While you may have thought you just copied "Lab" into the Clipboard, you actually copied the entire Label1 control (from

the form) into the Clipboard. Clicking the Edit menu anywhere outside the Properties Bar entry window causes the focus to revert back to the Label1 control on the form.

5. Click the "Form1" text appearing in the Properties Bar entry window to set the focus there.

6. From the Edit menu, choose Paste (again, clicking the Edit menu or anywhere outside the Properties Bar entry window causes the focus to revert back to the Label1 control on the form.) This causes the following message box to appear:

   "You already have a control named 'Label1'. Do you want to create a control array?".

   Select either the Yes or No command button. Notice that another copy of the Label1 box will appear in the upper left corner of the form.

Instead of choosing Edit commands with the mouse, you can select the desired text in the Properties Bar entry window and use the Edit shortcut keys. For example, you can select text in the Properties Bar entry window, then press CTRL+INSERT (while NUM LOCK is off) to copy text to the Clipboard. You can press SHIFT+INSERT to paste Clipboard text into the Properties Bar entry window. You can press ALT+BACKSPACE to Undo a Cut, Copy, or Paste.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtDes

Deleting VB Control Moves Associated Code to Object: (General)

Article ID: Q73808

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

When you delete a control in the Visual Basic environment, the code that you wrote for that control is not deleted, but is instead moved to the Object: (General) area for code. Some programmers may not realize that these detached (unused) procedures still exist and consume memory. The detached code is available for calling, copying, or reuse. For example, when you add a control that shares the same name as a detached procedure, the detached event procedure reassociates with that control.

If you want to delete both the control and its associated code, you need to manually select and delete the code in each event procedure for that control in addition to deleting the control itself.

This behavior is by design in Microsoft Visual Basic programming system for Windows, versions 1.0 and 2.0.

MORE INFORMATION

The following example demonstrates that your code goes into the Object: (General) area after you delete the associated control (or object):

1. From the File menu, choose New Project.

2. Double-click a command button from the Toolbox. This puts a Command1 button on your form.

3. Double-click the Command1 button. This brings up the code window for the Command1_Click event procedure.

4. Enter the following code inside the Command1_Click procedure:

```
Sub Command1_Click()
 Print "Hello"          ' Enter this statement.
End Sub
```

5. Press F5 to run your code. Click the Command1 button to see the text "Hello" display on Form1 in the upper left corner. From the Run menu, choose End to stop the program.

6. Click the Command1 button on Form1 to set the focus there and either press DELETE or choose the Delete command from the Edit menu.

This deletes the Command1 button from the form.

At this point, some programmers may incorrectly assume that the code associated with the Command1 button was also deleted. Actually, the code associated with any deleted object is automatically moved into the Object: (general) area of the Code window for that form. You can find the detached procedures in the Code window by choosing (general) from the Object: box, and choosing the procedures from the Proc: box. (Click the DOWN arrow symbol on the right of the Object: and Proc: boxes to see your choices.)

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtDes

PRB: VB Help Misleading Error: Unable to Find Windows Help.EXE

Article ID: Q76549

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

Starting Help when Windows has low system resources may result in a misleading error such as:

   Unable to find Windows Help.EXE

CAUSE

Windows displays this error if it has less than 5 percent of free system resources.

RESOLUTION

This problem is not destructive in any way. To regain access to the Visual Basic Help system, you must first close Windows applications until you have more than 5 percent of free system resources.

MORE INFORMATION

Steps to Reproduce Problem

1. Start Visual Basic.

2. Check Windows free resources (choose About from the Windows Help menu). If free resources are less than 5 percent, proceed to step 4.

3. Start another Windows application. Go to step 2.

4. From the Visual Basic online Help, choose the Index button.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtDes

Using PAGE DOWN and PAGE UP Keys Within VB.EXE Environment

Article ID: Q76559

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

At design time in the Code window of the Visual Basic programming environment (VB.EXE), you can use the PAGE DOWN and PAGE UP keys as shortcut keys to go from one event procedure to another. Other helpful shortcut keys are listed below.

MORE INFORMATION

Below is an example that demonstrates certain conditions you need to follow before you can use the PAGE DOWN and PAGE UP shortcut keys to their fullest potential in the Code window:

1. Start Visual Basic with a New Project.

2. Place some command buttons or any other objects on the form.

3. Double-click the form to bring up the Code window.

4. Place code within the various event procedures. For example, place some various Print statements in the following event procedures:

   - Form_Click
   - Command1_Click
   - Label1_Click
   - Form_Load

   (Note that you will need to have a command button and a label placed on Form1 before adding Print statements in the event procedures mentioned above.)

5. Press the PAGE DOWN or PAGE UP key and notice how the VB.EXE environment moves from one event procedure to another. The PAGE DOWN and PAGE UP work in a such a way that you are looking at the event procedures in an alphabetic order, except that the "(general)" event procedure is always on the top of the list even if it contains no code.

6. Notice that only the event procedures that contain code are displayed. Also note that PAGE DOWN and PAGE UP wrap around continuously. To activate PAGE UP and PAGE DOWN, the focus (the I-beam mouse pointer) must be in the Code window. When you have pages

of code within an event procedure, there are times when the PAGE DOWN and PAGE UP seem to perform differently, but you need to visualize paging up or down a listing of event procedures in a printout to see how these routines are designed to work.

Other Shortcut Keys in VB.EXE

The F1 function key invokes Visual Basic Help. When you receive an error after pressing the F5 key to run your code, you can press the F1 key to get additional information on that error.

F5 is the shortcut key to run a program.

F7 activates the Code window.

ALT+PRINT SCREEN is a Windows feature to copy the active window into the Clipboard. PRINT SCREEN copies the entire screen into the Clipboard. CTRL+INSERT copies selected text into the Clipboard.

F8 single-steps through a program in the VB.EXE environment.

F9 toggles breakpoints on and off.

F12 chooses the Save Project As command from the File menu.

For additional shortcut keys, search for "shortcuts" under Help in the VB.EXE environment, and search for "system keys" in the Windows Help.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtDes

CTRL+HOME Commits Current Line to VB Syntax Checking/Parsing

Article ID: Q76561

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

In the Visual Basic for Windows Code window, edit keys that move the cursor from a line will commit that line to syntax checking (and code parsing) by the Visual Basic for Windows editor. This is not a problem with Visual Basic for Windows, but is by design.

Note that you can turn off syntax checking by choosing the Syntax Checking command from the Code menu.

MORE INFORMATION

Visual Basic for Windows checks each line of code as it is entered for syntax and usage. This is a feature of the Visual Basic for Windows editing environment that prevents entry errors. Syntax checking is done before performing any edit function in which the cursor will leave the current line. Thus, any edit key combination that moves the cursor from that line will initiate the checking process.

Steps to Reproduce Problem

1. Start Visual Basic.

2. Open the Global module.

3. Type "This is a test" and press CTRL+HOME.

An error message of "Expected: Statement" will be displayed. You may not expect the error to occur because the cursor has not yet left the line of code with the error; however, CTRL+HOME normally moves the cursor off of the line, and therefore the line is checked before doing the edit operation, resulting in the error message.

Similar behavior results when using other edit keys that move the cursor from the current line, such as ENTER, PAGE UP, PAGE DOWN, CTRL+END, UP ARROW, DOWN ARROW, and so on.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtDes

VB Forms with Menus Cannot Have Fixed Double BorderStyle

Article ID: Q76630

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Windows versions 3.0 and 3.1

SUMMARY

Because of Windows version 3.0 and 3.1 limitations, forms with menus cannot have the BorderStyle property set to Fixed Double. To have menus, a form's BorderStyle property must be either None, Fixed Single, or Sizable.

MORE INFORMATION

Steps to Reproduce Problem

1. Run Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. In the Menu Design window, create a menu on Form1.

3. Set the BorderStyle of Form1 to Fixed Double.

4. Run the program.

Note that the border style is fixed single.

Because of a Windows problem with menus on forms with fixed double borders, Visual Basic does not paint the menus correctly. For this reason, Visual Basic does not allow this particular combination of a menu on a form with a fixed double border.

For more information on this limitation, query on the following words in the Microsoft Knowledge Base:

   visual basic and menu and caption and bar

Additional reference words: 1.00 3.00 3.10
KBCategory:
KBSubcategory: PrgCtrlsStd EnvtDes

PRB: Invalid in Immediate Window Error When Creating Variable

Article ID: Q76636

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

SYMPTOMS

One of the following error messages may occur when you attempt to create a new variable in the VB.EXE Immediate window:

Invalid in Immediate Window

-or-

Invalid in Debug Window

CAUSE

This error message may occur if your program has encountered a serious error (for example, "Out of Stack Space") from which the program cannot recover to continue. The current program must be able to continue for variables to be created in the Immediate window.

RESOLUTION

Exit Visual Basic.

MORE INFORMATION

Steps to Reproduce Problem

1. Start Visual Basic, or choose New Project from the File menu if Visual Basic is already running.

2. Double-click Form1 to open a code window. In the Form_Click event procedure, enter the following code:

   ```
   Call Form_Click
   ```

3. Execute the program and click Form1. An "Out of Stack Space" error will be displayed.

4. Close the error message window and enter the following code in the Immediate window:

   ```
   A$ = "123"
   ```

5. At this point, you will receive one of the error messages listed above.

If not, repeat steps 3 and 4.

Additional reference words: 1.00 2.00 3.00 errmsg
KBCategory:
KBSubcategory: EnvtDes

PRB: ToolBox/Color Palette Menus Lose Focus After Single ESC

Article ID: Q76984

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

The ToolBox and Color Palette system menus lose their focus after only one press of the ESC key rather than two. Other system menus lose focus after two presses of ESC.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Reproduce Behavior

1. Start Visual Basic.

2. Click the ToolBox or the Color Palette.

3. Press ALT+MINUS to open its system menu.

4. Press ESC to close its system menu.

5. Press SPACEBAR to drop the system menu again.

Pressing SPACEBAR does not drop the system menu as it would in other Windows system menus. The focus on these two particular system menus is lost with one press of ESC. This is not a problem, but a design feature of the Visual Basic ToolBox and Color Palette windows.

This feature is unlike other Windows system menus in which two ESC key presses are required to remove the focus. (The first ESC closes the system menu, but the focus remains on it. The second ESC returns the focus to the original window.) The ToolBox and Color Palette system menus are not regular Windows system menus, and function differently.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtDes

PRB: Compatibility Problems with Adobe Type Manager

Article ID: Q77645

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

The following problems may arise when using the Adobe Type Manager (ATM) with Visual Basic:

- FontName list is incorrect and/or contains duplicate names
- Unrecoverable Application Errors (UAEs) in Windows version 3.0 or General Protection (GP) faults in Windows version 3.1.
- Incorrect screen font displayed when using ATM fonts

STATUS

Adobe Type Manager, is manufactured by vendors independent of Microsoft; we make no warranty, implied or otherwise, regarding this product's performance or reliability.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: EnvtDes

Restart in VB Break Mode if Delete Blank Line Above End Sub

Article ID: Q78074

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Deleting a blank line above the End Sub/End Function or below the Sub/Function statement will generate the message

```
You will have to restart your program
after this edit--proceed anyway?
```

while in break mode in the VB.EXE environment. This behavior is by design.

This information applies to Microsoft Visual Basic programming system 1.0 for Windows.

MORE INFORMATION

Deleting the line following the Sub or Function statement requires you to restart when in break mode. This also occurs when deleting the line preceding the End Sub or End Function statement of any procedure. The Visual Basic edit manager treats both of these deletions as modifications to the first or last lines, both of which require a restart when in break mode.

The following steps will force a restart in a program while in break mode.

1. In a new project, choose Start from the Run Menu.

2. Press CTRL+BREAK to suspend execution of the application and enter break mode.

3. Press F7, or from the Code menu, choose View Code to bring up the code window.

4. The text cursor should be on the blank line between the following procedure statements:

   ```
   Sub Form_Click ()

   End Sub
   ```

5. Press DEL to delete the blank line between the Sub Form_Click() and End Sub lines.

The following message will be displayed:

```
You will have to restart your program
after this edit--proceed anyway?
```

The above message is also displayed when the cursor is on the second line and you press the BACKSPACE key once, or if the cursor is at the beginning of the last line of a procedure (at the beginning of the End Sub line) and you press the BACKSPACE key once.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtDes

PRB: Printer Error When Printing VB Form to Text-Only Printer

Article ID: Q78075

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

The message "Printer Error" displays when you print a form from Visual Basic to a text-only printer. The text-only printer does not have the graphics capability to print the Visual Basic form. Windows traps the printer error and displays the "Printer Error" dialog box.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Reproduce Problem

1. From the Windows Control Panel, choose the Printers icon, and select Generic Text / Text Only as the default printer. (You may need to install the Generic / Text Only printer from the Control Panel to make this option available.)

2. Start Visual Basic.

3. From the File menu, choose Print. The current form and code are selected by default in the print dialog box.

4. Choose the OK button to print. Windows displays the "Printer Error" dialog box.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtDes

PRB: Printing with HPPCL5A.DRV to HP LaserJet III Cuts Line

Article ID: Q78079

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

Choosing Print from the Visual Basic File menu to print source code truncates one line of code per page of output when printing to a Hewlett-Packard (HP) LaserJet series III printer using the HPPCL5A.DRV printer driver.

CAUSE

This is a problem with the Hewlett-Packard LaserJet series III printer driver version 3.42 for Windows.

STATUS

Microsoft has confirmed this to be a problem with the HPPCL5A printer driver version 3.42. This problem was corrected by the HP III driver version 30.3.85 included with Microsoft Word for Windows version 2.0.

Additional reference words: 1.00 2.00 HP laser jet truncate lose
KBCategory:
KBSubcategory: EnvtDes

High Granularity Setting Affects Windows/VB Form Resizing

Article ID: Q79386

The information in this article applies to:

- All programs written for Microsoft Windows version 3.0 and version 3.0a, including programs written with Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

If you set the granularity of Windows' invisible sizing grid to a value greater than zero, you may notice that form resizing is no longer smooth. This behavior is correct and can be changed by setting the granularity back to zero.

MORE INFORMATION

The Windows Control Panel program group contains several icons that allow you to customize the Windows environment. The Desktop program allows you to specify the Windows granularity setting. If you notice that Visual Basic forms do not smoothly resize, but instead resize in "chunks," the problem may be caused by a granularity setting that is too high.

To change the Windows granularity setting, do the following:

1. Open the Windows Main program group.

2. Double-click the Control Panel icon.

3. Double-click the Desktop icon.

4. Move to the granularity text box in the lower right portion of the dialog box.

5. Click the up or down scroll arrow to the right of the text box to increase or decrease the size of the grid. Or, type the number you want in the text box.

   Note: The allowable range of values is 0-49 inclusive. Setting the granularity to zero will produce the smoothest form resizing.

6. Choose OK.

Reference(s):

"Microsoft Windows 3.0 Graphical Environment: Users Guide," version 3.0, page 157

Additional reference words: 1.00 3.00 3.00a W_Win3
KBCategory:

KBSubcategory: EnvtDes

Helv and Tms Rmn FontNames Not Available in Windows 3.1

Article ID: Q84470

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Helv and Tms Rmn are no longer listed in the Settings box of the Properties bar as selections for the FontName property in Visual Basic under Windows 3.1. By default, Windows 3.1 maps the fonts Helv and Tms Rmn to MS Sans Serif and MS Serif, respectively.

MORE INFORMATION

Visual Basic uses Helv as the default setting for the FontName property of forms and controls. That font is still the default setting for FontName in Visual Basic, even though it is no longer a system font in Windows 3.1. Helv and Tms Rmn are no longer listed in the drop-down list box in the Properties bar.

In Windows 3.1, Helv is mapped to MS Sans Serif and Tms Rmn is mapped to MS Serif by default. This appears in the [FontSubstitutes] section of the WIN.INI file. Therefore, FontName can still be set to Helv or Tms Rmn at run-time without producing any errors, even though they are no longer available. Windows will instead use the existing fonts to which they are mapped.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtDes

VB Uses Bitmap Fonts when TrueType FontSize Less Than 7 Points

Article ID: Q84483

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

The Microsoft Windows version 3.1 operating environment provides you with TrueType scalable fonts that can be used in Microsoft Visual Basic for Windows applications. Visual Basic for Windows supports TrueType fonts for font sizes of 7 points or greater -- depending on the video driver installed. Smaller fonts are mapped to available bitmap fonts, based on the fonts available for the video driver installed.

CAUSE

This is not a problem with Visual Basic for Windows. This is how Windows manages fonts. This is expected behavior in Windows when using TrueType fonts that are less than 7 points in size.

STATUS

There is no way to force Visual Basic for Windows to use TrueType fonts for font sizes less than 7 points. This is by design.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

Microsoft Windows 3.1 utilizes automatic bitmap font substitution, which is done to preserve readability at small sizes when they are displayed. At very small point sizes (4 to 7 points on standard VGA video resolutions), most Type 2 fonts are substituted with a hand-tuned bitmap font to preserve readability. This can cause the style of the font to change. For example, the Times New Roman font shipped with Windows version 3.1 appears as the Small Fonts font for sizes 4 - 6 and MS Serif for sizes 6.25 - 8.25, rather than its native face it has at larger sizes.

The program below demonstrates this scenario. The program attempts to print a message using the Arial font in sizes from 1 to 9. Visual Basic for Windows uses the font Small Fonts for font sizes less than 7 and depending on the video driver installed may use Arial for sizes between 7 and 8.25. Using the standard VGA driver, Arial is used for fonts sizes greater then 8.25.

Steps to Demonstrate This Behavior
---

1. Start Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Enter the following code into the Form_Click procedure:

```
Sub Form_Click ()
   For i = 1 To 9 Step .25
      FontName = "Arial"
      FontSize = i
      Print Str$(i); Chr$(9); Str$(FontSize); Chr$(9); FontName
   Next i
End Sub
```

3. Press the F5 key to run the program, and click the form. Notice that the Arial TrueType font is used only for font sizes of 8.25 or larger.

Additional reference words: 1.00 2.00 3.00 3.10
KBCategory:
KBSubcategory: EnvtDes

VB for Windows Trappable Errors List of Changes/Additions

Article ID: Q93711

The information in this article applies to:

- The Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0

SUMMARY

This article lists error codes, messages, and explanations of the errors that you can trap at run time using the On Error statement and the Err function in Visual Basic for Windows, version 2.0.

The first list below documents the trappable errors for Visual Basic for Windows, version 1.0 that are no longer trappable errors in Visual Basic for Windows, version 2.0.

The second list below documents the new trappable errors in Visual Basic for Windows, version 2.0 that are not listed as trappable errors in Visual Basic for Windows, version 1.0.

This article is based on the online help documentation of the trappable error listing in Visual Basic for Windows, version 2.0. For more information on these errors look in the online help "trappable error" listing and select the error you would like more information on.

MORE INFORMATION

Listed below are trappable errors in Visual Basic for Windows, version 1.0 that are not trappable errors in Visual Basic for Windows, version 2.0:

```
260   No timer available
280   DDE channel not fully closed; awaiting response from foreign
      application
293   DDE method invoked with no channel open
296   PasteLink already performed on this control
297   Can't set LinkMode; invalid LinkTopic
340   Control array element 'item' does not exist
345   Reached limit: cannot create any more controls on this form
381   Invalid property array index
423   Property or control 'item' not found
430   No currently active control
431   No currently active form
520   Can't empty Clipboard
521   Can't open Clipboard
```

Listed below are the trappable errors in Visual Basic for Windows, version 2.0 that are not trappable errors in Visual Basic for Windows, version 1.0:

```
91    Object variable not set
92    For loop not initialized
```

93 Invalid pattern string
94 Invalid use of Null
95 Cannot destroy active form instance
298 DDE requires DDEML.DLL
387 'item' property can't be set on this control
391 Name not available
392 MDI Child forms cannot be hidden
393 'item' property cannot be read at run time
394 'item' property is write-only
403 MDI forms cannot be shown modally
404 MDI child forms cannot be shown modally
426 Only one MDI form allowed
600 Set value not allowed on collections
601 Get value not allowed on collections
602 General ODBC error: '<error>'
603 ODBC - SQLAllocEnv failure
604 ODBC - SQLAllocConnect failure
605 OpenDatabase - invalid connect string
606 ODBC - SQLConnect failure '<error>'
607 Visual Basic 2.0 attempted on unopened database
608 ODBC - SQLFreeConnect error
609 ODBC - GetDriverFunctions failure
610 ODBC - SQLAllocStmt failure
611 ODBC - SQLTables (TableDefs.Refresh) failure: '<error>'
612 ODBC - SQLBindCol failure
613 ODBC - SQLFetch failure: '<error>'
614 ODBC - SQLColumns (Fields.Refresh) failure: '<error>'
615 ODBC - SQLStatistics (Indexes.Refresh) failure: '<error>'
616 Table exists - append not allowed
617 No fields defined - cannot append table
618 ODBC - SQLNumResultCols (Create Dynaset) failure: '<error>'
619 ODBC - SQLDescribeCol (Create Dynaset) failure '<error>'
621 Row-returning SQL is illegal in ExecuteSQL method
622 CommitTrans/Rollback illegal - Transactions not supported
623 Name not found in this collection
624 Unable to build data type table
625 Data type of field '<field name>' not supported by target database
626 Attempt to Move past EOF
627 Dynaset is not updatable or Edit method has not been invoked
628 <method>: Dynaset method illegal - no scrollable cursor support
629 Warning: <operation> (ODBC - SQLSetConnectOption failure)
630 Property is read-only
631 Zero rows affected by Update method
632 Update illegal without previous Edit or AddNew method
633 Append illegal - field is part of a TableDefs collection
634 Property value only valid when Field is part of a Dynaset
635 Cannot set the property of an object which is part of a Database object
636 Set field value illegal without previous Edit or AddNew method
637 Append illegal - Index is part of a TableDefs collection
638 Visual Basic 2.0 attempted on unopened Dynaset
639 Field type is illegal
640 Field size illegal for specified field type
641 '<item>' illegal - no current record
642 Reserved parameter must be FALSE
643 Property not found

644 ODBC - SQLConfigDataSource error '<error>'
645 ODBC driver does not support exclusive access to Dynasets
646 GetChunk: Offset/Size argument combination illegal
647 Delete method requires a name argument
648 ODBC objects require VBODBCA.DLL
708 File not found: <file name>
710 File already open: <file name>
712 Device I/O error: <device>
713 File already exists: <file name>
716 Disk full: <drive>
719 Bad file name: <file name>
722 Too many files: <file name>
725 Permission denied: <file name>
730 Path access error: <path name>
731 Path not found: <path name>
732 Must have startup form or Sub Main ()
735 Can't save file to TEMP
740 Invalid procedure name
742 Not enough room to paste contents into current line
743 Can't set next statement
744 Search text not found
745 Text would be too long. Edit prevented
746 One or more replacements too long and not made
747 Syntax errors produced while replacing
748 An undo error has occurred. Further undo is unavailable
749 Watch expression too long, expression truncated
750 An expression must be selected
752 Error reading Tutorial file 'item'
753 Tutorial directory 'item' not found
754 Can't find file 'item'
755 Not enough memory to load help file.
756 Duplicate procedure name
757 Can't find Windows Help .EXE file
758 Control must be same type as rest of array
759 Array already has a control at index 'item'
760 Not a legal object name: 'item'
761 Must specify which item(s) to print
762 Can't clear Index property without changing name
764 <name> is a control name
765 Controls without the Align property cannot be placed directly on the MDI form
766 Event handler must be a Sub procedure
768 Event procedure argument has incorrect data type
769 Menu subitem skipped a level
770 Parent or top-level menu item may not be checked
771 Can't assign shortcut key to menu name
772 Can't use separator bar as menu name
773 Menu control must have a name
774 Menu control array element must have an index
775 Menu control array indices must be in ascending order
776 Menu control array elements must be contiguous and within the same submenu
777 Shortcut key already assigned
778 Separator bar may not be checked or disabled, or have an shortcut key
779 At least one submenu item must be visible
780 Valid range: [1...32]

781 Valid range: [24...1188]
783 Separator may not be the window List menu
784 Can't have more than one window List menu
785 New not allowed on this type
20000 Can't load Custom Control DLL: 'item'
20001 Can't unload Custom Control DLL; in use
20002 Can't quit at this time
20003 You'll have to restart your program after this edit--proceed anyway?
20004 'item' has been changed; must reset
20005 Reset halted programs so Code window can be closed?
20006 Not enough stack space to enter break mode--continue?
20007 Not enough stack space to enter break mode for error--must reset
20008 Line too long to edit--edit truncated line?
20009 Search complete
20011 Invalid command-line argument 'item'
20012 Save changes to 'item'?
20013 You already have a control named 'item'. Do you want to create a control array?
20014 Error loading 'item'. The code associated with this form could not be loaded. Continue loading form?
20015 Error loading 'item'. An error was encountered loading a property. Continue?
20016 Error loading 'item'. A control could not be loaded due to load error. Continue?
20017 Form had old file format
20018 Replace existing 'item'?
20019 'item' does not exist
20020 'item' already exists in project
20021 <Filename> is a Read-Only file
20022 Errors during load. Refer to <log file> for details
20023 '<item>' could not be loaded
20024 Version number missing or invalid; Visual Basic 2.0 assumed
20025 String value too long to process; form load aborted

Trappable Errors for Grid Control:

30000 Cannot use RemoveItem on a fixed row
30001 Cannot use AddItem on a fixed row
30002 Grid does not contain that row
30004 Invalid column number for alignment
30005 Invalid alignment value
30006 Unable to allocate memory for grid
30008 Not a valid picture type
30009 Invalid row value
30010 Invalid column value
30011 Unable to register the memory manager
30013 Invalid row height value
30014 Invalid column width value
30015 Cannot remove last non-fixed row
30016 FixedRows must be one less than Rows value
30017 FixedCols must be one less than Cols value
30018 Rows must be one more than FixedRows value
30019 Cols must be one more than FixedCols value

Trappable Errors for OLE Control:

30000 OLE_OK
30001 OLE_WAIT
30002 OLE_BUSY
30003 OLE_ERROR_PROTECT_ONLY
30004 OLE_ERROR_MEMORY
30005 OLE_ERROR_STREAM
30006 OLE_ERROR_STATIC
30007 OLE_ERROR_BLANK
30008 OLE_ERROR_DRAW
30009 OLE_ERROR_METAFILE
30010 OLE_ERROR_ABORT
30011 OLE_ERROR_CLIPBOARD
30012 OLE_ERROR_FORMAT
30013 OLE_ERROR_OBJECT
30014 OLE_ERROR_OPTION
30015 OLE_ERROR_PROTOCOL
30016 OLE_ERROR_ADDRESS
30017 OLE_ERROR_NOT_EQUAL
30018 OLE_ERROR_HANDLE
30019 OLE_ERROR_GENERIC
30020 OLE_ERROR_CLASS
30021 OLE_ERROR_SYNTAX
30022 OLE_ERROR_DATATYPE
30023 OLE_ERROR_PALETTE
30024 OLE_ERROR_NOT_LINK
30025 OLE_ERROR_NOT_EMPTY
30026 OLE_ERROR_SIZE
30027 OLE_ERROR_DRIVE
30028 OLE_ERROR_NETWORK
30029 OLE_ERROR_NAME
30030 OLE_ERROR_TEMPLATE
30031 OLE_ERROR_NEW
30033 OLE_ERROR_OPEN
30034 OLE_ERROR_NOT_OPEN
30035 OLE_ERROR_LAUNCH
30036 OLE_ERROR_COMM
30037 OLE_ERROR_TERMINATE
30038 OLE_ERROR_COMMAND
30039 OLE_ERROR_SHOW
30040 OLE_ERROR_DOVERB
30041 OLE_ERROR_ADVISE_NATIVE
30042 OLE_ERROR_ADVISE_PICT
30043 OLE_ERROR_ADVISE_RENAME
30044 OLE_ERROR_POKE_NATIVE
30045 OLE_ERROR_REQUEST_NATIVE
30046 OLE_ERROR_REQUEST_PICT
30047 OLE_ERROR_SERVER_BLOCKED
30048 OLE_ERROR_REGISTRATION
30050 OLE_ERROR_TASK
30051 OLE_ERROR_OUTOFDATE
30052 OLE_ERROR_CANT_UPDATE_CLIENT
30053 OLE_ERROR_UPDATE

31001 Out of memory
31002 Property is write-only
31003 Can't open Clipboard
31004 No object

31005 Object closed
31007 Can't paste
31008 Invalid property value
31009 Object not empty
31010 Property is read-only
31011 Type of object cannot be created
31014 This action is reserved for future use
31015 Cannot execute object
31016 Server class was not specified before the registration database was accessed
31017 Invalid format on set data or set data text
31018 Class is not set
31019 Source document is not set
31020 Source item is not set

Additional reference words: 2.00
KBCategory:
KBSubcategory: EnvtDes

How to Use Visual Basic Vers 1.0, 2.0, & 3.0 on Same Computer

Article ID: Q94697

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article describes how to set up Visual Basic version 2.0 or 3.0 and keep Visual Basic version 1.0 on your computer. There are two issues involved when attempting to maintain two versions of Visual Basic on the same computer. First, you need to set up Visual Basic version 2.0 or 3.0 so that it does not overwrite the Visual Basic version 1.0 files. Second, you must manage the compatibility between the two versions.

MORE INFORMATION

To keep Visual Basic version 1.0 on your computer, install Visual Basic version 2.0 or 3.0 in a different directory. By default, the Visual Basic version Setup program attempts to copy the files to the \VB directory, but it will ask you if you want to put it in a different directory.

If you have the Professional Toolkit for Visual Basic version 1.0 and you want to preserve the custom control (or .VBX) files, place them in a directory other than \WINDOWS or \WINDOWS\SYSTEM. The Setup program for the Professional Toolkit for Visual Basic 1.0 gives you the option to place a copy of the custom control files in a separate directory. These files are normally placed in a subdirectory called VBX in the Visual Basic directory.

If you requested an extra copy of the custom control files, they'll remain separated from the Visual Basic version 2.0 custom control files, so you do not need to do anything. However, if you didn't request a copy, copy the Visual Basic version 1.0 .VBX files from the \WINDOWS\SYSTEM directory to another directory before running the Visual Basic version 2.0 Setup program. In addition to the VBX files, you also need to copy GSW.EXE, GSWDLL.DLL, and COMMDLG.DLL from the \WINDOWS\SYSTEM directory to the other directory.

The Setup program for the Visual Basic Standard and Professional editions copies the .VBX, .EXE, and .DLL files to the \WINDOWS\SYSTEM directory. If a .VBX, .EXE, or .DLL file already exists in the \WINDOWS\SYSTEM directory with that name, the Setup program changes the file extension from .VBX, .EXE, or .DLL to .OLD.

If you already ran the Visual Basic version 2.0 or 3.0 Setup program, you can recover the version 1.0 custom control files by copying the .OLD files from the \WINDOWS\SYSTEM directory to a different directory. Then rename the .OLD files giving them the appropriate extension (.VBX, .EXE, or .DLL). For example, rename GSW.OLD to GSW.EXE, GSWDLL.OLD to GSWDLL.DLL, and

COMMDLG.OLD to COMMDLG.DLL. Then rename all other .OLD files to .VBX files.

For the most part, the code for Visual Basic version 1.0 applications is upwardly compatible. In other words, you should be able to run all version 1.0 applications in version 2.0 or 3.0 with few or no changes. When you load a version 1.0 project into Visual Basic version 2.0 or 3.0, you will be informed that your files are in the old format. When you save the project, Visual Basic version 2.0 or 3.0 converts the files into the new version 2.0 or 3.0 format. Once the version 1.0 project files are saved in the new version's format, you cannot load the project files back into Visual Basic version 1.0.

Once you install Visual Basic version 2.0 or 3.0, any version 1.0 .EXE files that use custom controls will likely use the version 2.0 or 3.0 custom controls. This happens because when you installed the later version, its controls replaced the earlier version's controls in the \WINDOWS\SYSTEM directory. This should work well because the Visual Basic version 2.0 Professional Toolkit controls are upwardly compatible from the version 1.0 Professional Toolkit controls.

If you need to use Visual Basic version 1.0 custom controls, put them in the same directory as the version 1.0 .EXE that uses them. Then the .EXE will find the version 1.0 controls first. However, there is no guarantee the version 1.0 custom controls will be used because another .EXE may have already loaded the version 2.0 controls.

Additional reference words: 2.00 1.00 3.00
KBCategory:
KBSubcategory: EnvtDes

Add Graph Causes Err: GSW.EXE and GSWDLL.DLL Version Mismatch

Article ID: Q96007

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 2.0

SUMMARY

Loading the graph control into a project when different versions of GSW.EXE and GSWDLL.DLL exist on the same computer can cause one of the following two error messages:

```
GSW.EXE and GSWDLL.DLL Version Mismatch
Need Graphics Server 2.00 or later
```

Because the graph control uses a graphing and charting library (GSWDLL.DLL) along with a graphics server (GSW.EXE) to provide its graphing capabilities, it must have the same versions of these two files.

To work around the problem, place the 2.0 version of all three files (GRAPH.VBX, GSW.EXE, and GSWDLL.DLL) in the \WINDOWS\SYSTEM directory, and delete or move the older versions out of the following directories:

- The \WINDOWS directory
- The \WINDOWS\SYSTEM directory
- All directories located on the MS-DOS path

MORE INFORMATION

The setup program for the Professional Edition of Visual Basic version 2.0, correctly updates the GRAPH.VBX and its two auxiliary files if previous versions of the files exist in the \WINDOWS\SYSTEM directory. If prior versions of the control reside in a different directory, such as the \WINDOWS directory, the setup program for Visual Basic version 2.0 will not rename or remove the older versions. However, it does place the version 2.0 versions of the controls in the \WINDOWS\SYSTEM directory.

When you load the graph control, either by choosing Add File from the File menu or by adding GRAPH.VBX to the AUTOLOAD.MAK file, the graph control searches the directories for its two auxiliary files in this order:

1. The directory where GRAPH.VBX resides, unless it's \WINDOWS\SYSTEM
2. The \WINDOWS directory
3. The \WINDOWS\SYSTEM directory
4. The directories on MS-DOS path

If GRAPH.VBX resides in the \WINDOWS\SYSTEM directory the search begins with step 2. The error message, therefore, can occur when a copy of GRAPH.VBX resides in the \WINDOWS\SYSTEM directory and an older version of either GSW.EXE or GSWDLL.DLL resides in the \WINDOWS directory while 2.0 versions reside in the \WINDOWS\SYSTEM directory with GRAPH.VBX.

Steps to Reproduce Error Message

1. Run SETUP.EXE from the Professional Toolkit for Visual Basic version 1.0.

2. Install the Toolkit in the \VB1 directory using these two options:

   - Select the option to install Controls/Samples.
   - Select Yes, when asked if you want to install duplicate copies of the Toolkit controls in the \VB1\VBX directory.

   This will install copies of the custom controls in the \WINDOWS\SYSTEM and \VB1\VBX directories.

3. Run SETUP.EXE from the Visual Basic Professional Edition version 2.0 for Windows.

4. Install Visual Basic version 2.0 in the default directory (\VB) using the Complete Installation option.

At this point, the Visual Basic version 1.0 controls in the \WINDOWS\SYSTEM directory are updated to their 2.0 versions, including GRAPH.VBX, GSW.EXE, and GSWDLL.DLL.

5. Rename GSW.EXE in the \WINDOWS\SYSTEM directory to GSW.OLD. Using File Manager, navigate to the \WINDOWS\SYSTEM directory and select GSW.EXE from the list of files. From the File menu, choose Rename (ALT, F, N). In the To box, enter GSW.OLD.

   By doing this, you will retain a 2.0 version of the Graph control's server program.

6. Copy GSW.EXE from the \VB1\VBX directory into the \WINDOWS\SYSTEM directory. Using File Manager, navigate to the \VB1\VBX directory and select GSW.EXE from the list of files. From the File menu, choose Copy (ALT, F, C).

   Now you have conflicting versions of the GRAPH.VBX and GSW.EXE files in the same directory.

7. Run Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

   The AUTOLOAD.MAK file, installed with Visual Basic 2.0, will attempt to load all custom controls shipped with the Professional Edition. When the loading process reaches the graph control, the following error message occurs:

      GSW.EXE & GSWDLL.DLL version mismatch

   Upon closing the error message box, a second error message box appears:

      Can't load Custom Control DLL: C:\WINDOWS\SYSTEM\GRAPH.VBX

   The two error messages also occur when you choose Add File from the File menu

Even though this example incorrectly updates only the GSW.EXE file, the same error messages occur when all three files related to the graph control have different versions.

Additional reference words: 2.00
KBCategory:
KBSubcategory: EnvtDes

PRB: Placing Controls inside Container Controls

Article ID: Q104166

The information in this article applies to:

- Standard and Professional Edition of Microsoft Visual Basic for Windows, versions 1.0, 2.0, and 3.0
- Standard and Professional Edition of Microsoft Visual Basic for MS-DOS, version 1.0

SUMMARY

To place a control correctly within a container control, use one of the following methods.

- Select an existing control, and from the Edit menu, choose either Cut or Copy. Then select the container control, and from the Edit menu, choose Paste.

- Find the icon for the control in the Toolbox. Click it, and then drag a rectangle inside the border of the container. More specifically, click a Toolbox icon, and then release the mouse button. Move the mouse cursor inside the border of the container control. The mouse cursor changes to cross hairs. Now click the mouse button and hold it down. Move the mouse to the bottom right. Then release the mouse button.

You cannot place a control inside a container control by double-clicking an icon in the Toolbox or by dragging a control onto a container control. These actions place the control on the form in front of the container rather than inside the container control.

MORE INFORMATION

Container controls supplied with the standard edition are the frame and picture box. The container controls supplied with the professional edition are the 3-D frame, 3-D panel, and the gauge. These controls can also be placed inside container controls.

When you place a control inside a container control, such as a frame, the containerized control:

- appears completely within the border of the container control and in front of the background of the container control

- maintains its position relative to the container control

Additional reference words: 1.00 2.00 3.00 B_VBmsdos
KBCategory: Envt
KBSubCategory: EnvtDes

Category Keywords for All Visual Basic KB Articles

Article ID: Q108753

The information in this article applies to:

- Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Each article in the Visual Basic for Windows collection contains at least one keyword (called a KBSubcategory keyword) that places the article in an appropriate category. This article lists all the KBSubcategory keywords.

MORE INFORMATION

Category & Subcategory Description	KBSubcategory Keyword
Setup / Installation (Setins)	Setins
Environment-specific Issues (Envt)	
VB Design Environment	EnvtDes
Run-Time Environment	EnvtRun
Programming (Prg)	
Visual Basic Forms and Controls	
Standard Controls / Forms	PrgCtrlsStd
Custom Controls	PrgCtrlsCus
Third-Party Controls	PrgCtrlsThird
Optimization	
Memory Management	PrgOptMemMgt
General Optimization Tips	PrgOptTips
General VB Programming	PrgOther
Advanced programming (APrg)	
Network	APrgNet
Windows Programming (APIs / DLLs)	
Printing	APrgPrint
Graphics	APrgGrap
Windowing	APrgWindow
INI Files	APrgINI
Other API / DLL Programming	APrgOther
Data Access	
ODBC	APrgDataODBC
IISAM	APrgDataIISAM
Access	APrgDataAcc
General Database Programming	APrgDataOther
3rd Party DLL's	APrgThirdDLL

Inter-Application Programmability (IAP)

OLE	IAPOLE
DDE	IAPDDE
3rd Party Interoperability	IAPThird

Tools (Tls)

Setup Toolkit / Wizard	TlsSetWiz
Control Development Kit (CDK)	TlsCDK
Help Compiler (HC)	TlsHC

References (Refs)

Documentation / Help File Fixes	RefsDoc
Product Information	RefsProd
Third-Party Information	RefsThird
PSS-Only Information	RefsPSS

Using Keywords to Query the KB

At Microsoft, we use the subcategory keywords to organize the articles for Help files and for the FastTips Catalog. You can use them to query the Microsoft Knowledge Base for Visual Basic articles that apply to that category or subcategory. For example, you can find all the general database programming articles by querying on the following words in the Microsoft Knowledge Base:

```
visual and basic and APrgDataOther
```

Use the asterisk (*) wildcard to find articles that fall into the general categories or into an intermediate subcategory. The first element in each keyword is the category. For example, to find all the articles that apply to Visual Basic Forms and Controls regardless of whether they are standard, custom, or third-party controls, use the following words to query the Microsoft Knowledge Base:

```
visual and basic and PrgCtrls*
```

To find all advanced programming articles, query on these words:

```
visual and basic and APrg*
```

Add KBSubcategory Keyword to Each Article

When contributing an article to the Visual Basic Knowledge Base, add the appropriate KBSubcategory keyword to the bottom of the article on the KBSubcategory line. Each article in the Visual Basic for Windows collection contains the following section at the bottom of the article:

```
Additional reference words:
KBCategory:
KBSubcategory: <keyword>
```

An article usually has only one subcategory keyword, but it may have more.

If you are interested in contributing, please obtain the guidelines by

querying on the following words in the Microsoft Knowledge Base:

   visual and basic and kbguide and kbartwrite

Additional reference words: 3.00 dskbguide subcatkey
KBCategory:
KBSubcategory: RefsPSS

Can't Use Multiple & (for Access Keys) in a VB Menu Control

Article ID: Q73372

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

When creating a menu control that has multiple & (ampersand) characters to mark the access keys in the caption (for example, &a&b&c&d), the menu will appear with an underline under the character after the last &. However, the access key will respond to the character following the first &. This is not a problem with Visual Basic, but rather with the Windows operating environment.

To work around this problem, do not put multiple & characters in the menu caption; just use one & character per caption.

MORE INFORMATION

Steps to Reproduce Problem

1. From the File menu, choose New Project (ALT+F+N).

2. From the Window menu, choose Menu Design Window (ALT+W+M).

3. Enter "&A&B&C&D" (without the quotation marks) for the Caption.

4. Enter "ABCD" (without the quotation marks) for the CtlName.

5. Choose the Done button.

6. Click the menu item ABCD on Form1.

7. Enter the statement Print "ABCD" in the click event for the menu item ABCD as follows:

```
Sub ABCD_Click ()
    Print "ABCD"
End Sub
```

8. Run the program.

When the program is run, the D in the menu caption will be underlined, but the menu responds to ALT+A, not ALT+D.

Additional reference words: 1.00 2.00 3.00
KBCategory:

KBSubcategory: EnvtRun

Cannot Tile or Cascade Programs Created with Visual Basic

Article ID: Q73698

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0
- Microsoft Windows, version 3.0

SUMMARY

Applications that have been created with Microsoft Visual Basic do not tile or cascade as do other Windows applications.

MORE INFORMATION

Visual Basic creates applications that are pop-up windows. This window style does not respond to the tile or cascade message sent from the Windows version 3.0 Task List or other applications that support the cascade and tile features.

You can verify this action by launching two applications created in Visual Basic, then bringing up the Windows Task List by pressing CTRL+ESC, and from the Task List choosing either the Cascade or Tile button. Notice that nothing has changed in the arrangements of these two Visual Basic application windows. You may have expected the Visual Basic application windows to cascade or tile as other Windows applications do, but they will not do so.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtRun

Some VB.EXE Main Menu Commands Can Be Invisible at Run Time

Article ID: Q73699

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0
- Microsoft Windows, version 3.0

SUMMARY

If you shrink the width of VB.EXE's main menu (and Properties Bar) such that menu commands automatically wrap to the next line, wrapped menu commands may be invisible at run time in the VB.EXE environment.

You can work around this visibility problem by using the ALT key to access the invisible menu commands on the shrunken menu, or by avoiding shrinking the width beyond the point where the menus wrap.

MORE INFORMATION

Steps to Reproduce Problem

1. Start Visual Basic (VB.EXE).

2. Place the mouse pointer on the far right side of the Properties Bar such that the mouse changes to a double-headed pointer, ready for resizing the Properties Bar.

3. Press and hold down the left button of the mouse and drag the right edge of the Properties Bar toward the left side of the screen so that the Bar ends up being about 3 inches in width, then release the mouse button. This should cause the Window and Help menu commands to automatically wrap the next line.

4. From the Run menu, choose Start, or press F5. The Window and Help menus are now invisible because they are wrapped beyond the edge of their window.

5. Click the Immediate Window (in the lower right corner) and bring the Immediate Window in front of the Form1 window.

6. Try bringing up either the Window menu by pressing ALT+W or the Help menu by pressing ALT+H. You will see the appropriate menu on the screen, but it will appear disconnected below the Main Menu (or Properties Bar) even though the menu is still functional. If you don't perform step 5, you won't be able to select the invisible menus with the ALT key (or with the mouse).

This behavior is due to the way that Windows, version 3.0 manipulates menus and the design of the Visual Basic for Windows, version 1.0 interface. To

avoid this behavior, Microsoft recommends keeping the main menu sufficiently wide such that menus are not wrapped.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtRun

UAE or GP Fault with VB .EXE Acting as Windows 3.0 Shell

Article ID: Q73801

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, versions 1.0 and 2.0

SUMMARY

Compiled Visual Basic .EXE applications can be used as the Windows shell in Windows version 3.1 on an 80386 computer or better.

However, an .EXE application created by Visual Basic cannot be used as the Windows shell in Windows version 3.0. Attempting to run an application as the Windows version 3.0 shell results in an Unrecoverable Application Error (UAE).

This is a design limitation of Windows version 3.0. It is not a limitation in Windows version 3.1, with one exception. When you run the .EXE program in Windows version 3.1 standard mode on a 80286 computer, a General Protection (GP) fault occurs at the same point where a UAE occurs in Windows version 3.0.

Basically, you can use a Visual Basic .EXE program as a Windows shell only on an 80386 computer or better. This information applies only to Visual Basic .EXE programs.

MORE INFORMATION

A user-defined shell application can be specified in the Windows system initialization (SYSTEM.INI) file. The default shell is PROGMAN.EXE (the Program Manager). If a Visual Basic program is specified as the customized Windows 3.0 shell, a Windows 3.0 UAE occurs on any attempt to run Windows version 3.0 from the MS-DOS command line. This problem does not occur with Windows version 3.1.

A Visual Basic application cannot be run as the Windows 3.0 shell because it does not contain the special set of startup code required by a Windows 3.0 shell application. The only way to create a Windows 3.0 shell application is to use the C Compiler and the Windows Software Development Kit (SDK) to write a non-Visual Basic application.

Steps to Reproduce Problem

Warning: The following steps require changing the Windows system initialization file (SYSTEM.INI) in a manner such that Windows version 3.0 will not run successfully unless the file is restored from MS-DOS. The

file can be restored from MS-DOS by using a backup copy of the SYSTEM.INI file or by restoring the SYSTEM.INI file with a text editor in MS-DOS.

1. Start Visual Basic.
2. From the File menu, choose New Project.
3. From the File menu, choose Make .EXE program.
4. Choose the OK button to select Project1.EXE as the .EXE filename.
5. Exit Visual Basic.
6. Start Windows Notepad.
7. From the File menu, choose Open.
8. In the Filename text box, type C:\WINDOWS\SYSTEM.INI including the correct path for the SYSTEM.INI file on your computer.
9. Choose the OK button.
10. Change the line the reads:

    SHELL=PROGMAN.EXE

    to this line:

    SHELL=C:\VB\PROJECT1.EXE

    changing the path to the correct path to the file created in step 4.
11. From the File menu, choose Save.
12. Exit Notepad.
13. From the Windows Program Manager File menu, choose Exit. You should return to MS-DOS.
14. At the MS-DOS command prompt, start Windows.

When you attempt to start Windows version 3.0, a UAE occurs. You will need to reboot (restart) your computer and modify the SYSTEM.INI file using a text editor in MS-DOS to reverse the change made in step 10.

Additional reference words: 1.00 2.00 3.00 286 386
KBCategory:
KBSubcategory: EnvtRun

F5 in Run Mode with Focus on Main Menu Bar Acts as CTRL+BREAK

Article ID: Q74348

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

A Microsoft Visual Basic for Windows program will break at run time under the following simultaneous conditions:

1. You run the program in the Visual Basic for Windows development environment.

2. The Visual Basic for Windows menu bar has the focus.

3. You press the F5 key.

The program will break when the F5 key is pressed and the Immediate Window will get the focus. This is not a problem with Visual Basic for Windows, but rather a design feature.

This information only applies to an application run in the Visual Basic for Windows development environment, not as an .EXE program.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The F5 key acts as the shortcut key for the Visual Basic for Windows Run menu. Because Start, Continue, and Break all share the same menu item under the Run menu, F5 acts differently depending upon the state of execution of a program. It acts as the Run key in the Visual Basic version 1.0 for Windows environment. It also serves as the Break key once the application is running and the focus is on the Visual Basic for Windows menu bar. After execution has been "broken" with the Break key, the F5 key serves as the Continue key.

To demonstrate the different modes of the F5 key, do the following:

1. Run Visual Basic for Windows.

2. From the File menu, select New Project (press ALT, F, N).

3. Press the F5 key to run the program.

4. Using the mouse, click the Visual Basic for Windows menu bar.

5. Press the F5 key to break the program. The Immediate window will be given the focus after you press the F5 key.

6. Press the F5 key again to continue execution of the program.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtRun

PRB: Access Key Causes Different Event Order than Mouse Click

Article ID: Q74905

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Visual Basic for MS-DOS, version 1.0

SYMPTOMS

In Visual Basic, events may be generated in a different order if you choose a control (such as a button, a check box, or an option box) using an access key rather than with the mouse. The events that occur in a different order are Click, LostFocus, and GotFocus.

WORKAROUND

By inserting the DoEvents statement as the very first statement in the Click event handler, you can cause the LostFocus and GotFocus events to be handled before the body of the Click event handler.

STATUS

This behavior is by design. It is not a bug in Visual Basic.

MORE INFORMATION

You can create an access key at design time by changing the Caption property of a control to include an ampersand (&). The access key is the character after the ampersand, and at run time you press ALT+character to choose the control. (See page 120 of the "Microsoft Visual Basic: Programmer's Guide" version 1.0. manual.)

When you press an access key (ALT+character) to choose a control, the Click event is generated before the LostFocus and GotFocus event; however, when you choose a control by clicking the mouse, the LostFocus and GotFocus events are generated before the Click event.

The example below shows this different order of events. The example uses command buttons, but also applies to Check and Option boxes:

1. Open a new form and create two command buttons.

2. Enter the code as shown further below.

3. Change the Caption property of Command2 to "Command&2"

4. Run the program.

5. a. When Command1 has the focus and you click Command2, the
      following events are generated in the following order:

      Command1_LostFocus
      Command2_GotFocus
      Command2_Click

   b. When Command1 has the focus and you press the access key, ALT+2,
      the following events are generated in the following order:

      Command2_Click
      Command1_LostFocus
      Command2_GotFocus

Sample Code:

```
Sub Command1_Click ()
   Print "Command1_click"
End Sub

Sub Command1_LostFocus ()
   Print "Command1_lostfocus"
End Sub

Sub Command1_GotFocus ()
    Print "Command1_gotfocus"
End Sub

Sub Command2_Click ()
    Print "Command2_click"
End Sub

Sub Command2_LostFocus ()
   Print "Command2_lostfocus"
End Sub

Sub Command2_GotFocus ()
   Print "Command2_gotfocus"
End Sub
```

Additional reference words: 1.00 2.00 3.00 vbmsdos
KBCategory:
KBSubcategory: PrgCtrlsStd EnvtRun

Determining Whether TAB or Mouse Gave a VB Control the Focus

Article ID: Q75411

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can determine whether a Microsoft Visual Basic for Windows control received the focus from a mouse click or a TAB keystroke by calling the Microsoft Windows API function GetKeyState in the control's GotFocus event procedure. By using GetKeyState to check if the TAB key is down, you can determine if the user pressed the TAB key to get to the control. If the TAB key was not used and the control does not have an access key, the user must have used the mouse to click the control to set the focus.

MORE INFORMATION

The GetKeyState Windows API function takes an integer parameter containing the virtual key code for the desired key states. GetKeyState returns an integer. If the return value is negative, the key has been pressed.

The following is a code example. To use this example, start with a new project in Visual Basic for Windows. Add a text box and a command button to Form1. Enter the following code in the project's GLOBAL.BAS module:

```
' Global Module.
Declare Function GetKeyState% Lib "User" (ByVal nVirtKey%)
Global Const VK_TAB = 9
```

Add the following code to the GotFocus event procedure for the Text1 text box control:

```
Sub Text1_GotFocus()
   If GetKeyState(VK_TAB) < 0 Then
     Text1.SelStart = 0
     Text1.SelLength = Len(Text1.Text)
   Else
     Text1.SelLength = 0
   End If
End Sub
```

Run the program. If you use the TAB key to move the focus from the command button to the text box, you should see the text in the text box selected. If you change the focus to the text box by clicking it with the mouse, the text will not be selected.

An access key is assigned by using an ampersand (&) in the control's caption property. If the control has an access key, you may also want to check the state of the virtual ALT key by using GetKeyState to see if the user used the access key to change the focus. The virtual key code for ALT, actually known as VK_MENU, is 12H (&H12).

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtRun APrgOther

How to Use CodeView for Windows (CVW.EXE) with Visual Basic

Article ID: Q75612

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

You can use CodeView for Windows (CVW) to debug a dynamic link library (DLL) or custom control that is called from Visual Basic.

Note that you can build custom controls using the Control Development Kit (CDK) for Visual Basic. The Visual Basic CDK, formerly shipped separately as an add-on product from Microsoft, is now shipped as part of Microsoft Professional Toolkit for Microsoft Visual Basic version 1.0 for Windows.

MORE INFORMATION

CVW.EXE, CodeView for Windows, is distributed with the Microsoft Windows Software Development Kit (SDK).

CVW can be a useful tool for debugging DLLs and custom controls written for a Visual Basic program.

CVW takes the following command line arguments:

```
[path]CVW.EXE /L [dynamic link library] [executable program]
```

where:

[dynamic library] is your DLL or custom control.

[executable program] is the EXE that uses your DLL/custom control.

The "/L" option tells CVW that this is a DLL or custom control.

You can invoke CVW from the Windows Program Manager in any of the following ways:

- From the Windows Program Manager File menu, choose New, and specify CVW.EXE as a Program Item with proper arguments. You can then double-click the CVW icon to run CVW.EXE.
- From the Windows Program Manager File menu, choose Run, and enter CVW.EXE and its command line arguments.
- Invoke CVW with no arguments, and at the prompts, enter the

program name and DLL/VBX that you want to debug.

The example below demonstrates how to invoke CIRCLE3.VBX, which comes with the Microsoft Visual Basic Control Development Kit (CDK):

1. Run CVW.EXE from the Program Manager as specified below.

   [path]CVW.EXE /L [path]CIRCLE3.VBX [path]VB.EXE

   Note: You can just specify an .EXE program that was written in the Visual Basic environment instead of specifying the VB.EXE environment itself. If you do this, skip steps 7, 8, and 9 below.

   Note: If you invoke CVW.EXE with no command line arguments, CVW.EXE will prompt you for command line arguments. Specify the VB.EXE file that uses the *.VBX file. CVW.EXE will then prompt you for "Additional DLLs...". Specify the *.VBX file at this prompt. Skip to step 4.

2. When CVW is loaded into the debug monitor, the following message will be displayed:

   No Symbolic information for VB.EXE

3. Choose the OK button to load the Visual Basic program.

4. From the File menu, choose Open Module to load the CIRCLE3.VBX source code. You should see a list of "c" source code in the list box. Select CIRCLE.C, which corresponds to the CIRCLE3.VBX source code.

5. Locate the WM_LBUTTONDBLCLK message and set a breakpoint on the first "IF" statement.

6. Press F5 to run your Visual Basic program.

7. From the File menu, choose Add File. In the Files box, select the CIRCLE3.VBX file. The CIRCLE3 tool appears in the toolbox.

8. Select the custom control from the toolbox and add it to your form.

9. Press F5 to run your program.

10. Double-click the circle. When your breakpoint is encountered, focus will be set to CVW and execution will stop at your breakpoint. You can now step through your program.

11. Press F5 to return to the Visual Basic program.

Reference(s):

"Programming Windows: the Microsoft Guide to Writing Applications for Windows 3," by Charles Petzold, Microsoft Press, 1990

"Peter Norton's Windows 3.0 Power Programming Techniques," by Peter Norton and Paul Yao, Bantam Computer Books, 1990

"Microsoft Windows Software Development Kit: Programming Tools," version 3.0

WINSDK.HLP file shipped with Microsoft Windows 3.0 Software Development Kit

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtRun

Simulating ON KEY and Key Trapping by Using the KeyDown Event

Article ID: Q75858

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Although there is no ON KEY GOSUB statement in Visual Basic, you can achieve an effect identical to ON KEY event handling. Visual Basic forms and controls that are able to get focus have a KeyDown event procedure that can simulate the effects of the ON KEY statements in Basic interpreters and compilers for MS-DOS. In fact, the KeyDown event procedure is more powerful and flexible than the ON KEY statement.

MORE INFORMATION

Pressing a key while a Visual Basic form or control has the focus executes that form or control KeyDown event procedure. Within the KeyDown event procedure, you can call a global procedure and pass the actual key states to the global procedure. You can use this to create an effect in Visual Basic for Windows that is identical to the effect caused by trapping ON KEY events in Basic interpreters and compilers for MS-DOS. In Visual Basic, you can also pass the name of the control or form where the KeyDown event occurred, so the global procedure will know which control or form called it.

Here's a small example:

1. In the Visual Basic Project window, double-click a form or module (GLOBAL.BAS in Visual Basic version 1.0) to bring up the code window. Move to the general-declaration section of the form or module. Then from the Visual Basic Code menu, choose Load text, and load the CONSTANTS.TXT file that came with Visual Basic.

   Note: in Visual Basic version 1.0, if you already have text in the GLOBAL.BAS file, create a new module, add the CONSTANTS.TXT file to the new module, and then cut and paste the text into the GLOBAL.BAS file.

2. Add two text boxes (Text1 and Text2) to a form.

3. In the Text1_KeyDown event procedure, add the following code:

   ```
   Call OnKeyGoSub(KeyCode, Shift, Text1)
   ```

4. In the Text2_KeyDown event procedure, add the following code:

   ```
   Call OnKeyGoSub(KeyCode, Shift, Text2)
   ```

5. Add a Label (Label1) to to the form.

6. In the general-declaration section for the form, add this procedure:

```
Sub OnKeyGoSub (KeyCode%, Shift%, Ctrl As Control)
   Select Case KeyCode%
      Case KEY_MENU: Key$ = ""
      Case KEY_SHIFT: Key$ = ""
      Case KEY_CONTROL: Key$ = ""
      Case KEY_F1: Key$ = " F1 "
      Case KEY_UP: Key$ = " UP key"
      Case KEY_CAPITAL: Key$ = "CAP LOCKS"
      Case Else: Key$ = Chr$(KeyCode%)
   End Select
   Select Case Shift%
      Case SHIFT_MASK: Shft$ = "Shift"
      Case ALT_MASK: Shft$ = "Alt"
      Case CTRL_MASK: Shft$ = "Ctrl"
      Case Else: Shft$ = ""
   End Select
   Label1.Caption="Key:"+ Shft$+ " "+ Key$
End Sub
```

7. Run the program. Move back and forth between the two text boxes using either the TAB key or the mouse. Experiment with any key in combination with the ALT, CTRL, and SHIFT keys. Also, try the F1 and UP ARROW keys.

The above example is limited, but shows you how to simulate the ON KEY statements or key trapping in Visual Basic by placing the call to the key trap procedure in any KeyDown event procedure.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtRun

Sending Keystrokes from Visual Basic to an MS-DOS Application

Article ID: Q77394

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The "Microsoft Visual Basic: Language Reference" version 1.0 manual states that the SendKeys function cannot be used to send keystrokes to a non-Windows application. Listed below is a method that can be used to work around this limitation.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The Microsoft Visual Basic for Windows SendKeys function can send keystrokes to the currently active window (as if the keystrokes had been typed at the keyboard). Although it is not possible to send keystrokes to a non-Windows application with SendKeys directly, you can place text on the Clipboard and use SendKeys to paste that text into an MS-DOS application that is running in a window (or minimized as an icon.)

To run an MS-DOS application in a window, you MUST be running in Windows 386 enhanced mode. You must also make sure that the MS-DOS application's PIF file has been set to display the application in a window rather than full screen. Use the Windows PIF Editor to make this modification, if necessary.

An example of sending keystrokes to an MS-DOS session running in a window is given below:

1. Start a MS-DOS session (running in a window).

2. Start Visual Basic for Windows.

3. Enter the following into the general declarations section of the form:

   ```
   Dim progname As String
   ```

4. Draw two labels on the form. Change the first label's caption to "Dos App Title." Change the second label's caption to "Keys to send."

5. Draw two text boxes on the form (next to each of the previously

drawn labels). Delete the default contents of these text boxes. These controls will be used to allow the user to enter the MS-DOS application window title and the keystrokes to send to it. Change the Name property of these text boxes to "DosTitle" and "DosKeys," respectively.

6. Draw a command button on the form and change its caption to "Send keys."

7. Attach the following code to the command button click procedure:

```
progname = "Microsoft Visual Basic"
clipboard.Clear
clipboard.SetText DosKeys.Text + Chr$(13)  ' Append a <CR>.
AppActivate DosTitle.Text
SendKeys "% ep", 1
AppActivate progname
```

   If the text that you send is the DIR command or another command that takes time, the AppActivate call immediately following the SendKeys call will interrupt the processing. The AppActivate call should be placed in a timer with the appropriate interval set and the timer enabled in the command_click procedure. The timer should be disabled before exiting the timer.

8. Run the program.

9. Enter the window title of the MS-DOS application into the DosTitle text box. The default window title for an MS-DOS session is "DOS." If you would like to change the window title of an MS-DOS application, you should use the PIF Editor.

10. Enter the keystrokes to send into the DosKeys text box (for example, "DIR").

11. Click the Send Keys command button. The keystrokes will be sent to the Clipboard and then pasted into the MS-DOS window.

If this technique is used in a compiled Visual Basic for Windows program, you should change the progname assignment from "Microsoft Visual Basic" to the executable file name. Also, if you would like to see the text being placed onto the Clipboard, you can open the Windows Clipboard viewer.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtRun

"Error Loading DLL" if VB Compiled .EXE Has Same Name as DLL

Article ID: Q79598

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

If you create a compiled executable program with the same base name as a dynamic link library (DLL) that is called from the program, an "Error Loading DLL" message will be displayed when the compiled program attempts to call the DLL. If the DLL is loaded before the .EXE program is run (for example, if the DLL is in use from another application) then the executable program will not run at all.

Similarly, if an .EXE program has the same name as a loaded device driver (.DRV) and the driver is loaded before you run the .EXE program, then your executable program will not run. For example, if you name your executable program TIMER.EXE, it will not run because Windows has already loaded a device driver named TIMER.DRV.

This behavior is how Windows is designed to operate. It is not a problem with Microsoft Visual Basic, because the behavior can occur with any Windows application, and may occur between any two Windows modules (either from executable programs or DLLs).

MORE INFORMATION

This behavior occurs because Windows checks, by module name, to see if a program is already loaded before it tries to execute that program. If the requested module is already loaded, Windows creates another instance of that module. Thus, attempting to load a DLL with the same module name as an already executing program will fail (usually with the error "Error Loading DLL"), and attempting to start an executable program with the same module name as an already loaded DLL will not execute the program.

Because the module name for a DLL is often the same as the name of the DLL itself (although this can be varied using the LIBRARY entry of the module definition file used when creating the DLL), and the module name for a compiled Visual Basic program is the same as the original base .EXE file name, attempting to load a DLL and a Visual Basic .EXE program that share the same name will often result in one of the above errors. To avoid this problem, either recompile the Visual Basic program and change the .EXE filename, or recreate the DLL, changing the LIBRARY entry in the module definition file.

Additional reference words: 1.00 2.00 3.00
KBCategory:

KBSubcategory: EnvtRun

VB Error Using Shell: Cannot Find DLL, Insert in Drive A

Article ID: Q80404

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

When a Visual Basic for Windows application shells to a Microsoft Windows application that expects to find a dynamic link library (DLL) in its own directory, Visual Basic for Windows may generate the following error message and fail to start the application:

   Cannot Find <DLL NAME>, Please Insert in Drive A

This error occurs because the application being shelled to expects to find the DLL in the current directory, the MS-DOS path, or the Windows directory. Shelling to an application in code does not change the current directory, even if you specify the path to the application in the Shell statement.

One solution is to use Visual Basic for Windows' ChDir statement to change the current directory to the directory containing the DLL before attempting to shell to the application. An alternative solution is to copy the DLL to the Windows directory, or include the path where the DLL is located in the MS-DOS path.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The following is a pseudocode example that shows how to use the ChDir statement to make the application's directory the current directory. The C:\APPS directory and the .EXE name MYAPP.EXE are arbitrary names selected to represent the location of the application being shelled to and an .EXE name, respectively.

```
    Sub Form_Click ()
        ChDir "c:\Apps"     ' The name of the directory containing
                            ' the needed DLL.
        x% = Shell("c:\Apps\MyApp.EXE", 1)
    End Sub
```

Note: If the application is on a different drive, use the ChDrive statement first to change drives before using the ChDir statement.

Additional reference words: 1.00 2.00 3.00
KBCategory:

KBSubcategory: EnvtRun

VB CURDIR$ Function Not Reliable to Determine Program Location

Article ID: Q80611

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The CURDIR$ function returns the path to the current directory on the currently selected drive. Because the current directory is not necessarily the directory where the current Visual Basic program resides, the CURDIR$ function is not a reliable means for determining the location of the currently executing program. This information applies to the CURDIR$ function, and to the drive list box, directory list box, and file list box controls in Visual Basic.

MORE INFORMATION

You can use any of the following methods to start a program under Windows:

- From the MS-DOS prompt, type:

  ```
  WIN drive:\pathname\program name
  ```

- From Windows Program Manager, choose Run from the File menu, and enter the full pathname of the executable program.

- From Windows File Manager, choose Run from the File menu, and enter the full path of the executable program.

- From Windows Program Manager, choose New from the File menu, and create a new program item for the executable program. Double-click the resulting icon.

- From Windows File Manager, use the mouse to choose the appropriate drive and directory containing the executable file, and double-click the executable filename.

If the program is launched using the first, third, or fifth method above, the CURDIR$ value will return the current directory at the time Windows was launched or at the time the program was started from File Manager. The current directory can be checked by opening File Manager and reading the current directory from the bar below the drive buttons; for the fifth method, the File Manager's current directory will actually be the directory where the started program resides.

If the program is launched using the second method from the Windows Program Manager, the CURDIR$ value will be the path to the location

of the program that was started.

If the program is launched using the fourth method, the CURDIR$ value is the working directory you specified for the icon, or if you left the working directory blank, CURDIR$ returns the Windows directory.

Note that the current directory of an MS-DOS session does not necessarily indicate the current directory that will be returned by CURDIR$.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtRun

How to Get Windows Version Number in VB with GetVersion API

Article ID: Q80642

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

From a Microsoft Visual Basic for Windows program, you can find out which version of Windows is running by calling the Windows API GetVersion() function from the Windows Kernel module. The GetVersion() function can help your application accommodate any known differences, if any, in the way API calls operate between different versions of Windows (such as differences
between API parameters or return values).

MORE INFORMATION

The step-by-step example given below demonstrates how to make the GetVersion() function call. GetVersion() takes no parameters, and the return value is a WORD value -- which translates to an integer in Visual Basic for Windows.

The return value specifies the major and minor version numbers of Windows. The high order byte specifies the minor version and the low order byte specifies the major version number.

Step-by-Step Example

1. Create a form with a text box and a command button.

2. Add the following declaration to the General Declarations section:

```
Declare Function GetVersion Lib "kernel" () As Integer
```

3. Add following code to the command button Click event:

```
Sub Command1_Click ()
   i% = GetVersion()
   ' Lowbyte is derived by masking off high byte.
   lowbyte$ = Str$(i% And &HFF)
   ' Highbyte is derived by masking off low byte and shifting.
   highbyte$ = LTrim$(Str$((i% And &HFF00) / 256))
   ' Assign Windows version to text property.
   text1.text = lowbyte$ + "." + highbyte$
End Sub
```

Additional reference words: 1.00 2.00 3.00
KBCategory:

KBSubcategory: APrgWindow EnvtRun

PRB: Device Unavailable Msg When Change Path & Drive Door Open

Article ID: Q80645

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SYMPTOMS

Changing the Path property of a directory list box or a file list box to a floppy drive that has an open drive door or no disk present results in the following error:

   Error: 68 Device unavailable

Rather than giving this more expected error:

   Error: 71 Disk not ready

This occurs whether you run the program in the VB.EXE environment or as an .EXE file.

STATUS

This behavior is by design.

MORE INFORMATION

The following definitions for errors 68 and 71 can be found in the Visual Basic online Help:

Error 68 Device Unavailable

The device you are trying to access is not online or does not exist.

Error 71 Disk Not Ready

There is no disk in the drive specified, or the drive door is open. Insert a disk in the drive, close the door, and retry the operation.

Attempting to open a file on a drive that has an open door or missing disk generates Error 71, "Disk Not Ready."

Steps to Reproduce Behavior

1. Start Visual basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Create the following controls on Form1:

Control	Control Name
Drive list box	Drive1
File list box	File1

3. Add the following code to the Drive1_Change event procedure:

```
Sub Drive1_Change ()
   On Error GoTo Trap
   File1.path = drive1.drive
   Exit Sub
Trap:
   MsgBox "Error:"+Str$(Err)+" "+Error$(Err)
   Resume Next
End Sub
```

4. From the Run menu, choose Start (ALT, R, S) to run the program.

Changing the drive in the drive list box to a drive that is open or that contains no disk causes a message box to display:

```
Error: 68 Device unavailable
```

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtRun

How to Right Justify Numbers Using Format$

Article ID: Q95945

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Use the following two-step process to right justify numbers in a string by using the format$ function:

1. Format the number into a string by using the usual numeric conversion characters (0 # . ,).

2. Format the resulting string by using a format string consisting of a number of @ characters equal to the length of the format string used in step 1.

The following example Sub procedure formats several numbers using the seven character formats $##0.00 and @@@@@@@:

```
Sub Form_Click ()
   Print "|" + Format$(Format$(1.5, "$##0.00"), "@@@@@@@") + "|"
   Print "|" + Format$(Format$(12.5, "$##0.00"), "@@@@@@@") + "|"
   Print "|" + Format$(Format$(123.5, "$##0.00"), "@@@@@@@") + "|"
End Sub
```

Here is the output:

```
|  $1.50|
| $12.50|
|$123.50|
```

MORE INFORMATION

You can automatically generate the @ format string by using Len and String$ as in this example:

```
Function rFormat (value As Variant, fmt As String) As Variant
   rFormat = Format(Format(value, fmt), String$(Len(fmt), "@"))
End Function
```

Additional reference words: 2.00 3.00 align alignment right-justify
KBCategory:
KBSubcategory: EnvtRun

Programming a Delay Using the Timer Function

Article ID: Q96069

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 1.0, 2.0, and 3.0
- Standard and Professional Editions of Microsoft Visual Basic for MS-DOS, version 1.0

SUMMARY

You can delay execution of your code for a specific time interval by using the Timer function.

With Visual Basic for MS-DOS, you cannot use the SLEEP statement to do this while forms are showing. An attempt to do so causes this error:

   Invalid when forms are showing.

To use the Timer function to pause for a number of seconds, store the value of Timer in a variable. Then use a loop to wait until the Timer returns a a specified number of seconds greater than the stored value. If the delay loop will execute when midnight passes, compensate by reducing the starting Timer value by the number of seconds in a day (24 hours * 60 minutes * 60 seconds). Calling DoEvents from within the loop allows events to be processed during the delay.

MORE INFORMATION

Code Example

```
Sub Form_Click ()
   Print "hello ";
   Call Pause(2)  ' delay for 2 seconds
   Print "world"
End Sub

Sub Pause (ByVal nSecond As Single)
   Dim t0 As Single
   t0 = Timer
   Do While Timer - t0 < nSecond
      Dim dummy As Integer
      dummy = DoEvents()
      ' if we cross midnight, back up one day
      If Timer < t0 Then
         t0 = t0 - 24 * 60 * 60
      End If
   Loop
End Sub
```


Additional reference words: B_VBasic B_VBMSDOS 1.00 2.00 3.00 wait

KBCategory:
KBSubcategory: EnvtRun

How to Emulate Overtype Mode in a Visual Basic Text Box

Article ID: Q96210

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Although Visual Basic text boxes do not support an Overtype (replace) mode where the text you type replaces the text already there, you can write code to support it. The example below demonstrates one method for implementing an overtype mode in a Visual Basic text box.

MORE INFORMATION

Microsoft Visual Basic for Windows text box controls default to Insert mode, inserting the text you type rather than replacing what is already there. To emulate the Overtype mode, you need to add code to the KeyPress and KeyDown events for a text box. The following is an example.

Steps to Create Example Program

1. Run Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add one text box to Form1 named Text1.

3. Add one label box to Form1 named Label1.

4. Enter the following code in the Form1 General section:

```
Dim insert As Integer   'Insert flag
```

4. Enter the following code in the Form1 Load procedure:

```
Sub Form_Load ()
   insert = False        'initialize insert flag
   label1.Caption = "Insert"
End Sub
```

3. Enter the following code in the Text1 KeyPress procedure:

```
Sub Text1_KeyPress (keyascii As Integer)
   If keyascii <> 8 Then                'if keyascii not Backspace
      If insert Then                    'check insert flag
         string1$ = text1.Text
         pos& = text1.SelStart + 1      'get cursor position
         If (pos& > Len(string1$)) Then  'if cursor is at the end then
```

```
                                        'append keystroke to end
            string1$ = text1.Text + Chr$(keyascii)
         Else                           'else place keystroke in
                                        'correct position in text
            Mid(string1$, pos&, 1) = Chr$(keyascii)
         End If
         text1.Text = string1$
         text1.SelStart = pos&
         keyascii = 0
      End If
   End If
End Sub
```

4. Enter the following code in the Text1 KeyDown procedure:

```
Sub Text1_KeyDown (keycode As Integer, Shift As Integer)
   If keycode = 45 Then    'If the insert key was pressed
      insert = Not insert   'toggle insert flag
      If insert Then
         label1.Caption = "Overwrite"
      Else
         label1.Caption = "Insert"
      End If
   End If
End Sub
```

5. Press the F5 key to run the program. When you click Insert, the Label1 label shows the current mode of the Text1 text box.

Additional reference words: 2.00 3.00 overtype typeover replace
KBCategory:
KBSubcategory: EnvtRun

'Error in loading DLL' When LIBRARY Name Not Same as Filename

Article ID: Q98309

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The "Error in loading DLL" error message will occur if you call a DLL and the LIBRARY name of the DLL is different from the filename. This is by design. Visual Basic ensures that the LIBRARY name and filename of a DLL match. If they don't match, Visual Basic generates the "Error in loading DLL" error.

Visual Basic 3.0 does not require that the LIBRARY name and the filename be the same for a DLL. However, unless you are designing a DLL specifically to be called from Visual Basic 3.0 or some other application not written using Visual Basic, we recommend that you use the same name for both the LIBRARY name and filename of a DLL.

MORE INFORMATION

When creating a Windows DLL, you must specify the LIBRARY name of the DLL in the module-definition (.DEF) file for the DLL. In order to call any procedure contained within the DLL from Visual Basic, the LIBRARY name given in the module-definition file must be the same as the filename for the DLL.

Steps to Reproduce the Error Message

Perform the following steps to build a DLL that will lead to a "Error in loading DLL" error when called from Visual Basic. To build the following application, you will need to use a C compiler capable of creating Windows Dynamic Link Libraries (DLLs).

1. Create a C source code file that contains the following code and save the file as TEST.C.

```
#include <windows.h>

VOID FAR PASCAL test ( VOID );
VOID FAR PASCAL test ( VOID )
{
   //The contents of any procedure in the DLL is not important
   //Define this procedure to be called from Visual Basic
   return;
}
```

```
//--------------------------------------------------------------------
// Initialize library. This routine is called when the first
// client loads
// the DLL.
//--------------------------------------------------------------------
int FAR PASCAL LibMain
(
   HANDLE hModule,
   WORD   wDataSeg,
   WORD   cbHeapSize,
   LPSTR  lpszCmdLine
)
{
   // Avoid warnings on unused (but required) formal parameters
   wDataSeg = wDataSeg;
   cbHeapSize = cbHeapSize;
   lpszCmdLine = lpszCmdLine;

   return 1;
}

//--------------------------------------------------------------------
// WEP
//--------------------------------------------------------------------
int FAR PASCAL WEP(int fSystemExit);

//--------------------------------------------------------------------
// Performs cleanup tasks when the DLL is unloaded.  WEP() is
// called automatically by Windows when the DLL is unloaded (no
// remaining tasks still have the DLL loaded).  It is strongly
// recommended that a DLL have a WEP() function, even if it does
// nothing but returns success (1), as in this example.
//--------------------------------------------------------------------
int FAR PASCAL WEP
(
    int fSystemExit
)
{
    // Avoid warnings on unused (but required) formal parameters
    fSystemExit = fSystemExit;

    return 1;
}
```

2. Create a module-definition file (DEF) file that contains the following code and save the file as TEST.DEF.

```
LIBRARY DIFFNAME

DESCRIPTION 'Sample DLL where LIBRARY name != filename'

EXETYPE WINDOWS

CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD SINGLE MOVEABLE

EXPORTS
```

```
WEP @1 RESIDENTNAME
TEST @2
```

3. Compile TEST.C from the command line as follows:

```
CL /c /ASw /W3 TEST.C
```

4. Link the resulting TEST.OBJ file as follows:

```
LINK /NOE /NOD TEST.OBJ+LIBENTRY.OBJ,TEST.DLL,,LIBW+SDLLCEW,TEST.DEF;
```

5. Copy TEST.DLL to the Windows directory.

6. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

7. Add the following code to the general declarations section of Form1:

```
Declare Sub Test Lib "TEST.DLL" ()
```

8. Add the following code to the Form_Load event of Form1:

```
Sub Form_Load ()
   Call TEST
End Sub
```

9. From the Run menu, choose Start (ALT, R, S) or press the F5 key to run the program.

Execution will break on the Call statement in the Form_Load event, and you will receive the error "Error in loading DLL."

To avoid this error, change the LIBRARY name in TEST.DEF, under step 2, from DIFFNAME to TEST. Then do step 4 to link in the new module-definition file. Follow steps 5 through 8 again and you should no longer see the "Error in loading DLL" error message.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: EnvtRun

PRB: Some ATI Video Drivers Hang When Using MSOUTLIN.VBX

Article ID: Q100194

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic programming system for Windows, version 3.0

SYMPTOMS

If you use an OutLine control in a Visual Basic project and you are using an ATI Mach 32 video driver this could cause your computer to hang (stop responding to input).

CAUSE

This is a problem with the ATI video driver not a problem with Visual Basic. The m32-86.drv and Mach32.drv drivers have been reported to cause this problem.

RESOLUTION

An updated driver may solve the problem. To contact ATI Technologies concerning an updated driver call the following number.

ATI Technologies Inc. (416) 756-0711 ATI technical support

Additional reference words: 3.00
KBCategory:
KBSubcategory: EnvtRun PrgCtrlsCus

Category Keywords for All Visual Basic KB Articles

Article ID: Q108753

The information in this article applies to:

- Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Each article in the Visual Basic for Windows collection contains at least one keyword (called a KBSubcategory keyword) that places the article in an appropriate category. This article lists all the KBSubcategory keywords.

MORE INFORMATION

Category & Subcategory Description	KBSubcategory Keyword
Setup / Installation (Setins)	Setins
Environment-specific Issues (Envt)	
VB Design Environment	EnvtDes
Run-Time Environment	EnvtRun
Programming (Prg)	
Visual Basic Forms and Controls	
Standard Controls / Forms	PrgCtrlsStd
Custom Controls	PrgCtrlsCus
Third-Party Controls	PrgCtrlsThird
Optimization	
Memory Management	PrgOptMemMgt
General Optimization Tips	PrgOptTips
General VB Programming	PrgOther
Advanced programming (APrg)	
Network	APrgNet
Windows Programming (APIs / DLLs)	
Printing	APrgPrint
Graphics	APrgGrap
Windowing	APrgWindow
INI Files	APrgINI
Other API / DLL Programming	APrgOther
Data Access	
ODBC	APrgDataODBC
IISAM	APrgDataIISAM
Access	APrgDataAcc
General Database Programming	APrgDataOther
3rd Party DLL's	APrgThirdDLL

Inter-Application Programmability (IAP)

OLE	IAPOLE
DDE	IAPDDE
3rd Party Interoperability	IAPThird

Tools (Tls)

Setup Toolkit / Wizard	TlsSetWiz
Control Development Kit (CDK)	TlsCDK
Help Compiler (HC)	TlsHC

References (Refs)

Documentation / Help File Fixes	RefsDoc
Product Information	RefsProd
Third-Party Information	RefsThird
PSS-Only Information	RefsPSS

Using Keywords to Query the KB

At Microsoft, we use the subcategory keywords to organize the articles for Help files and for the FastTips Catalog. You can use them to query the Microsoft Knowledge Base for Visual Basic articles that apply to that category or subcategory. For example, you can find all the general database programming articles by querying on the following words in the Microsoft Knowledge Base:

```
visual and basic and APrgDataOther
```

Use the asterisk (*) wildcard to find articles that fall into the general categories or into an intermediate subcategory. The first element in each keyword is the category. For example, to find all the articles that apply to Visual Basic Forms and Controls regardless of whether they are standard, custom, or third-party controls, use the following words to query the Microsoft Knowledge Base:

```
visual and basic and PrgCtrls*
```

To find all advanced programming articles, query on these words:

```
visual and basic and APrg*
```

Add KBSubcategory Keyword to Each Article

When contributing an article to the Visual Basic Knowledge Base, add the appropriate KBSubcategory keyword to the bottom of the article on the KBSubcategory line. Each article in the Visual Basic for Windows collection contains the following section at the bottom of the article:

```
Additional reference words:
KBCategory:
KBSubcategory: <keyword>
```

An article usually has only one subcategory keyword, but it may have more.

If you are interested in contributing, please obtain the guidelines by

querying on the following words in the Microsoft Knowledge Base:

   visual and basic and kbguide and kbartwrite

Additional reference words: 3.00 dskbguide subcatkey
KBCategory:
KBSubcategory: RefsPSS

PRB: Making .EXE Gives Error: Wrong Version of Runtime DLL

Article ID: Q112770

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

If a copy of an executable is running on your computer, and you try to create a new executable with the same name, you'll receive this error:

   Wrong version of runtime DLL.

followed by another error:

   Unexpected error quitting.

CAUSE

This error is caused by the fact that Windows doesn't load a new copy of a module if it thinks the module is already in memory. Because one instance of your .EXE file is running already, attempting to run the re-compiled version really just increments the usage count of the .EXE that is already running.

The Visual Basic error occurs because VBRUN300.DLL tries to open the .EXE file and read resources out of it. But the .EXE file you're reading from is different from the .EXE file that is actually running (because you did a 'Make EXE' over the old one), so VBRUN300 has problems. It signals with the "Wrong version of run-time DLL" error. It reports this error because it is under the assumption that because it could not read the .EXE file, it doesn't have the correct version of VBRUN needed to execute that .EXE file.

RESOLUTION

To work around the problem, either rename the .EXE when you create a new executable, or exit from the executable that is currently running.

Additional reference words: 3.00
KBCategory: Envt
KBSubcategory: EnvtRun

(Complete) Tutorial to Understand IEEE Floating-Point Errors

Article ID: Q42980

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic Programming System for Windows, versions 1.0, 2.0, and 3.0
- The Standard and Professional Editions of Microsoft Visual Basic for MS-DOS, version 1.0
- Microsoft QuickBasic for MS-DOS, versions 3.0 (QB87.EXE coprocessor version only), 4.0, 4.0b, and 4.5
- Microsoft Basic Compiler for MS-DOS and MS OS/2, versions 6.0 and 6.0b
- Microsoft Basic Professional Development System (PDS) for MS-DOS and MS OS/2, versions 7.0 and 7.1

SUMMARY

Floating-point mathematics is a complex topic that confuses many programmers. The tutorial below should help you recognize programming situations where floating-point errors are likely to occur and how to avoid them. It should also allow you to recognize cases that are caused by inherent floating-point math limitations as opposed to actual compiler bugs.

MORE INFORMATION

Decimal and Binary Number Systems

Normally, we count things in base 10. The base is completely arbitrary. The only reason that people have traditionally used base 10 is that they have 10 fingers, which have made handy counting tools.

The number 532.25 in decimal (base 10) means the following:

```
   (5 * 10^2) + (3 * 10^1) + (2 * 10^0) + (2 * 10^-1) + (5 * 10^-2)
       500    +     30     +      2     +     2/10    +    5/100

   _________
   =  532.25
```

In the binary number system (base 2), each column represents a power of 2 instead of 10. For example, the number 101.01 means the following:

```
   (1 * 2^2) + (0 * 2^1) + (1 * 2^0) + (0 * 2^-1) + (1 * 2^-2)
       4     +      0    +     1     +      0     +    1/4

   _________
   =  5.25  Decimal
```

How Integers Are Represented in PCs

Because there is no fractional part to an integer, its machine representation is much simpler than it is for floating-point values. Normal integers on personal computers (PCs) are 2 bytes (16 bits) long with the most significant bit indicating the sign. Long integers are 4 bytes long. Positive values are straightforward binary numbers. For example:

```
 1 Decimal = 1 Binary
 2 Decimal = 10 Binary
22 Decimal = 10110 Binary, etc.
```

However, negative integers are represented using the two's complement scheme. To get the two's complement representation for a negative number, take the binary representation for the number's absolute value and then flip all the bits and add 1. For example:

```
4 Decimal = 0000 0000 0000 0100
            1111 1111 1111 1011     Flip the Bits
-4        = 1111 1111 1111 1100     Add 1
```

Note that -1 Decimal = 1111 1111 1111 1111 in Binary, which explains why Basic treats -1 as logical true (All bits = 1). This is a consequence of not having separate operators for bitwise and logical comparisons. Often in Basic, it is convenient to use the code fragment below when your program will be making many logical comparisons. This greatly aids readability.

```
CONST TRUE = -1
CONST FALSE = NOT TRUE
```

Note that adding any combination of two's complement numbers together using ordinary binary arithmetic produces the correct result.

Floating-Point Complications

Every decimal integer can be exactly represented by a binary integer; however, this is not true for fractional numbers. In fact, every number that is irrational in base 10 will also be irrational in any system with a base smaller than 10.

For binary, in particular, only fractional numbers that can be represented in the form p/q, where q is an integer power of 2, can be expressed exactly, with a finite number of bits.

Even common decimal fractions, such as decimal 0.0001, cannot be represented exactly in binary. (0.0001 is a repeating binary fraction with a period of 104 bits!)

This explains why a simple example, such as the following

```
SUM = 0
FOR I% = 1 TO 10000
   SUM = SUM + 0.0001
NEXT I%
PRINT SUM                   ' Theoretically = 1.0.
```

will PRINT 1.000054 as output. The small error in representing 0.0001 in binary propagates to the sum.

For the same reason, you should always be very cautious when making comparisons on real numbers. The following example illustrates a common programming error:

```
item1# = 69.82#
item2# = 69.20# + 0.62#
IF item1# = item2# then print "Equality!"
```

This will NOT PRINT "Equality!" because 69.82 cannot be represented exactly in binary, which causes the value that results from the assignment to be SLIGHTLY different (in binary) than the value that is generated from the expression. In practice, you should always code such comparisons in such a way as to allow for some tolerance. For example:

```
IF (item1# < 69.83#) AND (item1# > 69.81#) then print "Equal"
```

This will PRINT "Equal".

IEEE Format Numbers

QuickBasic for MS-DOS, version 3.0 was shipped with an MBF (Microsoft Binary Floating Point) version and an IEEE (Institute of Electrical and Electronics Engineers) version for machines with a math coprocessor. QuickBasic for MS-DOS, versions 4.0 and later only use IEEE. Microsoft chose the IEEE standard to represent floating-point values in current versions of Basic for the following three primary reasons:

1. To allow Basic to use the Intel math coprocessors, which use IEEE format. The Intel 80x87 series coprocessors cannot work with Microsoft Binary Format numbers.

2. To make interlanguage calling between Basic, C, Pascal, FORTRAN, and MASM much easier. Otherwise, conversion routines would have to be used to send numeric values from one language to another.

3. To achieve consistency. IEEE is the accepted industry standard for C and FORTRAN compilers.

The following is a quick comparison of IEEE and MBF representations for a double-precision number:

	Sign Bits	Exponent Bits	Mantissa Bits
IEEE	1	11	52 + 1 (Implied)
MBF	1	8	56

For more information on the differences between IEEE and MBF floating-point representation, query in the Microsoft Knowledge Base on the following words:

IEEE and floating and point and appnote

Note that IEEE has more bits dedicated to the exponent, which allows it to represent a wider range of values. MBF has more mantissa bits, which allows it to be more precise within its narrower range.

General Floating-Point Concepts

It is very important to realize that any binary floating-point system can represent only a finite number of floating-point values in exact form. All other values must be approximated by the closest representable value. The IEEE standard specifies the method for rounding values to the "closest" representable value. QuickBasic for MS-DOS supports the standard and rounds according to the IEEE rules.

Also, keep in mind that the numbers that can be represented in IEEE are spread out over a very wide range. You can imagine them on a number line. There is a high density of representable numbers near 1.0 and -1.0 but fewer and fewer as you go towards 0 or infinity.

The goal of the IEEE standard, which is designed for engineering calculations, is to maximize accuracy (to get as close as possible to the actual number). Precision refers to the number of digits that you can represent. The IEEE standard attempts to balance the number of bits dedicated to the exponent with the number of bits used for the fractional part of the number, to keep both accuracy and precision within acceptable limits.

IEEE Details

Floating-point numbers are represented in the following form, where [exponent] is the binary exponent:

```
X =  Fraction * 2^(exponent - bias)
```

[Fraction] is the normalized fractional part of the number, normalized because the exponent is adjusted so that the leading bit is always a 1. This way, it does not have to be stored, and you get one more bit of precision. This is why there is an implied bit. You can think of this like scientific notation, where you manipulate the exponent to have one digit to the left of the decimal point, except in binary, you can always manipulate the exponent so that the first bit is a 1, since there are only 1s and 0s.

[bias] is the bias value used to avoid having to store negative exponents.

The bias for single-precision numbers is 127 and 1023 (decimal) for double-precision numbers.

The values equal to all 0's and all 1's (binary) are reserved for representing special cases. There are other special cases as well, that indicate various error conditions.

Single-Precision Examples

 2 = 1 * 2^1 = 0100 0000 0000 0000 ... 0000 0000 = 4000 0000 hex
    Note the sign bit is zero, and the stored exponent is 128, or
    100 0000 0 in binary, which is 127 plus 1. The stored mantissa is
    (1.) 000 0000 ... 0000 0000, which has an implied leading 1 and
    binary point, so the actual mantissa is 1.

-2 = -1 * 2^1 = 1100 0000 0000 0000 ... 0000 0000 = C000 0000 hex
    Same as +2 except that the sign bit is set. This is true for all
    IEEE format floating-point numbers.

 4 = 1 * 2^2 = 0100 0000 1000 0000 ... 0000 0000 = 4080 0000 hex
    Same mantissa, exponent increases by one (biased value is 129, or
    100 0000 1 in binary.

 6 = 1.5 * 2^2 = 0100 0000 1100 0000 ... 0000 0000 = 40C0 0000 hex
    Same exponent, mantissa is larger by half -- it's
    (1.) 100 0000 ... 0000 0000, which, since this is a binary
    fraction, is 1-1/2 (the values of the fractional digits are 1/2,
    1/4, 1/8, etc.).

 1 = 1   * 2^0 = 0011 1111 1000 0000 ... 0000 0000 = 3F80 0000 hex
    Same exponent as other powers of 2, mantissa is one less than
    2 at 127, or 011 1111 1 in binary.

.75 = 1.5 * 2^-1 = 0011 1111 0100 0000 ... 0000 0000 = 3F40 0000 hex
    The biased exponent is 126, 011 1111 0 in binary, and the mantissa
    is (1.) 100 0000 ... 0000 0000, which is 1-1/2.

2.5 = 1.25 * 2^1 = 0100 0000 0010 0000 ... 0000 0000 = 4020 0000 hex
    Exactly the same as 2 except that the bit which represents 1/4 is
    set in the mantissa.

0.1 = 1.6 * 2^-4 = 0011 1101 1100 1100 ... 1100 1101 = 3DCC CCCD hex
    1/10 is a repeating fraction in binary. The mantissa is just shy
    of 1.6, and the biased exponent says that 1.6 is to be divided by
    16 (it is 011 1101 1 in binary, which is 123 in decimal). The true
    exponent is 123 - 127 = -4, which means that the factor by which
    to multiply is 2**-4 = 1/16. Note that the stored mantissa is
    rounded up in the last bit. This is an attempt to represent the
    unrepresentable number as accurately as possible. (The reason that
    1/10 and 1/100 are not exactly representable in binary is similar
    to the way that 1/3 is not exactly representable in decimal.)

0   = 1.0 * 2^-128 = all zeros -- a special case.

Other Common Floating-Point Errors

The following are common floating-point errors:

1. Round-off error

   This error results when all of the bits in a binary number cannot
   be used in a calculation.

Example: Adding 0.0001 to 0.9900 (Single Precision)

Decimal 0.0001 will be represented as:

(1.)10100011011011100010111 * 2^(-14+Bias) (13 Leading 0s in Binary!)

0.9900 will be represented as:

(1.)11111010111000010100011 * 2^(-1+Bias)

Now to actually add these numbers, the decimal (binary) points must be aligned. For this they must be Unnormalized. Here is the resulting addition:

```
 .00000000000000011010001101 * 2^0  <- Only 11 of 23 Bits retained
+.11111101011100001010011 * 2^0
_____________________________________
 .11111101011101110011000 * 2^0
```

This is called a round-off error because some computers round when shifting for addition. Others simply truncate. Round-off errors are important to consider whenever you are adding or multiplying two very different values.

2. Subtracting two almost equal values

```
 .1235
-.1234
 _____
 .0001
```

This will be normalized. Note that although the original numbers each had four significant digits, the result has only one significant digit.

3. Overflow and underflow

This occurs when the result is too large or too small to be represented by the data type.

4. Quantizing error

This occurs with those numbers that cannot be represented in exact form by the floating-point standard.

5. Division by a very small number

This can trigger a "divide by zero" error or can produce bad results, as in the following example:

```
A = 112000000
B = 100000
C = 0.0009
X = A - B / C
```

In QuickBasic for MS-DOS, X now has the value 888887, instead of the correct answer, 900000.

6. Output error

   This type of error occurs when the output functions alter the values they are working with.

Additional reference words: 1.00 2.00 3.00 4.00 4.00b 4.50 6.00 6.00b 7.00 7.10 IEEETUTR
KBCategory:
KBSubcategory: RefsProd

How to Contribute Visual Basic Articles to the Microsoft KB

Article ID: Q70220

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic Programming System for Windows, version 3.0

SUMMARY

This article explains how you can contribute Visual Basic articles to the Microsoft Knowledge Base.

MORE INFORMATION

There are at least four different ways you can contribute to the Microsoft Knowledge Base:

- Contribute a fully-tested code sample containing ample comments and a short description.
- Contribute a rough draft of an article that is complete with fully-tested technical information but that needs rewriting, editing, and formatting.
- Contribute a fully-tested, well-written article that needs formatting.
- Contribute a fully-tested, well-written article in the standard format.

All of these are equally valuable. Choose the method that is most comfortable and quickest for you to do. In all cases, you will be given credit for the article either in the Author field or in the body of the article.

Submission Guidelines

When submitting an article, please ensure that it is fully tested and includes all the following information:

- Name of the author(s) or contributor(s)
- Name of the person(s) who signed off on the technical review
- Product name(s)
- Product version number(s)
- Operating system(s)

Contributions from Microsoft Employees

After your article has been technically reviewed, include the name of the technical reviewer in your article, and mail it to the Knowledge Base Lead (KBL) for your product. The KBL will make sure the article is added to the KB and edited or rewritten as quickly as possible.

Contributions from Outside Microsoft

If you are not a Microsoft employee, you can still contribute. You need to find a Microsoft employee to sponsor you and review your article for technical accuracy.

If you need us to assign a sponsor for you, please send mail to:

   y-kbfeed@Microsoft.com

on the Internet. You can do this in CompuServe Mail by putting the following on the TO: line of your message:

   >Internet: y-KBFeed@Microsoft.Com

In the body of the message, please include the name of the product name, a short summary of your article, and your telephone number. We will have a Microsoft sponsor call you to discuss your article. The sponsor will advise you on the appropriateness of your article for the KB. If your article is selected for the KB, your name will be included in the body of the article as the contributor so that readers can contact you directly if they have questions.

Writing Style

You don't have to be a writer to contribute to the KB. The Developer Support Knowledge Base team can rewrite your material for you. All we need is complete, well-tested technical information.

If you would prefer to write your own article, please follow these and the other general guidelines listed in this article:

- Emulate the writing style and standards used in the Visual Basic manuals and Help menu.

- Use active voice, present tense, short sentences, and bullets to make your writing clear, crisp, and easy for the reader to follow.

- When writing about bugs, list the symptoms first followed by the cause (if known), and then the workaround or resolution.

- Spell check your article.

The goal of good writing is to be as invisible as possible to the reader.

Format Style

If you want to put your article in standard format, follow these guidelines. Each article should be organized into sections. There are two basic format styles:

- Informational and how to articles use the SUMMARY/MORE INFORMATION format style. This article is an example of this format.

- Bugs and other problem articles use the SYMPTOMS/CAUSE/RESOLUTION

format style. For an example, query on the following words in the Microsoft Knowledge Base:

visual and basic and bug

Look at any one of the BUG or FIX articles. Many of these articles have a WORKAROUND section instead of a RESOLUTION section. In addition, many also include a STATUS section that either confirms the problem as a bug or tells the reader that the behavior described in the article is by design.

Reference Section

Many articles also include a REFERENCES section at the bottom of the article. When referring to the manuals or products, please use the official names. For example:

- Microsoft Visual Basic Programming System for Windows, version 3.0, Professional Edition, "Professional Features Book 1," page 161, "SeeThru Property" section.

- Microsoft Visual Basic Programming System for Windows, version 3.0, "Language Reference," page 333, "List Property" section.

Categorization and Reference Keywords

Add the following section to the bottom of each article:

Additional reference words:
KBCategory:
KBSubcategory:

On the top line, fill in any reference words that you think may be useful when querying for the article

Leave the middle line blank.

On the bottom line, fill in the appropriate Visual Basic subcategory keywords. To find the list of available keywords, query on the following words in the Microsoft Knowledge Base:

visual and basic and kbguide and subcatkey

Boilerplates

There are several boilerplates that go on many if not all articles. Put the following at the top of every article to identify the product and version numbers. Modify this boilerplate to give the actual product name:

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic Programming System for Windows, version 3.0

If Microsoft has confirmed that the problem described in the article is a bug, please add the following STATUS section below the WORKAROUND or RESOLUTION section:

STATUS
======

Microsoft has confirmed this to be a bug in the product(s) listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Miscellaneous Guidelines

- Always use the complete name of the product. For example, use Visual Basic, not VB. Use Microsoft Access, not Access.

- Use the uppercase and lowercase standards as shown in the manuals. For example, controls such as command button and list box are in lowercase.

- Write menu instructions using this wording:

  From the File menu, choose Save Project.

- When you give code examples, always use Visual Basic's default control and procedure names, such as Command1, List1, Option1. This will avoid reader misunderstandings.

- When you describe how to create a Visual Basic form, use complete sentences, and use the exact terminology used in the manual. Avoid using line-drawing characters in an article because of editing and interpretation problems. Instead, describe in complete sentences what controls go where and in what order.

- Use active voice and engage the reader by using the second person (you) instead of using "the user," "the customer," or "one." Use "user" only when it is necessary to differentiate the user from the developer.

- Use active voice such as: After you start Visual Basic, use x, call the y function, and click the z button. Avoid passive voice such as: After Visual Basic is started, x is used, the y function is called, and the z button is clicked.

  After writing your article, check for passive voice by looking at your use of the words have, is, and are. Where possible, change passive voice to active voice.

- Enclose error messages in double quotation marks, and be sure to quote the exact error message displayed on the screen by the software.

- Attempt to structure your article in a series of steps. There are many examples of this throughout the Visual Basic Knowledge Base.

- List all version numbers mentioned in the article to the hundredths

place in the following section at the bottom of every article. This will satisfy all potential article-body queries in the online services.

Additional reference words: 1.00 2.00 3.00 dskbguide kbartwrite
KBCategory:
KBSubcategory: RefsPSS

Why Cooper Software Is Listed in Visual Basic's Copyright

Article ID: Q72747

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0 and 2.0

SUMMARY
=======

The Microsoft Visual Basic copyright notice acknowledges Cooper Software in both the sign-on dialog box and in the About dialog box from the Help menu. Visual Basic uses technology from a forms engine purchased from Cooper Software. The acknowledgment in Visual Basic is part of the contract between Microsoft and Cooper Software.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: RefsThird

Technical Data Sheets Available for Visual Basic for Windows

Article ID: Q77906

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY
=======

You can obtain detailed sales literature in the United States by calling Microsoft End User Sales at (800) 426-9400. Ask for the Visual Basic for Windows package or the Visual Basic for MS-DOS package.

Outside of the United States, you can obtain these data sheets by contacting your local Microsoft subsidiary or dealer.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: RefsPSS

Visual Basic Online Help Example Errors

Article ID: Q78772

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

There are several code examples in Visual Basic version 1.0's online Help that do not behave as expected if actually copied and run. The corresponding examples in the "Visual Basic: Language Reference" manual contain the same errors.

STATUS

These problems do not exist in later versions of Visual Basic for Windows.

MORE INFORMATION

Under the topic "ActiveControl, Active Form Properties," the second example demonstrating these properties contains an omission of the Clipboard object. When copied and run as is, the error message "Method Not Applicable For This Object" will be displayed.

The ActiveControl example contains the line in the EditCut_Click event procedure for the menu item:

```
   SetText Screen.ActiveControl.SelText
```

This should be changed to read:

```
   Clipboard.SetText Screen.ActiveControl.SelText
```

Under the topic titled "Fonts Property," the example shows setting the FontName = "". This will cause a run time error "Invalid Property Value".

The example will also fail when attempting to select a printer FontName as the screen FontName where no associated screen font exists under Windows. For example, when the printer LinePrinter font is selected for the screen, an error will occur because the screen does not support this font. The examples for the topics "FontName Property" and "FontCount Property" if modified as suggested in the online Help to print the available printer fonts to the screen will fail for the same reason.

The example for the Fonts Property follows:

```
    'Fonts Property Example
```

```
    Sub Form_Click ()

        Static X%       ' A static variable.
        AutoRedraw = -1 ' Keep screen text.
        If X% = Printer.FontCount Then  ' Check for last font.
                X% = 0  ' Set X%.
                Print   ' Print blank line.
                FontName = ""   ' Reset to default font.
        End If
        If X% = 0 Then Print "Printer Fonts"    ' Print header.
        FontName = Printer.Fonts(X%)    ' Set FontName.
        Print X%; "This is " + FontName + " font"  ' Print message.
        X% = X% + 1     ' Set X%.

    End Sub
```

As stated above, this fails in two ways. The line resetting the FontName property is syntactically incorrect. Also, the logic may fail because of no corresponding screen font for the printer font.

Modifying the example to address both problems requires an On Error trap routine and saving the values of FontName, FontBold, and FontSize explicitly. The following example works properly.

```
    'Fonts Property Example
    Sub Form_Click ()
        On Error GoTo errHandler
        Static x%   ' A static variable.
        Static savename$, savebold%, savesize' <<< added this!!

        AutoRedraw = -1 ' Keep screen text.
        If x% = Printer.fontcount Then  ' Check for last font.
            x% = 0                ' Set X%.
            FontName = savename$ ' <<<add this! Reset to default font.
            Fontsize = savesize  '  and this
            Fontbold = savebold% '  and this
            Print                ' Print blank line.
        End If
        If x% = 0 Then
            Print "Printer Fonts"    ' Print header.
            savename$ = FontName      ' save all these
            savebold% = Fontbold      ' to original settings
            savesize = Fontsize       ' now
        End If
        FontName = Printer.Fonts(x%)   ' Set FontName.
        Print x%; "This is " + FontName + " font"  ' Print message.

    ExitSub:
        x% = x% + 1 ' Set X%.

    Exit Sub

    errHandler:

        Print x%; "This is " + Printer.Fonts(x%) + " name"
        Resume ExitSub:
```

```
    End Sub
```

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: RefsDoc

List of Visual Basic Companion Products and Services Available

Article ID: Q78962

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

A file is available that lists Visual Basic companion products and services available as of March 15, 1992. This file can be found in the Software/Data Library by searching on the word VBADDONS, the Q number of this article, or S13242. VBADDONS was archived using the PKware file-compression utility.

The VBADDONS file ("Visual Basic Companion Products and Services") contains the following sections:

- Custom controls and .DLLs
- Data access/connectivity
- Graphics utilities and clip-art
- Publications and services
- Trademarks
- Where to send additions or corrections

Additional reference words: 1.00 third-party add-on
KBCategory:
KBSubcategory: RefsThird

LONG: Visual Basic Companion Products & Services (Complete)

Article ID: Q78963

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article lists Visual Basic companion products and services.

MORE INFORMATION

This list is organized into the following sections:

- Custom Controls and .DLLs
- Data Access
- Report Writers
- LAN\Host Connectivity and Communications
- Pen\Multimedia
- Visual Basic Libraries and Tools
- Windows Programming Tools and Utilities
- Help File authoring tools
- Graphics Utilities and Clip-Art
- Publications and Training

For the latest version of this list, please obtain "Component Objects and Companion Products for Visual Basic," which is available from:

Fawcette Technical Publications
280 Second Street, Suite 200,
Los Altos, CA 94022-3603
415-917-7650
$7 per copy ($4 each for orders of 20 or more).

CUSTOM CONTROLS AND .DLLS

ADDE, 17, Rue Louise Michel, 92301 Levallois-Perret, France
Contact: Xavier Ledur +33-1-47-58-78-41
Map Custom Control -- a window in which one or more overlapped geographical maps (cities, countries, networks) are displayed. Zooming and positioning functions are integrated in the control. The control can load maps from "Cartes et Bases Windows," an existing Microsoft Windows graphical environment map package. Sample maps from ADDE catalog come with it.

Autodesk, Inc., 2320 Marinship Way, Sausalito, CA 94965
Contact: (415) 332-2344
Fax: (415) 331-8093
Autodesk Animation Player for Visual Basic -- dynamic link library that provides Visual Basic users with easy to use animation functionality to add

to Windows applications. The control plays industry-standard FLI and FLC animations from hard disks or CD-ROMs.

Crescent Software, Inc., 11 Bailey Ave, Ridgefield, CT 06877 USA
Contact: Don Malin (203) 438-5300
Fax: (203) 431-4626
QuickPak Professional for Windows -- contains custom controls and a general purpose set of utilities for use with Microsoft Visual Basic programming system. QuickPak Professional for Windows provides routines for quickly sorting and searching data, performing fast file operations, expression evaluation, and other useful tasks.

Desaware, 5 Town & Country Village #790, San Jose, CA 95128
Contact: Gabriel Appleman (213) 943-3305 or Dan Appleman (408) 377-4770
Fax: (408) 371-3530
CompuServe: Dan Appleman 70303,2252
Custom Control Factory -- an interactive development tool for creating custom controls including Animated Pushbuttons, Multistate Buttons, enhanced buttons, checkbox and option button controls for Windows applications.
CCF-Cursors -- provides you with complete control over cursors (mouse pointers) in Visual Basic applications. Create your own cursors or convert icons to cursors, and much more. Includes over 50 cursors.
SpyWorks-VB -- is an advanced development tool for use with Visual Basic. The SpyWorks-VB package supports true owner-draw list boxes, in which you can draw each entry under program control in any manner that you wish.

FarPoint Technologies, Inc., 585A Southlake Boulevard,
Southport Office Park Richmond, VA 23236
Sales/Info: (800) 645-5913 or (804) 378-0432
Tech Support: (804) 378-1011
FAX: (804) 378-1015
Visual Architect for Visual Basic -- Custom controls and other tools for creating advanced applications, including a complete spreadsheet control.

INSYS, 268 Rue du Faubourg Saint-Antoine, 75012 Paris, France
Contact: M. Quentin +33-1-40-04-6-36
Insys Classes -- a collection of Visual Basic custom controls for business oriented computing and communications applications, including: structured text fields (numeric, alphanumeric, masked input), hierarchical list boxes, structured list boxes, date/time management controls with spin buttons, CPIC control, and a simple spreadsheet control.

MicroHelp, Inc., 4359 Shallowford Industrial Parkway, Marietta, GA 30066
Contact: Mark Novisoff (404) 516-0899 or 1-800-922-3383
Fax: (404) 516-1099
VB Tools 2.0 -- designed to add "pizazz" to Visual Basic programs. It includes over 30 custom controls such as a grid, MDI child windows, 256 color control, and icon tag: also information on how to use Windows API services, ASM routines, utility modules, a program providing $INCLUDE capabilities, and much more.
MicroHelp Muscle -- libraries to include in Visual Basic applications.

OutRider Systems, P.O. Box 271669, Houston, TX 77277-1669
Contact: Jim Nech (713) 521-0486
ButtonTool -- custom control that enables developers to create many new button types and styles using bitmaps, icons, and metafiles as backgrounds.

Edit Tool -- custom control mask for a custom edit box that formats date, time, dollar, and numerical data.

Pinnacle Publishing, P.O. Box 8099, Federal Way, WA 98003
Contact: David Johnson (800) 231-1293 or (206) 941-2300
Graphics Server for Visual Basic -- custom control for integrating graphing and charting capabilities into Visual Basic applications. It includes pie charts, bar charts and a variety of other graphs in 2D or 3D.

Sheridan Software Systems, Inc., 65 Maxess Road, Melville, NY 11747
Contact: Joseph Modica (516) 753-0985
Fax (516) 293-4155
BBS numbers: 2400 Baud: (516) 753-5452 9600 baud: (516) 753-6510
VB Extenders -- 3-D Widgets (Versions 1, 2, and 3) are collections of custom controls that support three-dimensional text boxes and controls on Visual Basic forms. It includes standard set of six, plus controls for more advanced functionality such as list boxes and menu options.

Software Paths Ltd., Clonmel House, 17 Harcourt Street
Dublin 2, Ireland
Contact: 010 353 1 780039
Fax : 010 353 1 780142
Data Validation Control -- offers automatic data-validation for text, integer, floating point, date, time and currency values. Text validation uses regular expressions, allowing complex pattern matching to be provided automatically. Time, date and currency validation use the international settings from WIN.INI or may be specified by the programmer.

TeraTech, 3 Choke Cherry Road, Suite 360, Rockville, MD 20850
Contact: (301) 424-3903
Fax: (301) 762-8185
Dazzle/VB -- custom control that displays realistic images in Visual Basic (256 color) with Dazzle's special effects (wipe, fade); or zoom or adjust colors. Also available in a professional version with true grey scale support, on the fly compression, color support, and palette control.
Creating Visual Basic Custom Controls and .DLLs -- programmers can create Dynamic Link Libraries (DLLs) that are callable from Visual Basic using any of the following language tools. The .DLL must use Pascal calling conventions (the standard for Microsoft Windows). Custom controls are created with the Control Development Kit.

Microsoft C/C++ 7.0	Microsoft Macro Assembler 6.0
Microsoft FORTRAN 5.1	Microsoft COBOL 4.5
Microsoft Quick C for Windows	Borland Turbo Pascal for Windows
Borland C++	Watcom C
Zortech C++	

DATA ACCESS

Aaerdeus, Inc., 302 College Avenue, Palo Alto, CA 94306
Contact: Randy Burns (415) 325-7529
SQL Express -- dynamic link library and set of sample programs that allow Microsoft SQL Server to be used with Microsoft Visual Basic.

Abacus Accounting Systems Inc., P.O. Box 3835, Postal Station "D"
Edmonton, AB T5L 4K1 10335-172 Street, Suite 208, Edmonton,
AB T5S 1K9 Canada

Contact: Tom Dawson (403) 489-5994
Fax: (403) 486-4335
vxBase -- DLL that allows Visual Basic programmers to create xBASE applications for Windows in hours. It's all in the functions: vxAppendBlank through vxZap -- 86 functions in all. Browse object supports user-definable tables, on-screen editing, and visual relationships. Available as shareware on MSBASIC forum on CompuServe or directly from Comsoft.

Akros, Inc., 115 N. Center Street, Ste. 204, Northville, MI 48167
Contact: (313) 347-3556
Fax: (313) 347-3765
VBPX -- provides a seamless interface between Visual Basic and the Borland Paradox Engine. Contains over 50 functions in a single DLL for single and multi-user support, sample application (with source code) and is runtime and royalty free. Provides cost-effective application development.

AJS, PO Box 83220, Los Angles, CA 90083
Contact: Jim Taylor (800) 992-3383 or (310)215-9145
Visual/db -- Visual Basic developers can access dBase-compatible data and index files using the Visual/db Source Document Relational Database Management System. This is a standalone, single user, DBMS engine that includes VB source code for creating stand alone applications.

Apex Software Corporation, 4516 Henry Street, Suite 401
Pittsburgh, PA 15213
Tel. (412) 681-4343
Contact: Richard F. DiGiovani (818) 594-7293
Agility/VB -- a database developer's tool for Visual Basic based on Apex's powerful Apex Database Library. It is provided as a set of DLL functions callable from Visual Basic programs, which the programmer defines and relates to each other using a graphical View Editor. Provides complete access to dBASE IV compatible files.

Blue Rose Software, Box 29574 Atlanta, GA 30359-0574
Contact: Richard Denton (404) 717-1225
DATABasic -- a B-tree database engine for use with Visual Basic featuring speed, flexibility, small libraries, ease of maintenance, and rapid software development. It provides an integrated development environment. DATABasic eliminates an entire class of programming bugs -- synchronization bugs between code and databases. (Includes source code at no extra charge.)

Borland International, 1800 Green Hill Rd, Scotts Valley, CA 95067
Contact: 408-439-1639
Paradox Engine Version 2.0 -- includes DLL for developing Windows applications. You can create, read, and write Paradox tables, records, and fields. Supports multi-user database functions such as multi-user file locking, record locking, and password protection. Applications created with the Paradox engine ship run time and royalty free.

Channel Computing, Inc., 53 Main Street, Newmarket, NH 03857
Contact: Max Klein (603) 659-2832
Forest & Trees -- a Data Access and Reporting Tool that lets Visual Basic system users build an "electronic dashboard" to collect, combine, and automatically monitor information from a wide range of spreadsheets, database files and database servers.

Copia International, 1342 Avalon Court, Wheaton, IL 60187

Contact: Dorothy Gaden (708) 682-8898
AccSys for Paradox -- with the Microsoft Visual Basic, it provides the programmer with total control over Paradox table files, primary and secondary index files. Developers can create, read, write, modify, and update Paradox files without having to control the internal file format.

Coromandel, 70-15 Austin Street, Third Floor, Forest Hills, NY 11375
Contact: Narayan Laksham, Director of Marketing, (800) 535-3267, (718) 793-7963
Fax (718) 973-9710
ObjecTrieve for Visual Basic -- an ISAM DLL for Microsoft Windows and Visual Basic. It is capable of storing and retrieving binary large objects (BLOBS) such as scanned images, video, documents, bitmaps, etc. It includes Visual Basic declarations and sample code.
DbControls -- database custom controls for Visual Basic. Build database applications without writing any code. Uses ObjecTrieve's database engine, with support for binary large objects (BLOBS), multiple variable-length fields in the same record, unlimited number of indexes, and non-contiguous multi-key parts.
DbControls for dBASE -- dtabase custom controls for Visual Basic. Read and write dBASE III files without writing any code. Create new dBASE files from your Visual Basic Applications.
DbControls for Btrieve -- database custom controls for Visual Basic. Read and write Btrieve files without writing any code.
Integra SQL -- complements and extends the Visual Basic system by providing high-performance relational database functionality, including building, querying, updating and reporting of facilities.

DatTel Communications Systems, Inc., 3508 Market Street, Suite 415
Philadelphia, PA 19104
Contact: Ravi Gururaj (215) 564-5577
DataLIB -- dynamic link library (DLL) that allows Visual Basic programmers to read and write Excel, Lotus 1-2-3, dBASE, and DIF, SYLK and ASCII files. Includes all Visual Basic declarations and sample application.

Daytris Inc., 81 Bright Street, Suite 1E, Jersey City, NJ 07302
Contact: Todd C. Fearn (201) 200-0018
CDB for Windows -- sophisticated database toolkit for Windows developers offering multi-user ISAM functionality, relational and network data models, client server implementations, portability to MS-DOS and UNIX platforms, and royalty free distribution of object files.

ETN Corporation, RD4 Box 659, Montoursville, PA 17754-9433
Contact: Wynne Yoder (717) 435-2202
PowerLibW -- library (DLL) of over 90 functions and a DBMS server that provides dBX-base compatible I/O that the Microsoft Visual Basic programmer may access. Supports expressions, filters, indexes, memos, relations, and multiple database access.
Top D.B.A. -- utility for creating and modifying files (and accessing data) used in the program development/testing phase of Visual Basic application production via compatible DDE capabilities.

Gupta Technologies, 1040 Marsh Road, Menlo Park, CA 94025
Contact: (415) 321-9500
SQLBase Server -- multi-user SQL database engine that supports crash recovery, password protection, on-line backup, and remote monitoring. Gupta has DLLs that provide access to the server from Visual Basic client apps.

MDBS, PO BOX 6089, Lafayette, IN 47903
Fax: (317) 448 6428
Contact: Gary Rush (317) 447-1122
MDBS VI -- ISAM engine for Windows that has a Visual Basic interface for creating sophisticated, powerful, Windows database applications. Includes a Visual Basic global module and documentation for using Visual Basic with MDBS VI.

Microsoft Corporation, One Microsoft Way, Redmond, WA 98027
Contact: Microsoft Inside Sales (800) 227-4679
Microsoft Visual Basic Library for SQL Server -- write Visual Basic applications for Microsoft SQL Server using this library.

Natural Language, Inc., 2910 Seventh Street, Berkeley, California 94710
Contact: Mark Foster, (510) 849-8244 Paul Ricci, VP Marketing (510) 849-8217
Fax: (510) 841-3628
Natural Language -- dynamic link library (DLL) that translates English queries into SQL. Allows Visual Basic programmers to provide their users with English-language interfaces to SQL databases.

Novell, Inc., 5918 West Courtyard Drive, Austin, TX 78732
Contact: Mary K. Ellsworth (512) 794-1488
Btrieve for Windows Developer's Kit -- a complete toolkit that enables Visual Basic developers to write applications with Btrieve, Novell's key-indexed record manager.

Outrider Systems, Inc., P.O. Box 271669, Houston, TX 77277-1669
Contact: Jim Nech (713) 521-0486
vBaseTool -- database engine that supports xBase III compatible data, index, and memo fields.

Pioneer Software, 5540 Centerview Drive, Suite 324
Raleigh, North Carolina 27608
Contact: Sales: (800) 876-3101 or (919) 859-2220 Richard Holcomb, VP of marketing
Q+E Database Library -- collection of DLLs that support access to database resident information from Visual Basic applications. API supports development of low memory usage, high performance, database-independent Visual Basic applications. Connect to SQL databases from Oracle, Sybase, Ingres, SQL Server, Microsoft, and Novell. Connect to DB2. Connect to Paradox, dBASE, Btrieve, Excel XLS and ASCII text files.
Q+E Database/VB -- custom controls for Visual Basic allow you to create full-featured, multi-user database applications without writing any code. dBASE-compatible format supports record locking. Pictures and bitmaps can be stored directly in the database. Complete database creation and maintenance utility included.

PowerFlex Corp, Victoria, Australia
Contact: (03) 882 7599
PFX C-Lib -- finely-crafted DLL that allows you to access the data in your current POWERFlex or Dataflex file from Visual Basic.

Quadbase Systems, Inc., 790 Lucerne Drive, Suite 51, Sunnyvale, CA 94086
Fax: (408) 738-6980
Contact: Fred Luk (408) 738-6989

Quadbase-SQL for Windows -- a DLL (dynamic link library) that is a full-featured, compact, and high performance relational database engine for Visual Basic programmers to build single and/or multi-user applications that require advanced database features and industry standard SQL. The system can directly sccess dBase IV, Lotus 123, Foxpro index, and Clipper index files.

Raima Corporation, 3245 146th Place S.E., Suite 230, Bellevue, WA 98007
Contact: (206) 747-5570
Marketing contact: Bill Pieser
db_VISTA III Database Management System -- combines both relational and network model database technologies for high-performance Visual Basic application development. API can be easily called from Visual Basic for database application development. Sample application in Visual Basic available upon request.

SQLSoft, 10635 N.E. 38th Place, Bldg. 24, Suite B, Kirkland, WA 98942
Contact: James O'Farrell (206) 822-1287
VBOAS Design Kit V1.0 (Visual Basic Object Access for SQL Server) -- provides production application developers with high level Visual Basic object access to Microsoft/Sybase SQL Server. In just a few lines of Visual Basic code, you can connect to SQL Server, load data into Visual Basic objects and execute TransAct SQL statements. Extensive, on-line Windows help documents the usage of SQLVB Design Kit V1.0.

Sequiter Software Inc., #209, 9644-54 Ave., Edmonton, AB, Canada T6E 5V1
Tel. (403) 437-2410, Fax (403) 436-2999,
Europe Tel. +33.20.24.20.14, Europe fax +33.20.24.20.90
Contact: Ben Krueger (403) 437-2410
CodeBase 4.5 -- complete multi-user, multi-platform library for database management. Compatible with dBASE IV/III, Clipper, and FoxPro 2.0 data, index and memo files. Includes a Windows DLL for Visual Basic and on-line documentation with Visual Basic declarations and examples.

Software Source, 42808 Christy St. Ste 222, Fremont, CA 94538
Fax (415) 651-6039
Contact: Sam Cohen (415) 623-7854
VB/ISAM -- extends Visual Basic with a set of simple functions to read and write data file records by alphanumeric key. Capabilities include field-structured (Get and Put) or unstructured access, read next, previous, or approximate record, variable-length records and keys, and very large records (up to 32KB) and files (up to 512MB).

TechGnosis, Inc., One Park Place, 621 N.W. 53rd Street, Suite 340
Boca Raton, FL 33487
Contact: Keith Toleman (407) 997-6687
SequeLink -- client-server data access for Visual Basic system. Provides access to OS/2, UNIX, VAX/VMS, and AS400 servers. Supported databases include Oracle, Sybase, Ingres, SQL Server, DBM, RDB, and SQL 400.

Unelko Corporation, 7428 E. Caren Drive, Scottsdale, AZ 85260
Contact: Tony Pitman (602) 991-7272
Fax: (602) 483-7674
Bridgit -- dynamic link library that contains functions to allow full access to dBase III files, indexes, and memos. Two versions will be available: one for dBase III and the other for Clipper index files.

XDB Systems, 14700 Sweitzer Lane, Laurel, MD 20707
Contact: (800) 488-4948
Fax: (301)317-7701
XDB -- DLL gives serious SQL power in Windows. It provides 100% of IBM's DB2 SQL on your PC. Provides advanced SQL functionality such as dynamic SQL, cascading referential integrity, concurrency control, transaction processing, backup, recovery, and data security. Also supports DDE.

REPORT WRITERS

Crystal Services, 1050 West Pender Street, Ste 2200
Vancouver, B.C. V6E357
Contact: Greg Kerfoot 604-681-3425
Quick Reports For Windows -- a Windows report writer that can access data from dBase, Paradox and Btrieve databases. The product is a WYSIWYG report designer that allows user to pick fields from their databases and place them on a report and print this report to a window or printer.

Zen Software, Inc., 72 Bart Road, Monroe, CT 05468
Contact: Harlan Cooper (203) 268-6015
Excel Reporter -- Windows-based report writer. Allows developers and end users to produce reports, forms and mailing labels from the data stored in database files. Can be used as a standalone or called from within a Visual Basic program via DDE.

LAN/HOST CONNECTIVITY AND COMMUNICATIONS

Attachmate, 13231 S.E. 36th Street, Bellevue, WA 98006
Contact: Posy Gering or Mike New (800) 426-6283
Extra for Windows 3.2 -- gives Visual Basic developers access to IBM mainframes. Programs can be written to automatically integrate mainframe information with PC applications using DDE, DLL calls, and Visual Basic custom controls.

CNA Computer Systems Engineering, Inc., P.O. Box 70248, Bellevue, WA 98007
Contact: John Evans (206) 861-4736
ConnX -- connectivity tool allowing record level communication between Visual Basic applications and indexed or sequential VAX RMS files while supporting user and file level security.

Crescent Software, Inc., 11 Bailey Ave, Ridgefield, CT 06877 USA
Contact: Don Malin (203) 438-5300 Fax: (203) 431-4626
PDQComm for Windows -- complete collection of routines that make it easy to add communications capabilities to programs written in Visual Basic.

Digital Communications Associates, Inc., 1000 Alderman Drive
Alpharetta, GA 30202-4199
Contact: Margaret Owens (404) 442-4521
IRMA Workstation for Windows' (IWW) Standard IRMA Scripting Language and the Crosstalk products' Crosstalk Application Scripting Language (CASL) -- enable developers to write scripts that transfer information to and from mainframes or information services using Microsoft Visual Basic applications through dynamic data exchange (DDE). Supports XModem and ZModem transfer protocols.

Distinct Corporation, P.O. Box 3410, Saratoga, CA 95070-1410
Contact: Chris Apap-Bologna (408) 741-0781

Distinct TCP/IP Software Development Kit Berkeley Sockets, RPC/XDR and NFS toolkit for the Microsoft Windows environment -- includes Visual Basic declarations. Allows developers to write custom TCP/IP network applications or distributed applications for Windows. Accessed using a DLL.

Dome Software Corporation, 655 West Carmel Drive, Suite 151
Carmel, IN 46032
Fax: 317-573-8109
Contact: Ken Jones (317) 573-8100
Parley -- client server product that provides access to VAX or mainframe data. It provides a network independent communication layer that fully integrates a Visual Basic application into a variety of corporate data sources (SQL and non-SQL sources).

The Frustum Group, Inc., 122 East 42nd Street, Suite 1700
New York, NY 10168
Contact: Chris Davis (212) 984-0760 or (800) 548-5660
Fax: (212) 687-8119
TransPortal PRO -- data-exchange toolkit that integrates Visual Basic applications with on-line host applications (3270, 5250, or VT100). DLL can be used to read from, write to, and send keystrokes directly to host application. Includes Visual Basic declarations.

FutureSoft, 1001 South Dairy Ashford, Suite 203, Houston, TX 77077
Contact: Teri Taylor (713) 496-9400
DynaComm -- with each DynaComm product, Visual Basic system users will be able to visually link their applications to DynaComm using DynaComm custom controls. Planned to support IBM, HP, NEC, and Data General mainframes.

Groupe Bull, 7, Rue Ampere, 91343 Massy, France
Phone: +33-1-69-93-90-90
Affinity-Visual -- fully integrates the Microsoft Visual Basic system with Bull's Affinity product. Affinity-Visual provides full Windows graphical display services to existing host applications throughout Bull environments.

JSB Corporation, 108 Wispering Pines Drive, Suite 115
Scotts Valley, CA 95066
Contact: (408) 438-8300
Fax: (408) 438-8360
JSB MutiView Desktop PC to Unix integration product -- supports DDE links between Visual Basic and existing remote UNIX applications. Additionally, it provides custom controls that provide communications links to UNIX applications to allow Visual Basic programs to be clients of UNIX systems.

Microcom Inc., 55 Federal Road, Danbury, CT 06810
Contact: (800) 822-8224 or Howard Luxenberg (203) 730-4378
MicroCourier -- complete communication package for Windows for under $100. Includes sample applications written in Visual Basic with full source code.

MicroHelp, Inc., 4359 Shallowford Industrial Parkway, Marietta, GA 30066
Contact: Mark Novisoff (404) 516-0899 or 1-800-922-3383
Fax: (404) 516-1099
MicroHelp Communications Library -- communications routines for Visual Basic invoked exactly like SubPrograms and Functions, including automatic file transfer routines using XModem, XModem CRC, YModem, YModem-Batch, ZModem, CompuServe B, and ASCII transfers.

MicroHelp Network Library -- access to network interface routines. Supports Novell, Lantastic, and NETBios compatible networks.

Microsoft Corporation, One Microsoft Way, Redmond, WA 98027
Contact: Microsoft Inside Sales (800) 227-4679
Microsoft LAN Manager Toolkit for Visual Basic -- tools to customize a LAN Manager-based network using Microsoft Visual Basic. Includes a graphing facility for displaying performance information and other system stats. Sample utilities for common network management and diagnostic applications.

NetManage, Inc., 20823 Stevens Creek Blvd., Suite 100, Cupertino, CA 95014
Contact: Sales Dept. (408) 973-7171 Dan Geisler
Chameleon TCP/IP for Windows -- TCP/IP application package for Windows. Includes TELNET, FTP, TFTP, SMTP/mail, name services, PING, network management and diagnostics. Implemented as a Windows DLL callable from Visual Basic applications as both client and server.
RPC-SDK: ONC Development Tools -- software development kit for building distributed applications in Windows using Sun ONC RPC/XDR. Windows DLL callable from Visual Basic applications as RPC client and server.
NEWT/SDK -- software development kit for Windows 3.0 TCP/IP communications protocol. Offers the Visual Basic programmer direct access to the Berkeley 4.3BSD socket interface, FTP and SMTP.

Rochester Software Connection, 4909 Highway 52 North, Rochester, MN 55901
Contact: John Freund, Vice President of Sales & Marketing,
(507) 288-5922, (800) 829-3555
ShowCase WindowLink -- DLL allows you to link Visual Basic applications to IBM AS/400 systems. Includes Visual Basic declarations and sample code.

Symbiotics, 725 Concord Ave, Cambridge, MA 02138
Contact: (800) 989-9174
NetWorks!Connect -- allows you to write programs that talk to each other over a network using the language functions and commands you already know. Fully compatible with Novell NetWare LAN Manger, and Banyan Vines. Also Sun and HP UNIX platforms.

TechGnosis, Inc., One Park Place, 621 N.W. 53rd Street, Suite 340
Boca Raton, FL 33487
Contact: Keith Toleman (407) 997-6687
SequeLink Engine -- software development toolkit enabling workstation access to host-based data and applications. Extends the functionality of the company's SequeLink client/server architecture by enabling host operating systems, applications, and non-relational DBMSs to act as servers for Windows applications.

Wall Data Incorporated, 17769 N.E. 78th Place, Redmond, WA 98052
Contact: Catherine Rudolph (Marketing Communications) (800)48-RUMBA
Fax: (206) 885-9250
Rumba Application Development Kit -- complete development environment enables Visual Basic developers to change how users interact with PC and host applications. Includes advanced tools for creating connectivity links.
Rumba Tools for DDE and Rumba Tools for EHLLAPI -- enables advanced users to create simplified and transparent connectivity links between PCs and host computers. Rumba Tools for DDE allows Visual Basic applications to exchange data continuously with Rumba using DDE. Also allows Visual Basic applications to exchange data with Rumba using EHLLAPI.

PEN/MULIMEDIA
==========

New Media Graphics Corporation, 780 Boston Road, Billerica, MA 01821-0666
Contact: (800) 288-2207
Fax: (508) 663-6678
SuperVideo Windows -- a full line of video, framegrabbing, and compression boards for desktop multimedia applications on PC and MCA computers using a custom control. Display, capture, or compress full motion, true color video in any Windows 3.x application.

StylusTech Inc., Suite 300, Building 600, One Kendall Square
Boston, MA 02139
Contact: (617) 277-7007
Fax: (617) 277-8907
Pen InputMaster -- first of a series of pencentric extensions to Visual Basic. A multi-featured, combination custom control that supports three methods of data input: entry field, entry field with character guides, and pick list.

VISUAL BASIC LIBRARIES AND TOOLS
==========

Crescent Software, Inc., 11 Bailey Ave, Ridgefield, CT 06877 USA
Contact: Don Malin (203) 438-5300 Fax: (203) 431-4626
QuickPak Professional for Windows -- custom controls and a general purpose set of utilities for use with Visual Basic programming system. QuickPak Professional for Windows provides routines for quickly sorting and searching data, performing fast file operations, expression evaluation, and other useful tasks.

EMS Professional Shareware, 4505 Buckhurst Ct., Olney, MD 20832
Contact: (301) 924-3594 Fax (301) 963-2708
Public Domain Files -- file collection of public domain and shareware file collections for Visual Basic programmers. Over 300 applications written in Visual Basic and utilities.

Hewlett-Packard Company, 19310 Pruneridge Ave., M/S 49AW
Cupertino, CA 95014
Contact: Inquiry Manager (800) 452-4844
HP 82335B HP-IB for Windows and MS-DOS -- dynamic link library (DLL) and language interface for creating HP-IP (IEEE 488) instrument control programs for the most popular industry standard test equipment using Visual Basic. The HP-IB interface card is included.

Kofax Image Products, 3 Jenner Street, Irvine, CA 92718
Fax: (714) 727-3144
Contact: Emily Backus (714) 727-1733
Kofax Image Processing Platform (KIPP) -- application-development software and controller boards, compatible with the Visual Basic system, that serve as the foundation for creating PC-based document image processing applications and systems.

MicroHelp, Inc., 4636 Huntridge Drive, Roswell, GA 30075-2012
Contact: Mark Novisoff (404) 516-0899 or 1-800-922-3383
MicroHelp Muscle -- library for the professional programmer that includes hundreds of assembly language routines and several high-level Visual Basic routines.
VBXRef -- a comprehensive cross reference utility for Visual Basic

applications, including reference trees for procedures and variables.

National Instruments, 6504 Bridge Point Parkway, Austin, TX 78730-5039
Contact: Tim Dehne or Holly Matheny (512) 794-0100
NI-488.2 Windows Interface for Visual Basic -- links a Visual Basic application to the NI-488.2 Windows GPIB driver software. System boards for the IEEE 488 interface available as well. Products connect Visual Basic with thousands of industry-standard programmable instruments.
NI-DAQ for Windows -- NI-DAQ Windows Interface for Visual Basic applications using National Instruments' plug-in data acquisition boards. DLL with high-level data acquisition functions for developing data acquisition applications in Visual Basic.

Pinnacle Publishing, P.O. Box 8099, Federal Way, WA 98003
Contact: David Johnson (800) 231-1293 or (206) 941-2300
Graphics Server for Visual Basic -- custom control for integrating graphing and charting capabilities into Visual Basic applications. Includes pie charts, bar charts and a variety of other graphs in 2D or 3D.

Scientific Software Tools, Inc., 30 East Swedesford Road
Malvern, PA 19355
Contact: Elise Furman (215) 889-1454, Fax (215) 889-1630
DriverLINX\VB -- high-performance data-acquisition engine for developing custom applications using Microsoft Visual Basic. Quickly create sophisticated virtual instruments that you could only dream of in MS-DOS, in just days, using DriverLINX\VB. DriverLINX takes the form of a custom control that is added to the Toolbox of built-in Visual Basic controls.

Sheridan Software Systems, Inc., 65 Maxess Road, Melville, NY 11747
Contact: Joseph Modica (516) 753-0985, fax (516) 293-4155
BBS numbers: 2400 Baud: (516) 753-5452 9600 baud: (516) 753-6510
VB Assist -- Help utility that works alongside Visual Basic to speed application development with utilities to set properties and much more. Visual Basic 3.0 requires VBAssist version 2.0c or later.

TeraTech, 3 Choke Cherry Road, Suite 360, Rockville, MD 20850
Contact: (301) 424-3903
Fax: (301) 762-8185
ProMath/VB -- many mathematical, scientific, and statistical functions. From integration to Bessel Functions to Curtosis and Skew. Complex numbers and FFT are all supported.

The Young Software Works, PO Box 185 Cooper Station, New York, NY 10276
Contact: (212) 982-4127
FAX: (212) 673-1715
VB Project Archiver -- project management utility for Visual Basic programmers. Provides project archiving capabilities using PKZip, LHARC, or other compression utilities. Can determine which code and form modules are active in a Visual Basic app for use as a simple version control system.

Ward Systems Group, Inc., 245 W. Patrick Street, Frederick, MD 21701
Contact: Marge Sherald (301) 662-7950
NeuroWindows -- a neural network programming tool, designed to work with Microsoft Visual Basic. It builds powerful neural network applications that perform a wide variety of pattern recognition and prediction tasks.

==========

Artisoft, 6920 Koll Center Parkway, Suite 209, Pleasanton, California 94566
Contact: (415) 426-5355 Corporate accounts national sales
manager: Brion Miller
Wired for Sound -- DLL that can add sound capabilities to any Visual Basic form. Plays sound through PC speaker or sound boards. Includes API_SPEC.TXT file with code examples for Visual Basic programmers.

Black Ice Software, Inc., Crane Road, Somers, NY 10589
Contact: (914) 277-7006 Laurie Welchoff; Jozef Nemeth, President
Fax: (914) 276-8418
TIFF SDK for Windows -- DLL that allows you to add TIFF 5.0 support to Visual Basic applications without learning the complexity of the Tagged Image File Format.

DemoSource, 8646 Corbin Avenue, Northridge CA, 91324-4130
Contact: Brian L. Berman (800) 888-8063 Fax (818) 772-2877
DemoSource -- a QuickLine voice library and VFEdit professional sound editor compatible with Visual Basic. It enables PCs to dispense prerecorded voice messages through standard touch-tone telephones for interactive mail order catalogs and automated outbound dialing systems for sales and telemarketing.

First Byte, 19840 Pioneer Avenue, Torrance, CA 90503
Contact: Michael Belanger (310) 793-0600 x 212 Sales rep/tech support
Monologue for Windows -- a DLL to make Visual Basic applications talk. It is a text-to-speech utility that converts text into speech, to PC speaker or sound board.

RealSound Inc., 4910 Amelia Earhart Drive, Salt Lake City, UT 84116
Fax (801) 359-2968
Contact: Janson Tanner (801) 359-2900
RealSound for Windows -- a DLL for Windows providing an exciting enhancement to Visual Basic in hardware-quality digitized sound.

Silicon Valley Products, Corp., 8 Paquatuck Avenue
East Moriches, NY 11940-0564
Contact: Paul Norris (516) 878-6438
QuickLine -- dynamic link library for use with Visual Basic to control TTI's telephone interface board for recording or playing messages, decoding telephone touch tones, and placement of calls.

The Stirling Group, 172 Old Mill Road, Schaumberg, IL 60193
Contact: Viresh Bhatia, Managing Partner (708) 307-9197,
(800) 3-SHIELD (800-374-4353)
Fax: (708) 307-9340
TbxSHIELD -- a dynamic link library that allows you to create toolbox controls to include in your applications. Controls can be of any size, shape, or style. It can be created quickly and easily and includes Visual Basic declarations and sample application.

VideoLogic, 245 First Street, Cambridge, MA 02142
Contact: Karyn Scott (617) 494-0530
DVA-4000/ISA -- digital video adapter that allows Visual Basic users to seamlessly integrate full-motion video with standard graphics and text in the Windows environment.

HELP FILE AUTHORING TOOLS

Blue Sky Software Corp., 7486 La Jolla Blvd. Ste 3, La Jolla, CA 92037
Contact: (619) 459-6365, (800) 677-4WIN
RoboHelp -- An automatic authoring tool that makes the process of creating a Windows Help System just a matter of pointing and clicking. The user just fills in the actual help text when prompted. Features a customized tool palette. Generates source code for context sensitive help, hypertext link, cross reference.

Software Interphase, Inc., 82 Cucumber Hill Road, Foster, RI 02825
Contact: 800-542-2742
Windows Help Magician -- create Windows Help files in a single integrated environment. Uses advanced functions, and hotkeys. Allows you to test a file instantly. Edit, test, write RTF file, compile and call WINHELP.EXE in the same environment.

WexTech Systems, Inc., 310 Madison Avenue, Ste 905, New York, NY 10017
Contact: Steve Wexler (212) 949-9595
Fax: (212) 949-4007
Doc-to-Help -- Word for Windows 2.0 utility that allows you to create professional-quality documentation and automatically convert that documentation into Windows context-sensitive online help for your Visual Basic application. Includes the Microsoft Windows Help Compiler.

GRAPHICS UTILITIES AND CLIP-ART

Data Techniques, 1000 Business Center Drive Suite 120, Savannah, GA 31405
Contact: (912) 651-8003
Image Man/VB -- object oriented Windows custom control that adds advanced image display and print capabilities to applications. Supports TIFF, PCX, GIF, EPSF, WMF, and BMP formats in 24 bit color.

Dynalink Technologies, P.O. Box 593, Beaconsfield, Quebec, Canada H9W 5V3
Contact: (800) 522-4624 Peter Krenjevich, (514) 489-3007
Clip'nSave 2.0 for Windows -- sreen capture and image conversion program. It can capture any part of a screen to include in a Visual Basic program or print. Reads and writes mono, gray, and color BMP, DIB, TIF, PCX, GIF, and EPS files.

Eikon Systems Inc., 989 East Hillside Blvd, Suite 260
Foster City, CA 94404
Sales: (800) 727-2793
Contact: Jeff Galvin (415) 349-4664
Scrapbook+ -- a Windows utility for managing Clipboard images, bitmaps, clip art, and other graphics. "Camera" tool allows you to create bitmap images of any portion of a screen. Can convert graphics between TIF, PCX, BMP, and MSP formats.

MicroCal, Inc., 22 Industrial Dr. E., Northampton, MA 01060
Contact: (800) 969-7720.
Origin -- powerful scientific and technical graphics software for Windows. Supports DDE for plotting data from Visual Basic applications.

TechSmith Corporation, 1745 Hamilton Road, Suite 300, Okemos, MI 48864
Contact: (517) 347-0800
SnagIt -- screen capture utility for Windows. DDE support allows you to

add screen capture capability to Visual Basic applications.
DDE Watch -- monitoring and debugging tool for dynamic data exchange.

PUBLICATIONS AND TRAINING
==========
Addison-Wesley Publishing Company, Inc., 1 Jacob Way, Reading, MA 01867
Orders: (800) 447-2226 or (617) 944-3700 or fax (617) 942-1117
Contact: (617) 944-3700 Editor: Julie Stillman x2773, (&
Claire Horne), Marketing: Ann Lane x2278
Using Visual Basic by William Murray and Chris Pappas -- a hands-on guide to learning, using and mastering Visual Basic. The book emphasizes how to design screens and place controls within Visual Basic. The authors lead readers through a series of applications that will serve as templates for applications. Includes disk.
Advanced Visual Basic by Mark Burgess -- due Summer, 1992.

Bantam Computer Books, 666 Fifth Avenue, New York, NY 10103
Contact: Jono Hardjowirogo (212) 492-9826
Visual Basic Programming with Windows Applications by Douglas Hergert -- a book oriented toward programmers with Basic experience interested in developing business solutions.

Brady (Prentice Hall, owned by S & S), Simon and Shuster, Inc.
15 Columbus Circle, New York, NY 10023
Sales: (800) 223-2348
Contact: Gene Smith (503) 639-9822
Visual Basic by Steven Holzner and The Peter Norton Computing Group -- a complete introduction to Visual Basic.

The Cobb Group, 9420 Bunsen Parkway, Suite 300, Louisville, KY 40220
Sales: (800) 223-8720
Contact: Melissa Haeberlin (502) 491-1900
Inside Visual Basic -- a 16-page monthly journal providing tips and techniques for using the Visual Basic programming system.

Cooper Software Inc., 3523 A Haven Avenue, Menlow Park, CA 94025-9986
Fax: (415) 364-0593
Contact: Alan Cooper (415) 364-9150
QRC -- Quick reference card for Microsoft Windows 3.0. Quick reference to all 597 Windows API calls.

ETN Corporation, RD4 Box 659, Montoursville, PA 17754-9433
Contact: (Technical Information): (717) 435-2202 Sales:
(800) 326-9273
Fax: (717) 435-2802
VB= mc^2: The Art of Visual Basic Programming by J.D. Evans -- a book about advanced Visual Basic programming and Windows application design. It includes a companion disk and extensive code samples and approaches Visual Basic from a different angle. Order directly from ETN.

Fawcette Technical Publishing, 299 California Ave, Suite 120
Palo Alto, CA   94306-1912
Contact: Jim Fawcette (415) 688-1808 Fax (415) 688-1812
Basic Pro -- a bimonthly periodical for Basic professionals covering both text-mode and Windows Basic development issues. Provides advertising space for developers of Basic language products and add-on products, in addition to regular letters to the editor, guest columnist, product review, and

upcoming industry event sections.

Microsoft Press, One Microsoft Way, Redmond, WA 98052-6399
Contact: Craig Johnson (206) 936-3895
The Microsoft Visual Basic Workshop -- a book and software package that is a one-stop source of imaginative and useful Visual Basic forms and subprograms to use in Microsoft Windows applications.
Microsoft Windows Multimedia -- Programmer's reference for creating Windows applications that access multimedia functionality.

Microsoft University, 10700 Northrup Way, Bellevue, WA 98004-1447
(206) 828-1507
Microsoft University Visual Basic -- Advanced Topics course, a 3-day course covering concepts needed to write sophisticated event-driven, graphical programs and design applications that integrate with DDE and Windows DLLs.

Osborne/McGraw Hill, 2600 10th Street, Berkeley, CA 94710
Sales: (510) 549-6614
Contact: Jeff Pepper (415) 549-6638
Visual Basic Inside and Out By Gary Cornell -- a complete review of the Visual Basic programming system for Windows.

Programmer's Warehouse, 8283 N. Hayden Road, Suite 195
Scottsdale, Arizona 85258
Contact: 800) 323-1809 or (602) 443-0580
Fax: (602) 443-0659
A full-service mail-order reseller for Visual Basic and all related companion products.

Que (Prentice Hall, owned by S & S), 11711 North College Avenue
Carmel, IN 46032
Tel. (800) 428-5331 (317) 573-2500
Using Visual Basic by Roger Jennings -- a book for beginning and intermediate programmers who want to write Visual Basic applications. Includes advanced features such as DDE, as well as a complete keyword reference section. Ships in April, 1992.
Visual Basic By Example by D.F. Scott -- beginning level overview with many programming examples. Ships in April, 1992.
Visual Basic Programmer's Reference -- Currently on hold.

Sams (Prentice Hall, owned by S & S), 11711 North College Avenue
Carmel, IN 46032
Tel. (800) 628-7360
First Book of Visual Basic by Orvis -- a structured tutorial for the novice computer user covering the Visual Basic language and modern programming practice.

Tab/McGraw Hill, 13311 Monteray Lane, Blue Ridge Summit, PA 17294
Tel: 717-794-2191 or 800-822-8138 (for orders)
Visual Basic: Easy Windows Programming by Namir Shammas -- a hands-on introduction to developing VB applications. Organized in a workbook format, each chapter teaches a specific task such as constructing interfaces, testing and debugging code, and producing executable files. Includes more than 50 ready to use programming examples. Ships in February, 1992.
Visual Basic Power Programming by Namir Shammas -- a book designed to go beyond the fundamentals of developing with Visual Basic. It provides a programmer's toolbox complete with routines for file management, text and

graphics manipulation, scientific plotting, and more. The package includes many reusable programs, modules, and forms. Ships in April, 1992.

Waite Group Press, 100 Shoreline Highway, Suite A-285
Mill Valley, CA 94941
Contact: (415) 331-0575
Visual Basic How-To by Robert Arnson, Dan Rosen, Mitchell Waite, and Jonathan Zuck -- a book and disk package that contains hundreds of Visual Basic solutions from how to make an interface to how to use the Windows API functions.
Visual Basic Super Bible by Bryan Scott, Taylor Maxwell -- explains each command, keyword, property, object and procedure of Visual Basic. 900 pages. All examples on disk. Ships April, 1992.

Windows Tech Journal, Oakley Publishing Company, PO Box 70167
Eugene, OR 97401-0110
Contact: J.D. Hildebrand (503) 747-0800 Fax: (503) 746-0071
Windows Tech Journal -- the monthly magazine of tools and techniques for Windows programmers. Annual subscription (12 issues) is $29.95.

Michael Risse
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
Tel. (206) 882-8080
Fax (206) 93-MS-FAX (206-936-7329)

Additional reference words:
KBCategory: Refs
KBSubcategory: RefsThird

Cobb Group's "Inside Visual Basic" Journal Article Titles

Article ID: Q83351

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

"Inside Visual Basic" is a monthly magazine published by The Cobb Group, Inc. The following article titles are reprinted with permission from the Cobb Group's "Inside Visual Basic" (c) January and February 1992 issues.

For more information, contact the Cobb Group at the following address and phone number:

The Cobb Group, Inc.
9420 Bunsen Parkway
Suite 300
Louisville, KY 40220
(800) 223-8720

January 1992 Issue Contents

"Creating a new control -- The combo dropdown list box"
"Wither Basic data-type codes"
"Keeping users informed with minimized icons"
"Speeding up list box clearing"
"Managing data in multiple database formats with QELIB"
"Displaying your forms faster"
"Simplify debugging and maintenance with a good naming convention"
"Creating smaller VB EXEs"
"Stop draggin' that text around"
"Soup to nuts software"

February 1992 Issue Contents

"Creating your own VB help system"
"VB classes available"
"VB books available"
"Where'd those !@#$%^ characters go?"
"Help is just a button away"
"Adding hot keys to your programs"
"Anyone need a sort?"
"Keeping your perspectives when resizing forms"
"Source code listings"

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: RefsPSS RefsThird

Visual Basic 3.0 Support Service Questions & Answers

Article ID: Q92552

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, version 3.0

---

1. Q. Where can I get information on Microsoft's no-charge startup and installation support services for Visual Basic version 3.0 for Windows?

   A. The telephone number to call for no-charge technical support for Microsoft Visual Basic version 3.0 for Windows is (206)646-5105. Your telephone company will bill you for long-distance charges only. This service provides support for setup and installation questions only and is available 6:00 A.M. to 6:00 P.M. Pacific time, Monday through Friday.

2. Q. Where can I get information on fee-based technical support and programming assistance for Microsoft Visual Basic version 3.0 for Windows?

   A. Microsoft OnCall is a fee-based service that provides technical support and programming assistance. The rate for this service is $2 per minute. The telephone number is (900)896-9876. Your telephone company will bill you for this service.

   If you are blocked from dialing the 900 number, you can call (206)646-5106. A $20 fee will be charged to your credit card for each call. Mastercard, Visa, and American Express cards are accepted.

   Microsoft offers additional comprehensive, fee-based technical support options. For more information about these options, please call Microsoft OnLine Sales at (800)443-4672.

3. Q. Where can I get information about support for Microsoft Visual Basic for Windows available on the CompuServe electronic information service?

   A. Microsoft technical support and programming assistance from other Visual Basic developers is available in the MSBASIC forum on CompuServe. Through the MSDN forum on CompuServe, you can also gain access to the Microsoft Knowledge Base, which contains descriptions of known problems and answers to many frequently asked questions. For more information, please call CompuServe at (800)848-8990 and ask for the CompuServe Information Manager software disk. The software disk provides you with an option that enables you to set up your own CompuServe account.

4. Q. Where can I get information about support for the Control Development Kit provided with the professional edition of Visual Basic for Windows?

A. Support for the Microsoft Control Development Kit is currently provided only through CompuServe in the MSBASIC forum in section 16 or through service requests in Microsoft OnLine support services. For more information about CompuServe, please call CompuServe at (800)848-8990. For more information about the Microsoft OnLine support services, please call (800)443-4672.

5. Q. Where can I get information about support for the Crystal Reports Custom Control and associated features in Microsoft Visual Basic version 3.0 for Windows?

   A. Support for the Crystal Reports Custom Control is provide solely by Crystal, a company separate from Microsoft. There is a detailed listing of all support options available to you from Crystal in the last 2 pages of the "Microsoft Visual Basic version 3.0 Professional Features Book 2" manual.

6. Q. What is the Microsoft Download Service? how do I access it?

   A. Microsoft Download Service (MSDL) is a Bulletin Board system (BBS) that can be accessed by any user with a computer and a modem. The MSDL contains application notes, drivers, and other support files from Microsoft. MSDL supports 1200, 2400, and 9600 baud (V.32 and V.42) with 8 data bits, 1 stop bit, and no parity. The supported protocols are Xmodem, Xmodem-1K, Ymodem (batch), Kermit, Super Kermit (Sliding Windows), and Zmodem. To connect to MSDL, call (206)936-6735 and follow the instructions.

7. Q. What is the Microsoft Developer Network? How do I get it?

   A. The Microsoft Developer Network (MSDN) is a newsletter and CD available together or separately. The newsletter is published every other month and the CD is published quarterly. Both the newsletter and the CD contain technical information for all developers who write applications using Microsoft operating systems or development tools. The CD contains code samples, technical articles, development tools, and the Microsoft Knowledge Base. For more information, please call (800)227-4679, or call (800)759-5474 to join.

8. Q. Where can I place an order or get upgrade and pricing information about Microsoft Visual Basic version 3.0 for Windows?

   A. For information regarding product updates, prices, and sales, please call Microsoft Customer Service at (800)426-9400. Note that no technical support is provided on this line.

Additional reference words: 3.00 ivrfax fasttips
KBCategory:
KBSubcategory: RefsProd

Name Property Cannot Be Set When Using Implicit Property

Article ID: Q93214

The information in this article applies to:

- Microsoft Visual Basic for Windows, version 2.0

SUMMARY

On Page 126 of the Visual Basic Programmer's Guide, it incorrectly states that all controls have an implicit property you can use for storing or retrieving values. Some controls supplied with the Professional Edition of Visual Basic for Windows use the Name property as their implicit property, which you cannot use at run-time.

MORE INFORMATION

The following controls from the Visual Basic Professional Edition use the Name property as their implicit property:

   Common dialog
   MAPI session
   MAPI message
   Spin button

Attempting to access the implicit property of these controls results in one of the following errors:

   'Name' property cannot be read at run time
   'Name' property cannot be set at run time

You access the implicit property of a control (also known as the "value of a control" or the "default value of a control") by writing the control name with no property. For example, with a text box named Text1, you can write the following statement to assign a value to the Text property:

```
Text1 = "hello world"
```

The following list shows the implicit properties for all the controls in both the Standard and Professional Editions:

Standard Control	Implicit Property
Check box	Value
Combo box	Text
Command button	Value
Directory list box	Path
Drive list box	Drive
File list box	FileName
Frame	Caption
Grid	Text
Image	Picture

Label	Caption
Line	Visible
List box	Text
Menu	Enabled
OLE client	Action
Option button	Value
Picture box	Picture
Scroll bar vertical	Value
Scroll bar horizontal	Value
Shape	Shape
Text box	Text
Timer	Enabled

Professional Control	Implicit Property
3D check box	Value
3D command button	Value
3D frame	Caption
3D group push button	Value
3D option button	Value
3D panel	Caption
Animated button	Value
Common dialog	Name (not usable)
Communications	Input
Gauge	Value
Graph	QuickData
Key status	Value
MAPI session	Name (not usable)
MAPI message	Name (not usable)
Masked edit	Text
Multimedia MCI	Command
Pen BEdit	Text
Pen HEdit	Text
Pen ink on bitmap	Picture
Pen on-screen keyboard	Visible
Picture clip	Picture
Spin button	Name (not usable)

Additional reference words: 2.00 docerr
KBCategory:
KBSubcategory: RefsDoc PrgCtrlsStd PrgCtrlsCus

Visual Basic MCI Control TimeFormat Property Information

Article ID: Q94012

The information in this article applies to:

- The Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0
- The Professional Edition of Microsoft Visual Basic for Windows, version 2.0

SUMMARY

The Multimedia Device Control (MCI.VBX) TimeFormat property does not support all format settings with all device types. When you assign a value to TimeFormat that is not supported by the device, the TimeFormat retains its previous setting.

This article also describes MCI_FORMAT_MSF (2) and shows how to separate the 4 bytes of a time value.

MORE INFORMATION

To determine if the current device supports a particular TimeFormat setting, assign the value to TimeFormat. Then check TimeFormat to see if it returns the value assigned. For example:

```
For i = 0 To 10
   MMControl1.TimeFormat = i
   If MMControl1.TimeFormat = i Then
      MsgBox Format$(i) + " supported"
   Else
      MsgBox Format$(i) + " not supported"
   End If
Next
```

Some of the time formats, such as MCI_FORMAT_TMSF, provide four separate byte size numbers packed into one four byte long integer. The following sample statements show how you can extract the four bytes into separate variables:

```
byte1 =  MMControl1.Position And       &HFF&
byte2 = (MMControl1.Position And     &HFF00&) \     &H100
byte3 = (MMControl1.Position And   &HFF0000)  \   &H10000
byte4 = (MMControl1.Position And &H7F000000)  \ &H1000000
If (MMControl1.Position And &H80000000) <> 0 Then
   ' put sign bit back into byte4
   byte4 = byte4 + &H80
End If
```

The least significant byte is stored in byte1 and the most significant byte is stored in byte4.

The following list shows all possible settings for TimeFormat:

```
    0  MCI_FORMAT_MILLISECONDS
    1  MCI_FORMAT_HMS
    2  MCI_FORMAT_MSF
    3  MCI_FORMAT_FRAMES
    4  MCI_FORMAT_SMPTE_24
    5  MCI_FORMAT_SMPTE_25
    6  MCI_FORMAT_SMPTE_30
    7  MCI_FORMAT_SMPTE_30DROP
    8  MCI_FORMAT_BYTES
    9  MCI_FORMAT_SAMPLES
   10  MCI_FORMAT_TMSF
```

The TimeFormat setting MCI_FORMAT_MSF is described in the README.TXT file but is missing from the "Microsoft Visual Basic Professional Features Custom Control Reference" for version 2.0. The following description of MCI_FORMAT_MSF appears in the README.TXT file:

```
2  MCI_FORMAT_MSF  Minutes, seconds, and frames are packed into a
                   four-byte integer.  From least significant byte to
                   most significant byte, the individual data values
                   follow:

                   Minutes (least significant byte)
                   Seconds
                   Frames
                   Unused (most significant byte)
```

The TimeFormat property affects the following properties.

```
   Position
   From
   To
   Start
   Length
   TrackLength
   TrackPosition
```

Microsoft has confirmed that this information should be included in the "Microsoft Visual Basic Professional Features Custom Control Reference" for version 2.0. We will post new information here when the documentation has been updated with this additional information.

Additional reference words: 1.00 2.00 docerr
KBCategory:
KBSubcategory: RefsDoc

Corrections for Errors in Visual Basic Version 2.0 Manuals

Article ID: Q94373

The information in this article applies to:

- The Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, version 2.0

SUMMARY

Below are corrections for documentation errors in the manuals shipped with Microsoft Visual Basic for Windows Standard Edition and Professional Edition version 2.0.

This master list of corrections includes and adds to the corrections already found in the README.TXT file shipped with Visual Basic 2.0. Please use the article below as your master list for making corrections to the Visual Basic 2.0 manuals.

MORE INFORMATION

Microsoft Visual Basic for Windows Standard Edition version 2.0 includes the following two manuals:

- "Microsoft Visual Basic Programmer's Guide"
- "Microsoft Visual Basic Language Reference"

In addition, the Professional Edition version 2.0 also includes:

- "Microsoft Visual Basic Professional Features"

Corrections to "Microsoft Visual Basic Programmer's Guide," Version 2.0

Page	Section/Note
4	Visual Basic Documentation In the second bullet list item, replace "eight" with "seven."
6	Using Online Documentation In the third line, replace "eight" with "seven."
8	Figure 1.2 The Contents Screen This illustration does not show the actual Contents screen.
15	Starting Visual Basic

In the second table item under "Menu equivalent," change it to read, "Start command on the Run menu."

19 Setting Properties

In the second paragraph of step 3, change "...clicking the DOWN ARROW key at the right..." to "...clicking the down arrow at the right..."

23 Simple Animation

In the Setting column of the table, change "(White)" to "(Black)."

In the paragraph following the table, change "... and the BackColor property to 0 (Black)" to "... and the BackColor property to black"

26 File1_DblClick (Source Code)

Form1.Open.Picture = ...

Should read:

Form1.Image1.Picture = ...

66 Table 3.2 Operator(n)

The values for (n) are incorrect:
1,2,3,2,2 should be 1,3,2,0,4 (reading down the list).

201 Identifying the Current Mode

In the paragraph at the bottom of the page, change the phrase after the semicolon to "the unavailable buttons appear dimmed on the toolbar."

210 Using the Calls Dialog

Remove the "}" from the end of step 1.

216 Editing or Deleting a Watch Expression

In the numbered list at the top of the page, remove the "(s)" from the word "expression" in the second step.

220 Assigning Values to Variables and Properties

In the paragraph following the three lines of example statements, change the text to "The first statement alters a property of the currently active form, the second alters a property of the VScroll1 control, and the third assigns a value to a variable."

227 How to Handle Errors

The list of steps is incorrectly numbered. The paragraph now

numbered 2 should not be numbered. Remove the number 2 from that paragraph. Then replace the 3 in the following paragraph with 2 and replace the 4 in the last paragraph with 3.

229 Exiting an Error-Handling Routine

In the table that describes ways to exit an error-handling routine, make the following changes:

Replace the Resume entry with:

Resume (0)	Program execution resumes with the statement that caused the error or the most recently executed call out of the procedure containing the error-handling routine.

Change the Resume Next entry by removing the period at the end of the sentence and adding "or with the statement immediately following the most recently executed call out of the procedure containing the error-handling routine."

Change the Resume line entry by removing the period at the end of the sentence and adding "that must be in the same procedure as the error handler."

234 Change the note at the bottom of the page as follows:

Remove everything after the first sentence. Add the following:

If a Resume statement is executed, control returns to the most recently executed call out of the procedure containing the error handler. If a Resume Next statement is executed, control returns to whatever statement in the procedure containing the error-handling routine immediately follows the most recently executed call out of that procedure.

For example, in the Calls list shown in Figure 10.3, if procedure A has an enabled error handler and Procedures B and C don't, an error occurring in Procedure C will be handled by Procedure A's error handler. If that error handler uses a Resume statement, upon exit, the program continues with a call to Procedure B. However, if Procedure A's error handler uses a Resume Next statement, upon exit, the program will continue with whatever statement in Procedure A follows the call to Procedure B. In neither case does the error handler return directly to either the procedure or the statement where the error originally occurred.

420 The Directory List Box

In the code at the bottom of the page, change the first line as follows:

```
GoHigher = 0     ' Initialize for currently expanded directory.
```

421 The File List Box

Change the first paragraph as follows:

"The file list box displays files contained in the directory specified by the Path property at run time. You can display all the files in the current directory on the current drive by using the following statement:"

The paragraph that begins, "If you set the System property..." may be misleading. The following additional information is provided to clarify the meaning.

The default value for the System and Hidden properties is False. The default value for the Normal, Archive, and ReadOnly properties is True.

When Normal = True, any file that does not have the System or Hidden attribute is displayed. When Normal = False, you can still display files with ReadOnly and/or Archive attributes by setting the appropriate attribute to True (ReadOnly = True, Archive = True).

When System = True, any file with the System attribute is displayed unless it also has the Hidden attribute.

When Hidden = True, any file with the Hidden attribute is displayed unless it also has the System attribute.

To display any file that has both Hidden and System among its attributes, both Hidden and System must be True. However, files that have either Hidden or System attributes are displayed as well.

424 Writing Code for the WinSeek Application

In the second paragraph, change the first sentence as follows:

"The WinSeek application resolves this ambiguity by determining if the path of the dirList box is different from the currently highlighted directory."

425 The cmdSearch_Click Procedure

In the sample code shown, change the reference to "dirList,ListIndex" (note the comma) in the If statement to the following:

```
If dirList.Path <> dirList.List (dirList.ListIndex) Then
```

482 Change the last sentence in the paragraph at the top of the page to this:

"When the user activates the object (the graph), the server application (MS Graph) is invoked by the client application (Visual Basic), and the object's data is opened for editing."

541 New Keywords in Visual Basic 2.0

Include the keyword "Count" in the list.

Corrections to "Microsoft Visual Basic Language Reference," Version 2.0

Page	Section/Note

82 DateValue Function

Change the last sentence in the second paragraph of the Remarks section to this:

"For example, in addition to recognizing 12/30/1991 and 12/30/91, DateValue recognizes December 30, 1991 and Dec 30, 1991.

167 GetData Method

The following line of code is not correct:

```
Picture = Clipboard.GetData()
```

It should be:

```
Picture1.Picture = Clipboard.GetData()
```

And the following line is not correct:

```
Picture = LoadPicture()
```

It should be:

```
Picture1.Picture = LoadPicture()
```

320 Print Method

In the description of expressionlist at the top of the page, the term "text expression" should read "string expression."

386 Shell Function

Change the second sentence in the description of commandstring to this:

"If the program name in commandstring does not include a .BAT, .COM, .EXE, or .PIF extension, .EXE is assumed."

412 Text Box Control

The Toolbox Icon and figure shows the menu control, not the text box control.

489 Not Operator

Search in the Visual Basic Help menu for more current information about the Not operator.

None Me Keyword

The Me keyword is not documented in the "Microsoft Visual Basic Language Reference." For complete information about the Me keyword, search in the Visual Basic Help menu.

Corrections to "Microsoft Visual Basic Professional Features," Version 2.0

Custom Control Reference

Page	Section/Note
153	Two lines in the code example for the ExtraData property need to be corrected to produce the graph illustrated on that page. Change references to ThisPoint to Graph1.ThisPoint and add the following line as the first line of code: Graph1.GraphType = 2
247	DeviceID Property The second paragraph in Remarks that starts with "The device ID may be used..." is not true.
248	DisplayhWnd Property The DisplayhWnd property is not a valid property of the MCI control. The property in the manual should be hWndDisplay. The documentation on page 248 for DisplayhWnd actually applies to the hWndDisplay property.
263	Done Event The syntax for the Done event should be: "Sub MMControl_Done(NotifyCode As Long)

ODBC Object Reference

Page	Section/Note
11	"Creating a New Table" (Code) The following line of code is incorrect: Dim f1, f1, f3, f4, f5 as New Field It needs to be broken up into individual statements: Dim f1 as New Field Dim f2 as New Field Dim f3 as New Field Dim f4 as New Field Dim f5 as New Field

Help Compiler Guide

Page	Section/Note
126	Remove the extraneous text near the top of the page beginning with ".para." and ending with "end."

Additional reference words: 2.00
KBCategory:
KBSubcategory: RefsDoc

Visual Basic User Groups in the U.S.A. and Other Countries

Article ID: Q95831

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Below is a non-exhaustive list of Visual Basic User Groups throughout the world including their name, contact name, address, and if applicable their voice phone, FAX phone and CompuServe ID numbers. The list is broken down into the following three geographic regions:

1. United States
2. Canada
3. International Countries

The geographic regions are sorted alphabetically by state, province, or country within their region.

MORE INFORMATION

Visual Basic User Groups in the United States:

Tucson Computer Society
Contact: Bruce Fulton
516 E. Mabel
Tucson, AZ 85705
Voice Phone: 602/577-7700

Pasadena PC UG
Contact: Rod Ream
2026 S. 6th St.
Alahambra, CA 91830
Voice Phone: 818/280-6850

Berkeley PC UG
Contact: Gustavo Edelstyn
2625 Alcatraz Avenue #271
Berkeley, CA 94705
Voice Phone: 415/553-8739
CompuServe ID:71552,3052

San Fransisco PC UG
Contact: Dov Gorman
1127 Bancroft Way
Berkeley, CA 94702

Voice Phone: 510/339-3414

Sacramento PC UG
Contact: Larry Clarke
345 Pruewett Drive
Folsom, CA 95630
Voice Phone: 916/983-3950

Orange Coast PC UG
Contact: Wendy Sarrett
3700 Park View Lane #22D
Irvine, CA 92715
714/966-3925

Diablo Valley PC UG
Contact: Steve Israel
1635 School St. Suite 101
Morago, CA 94556-1125
Voice Phone: 510/376-7174

Napa Valley PC UG
Contact: Frank Sommer
1253 Monticello Road
Napa, CA 94558
Voice Phone: 707/258-2509

North Orange County CC
Contact: Bill Hinds
712 N. Clinton
Orange, CA 92667
Voice Phone: 714/633-4874
CompuServe ID:76516,2623

BASIC PRO Magazine
Contact: Jim Fawcette
c/o Basic Pro
299 California Ave - Suite 190
Palo Alto, CA 94306-1912

Silicon Valley Com. Soc.
Contact: Allan Colby
107 Lake Road
Portola Valley, CA 94028
Voice Phone: 415/851-4567
71257,760

Pinellas IBM PC UG
Contact: Thomas Kiehl
14155 102nd Avenue N
Largo, FL 34644

Chicago Computer Society
Contact: Allan Wolff
1560 N. Sandburg Terrace #1715
Chicago, IL 60610
Voice Phone: 312/787-8966
CompuServe ID:72430,2717

Indianapolis Computer Society
Contact: Bill Seltzer
2064 Emily Dr
Indianapolis, IN 46260

Indianapolis Comp. Soc.
Contact: Bill Seltzer
2064 Emily Dr
Indianapolis, IN 46260
Voice Phone: 317/549-9011

Kentucky Indiana PC UG
Contact: Tim Landgrave
200 Whittington Parkway Suite 100A
Louisville, KY 40222
CompuServe ID:71760,12

The Cobb Group
Contact: Blake Ragsdale
9420 Bunsen Parkway Suite 300
Louisville, KY 40220
CompuServe ID:71321,1127

Boston Computer Society
Contact: Jim Wieler
15 Lanark Road
Arlington, MA 02147
Voice Phone: 617/648-1768
CompuServe ID:72570,66

Boston Computer Society
Contact: Bill Goodridge
30 Woodfield Road
Wellesley, MA 02181
Voice Phone: 617/239-0958

Twin City PC UG
Contact: Bill Willis
5860 73rd Ave N. #207
Brooklyn Park, MN 55429
Voice Phone: 612/566-9464

Las Vegas Computer Soc.
Contact: Carl Jarnberg
3111 S. Valley View
Suite A214
Las Vegas, NV 89102
Voice Phone: 702/876-0603

ACGNJ
Contact: James Boyd
60 Feronia Wayt
Rutherford, NJ 07070
Voice Phone: 201/438-6166

Philadelphia Area Com. Soc.

Contact: Steve Longo
c/o LaSalle University
1900 West Olney
Philadelphia, PA 19141
Voice Phone: 215/951-1255

Houston Area League
Contact: Fred Thorlin
10819 Lakeside Forest Lane
Houston, TX 77042-1025
Voice Phone: 713/784-8906
CompuServe ID:73317,662

N. Texas PC UG
Contact: Woody Pewitt
1301 East Parkerville Road
Desoto, TX 75115
Voice Phone: 214/230-3485
CompuServe ID:71670,3203

Utah Blue Chips
Contact: Jim Murtha
7563 s. 960 east
Midville, UT 84047
FAX Phone: 801-533-8004

Pac N'West PC UG
Contact: Sean Bleichschmidt
12831 N.E. 14th Place
Bellevue, WA 98005
Voice Phone: 206/455-4317

Capital PC UG
Contact: Charles Kelly
1800 G St. NW Room 408
Washington DC 20550
Voice Phone: 202/357-9796
CompuServe ID:71044,1124

User Groups in Canada:

Philadelphia Area Com. Soc.
Contact: Steve Longo
c/o LaSalle University
1900 West Olney
Philadelphia, PA 19141
Voice Phone: 215/951-1255

Winnepeg PC UG
Contact: Kent Sharkey
210 Montgomery Ave
Winnipeg Manitoba, Canada
Voice Phone: 204/989-6870

Toronto Win UG

Contact: Don Roy
6327 Atherley Crescent
Mississaugua Ontario Canada L5N 2J1
Voice Phone: 416/826-0320
CompuServe ID:76675,1272

International User Groups:

Taiwan VB Program Group
Contact: Andy Kuo
U Lead Systems, Inc.
12F-A, 563 Chung Hsiao E. Rd - Section 4
Taipei, Taiwan R.O.C.
Fax Phone: 011-86-2-764-9599

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: RefsThird

Differences Between VCP Version 1.0 and VB Version 2.0 or 3.0

Article ID: Q98544

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic programming system for Windows, versions 2.0 and 3.0
- Microsoft Visual C++ programming system for Windows and MS-DOS, version 1.0
- Microsoft Visual Control Pack, version 1.0

SUMMARY

If you have the Professional edition of Microsoft Visual Basic version 2.0 or 3.0 for Windows, you have everything that the Microsoft Visual Control Pack (VCP) has and more.

All controls, tools, and documentation shipped with the Microsoft Visual Control Pack are identical to those same controls, tools, and documentation shipped with the Professional Edition of Microsoft Visual Basic versions 2.0 for Windows, with two exceptions:

- A new copy of the MSCOMM.VBX custom control that works with Visual C++ version 1.0 comes with the Visual Control Pack version 1.0.
- Enhanced Control Development Kit (CDK) documentation including helpful hints on creating custom controls for use with Microsoft Visual C++ version 1.0 comes with the Visual Control Pack version 1.0.

MORE INFORMATION

The Microsoft Visual Control Pack includes a newer MSCOMM.VBX custom control. This newer MSCOMM.VBX is slightly enhanced to work with Microsoft Visual C++ version 1.0. The newer control does not work any differently or any better than the one that comes with Visual Basic version 2.0 for Windows.

If you have Visual C++ version 1.0 and currently own the Professional Edition of Microsoft Visual Basic version 2.0 for Windows, you can get a free copy of the new MSCOMM.VBX custom control from Microsoft Visual Basic Product Support by calling (206)646-5105.

The new MSCOMM.VBX custom control and the enhanced CDK documentation come with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows. The enhanced CDK documentation was also shipped as part of the April 1993 release of the Microsoft Developer Network (MSDN) CD.

Additional reference words: 1.00 2.00 3.00 VC++
KBCategory:
KBSubcategory: RefsProd

Data Manager Source Code Available on CompuServe

Article ID: Q99643

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

The source code for the Visual Basic Data Manager is available. You may download the source code, modify, and distribute it royalty free. To obtain the source code, download DATAMGR.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  - GO MSL
  - Search for DATAMGR.EXE
  - Display results and download

- Microsoft Download Service (MSDL)
  - Dial (206) 936-6735 to connect to MSDL
  - Download DATAMGR.EXE

- Internet (anonymous FTP)
  - ftp ftp.microsoft.com
  - Change to the \softlib\mslfiles directory
  - Get DATAMGR.EXE

MORE INFORMATION

The source code has been made available because of a press release announcement made by Microsoft that stated that the source code for the Visual Basic version 3.0 Data Manager would be available on CompuServe.

Additional reference words: 3.00 CISFile3.00 softlib S14635
KBCategory:
KBSubcategory: RefsProd

International and U.S. Support for Crystal Reports

Article ID: Q100368

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

Microsoft supports setup and installation for the Crystal Reports product shipped with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows. For other Crystal Reports support, please contact Crystal Services, not Microsoft.

MORE INFORMATION

The following lists international and U.S. telephone numbers you can call to get technical support for Crystal Reports. Also listed is the CompuServe ID and mailing address for Crystal Reports support.

Canada/US

Crystal Services
Suite 2200 - 1050 West Pender Street
Vancouver, BC, Canada V6E 3S7

Phone: 604-669-8379 (8:00am - 5:00pm pacific time)
Fax: 604-681-7163
BBS: 604-681-9516

Product support via CompuServe:
Send CompuServe mail to : 71035,2430

England

Company: Contemporary Software
Phone: 273-483-979
Fax: 273-486-224

Netherlands

Company: Microscope
Phone 10-456-3799
Fax 10-456-5549

Australia

Company: Sourceware
Phone: 2-427-7999
Fax: 2-427-7255
"Ask for Tony Johnson"

For a complete list of Crystal Reports support offerings see the last three pages (PSS 1 - PSS 3) of the "Microsoft Visual Basic Professional Features Book 2" manual

Additional reference words: 3.00
KBCategory:
KBSubcategory: RefsProd PrgCtrlsCus

LONG: Corrections for Errors in VB Version 3.0 Manuals

Article ID: Q100369

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article is a master list of corrections for documentation errors in the Microsoft Visual Basic version 3.0 for Windows manuals.

Included are corrections to version 3.0 of the following manuals and files that ship with both the Standard and Professional Editions:

- Online Help file VB.HLP
- "Microsoft Visual Basic for Windows Programmer's Guide"
- "Microsoft Visual Basic for Windows Language Reference"

Also included are corrections to version 3.0 of the following manuals that ship with the Professional Edition only:

- "Microsoft Visual Basic for Windows Professional Features Book 1"
  - Custom Control Reference
  - Control Development Guide
  - Help Compiler Guide
- "Microsoft Visual Basic for Windows Professional Features Book 2"
  - Data Access Guide
  - Appendixes
  - Crystal Reports for Visual Basic User's Manual

This master list of corrections includes and adds to the correction list found in "Part 4: Notes for Microsoft Visual Basic Online Help" and "Part 5: Notes for Microsoft Visual Basic Programmer's Guide" of the README.TXT file shipped with the product. Please use this article as your master list for making corrections to the manuals and help file.

MORE INFORMATION

Corrections to the Online Help File VB.HLP

Rediminsioning an Array

Help says that you can rediminsion an array from -32768 to 32767. Actually, you cannot use the bottom number. That is, you can rediminsion an array from -32767 to 32767 as in this example:

```
Sub Form_Load ()
   Redim x%(-32767 To 32767)
End Sub
```

Error Message Help

Online Help is not available for dialogs or error messages that occur at design time. For ISAM errors, use the Search dialog in Help.

"Couldn't find installable ISAM"

An external file dynamic link library (DLL) file couldn't be found. This file is required for operations such as attaching or accessing external tables.

Possible causes:

1) An entry in the [Installable ISAM] section in VB.INI or <APPNAME>.INI is incorrect. For example, this error occurs if you're accessing a Paradox external table, and the Paradox entry of the .INI file points to a nonexistent directory. Exit Visual Basic, make necessary corrections in VB.INI or <APPNAME>.INI using Microsoft Windows Notepad or another text editor, restart Visual Basic, and try the operation again.

2) One of the entries in the [Installable ISAM] section in VB.INI points to a network drive, and that drive isn't connected. Check to make sure the network is available and the proper drive letter is established, and try the operation again.

Attributes Property

The Help topic for the attributes property and the DATACONS.TXT file incorrectly list DB_SYSTEMOBJECT as having a value of &H80000002. The correct value is H80000000.

Index Property (Data Access)

The information listed in the Index Property (Data Access) Help topic is not correct. Here is the correct information for this topic:

Applies To
  Table object

Description
  With data access, determines which existing index is the current index used to sort records in a Table and in recordsets created from that Table. The default is blank. Not available at design time; read/write at run time.

Syntax
  table.Index [= indexname]

Remarks

The order of the data in a table is determined by the order in which the data is added to the table. To alter the order of records fetched from the table when using a Table object, set the Index property to the name of an index in the Indexes collection of the Table's TableDef object. For example, to set the index to be used on a Seek against the Titles table:

```
Dim Tb as Table, Db as Database
Set Db = OpenDatabase("Biblio.MDB")
Set Tb = Db.OpenTable("Titles")
Tb.Index = "PubID"
Tb.Seek "=", 3
```

The specified index must already be defined. If you set the Index property to an index that doesn't exist, or if the index isn't set when you use the Seek method, an error occurs.

In the Professional Edition, you can create a new Index in a Table by creating a new Index object, setting its properties, then appending it to the Indexes collection of the Table's TableDef.

The records in a Table can be ordered only according to the indexes defined for it. To sort the Table records in some other order, create a new Index for the table and append it to the Table's Index Collection, or create a Dynaset or Snapshot that has a different sort order. To specify the sort order for Dynasets and Snapshots, use the Sort property after the Dynaset or Snapshot has been created. You can also set the order of a Dynaset or Snapshot by including an Order By clause in a SQL statement used to define the Dynaset or Snapshot.

The Index property of a control array element is not the same as the Index property of a data access object.

Data Type

String

OpenQueryDef Example Code

In the example, the name of the parameter is "Enter State" not "State Wanted," and the name of the existing query is "By State" not "Get State."

==========

Corrections to "Programmer's Guide"

==========

(Page 188) The New Keyword

In the example at the top of the page, the local form variable F is declared with the New keyword using the Dim statement. To make the form variable and the loaded form instance persist, use a Static or Global variable instead.

(Page 194) Determining the Type of an Object Variable

You can use the If...TypeOf statement to determine the

control type of a custom control:

If TypeOf object Is objecttype

The identifier you use for 'object' is the class name of the custom control. See the section "Specific Control Object Types" (P. 186) for more information.

(Page 461) The Options Property

The constant values shown are in hexidecimal and should be preceded with the &H notation. For example, DB_SQLPASSTHROUGH = &H40, not decimal 40. See online Help (Options Property) or the file DATACONS.TXT for the correct values.

(Page 454) BIBLIO.MAK and DATAMGR.EXE

The second sentence of the first paragraph should read:

If you installed the sample applications, you will find this application in the \DATACTRL subdirectory of the Visual Basic SAMPLES subdirectory (\VB\SAMPLES\DATACTRL).

The third sentence of the second paragraph shoul read:

You will find DATAMGR.EXE in the main Visual Basic directory (\VB).

(Page 456) Getting a Quick Start. Item 6.

Delete the second sentence which begins "Set the DataSource property for Label1 ..." under item 6.

(Page 458) Setting Database Properties at Design Time

This section incorrectly states that at design time the RecordSource property of the Data Control lists all tables and queries. The RecordSource property lists the tables in a database, not the queries.

At design time, if the DatabaseName or Connect property of the Data Control is set, the RecordSource property will retrieve a list of all available tables. If the user knows of a valid SQL query for the database, the RecordSource property will allow the query to be typed in, but it does not list the queries.

(Page 459) The Connect Property

In the table for the Connect property setting, change the Connect setting for Paradox from the following:

```
paradox;pwd=password;
```

to:

```
paradox 3.x;pwd=password;
```

NOTE: The database name in the Connect setting must match (except for case) the database name in the VB.INI file. See page 148 of "Professional Features Book 2."

(Page 460) The DatabaseName Property

The first paragraph on this page incorrectly says the RecordSource property of the Data Control lists all tables and queries. The RecordSource property lists the tables, not the queries.

At design time, if the DatabaseName or Connect property of the Data Control is set, the RecordSource property will retrieve a list of all available tables. If the user knows of a valid SQL query for the database, the RecordSource property will allow the query to be typed in, but it does not list the queries.

(Page 462) The RecordSource Property

The first sentence in the second paragraph should be changed to remove the reference to queries. Queries are not returned by the RecordSource Property. In other words, change the following:

At design time you can choose from a list of database tables and queries by first ...

to:

At design time you can choose from a list of database tables by first ...

In addition, the following text and example should be changed:

For example, the following SQL query returns all of the columns in the bibliography for authors who live in New York:

```
Data1.DatabaseName = "BIBLIO.MDB"
Data1.RecordSource = "Select * from Titles where state = 'NY'"
Data1.Refresh
```

The above should read:

For example, the following SQL query returns all of the columns in the bibliography for publishers based in New York:

```
Data1.DatabaseName = "BIBLIO.MDB"
Data1.RecordSource = "Select * from Publishers where
                      state ='NY'"
Data1.Refresh
```

(Page 465) Adding a New Record

In the second paragraph in this section, the last sentence should read, "Notice that using the buttons on the data control or one of the Move methods to move to another record will automatically save your added record."

(Page 530) Determining How an Object Is Displayed

In the first paragraph, the second sentence should read, "the Icon check box," not "th eIcon check box."

(Page 550) Creating Invisible Objects

In the sample code, the following line has incorrect syntax:

```
MyWord = ObjVar.SuggestWord MyWord
```

The code should look like this:

```
MyWord = ObjVar.SuggestWord (MyWord)
```

(Page 552) Limitations in Visual Basic

Under the discussion "Arrays and User-Defined Types," the third bulleted item should read: "You cannot assign the return value of a property or method to an array variable or a variable of a user-defined type."

(Page 554) Closing an Object

In the paragraph after the sample code, second sentence: It is not true that invoking a Close method on an object sets variables that refer to the object to Nothing.

(Page 582) Determining the Files You Need to Distribute

The following additional files are required for distributing your Visual Basic applications:

DLL Name	Required by (Professional Edition Only)
PDIRJET.DLL	Crystal Reports for Visual Basic
PDBJET.DLL	Crystal Reports for Visual Basic
MSAJT110.DLL	Crystal Reports for Visual Basic
MSAES110.DLL	Crystal Reports for Visual Basic
PDSODBC.DLL	ODBC and Crystal Reports for Visual Basic

(Page 643) Symbol Tables

The first bullet item under Module Symbol Table should be under Global Symbol Table:

- The actual text of the names of Sub and Function procedures

==========

Corrections to "Language Reference"

==========

(Page 21-22) Action Property (OLE)

In the Settings table, in Setting 5, the reference to None in the second sentence of the third paragraph should read as follows:

"If the Paste was not successful, the OleType property will be set to 3 (None)."

In Setting 12, the constant should be OLE_READ_FROM_FILE, not ReadFromFile. In Setting 14, the constant should be OLE_INSERT_OBJ_DLG.

(Page 41) AutoActivate Property

In the Note, replace the words "the double-click event" with "a DblClick event."

(Page 53) BorderStyle Property

The OLE control cannot have a setting of 2. Remove the setting and description for Setting 2 in the OLE control table.

(page 57) Caption property

For labels, the caption is limited to 1024 characters, not 2048.

(Page 65) Check Box Control

Add DataField and DataSource to the Properties list.

(Page 82) Color Property

The "Applies To" line should read "Common dialog (Color dialog)."

(Page 89) Common Dialog Control

Add "FilterIndex" and "MaxFileSize" to the Properties (File dialogs) list.

(Page 90) CompactDatabase Function

The Syntax line indicates that both the second and third function parameters are optional. This is incorrect. The second parameter is required. Use DB_LANG_GENERAL as the default.

(Page 93) Connect Property

In the Note, change "SourceTable" to "SourceTableName."

(Page 97) Copies Property

The "Applies To" line should read "Common dialog (Print dialog)."

(Page 97) Controls Collection

The following three statements are incorrect:

```
If TypeOf Frm.Controls(I) Is Not Menu Then
   Frm.Controls(I).Enabled = State
End If
```

Replace them with the following four statements:

```
If TypeOf Frm.Controls(I) Is Menu Then
Else
   Frm.Controls(I).Enabled = State
End If
```

(Page 100) CreateDatabase Function

Three corrections are necessary:

- In the code example, replace "False" with "DB_VERSION10."
- In the table above the code example, replace "DB_COMPACT_ENCRYPT" with "DB_ENCRYPT."
- The Syntax line indicates that both the second and third function parameters are optional. This is incorrect. The second parameter is required. Use DB_LANG_GENERAL as the default.

(Page 111) Data Control

Add UpdateControls and UpdateRecord to the Methods list.

(Page 112) Database Object

In the Properties list, the QueryTimeout Property should be identified as being available only in the Professional Edition.

(Page 117) DataText Property

In the code example, change the two instances of "MSDRAW" to "MSGRAPH."

(Page 134) DefaultExt Property

The "Applies To" line should read:

"Common dialog (File dialogs)."

(Page 144) Dir, Dir$ Functions

Line 11 of the sample program is incorrect. It reads:

```
If GetAttr(Path + DirName) And ATTR_DIRECTORY = ATTR_DIRECTORY Then
```

It should read:

```
If GetAttr(Path + DirName) = ATTR_DIRECTORY Then
```

(Page 185) Field Object

The Properties list should refer to SourceField and SourceTable, not SourceFieldName and SourceTableName.

(Page 187) Fields Property

The third line of the Example (Professional Edition Only) given at the bottom of the page is incorrect. It should be changed to:

TempIndex.Fields = "LName;FName"

The correct version has no space between the two fields.

(Page 195) FileTitle Property

The "Applies To" line should read:

"Common dialog (File dialogs)."
Add the following to the Remarks section:

Note: If the OFN_NOVALIDATE flag is set, the FileTitle property will not return a value.

(Page 198) Filter Property (Common Dialog)

At the beginning of the topic, add "Applies To...Common dialog (File dialogs)." In the Remarks section, after the third paragraph, add this text:

Here is an example of a Filter in which the user can choose text files or picture files that include bitmaps and icons:
Text(*.txt)|*.txt|Pictures(*.bmp;*.ico)|*.bmp;*.ico

(Page 199) FilterIndex Property

The "Applies To" line should read
"Common dialog (File dialogs)."

(Page 229) Frame Control

Add the Name Property to the Properties list.

(Page 231) FromPage, ToPage Properties

The "Applies To" line should read
"Common dialog (Print dialog)."

(Page 240) GetAttr Function

The final Sub...End Sub block in code should read as follows:

```
Sub File1_Click ()
   Const ATTR_READONLY = 1, ATTR_HIDDEN = 2     ' Declare
   Const ATTR_SYSTEM = 4, ATTR_ARCHIVE = 32     ' Constants.
   Dim Attr, FName, Msg                    ' Declare variables.
   If Right(Dir1.Path, 1) = "\" Then      ' See if root file.
      FName = Dir1.Path & File1.FileName  ' Get file path.
   Else
     FName = Dir1.Path & "\" & File1.FileName    ' Get file
                                                 ' path.
   End If
   Attr = GetAttr(FName)  ' Get attributes.
   If Attr > 7 Then Attr = Attr Xor ATTR_ARCHIVE  ' Disregard
                                                  'Archive.
   Select Case Attr       ' Look up attributes.
```

```
            Case 0: Msg = "Normal"
            Case ATTR_READONLY: Msg = "Read-Only"
            Case ATTR_HIDDEN: Msg = "Hidden"
            Case ATTR_HIDDEN +  ATTR_READONLY: Msg = "Hidden and Read-Only"
            Case ATTR_SYSTEM: Msg = "System"
            Case ATTR_READONLY +  ATTR_SYSTEM: Msg = "Read-Only and System"
            Case ATTR_HIDDEN +  ATTR_SYSTEM: Msg = "Hidden and System"
            Case ATTR_READONLY + ATTR_HIDDEN +  ATTR_SYSTEM:
              - Msg = "Read-Only," + Msg = " Hidden, and System"
         End Select
         MsgBox UCase(FName) & " is a " & Msg & " file."    ' Display
                                                            ' message.
      End Sub
```

(Page 256) hDC Property

The Usage line should read:

```
{[form.] [commondialog. | picturebox.] | Printer.}hDC
```

Also, the second paragraph of the Remarks should read:

"With a common dialog control, this property returns a device context for the printer selected in the Print dialog box when the..." (the rest of the text remains the same).

(Page 258) Height, Width Properties

The See Also line should refer to the "Width # Statement," not the "Width Statement."

(Page 268) hWnd Property

In the example code, the last argument in the SetWindowPos Declare (ByVal f as Long) is incorrect, it should be ByVal f as Integer. As written, the code generates a "Bad DLL Calling Convention" error. Change the Long to Integer and the call works.

In addition, the following line of code doesn't work:

```
mnuTopmost.Checked = Not mnuTopmost.checked
```

This is because the Value property of a check box does not accept -1 (the results of NOT on 0), which is the initial value. Replace the incorrect line of code with this code:

```
If mnuTopMost.Checked = 1 Then
   mnuTopMost.Checked = 0
Else
   mnuTopMost.Checked = 1
End if
```

(Page 274) Image Control

Add DataField and DataSource to the Properties list.

(Page 279) Index Property (Data Access)

The following information, in the Remarks section, is incorrect:

> To set this property with a data control, specify the TableDef, set the index, and then Refresh the control:
>
> ```
> Data1.RecordSource = "Publishers"
> Data1.Database.TableDefs("Publishers").Index = "PrimaryKey"
> Data1.Refresh
> ```

Replace it with the following:

> You cannot set the Index property with a data control. To use an indexed field in Visual Basic, use a SQL statement similar to the following example:
>
> ```
> Data1.RecordSource = "SELECT * FROM Publishers ORDER BY Zip"
> Data1.Refresh
> ```
>
> By using the ORDER BY clause in the SQL syntax, you can simulate the effect of the Index property.

(Page 279) Index Property (Data Access)

The "Applies To" says TableDef but should say "Table."

(Page 280) InitDir Property

The "Applies To" line should read:

"Common dialog (File dialogs)."

(Page 280) Indexes Collection

The See Also section makes reference to a "CreateIndex Method." This method does not exist. The reference should be omitted.

(Page 281) Input # Statement

The last paragraph, second-to-last sentence is incorrect. Change it to read as follows:

"For strings not delimited by double quotation marks, the end of a string is assumed when a comma or the end of a line is encountered."

(Page 297) KeyDown, KeyUp Events

The See Also should refer to the SendKeys Statement, not the SendKeys Method.

(Page 299) KeyPress Events

The See Also should refer to the SendKeys Statement, not the SendKeys Method.

(Page 303) Label Control

Add the DataField, DataSource, and Parent properties to the Properties list.

(Page 336-338) ListFields Method

In the second table, the fifth and sixth entries in the Field column should be SourceTable and SourceField, not SourceTableName and SourceFieldName. The code example and the headings of the table below it should also refer to SourceTable and SourceField.

(Page 345) ListTables Method

In Remarks, the first paragraph under the TableType field table should read:

"When you use the ListTables method to create a Snapshot, you can evaluate the contents of the Attributes field in the Snapshot by referring to the TableDef property settings table in the Attributes property topic.

(Page 361) Max, Min Properties (Common Dialog)

At the beginning of the topic, add:

"Applies To...Common dialog (Font, Print dialogs)."

(Page 363) MaxFileSize Property

The "Applies To" line should read:

"Common dialog (File dialogs)."

(Page 381) MousePointer Property

The Note section near the bottom of the page is incorrect. It should read:

Note When set for the Screen object, MousePointer changes for the Visual Basic application; that is, it only overrides the application's MousePointer settings.

(Page 390) Name Property

The "Applies To" line should include the Database object.

(Page 418) OpenQueryDef Method

In the example, the name of the parameter is "Enter State" not "State Wanted," and the name of the existing query is "By State" not "Get State."

(Page 432) Partition Function

In the code in Example 3, the second five lines of code duplicate the first five lines and should be deleted.

(Page 439) Picture Box Control

Add DataField and DataSource to the Properties list.

(Page 444) PopupMenu Method

In the Syntax line, there should be a comma immediately before the y.

(Page 455) PrinterDefault Property

The "Applies To" line should read:

"Common dialog (Print dialog)."

(Page 536-537) SourceFieldName, SourceTableName Properties

All references to SourceFieldName and SourceTableName in this topic should refer to "SourceField" and "SourceTable" instead.

(Page 538) SourceTableName Property

There should be a full entry for the "SourceTableName" topic. See online Help for the text of this topic.

(Page 565) Text Box Control

The second piece of art is incorrect. It should show a text box on a form but instead, it shows a menu title and menu items on a form. Also, add DataField and DataSource to the Properties list.

(Page 595) Validate Event

In the third paragraph following the Constants table, change "edit buffer" to "copy buffer."

(Page 619) Trappable Errors

In Appendix B, the odd header is wrong. It should read "Trappable Errors," not "Trappable Error Messages."

(Page 634) Trappable Error Messages

In Table B.6 ("Data Access Trappable Error Messages"), Error #3137 should be deleted.

==========

Corrections to "Professional Features Book 1 -- Custom Control Reference"

==========

(Page xxii) Visual Basic Executable (.EXE) Files

The Visual Basic run-time file is listed incorrectly. The first bulleted item should read VBRUN300.DLL, not VBRUN200.DLL.

(Page 69) CDHolding Property

Cross out the following paragraph. It is incorrect. It contradicts

the Remarks under CDTimeout Property:

When the Carrier Detect line is high (CDHolding=True) and the CDTimeout number of milliseconds has passed, the communications control sets the CommEvent property to MSCOMM_ER_CDTO (Carrier Detect Timeout Error), and generates the OnComm event.

(page 78) Input Property

The Remarks section lists the SendMessage syntax incorrectly. Replace it with this syntax:

```
pMSComm->SendMessage(UM_INPUT,cbData,lpData)
```

(Page 107) Graphs Within Graphs

This section states, "The graph control can have child windows. You can place other controls (including more graphs) within a graph."

This information is incorrect. The graph control in Visual Basic version 3.0 does not support child controls. You cannot place a control of any type on a graph and have it belong to the graph. The entire section should be removed.

(Page 147) Graph Control

In Example 1, the following line contains two "=" characters:

```
Graph1.LabelText = "Data point" = Str$(i%)
```

The line should read:

```
Graph1.LabelText = "Data point" + Str$(i%)
```

(Page 148) Graph Control

In Example 2, the following line contains two "=" characters:

```
Graph1.LabelText = "Label" = Str$(i%)
```

The line should read:

```
Graph1.LabelText = "Label" + Str$(i%)
```

(Page 176) Key Status Control

The table for the Value property incorrectly states that False is the default value. The default value is determined by the state of the keyboard.

(Page 180) MAPI Session Control

There should be no footnotes, since the MAPI controls are only available in Visual Basic.

(Page 186) MAPI Messages Control

There should be no footnotes, since the MAPI controls are only available in Visual Basic.

Corrections to "Professional Features Book 1 -- Help Compiler Guide"

(Page 68) Running Macros When a User Enters a Topic

Insert the following sentence after the first sentence: "Macro calls can be authored in footnotes that use an exclamation (!) as the reference mark."

(Page 117) Mapping Context Sensitive Objects

The text attached to the third bullet is incorrect. There must be exactly one space between the context number and the context string. The example on the following page is also incorrect. It should be corrected to reduce the multiple spaces to one space as follows:

```
[MAP]

Edit_Window 0x001
Control_Menu 0x002
Maximize_Icon 0x003
... and so on ...
```

Corrections to "Professional Features Book 2 -- Data Access Guide"

(Page 31) Creating New Table Definitions

Delete the following line of code from the example:

```
On Error Resume Next
```

(Page 58) Using the Options Argument

Change the example code near the middle of the page to this:

```
Dim Options%
Dim Db As Database
Dim T As Table
Set Db = OpenDatabase ("BIBLIO.MDB")
Options% = DB_DENYWRITE + DB_DENYREAD
Set T = Db.OpenTable ("Titles", Options%)
```

(Page 96-97) QueryDef Example Code

In the example, the name of the parameter is "Enter State" not "State Wanted," and the name of the existing query is "By State" not "Get State."

(Page 108) Transaction Logging

The reference to a trappable error (2004) in the last sentence is incorrect. There is no such error code. When a transaction log fills, it does not cause error 2004.

(Page 139) Accessing Paradox Tables

The following line is incorrect:

```
conn$ = "Paradox;"
```

It should read as follows:

```
conn$ = "Paradox 3.X"
```

(Page 154) Accessing Microsoft SQL Server Databases

The reference to two versions of INSTCAT.SQL (INSTCAT.SQL and INSTCAT.48) that are supposedly used differently depending on whether the backend is Microsoft SQL Server or Sybase, is an error. The single version of INSTCAT.SQL provided by Microsoft on the Visual Basic version 3.0 disks is complete and sufficient for both Microsoft SQL Server and Sybase SQL Server, versions 4.2 and later. The file named INSTCAT.48, if you have it, is not useful and can be deleted.

The instructions on how to run INSTCAT.SQL, which formerly were found in Appendix D of the version 2.0 "Professional Features" manual, are no longer included in the manual. Page 154 of the version 3.0 manual says you can find information on setup, configuration, and operational issues when accessing tables from SQL Server in a file named SQLSVR.HLP. In fact, this file does not exist. The correct file name is DRVSSRVR.HLP, and you should find it in the \WINDOWS\SYSTEM directory.

In the DRVSSRVR.HLP file, search on "INSTCAT.SQL" to find the syntax of the ISQL batch command that you need to use to run the INSTCAT.SQL file.

Additional reference words: 3.00 docerr
KBCategory:
KBSubcategory: RefsDoc

README.TXT for Standard Edition of VB ver 3.0 for Windows

Article ID: Q100492

The information in this article applies to:

- Standard Edition of Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

The following article contains the complete contents of the README.TXT file distributed with Microsoft Standard Edition of Visual Basic version 3.0 for Windows.

MORE INFORMATION

README.TXT

Release Notes for Microsoft (R) Visual Basic (TM) Standard Edition

Version 3.00

This document contains release notes for Microsoft Visual Basic Standard Edition version 3.0 for Windows. Information in this document is more current than that in the manuals or online Help.

How to Use This Document

To view README.TXT on screen in Windows Notepad, maximize the Notepad window.

To print README.TXT, open it in Windows Write, Microsoft Word, or another word processor. Then select the entire document and format the text in 10-point Courier before printing.

****
Read Part 1 - Software Installation Information - before installing.
****

Contents

Part	Description
1	Software Installation Information
2	Notes and Tips
3	Notes for "Learning Microsoft Visual Basic" Tutorial

4 Notes for Microsoft Visual Basic Online Help
5 Notes for Microsoft Visual Basic "Programmer's Guide"
6 Notes for Microsoft Visual Basic "Language Reference"

Part 1: Software Installation Information

To install Visual Basic, use the Program Manager or File Manager to start SETUP.EXE as you would any other Windows-based application. For example, if you are installing from drive A:

- From the Program Manager File menu, choose Run.

- In the Run dialog box, type A:SETUP and choose OK.

  Or

- From the File Manager, double-click the SETUP.EXE file icon on drive A.

File Sharing for OLE and Data Access

You will need SHARE.EXE to enforce file and byte range locking if the following programs are running simultaneously:

- Two Visual Basic applications that perform data access
- Two instances of one Visual Basic application that performs data access
- Microsoft Access and a Visual Basic application that performs data access

If you are running Microsoft Windows 3.0 or Windows 3.1 (i.e. not Windows for Workgroups), you will need to change your AUTOEXEC.BAT as follows:

```
SHARE /L:500 /F:5100
```

If the /L or the /F setting to SHARE has a larger value than listed here, leave the setting as it is rather than reducing it.

Please make this change as soon as possible in your system.

Note

If you ship .EXE files to users that use OLE or data access, they must add this setting to their AUTOEXEC.BAT file as well.

Visual C++ and GRID.VBX

If you plan to install Microsoft Visual C++(TM) Development System for Windows on your system, you may overwrite the grid control in your Windows \SYSTEM directory. Before installing Visual C++, make a backup of GRID.VBX. If you have already installed Visual C++, you can re-install GRID.VBX by

running Setup again, choosing the Custom Installation button, and then selecting the Microsoft Visual Basic option.

Part 2: Notes and Tips

Data Access Compatibility - Visual Basic 2.0 and 3.0

The new functionality of the Microsoft Access engine in Visual Basic 3.0 will affect the behavior of your VB 2.0 code. Visual Basic 3.0 will now assume that all SQL statements take the form of Access SQL (see the Access documentation). You can passthrough SQL to ODBC; as a second parameter to the CreateDynaset method, you must specify &H40 to indicate SQL passthrough. Note that if you used backend-specific SQL with CreateDynaset, you must add this parameter.

ATI Driver

If you have an ATI Wonder card, moving the data control around at design time may cause a GP fault. Contact ATI for more information.

BackColor Displayed Incorrectly with 256-Color Bitmaps

If you have a 256-color bitmap in any control (or the form) that has a BackColor property, it is possible that the background color of that object will not display correctly if no color in the bitmap's palette matches the background color.

"Communication Link Failure" Error

If you receive a "Communication Link Failure" error when executing queries against a Microsoft or Sybase SQL Server, you can retry the operation with asynchronous execution disabled. To do this, add the following entry to your VB.INI file:

```
[Debug]
RmtTrace=16
```

Visual Basic will continue to run synchronously until this line is removed from VB.INI.

OLE Class Names

To get a reference to a currently running OLE application by using its class name, you must use:

```
GetObject(, "classname")
```

that is, GetObject has to have an empty file name. A file name of "" will create a new instance of the server for class name. This is contradictory to what is documented.

OLE Control: In-Place Activation

The Visual Basic OLE 2.0 control supports "Inside-Out Activation," which corresponds to "activation on GetFocus." If a server designates this capability and you specify AutoActivate = GETFOCUS, then:

- The object is initially deactivated the first time the control gets the focus.

- The object is not fully deactivated when the control loses focus. Instead, the object is directed to tear down any floating user interface it may have, such as tool pallettes.

Note that AutoActivate = GETFOCUS is only supported if the server claims to be "inside-out-capable," further reducing the possibility of looping re-activation when a non-insitu server closes and gives focus back to the control.

OLE Control: Pasting Objects from the Clipboard

Applications that provide objects behave differently when an object is deleted. When you delete an OLE object (set Action = 10), the object's application may or not close. If the application does close, any objects on the Clipboard associated with that application may also be closed. Because of this, you may not be able to cut an object (copy, then delete), since deleting the object may also cause the data on the Clipboard to be deleted.

Another instance of this behavior is when you try to copy an object, then paste the object back onto itself. This action may cause an error, because in order to paste over an existing object, the existing object is first deleted. If the application associated with the object closes, and subsequently deletes any objects it has on the Clipboard, the Clipboard no longer contains an object to paste.

OLE Control: The PasteOK Property

The following applies to objects on the Clipboard that come from an OLE object:

- When PasteOK returns True, there is no guarantee that the Paste operation will succeed. For example, PasteOK returns True and the Paste fails when there is a linked object on the Clipboard and you are pasting into an object whose OleTypeAllowed property is set to 1 (Embedded).

- PasteOK returns False when there is a linked object on the Clipboard and you are pasting into an object whose OleTypeAllowed property is set to

0 (Linked).

OLE Control: Link Target

You cannot activate a linked object as hidden (set Verb = -3). You can, however, activate an embedded object as hidden.

OLE Constants in CONSTANT.TXT

OLE has also defined two new standard verbs relating to the two states that an in-place-active object can have. The following corresponding constants have been added to CONSTANT.TXT:

Constant	Description
VERB_INPLACEUIACTIVATE	Object fully in-place active, including floating UI. Only one at a time per top-level form can be in this state.
VERB_INPLACEACTIVATE	Object is semi-active; it is running and ready to respond to user input like clicking within the object or changing the mouse pointer as the user moves the mouse over different parts of the object. Any number of objects can be in this state at a time.

So, if you have a number of inside-out-capable objects on a form, where the user has specified AutoActivate = ONGETFOCUS, the objects take turns at being in-place-UI-Active. The new verbs allow the Visual Basic programmer to cause these objects to be in the in-place-active state. For example, if you want to create a form with several of these objects, and you want the form to be as responsive as possible to user input, you would put the following code for each control into your Form_Load event handler:

```
OLEControl.Verb = VERB_INPLACEACTIVATE
OLEControl.Action = OLE_ACTIVATE
```

See the file CONSTANT.TXT for more information.

Windows 3.0 and the PopupMenu Method

Under Windows 3.0, pop-up menus invoked during a MouseDown event in some cases do not recognize menu selections made with the mouse. If that occurs, you can still make a selection using the arrow keys and the Enter key.

Saving ASCII Forms in Source Code Managers

When using source code managers, you need to change the read-only bit on the binary file (.FRX) as well as the form file (.FRM) to save the form.

Part 3: Notes for "Learning Microsoft Visual Basic" Tutorial

Save Project Before Run

The Save Project Before Run Environment Option (under the Options menu) should be set to "No" when running the "Learning Visual Basic" tutorial. Some of the lessons may be impaired if this option is set to "Yes." By default, this setting is "No."

Part 4: Notes for Microsoft Visual Basic Online Help

Error Message Help

Online Help is not available for dialogs or error messages that occur at design time. For ISAM errors, use the Search dialog in Help.

"Couldn't find installable ISAM"

An external file dynamic link library (DLL) file couldn't be found. This file is required for operations such as attaching or accessing external tables.

Possible causes:

1) An entry in the [Installable ISAM] section in VB.INI or <APPNAME>.INI is incorrect. For example, this error occurs if you're accessing a Paradox external table, and the Paradox entry of the .INI file points to a nonexistent directory. Exit Visual Basic, make necessary corrections in VB.INI or <APPNAME>.INI using Microsoft Windows Notepad or another text editor, restart Visual Basic, and try the operation again.

2) One of the entries in the [Installable ISAM] section in VB.INI points to a network drive, and that drive isn't connected. Check to make sure the network is available and the proper drive letter is established, and try the operation again.

Part 5: Notes for Microsoft Visual Basic "Programmer's Guide"

Page	Section/Note
188	The New Keyword

In the example at the top of the page, the local form variable F is declared with the New keyword using the Dim statement. To make the form variable and the loaded form instance persist, use a Static or Global

variable instead.

194 Determining the Type of an Object Variable

You can use the If...TypeOf statement to determine the control type of a custom control:

```
If TypeOf object Is objecttype
```

The identifier you use for 'object' is the class name of the custom control. See the section "Specific Control Object Types" (p. 186) for more information.

461 The Options Property

The constant values shown are in hexidecimal and should be preceded with the &H notation. For example, DB_SQLPASSTHROUGH = &H40, not decimal 40. See online Help (Options Property) or the file DATACONS.TXT for the correct values.

462 The RecordSource Property

The following text and example should be changed:

For example, the following SQL query returns all of the columns in the bibliography for authors who live in New York:

```
Data1.DatabaseName = "BIBLIO.MDB"
Data1.RecordSource = "Select * from Titles where state = 'NY'"
Data1.Refresh
```

The above should read:

For example, the following SQL query returns all of the columns in the bibliography for publishers based in New York:

```
Data1.DatabaseName = "BIBLIO.MDB"
Data1.RecordSource = "Select * from Publishers where state = 'NY'"
Data1.Refresh
```

465 Adding a New Record

In the second paragraph in this section, the last sentence should read, "Notice that using the buttons on the data control or one of the Move methods to move to another record will automatically save your added record."

530 Determining How an Object Is Displayed

In the first paragraph, the second sentence should read, "the Icon check box," not "th eIcon check box."

550 Creating Invisible Objects

In the sample code, the following line has incorrect syntax:

```
       MyWord = ObjVar.SuggestWord MyWord

       The code should look like this:

       MyWord = ObjVar.SuggestWord (MyWord)

552    Limitations in Visual Basic

       Under the discussion "Arrays and User-Defined Types," the third
       bulleted item should read: "You cannot...Assign the return value
       of a property or method to an array variable or a variable of a
       user-defined type."

554    Closing an Object

       In the paragraph after the sample code, second sentence:  It is
       not true that invoking a Close method on an object sets variables
       that refer to the object to Nothing.

582    Determining the Files You Need to Distribute

       The following additional files are required for distributing
       your Visual Basic applications:

       DLL Name       Required by (Professional Edition Only)
       --------       ---------------------------------------
       PDIRJET.DLL    Crystal Reports for Visual Basic
       PDBJET.DLL     Crystal Reports for Visual Basic
       MSAJT110.DLL   Crystal Reports for Visual Basic
       MSAES110.DLL   Crystal Reports for Visual Basic
       PDSODBC.DLL    ODBC and Crystal Reports for Visual Basic

============================================================
Part 6: Notes for Microsoft Visual Basic "Language Reference"
============================================================

Page   Section/Note
----   ------------
21-22  Action Property (OLE)

       In the Settings table, in Setting 5, the reference to None in the
       second sentence of the third paragraph should read as follows:
       "If the Paste was not successful, the OleType property will be set
       to 3 (None)."  In Setting 12, the constant should be
       OLE_READ_FROM_FILE, not ReadFromFile. In Setting 14, the constant
       should be OLE_INSERT_OBJ_DLG.

41     AutoActivate Property

       In the Note, replace the words "the double-click event" with
       "a DblClick event."

53     BorderStyle Property

       The OLE control cannot have a setting of 2. Remove the setting and
       description for Setting 2 in the OLE control table.
```

65 Check Box Control

Add DataField and DataSource to the Properties list.

82 Color Property

The "Applies To" line should read "Common dialog (Color dialog)."

89 Common Dialog Control

Add "FilterIndex" and "MaxFileSize" to the Properties (File dialogs) list.

93 Connect Property

In the Note, change "SourceTable" to "SourceTableName."

97 Copies Property

The "Applies To" line should read "Common dialog (Print dialog)."

100 CreateDatabase Function

In the code example, replace "False" with "DBVERSION10." Also, in the table above the code example, replace "DB_COMPACT_ENCRYPT" with "DB_ENCRYPT."

111 Data Control

Add UpdateControls and UpdateRecord to the Methods list.

112 Database Object

In the Properties list, the QueryTimeout Property should be identified as being available only in the Professional Edition.

117 DataText Property

In the code example, change the two instances of "MSDRAW" to "MSGRAPH."

134 DefaultExt Property

The "Applies To" line should read "Common dialog (File dialogs)."

185 Field Object

The Properties list should refer to SourceField and SourceTable, not SourceFieldName and SourceTableName.

195 FileTitle Property

The "Applies To" line should read "Common dialog (File dialogs)." Add the following to the Remarks section:

Note: If the OFN_NOVALIDATE flag is set, the FileTitle property will not return a value.

198 Filter Property (Common Dialog)

At the beginning of the topic, add "Applies To...Common dialog (File dialogs)." In the Remarks section, after the third paragraph, add this text:

Here is an example of a Filter in which the user can choose text files or picture files that include bitmaps and icons:

```
Text(*.txt)|*.txt|Pictures(*.bmp;*.ico)|*.bmp;*.ico
```

199 FilterIndex Property

The "Applies To" line should read "Common dialog (File dialogs)."

229 Frame Control

Add the Name Property to the Properties list.

231 FromPage, ToPage Properties

The "Applies To" line should read "Common dialog (Print dialog)."

240 GetAttr Function

The final Sub...End Sub block in code should read as follows:

```
Sub File1_Click ()
  Const ATTR_READONLY = 1, ATTR_HIDDEN = 2   ' Declare constants.
  Const ATTR_SYSTEM = 4, ATTR_ARCHIVE = 32
  Dim Attr, FName, Msg   ' Declare variables.
  If Right(Dir1.Path, 1) = "\" Then      ' See if root file.
     FName = Dir1.Path & File1.FileName  ' Get file path.
  Else
     FName = Dir1.Path & "\" & File1.FileName    ' Get file path.
  End If
  Attr = GetAttr(FName)  ' Get attributes.
  If Attr > 7 Then Attr = Attr Xor ATTR_ARCHIVE ' Disregard Archive.
  Select Case Attr       ' Look up attributes.
     Case 0: Msg = "Normal"
     Case ATTR_READONLY: Msg = "Read-Only"
     Case ATTR_HIDDEN: Msg = "Hidden"
     Case ATTR_HIDDEN +  ATTR_READONLY: Msg = "Hidden and Read-Only"
     Case ATTR_SYSTEM: Msg = "System"
     Case ATTR_READONLY +  ATTR_SYSTEM: Msg = "Read-Only and System"
     Case ATTR_HIDDEN +  ATTR_SYSTEM: Msg = "Hidden and System"
     Case ATTR_READONLY + ATTR_HIDDEN +  ATTR_SYSTEM:
         - Msg = "Read-Only," + Msg = " Hidden, and System"
  End Select
  MsgBox UCase(FName) & " is a " & Msg & " file." ' Display message.
End Sub
```

256 hDC Property

The Usage line should read:

{[form.] [commondialog. | picturebox.] | Printer.}hDC

Also, the second paragraph of the Remarks should read, "With a common dialog control, this property returns a device context for the printer selected in the Print dialog box when the..." (the rest of the text remains the same).

258 Height, Width Properties

The See Also line should refer to the "Width # Statement," not the "Width Statement."

274 Image Control

Add DataField and DataSource to the Properties list.

280 InitDir Property

The "Applies To" line should read "Common dialog (File dialogs)."

297 KeyDown, KeyUp Events

The See Also should refer to the SendKeys Statement, not the SendKeys Method.

299 KeyPress Events

The See Also should refer to the SendKeys Statement, not the SendKeys Method.

303 Label Control

Add the DataField, DataSource, and Parent properties to the Properties list.

336-338 ListFields Method

In the second table, the fifth and sixth entries in the Field column should be SourceTable and SourceField, not SourceTableName and SourceFieldName. The code example and the headings of the table below it should also refer to SourceTable and SourceField.

345 ListTables Method

In Remarks, the first paragraph under the TableType field table should read: "When you use the ListTables method to create a Snapshot, you can evaluate the contents of the Attributes field in the Snapshot by referring to the TableDef property settings table in the Attributes property topic.

361 Max, Min Properties (Common Dialog)

At the beginning of the topic, add "Applies To...Common dialog (Font, Print dialogs)."

363 MaxFileSize Property

The "Applies To" line should read "Common dialog (File dialogs)."

390 Name Property

The "Applies To" line should include the Database object.

432 Partition Function

In the code in Example 3, the second five lines of code duplicate the first five lines and should be deleted.

439 Picture Box Control

Add DataField and DataSource to the Properties list.

444 PopupMenu Method

In the Syntax line, there should be a comma in front of the y.

455 PrinterDefault Property

The "Applies To" line should read "Common dialog (Print dialog)."

536-537 SourceFieldName, SourceTableName Properties

All references to SourceFieldName and SourceTableName in this topic should refer to "SourceField" and "SourceTable" instead.

538 SourceTableName Property

There should be a full entry for the "SourceTableName" topic. See online Help for the text of this topic.

565 Text Box Control

The second piece of art is incorrect. It should show a text box on a form but instead shows a menu title and menu items on a form. Also, add DataField and DataSource to the Properties list.

595 Validate Event

In the third paragraph following the Constants table, change "edit buffer" to "copy buffer."

619 Trappable Errors

In Appendix B, the odd header is wrong. It should read "Trappable Errors," not "Trappable Error Messages."

634 Trappable Error Messages

In Table B.6 ("Data Access Trappable Error Messages"), Error #3137 should be deleted.

Additional reference words: 3.00
KBCategory:
KBSubcategory: RefsDoc

README.TXT for Professional Edition of VB 3.0 for Windows

Article ID: Q100493

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY
=======

The following article contains the complete contents of the README.TXT file distributed with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

MORE INFORMATION
================

README.TXT

Release Notes for Microsoft (R) Visual Basic (R) Professional Edition
Version 3.00. (C) Copyright Microsoft Corporation, 1993

This document contains release notes for Microsoft Visual Basic Professional Edition version 3.0 for Windows. Information in this document is more current than that in the manuals or online Help.

How to Use This Document

To view README.TXT on screen in Windows Notepad, maximize the Notepad window.

To print README.TXT, open it in Windows Write, Microsoft Word, or another word processor. Then select the entire document and format the text in 10-point Courier before printing.

****
Read Part 1 - Software Installation Information - before installing.
****

========
Contents
========

Part	Description
1	Software Installation Information
2	Notes and Tips
3	Notes for "Learning Microsoft Visual Basic" Tutorial
4	Notes for Microsoft Visual Basic Online Help
5	Notes for Microsoft Visual Basic "Programmer's Guide"
6	Notes for Microsoft Visual Basic "Language Reference"
7	Notes for Microsoft Visual Basic "Custom Control Reference"

Part 1: Software Installation Information

To install Visual Basic, use the Program Manager or File Manager to start SETUP.EXE as you would any other Windows-based application. For example, if you are installing from drive A:

- From the Program Manager File menu, choose Run.
- In the Run dialog box, type A:SETUP and choose OK.

  Or

- From the File Manager, double-click the SETUP.EXE file icon on drive A.

File Sharing for OLE and Data Access

You will need SHARE.EXE to enforce file and byte range locking if the following programs are running simultaneously:

- Two Visual Basic applications that perform data access
- Two instances of one Visual Basic application that performs data access
- Microsoft Access and a Visual Basic application that performs data access

If you are running Microsoft Windows 3.0 or Windows 3.1 (i.e. not Windows for Workgroups), you will need to change your AUTOEXEC.BAT as follows:

```
SHARE /L:500 /F:5100
```

If the /L or the /F setting to SHARE has a larger value than listed here, leave the setting as it is rather than reducing it.

Please make this change as soon as possible in your system.

NOTE: If you ship .EXE files to users that use OLE or data access, they must add this setting to their AUTOEXEC.BAT file as well.

Visual C++ and GRID.VBX

If you plan to install Microsoft Visual C++(TM) Development System for Windows on your system, you may overwrite the grid control in your Windows \SYSTEM directory. Before installing Visual C++, make a backup of GRID.VBX. If you have already installed Visual C++, you can re-install GRID.VBX by running Setup again, choosing the Custom Installation button, and then selecting the Microsoft Visual Basic option.

Using Visual Basic 3.0 with SQL Server and Access 1.0

If you choose to install the SQL Server ODBC driver, you will find the file INSTCAT.SQL in the \VB\ODBC directory. This is a stored procedure file that the SQL Server administrator must run on the server to prepare the server to accept Visual Basic as a client. However, this version of INSTCAT.SQL is incompatible with Microsoft Access 1.0. If you are also using Access 1.0 as a client for that SQL Server, you should use the INSTCAT.SQL that was included with Access 1.0. Visual Basic 3.0 will work correctly with INSTCAT.SQL from Access 1.0 except that it will be unable to delete indexes from tables.

Setup and LZEXPAND.DLL

Visual Basic's Setup program uses LZEXPAND to install ODBC. If you encounter a system error while Visual Basic is calculating free disk space, check for this file in your Windows \SYSTEM directory. If an older version of LZEXPAND.DLL exists along your path, remove the file and continue with Setup.

==========
Part 2: Notes and Tips
==========

Data Access Compatibility - Visual Basic 2.0 and 3.0

The new functionality of the Microsoft Access engine in Visual Basic 3.0 will affect the behavior of your VB 2.0 code. Visual Basic 3.0 will now assume that all SQL statements take the form of Access SQL (see the Access documentation). You can passthrough SQL to ODBC; as a second parameter to the CreateDynaset method, you must specify &H40 to indicate SQL passthrough. Note that if you used backend-specific SQL with CreateDynaset, you must add this parameter.

ATI Driver

If you have an ATI Wonder card, moving the data control around at design time may cause a GP fault. Contact ATI for more information.

BackColor Displayed Incorrectly with 256-Color Bitmaps

If you have a 256-color bitmap in any control (or the form) that has a BackColor property, it is possible that the background color of that object will not display correctly if no color in the bitmap's palette matches the background color.

"Communication Link Failure" Error

If you receive a "Communication Link Failure" error when executing queries against a Microsoft or Sybase SQL Server, you can retry the operation with asynchronous execution disabled. To do this, add the following entry to your VB.INI file:

```
[Debug]
RmtTrace=16
```

Visual Basic will continue to run synchronously until this line is removed from VB.INI.

OLE Class Names

To get a reference to a currently running OLE application by using its class name, you must use:

```
GetObject(, "classname")
```

that is, GetObject has to have an empty file name. A file name of "" will create a new instance of the server for class name. This is contradictory to what is documented.

OLE Control: In-Place Activation

The Visual Basic OLE 2.0 control supports "Inside-Out Activation," which corresponds to "activation on GetFocus." If a server designates this capability and you specify AutoActivate = GETFOCUS, then:

- The object is initially deactivated the first time the control gets the focus.
- The object is not fully deactivated when the control loses focus. Instead, the object is directed to tear down any floating user interface it may have, such as tool pallettes.

Note that AutoActivate = GETFOCUS is only supported if the server claims to be "inside-out-capable," further reducing the possibility of looping re-activation when a non-insitu server closes and gives focus back to the control.

OLE Control: Pasting Objects from the Clipboard

Applications that provide objects behave differently when an object is deleted. When you delete an OLE object (set Action = 10), the object's application may or not close. If the application does close, any objects on the Clipboard associated with that application may also be closed. Because of this, you may not be able to cut an object (copy, then delete), since deleting the object may also cause the data on the Clipboard to be deleted.

Another instance of this behavior is when you try to copy an object, then

paste the object back onto itself. This action may cause an error, because in order to paste over an existing object, the existing object is first deleted. If the application associated with the object closes, and subsequently deletes any objects it has on the Clipboard, the Clipboard no longer contains an object to paste.

OLE Control: The PasteOK Property

The following applies to objects on the Clipboard that come from an OLE object:

- When PasteOK returns True, there is no guarantee that the Paste operation will succeed. For example, PasteOK returns True and the Paste fails when there is a linked object on the Clipboard and you are pasting into an object whose OleTypeAllowed property is set to 1 (Embedded).
- PasteOK returns False when there is a linked object on the Clipboard and you are pasting into an object whose OleTypeAllowed property is set to 0 (Linked).

OLE Control: Link Target

You cannot activate a linked object as hidden (set Verb = -3). You can, however, activate an embedded object as hidden.

OLE Constants in CONSTANT.TXT

OLE has also defined two new standard verbs relating to the two states that an in-place-active object can have. The following corresponding constants have been added to CONSTANT.TXT:

Constant	Description
VERB_INPLACEUIACTIVATE	Object is fully in-place active, including floating UI. Only one at a time per top-level form can be in this state.
VERB_INPLACEACTIVATE	Object is semi-active; it is running and ready to respond to user input like clicking within the object or changing the mouse pointer as the user moves the mouse over different parts of the object. Any number of objects can be in this state at a time.

So, if you have a number of inside-out-capable objects on a form, where the user has specified AutoActivate = ONGETFOCUS, the objects take turns at being in-place-UI-Active. The new verbs allow the Visual Basic programmer to cause these objects to be in the in-place-active state. For example, if you want to create a form with several of these objects, and you want the form to be as responsive as possible to user input, you would put the following code for each control into your Form_Load event handler:

```
     OLEControl.Verb = VERB_INPLACEACTIVATE
     OLEControl.Action = OLE_ACTIVATE
```

See the file CONSTANT.TXT for more information.

Windows 3.0 and the PopupMenu Method

Under Windows 3.0, pop-up menus invoked during a MouseDown event in some cases do not recognize menu selections made with the mouse. If that occurs, you can still make a selection using the arrow keys and the Enter key.

Saving ASCII Forms in Source Code Managers

When using source code managers, you need to change the read-only bit on the binary file (.FRX) as well as the form file (.FRM) to save the form.

Crystal Reports for Visual Basic

All prices in the Crystal Reports for Visual Basic editor, CRW.EXE, are listed in US dollars.

If you are calling Crystal Services from outside the United States for support or more information, please contact your local international long distance carrier if you need assistance.

Part 3: Notes for "Learning Microsoft Visual Basic" Tutorial

Save Project Before Run

The Save Project Before Run Environment Option (under the Options menu) should be set to "No" when running the "Learning Visual Basic" tutorial. Some of the lessons may be impaired if this option is set to "Yes." By default, this setting is "No."

Part 4: Notes for Microsoft Visual Basic Online Help

Error Message Help

Online Help is not available for dialogs or error messages that occur at design time. For ISAM errors, use the Search dialog in Help.

"Couldn't find installable ISAM"

An external file dynamic link library (DLL) file couldn't be found. This file is required for operations such as attaching or accessing external tables.

Possible causes:

1. An entry in the [Installable ISAM] section in VB.INI or <APPNAME>.INI is incorrect. For example, this error occurs if you're accessing a Paradox external table, and the Paradox entry of the .INI file points to a nonexistent directory. Exit Visual Basic, make necessary corrections in VB.INI or <APPNAME>.INI using Microsoft Windows Notepad or another text editor, restart Visual Basic, and try the operation again.

2. One of the entries in the [Installable ISAM] section in VB.INI points to a network drive, and that drive isn't connected. Check to make sure the network is available and the proper drive letter is established, and try the operation again.

Part 5: Notes for Microsoft Visual Basic "Programmer's Guide"

Page	Section/Note
188	The New Keyword

In the example at the top of the page, the local form variable F is declared with the New keyword using the Dim statement. To make the form variable and the loaded form instance persist, use a Static or Global variable instead.

Page	Section/Note
194	Determining the Type of an Object Variable

You can use the If...TypeOf statement to determine the control type of a custom control:

```
If TypeOf object Is objecttype
```

The identifier you use for 'object' is the class name of the custom control. See the section "Specific Control Object Types" (p. 186) for more information.

Page	Section/Note
461	The Options Property

The constant values shown are in hexidecimal and should be preceded with the &H notation. For example, DB_SQLPASSTHROUGH = &H40, not decimal 40. See online Help (Options Property) or the file DATACONS.TXT for the correct values.

Page	Section/Note
462	The RecordSource Property

The following text and example should be changed:

For example, the following SQL query returns all of the columns in the bibliography for authors who live in New York:

```
Data1.DatabaseName = "BIBLIO.MDB"
Data1.RecordSource = "Select * from Titles where state = 'NY'"
Data1.Refresh
```

The above should read:

For example, the following SQL query returns all of the columns in the bibliography for publishers based in New York:

```
Data1.DatabaseName = "BIBLIO.MDB"
Data1.RecordSource = "Select * from Publishers where state = 'NY'"
Data1.Refresh
```

465 Adding a New Record

In the second paragraph in this section, the last sentence should read, "Notice that using the buttons on the data control or one of the Move methods to move to another record will automatically save your added record."

530 Determining How an Object Is Displayed

In the first paragraph, the second sentence should read, "the Icon check box," not "th eIcon check box."

550 Creating Invisible Objects

In the sample code, the following line has incorrect syntax:

```
MyWord = ObjVar.SuggestWord MyWord
```

The code should look like this:

```
MyWord = ObjVar.SuggestWord (MyWord)
```

552 Limitations in Visual Basic

Under the discussion "Arrays and User-Defined Types," the third bulleted item should read: "You cannot...Assign the return value of a property or method to an array variable or a variable of a user-defined type."

554 Closing an Object

In the paragraph after the sample code, second sentence: It is not true that invoking a Close method on an object sets variables that refer to the object to Nothing.

582 Determining the Files You Need to Distribute

The following additional files are required for distributing your Visual Basic applications:

DLL Name Required by (Professional Edition Only)

PDIRJET.DLL	Crystal Reports for Visual Basic
PDBJET.DLL	Crystal Reports for Visual Basic
MSAJT110.DLL	Crystal Reports for Visual Basic
MSAES110.DLL	Crystal Reports for Visual Basic
PDSODBC.DLL	ODBC and Crystal Reports for Visual Basic

Part 6: Notes for Microsoft Visual Basic "Language Reference"

Page	Section/Note

21-22 Action Property (OLE)

In the Settings table, in Setting 5, the reference to None in the second sentence of the third paragraph should read as follows: "If the Paste was not successful, the OleType property will be set to 3 (None)." In Setting 12, the constant should be OLE_READ_FROM_FILE, not ReadFromFile. In Setting 14, the constant should be OLE_INSERT_OBJ_DLG.

41 AutoActivate Property

In the Note, replace the words "the double-click event" with "a DblClick event."

53 BorderStyle Property

The OLE control cannot have a setting of 2. Remove the setting and description for Setting 2 in the OLE control table.

65 Check Box Control

Add DataField and DataSource to the Properties list.

82 Color Property

The "Applies To" line should read "Common dialog (Color dialog)."

89 Common Dialog Control

Add "FilterIndex" and "MaxFileSize" to the Properties (File dialogs) list.

93 Connect Property

In the Note, change "SourceTable" to "SourceTableName."

97 Copies Property

The "Applies To" line should read "Common dialog (Print dialog)."

100 CreateDatabase Function

In the code example, replace "False" with "DBVERSION10." Also, in the table above the code example, replace "DB_COMPACT_ENCRYPT" with

"DB_ENCRYPT."

111 Data Control

Add UpdateControls and UpdateRecord to the Methods list.

112 Database Object

In the Properties list, the QueryTimeout Property should be identified as being available only in the Professional Edition.

117 DataText Property

In the code example, change the two instances of "MSDRAW" to "MSGRAPH."

134 DefaultExt Property

The "Applies To" line should read "Common dialog (File dialogs)."

185 Field Object

The Properties list should refer to SourceField and SourceTable, not SourceFieldName and SourceTableName.

195 FileTitle Property

The "Applies To" line should read "Common dialog (File dialogs)." Add the following to the Remarks section:

Note: If the OFN_NOVALIDATE flag is set, the FileTitle property will not return a value.

198 Filter Property (Common Dialog)

At the beginning of the topic, add "Applies To...Common dialog (File dialogs)." In the Remarks section, after the third paragraph, add this text:

Here is an example of a Filter in which the user can choose text files or picture files that include bitmaps and icons:

Text(*.txt)|*.txt|Pictures(*.bmp;*.ico)|*.bmp;*.ico

199 FilterIndex Property

The "Applies To" line should read "Common dialog (File dialogs)."

229 Frame Control

Add the Name Property to the Properties list.

231 FromPage, ToPage Properties

The "Applies To" line should read "Common dialog (Print dialog)."

240 GetAttr Function

The final Sub...End Sub block in code should read as follows:

```
Sub File1_Click ()
Const ATTR_READONLY = 1, ATTR_HIDDEN = 2       ' Declare constants.
Const ATTR_SYSTEM = 4, ATTR_ARCHIVE = 32
Dim Attr, FName, Msg   ' Declare variables.
If Right(Dir1.Path, 1) = "\" Then      ' See if root file.
   FName = Dir1.Path & File1.FileName  ' Get file path.
Else
   FName = Dir1.Path & "\" & File1.FileName    ' Get file path.
End If
Attr = GetAttr(FName)  ' Get attributes.
If Attr > 7 Then Attr = Attr Xor ATTR_ARCHIVE  ' Disregard Archive.
Select Case Attr       ' Look up attributes.
   Case 0: Msg = "Normal"
   Case ATTR_READONLY: Msg = "Read-Only"
   Case ATTR_HIDDEN: Msg = "Hidden"
   Case ATTR_HIDDEN +  ATTR_READONLY: Msg = "Hidden and Read-Only"
   Case ATTR_SYSTEM: Msg = "System"
   Case ATTR_READONLY +  ATTR_SYSTEM: Msg = "Read-Only and System"
   Case ATTR_HIDDEN +  ATTR_SYSTEM: Msg = "Hidden and System"
   Case ATTR_READONLY + ATTR_HIDDEN +  ATTR_SYSTEM: Msg = "Read-Only,"
      - + Msg = " Hidden, and System"
End Select
MsgBox UCase(FName) & " is a " & Msg & " file."   ' Display message.
End Sub
```

256 hDC Property

The Usage line should read:

{[form.] [commondialog. | picturebox.] | Printer.}hDC

Also, the second paragraph of the Remarks should read, "With a common dialog control, this property returns a device context for the printer selected in the Print dialog box when the..." (the rest of the text remains the same).

258 Height, Width Properties

The See Also line should refer to the "Width # Statement," not the "Width Statement."

274 Image Control

Add DataField and DataSource to the Properties list.

280 InitDir Property

The "Applies To" line should read "Common dialog (File dialogs)."

297 KeyDown, KeyUp Events

The See Also should refer to the SendKeys Statement, not the SendKeys Method.

299 KeyPress Events

The See Also should refer to the SendKeys Statement, not the SendKeys Method.

303 Label Control

Add the DataField, DataSource, and Parent properties to the Properties list.

336-338 ListFields Method

In the second table, the fifth and sixth entries in the Field column should be SourceTable and SourceField, not SourceTableName and SourceFieldName. The code example and the headings of the table below it should also refer to SourceTable and SourceField.

345 ListTables Method

In Remarks, the first paragraph under the TableType field table should read: "When you use the ListTables method to create a Snapshot, you can evaluate the contents of the Attributes field in the Snapshot by referring to the TableDef property settings table in the Attributes property topic.

361 Max, Min Properties (Common Dialog)

At the beginning of the topic, add "Applies To...Common dialog (Font, Print dialogs)."

363 MaxFileSize Property

The "Applies To" line should read "Common dialog (File dialogs)."

390 Name Property

The "Applies To" line should include the Database object.

432 Partition Function

In the code in Example 3, the second five lines of code duplicate the first five lines and should be deleted.

439 Picture Box Control

Add DataField and DataSource to the Properties list.

444 PopupMenu Method

In the Syntax line, add a comma immediately before the y.

455 PrinterDefault Property

The "Applies To" line should read "Common dialog (Print dialog)."

536-537 SourceFieldName, SourceTableName Properties

All references to SourceFieldName and SourceTableName in this topic should refer to "SourceField" and "SourceTable" instead.

538 SourceTableName Property

There should be a full entry for the "SourceTableName" topic. See online Help for the text of this topic.

565 Text Box Control

The second piece of art is incorrect. It should show a text box on a form but instead shows a menu title and menu items on a form. Also, add DataField and DataSource to the Properties list.

595 Validate Event

In the third paragraph following the Constants table, change "edit buffer" to "copy buffer."

619 Trappable Errors

In Appendix B, the odd header is wrong. It should read "Trappable Errors," not "Trappable Error Messages."

634 Trappable Error Messages

In Table B.6 ("Data Access Trappable Error Messages"), Error #3137 should be deleted.

Part 7: Notes for Microsoft Visual Basic "Custom Control Reference"

Page	Section/Note

xxii Visual Basic Executable (.EXE) Files

The Visual Basic run-time file is listed incorrectly. The first bulleted item should read VBRUN300.DLL, not VBRUN200.DLL.

147 Graph Control

In Example 1, the following line contains two "=" characters:

```
Graph1.LabelText = "Data point" = Str$(i%)
```

The line should read:

```
Graph1.LabelText = "Data point" + Str$(i%)
```

148 Graph Control

In Example 2, the following line contains two "=" characters:

```
Graph1.LabelText = "Label" = Str$(i%)
```

The line should read:

```
       Graph1.LabelText = "Label" + Str$(i%)

176    Key Status Control

       The table for the Value property incorrectly states that False is
       the default value. The default value is determined by the state
       of the keyboard.

180    MAPI Session Control

       There should be no footnotes, since the MAPI controls are
       only available in Visual Basic.

186    MAPI Messages Control

       There should be no footnotes, since the MAPI controls are
       only available in Visual Basic.

============================================================
Part 8: Notes for Microsoft Visual Basic "Data Access Guide"
============================================================

Page   Section/Note
------------------------------------------------------------
23     Creating New Table Definitions

       Delete the following line of code from the example:

   On Error Resume Next

Additional reference words: 3.00
KBCategory:
KBSubcategory: RefsDoc
```

PACKING.LST for Standard Edition of VB 3.0 for Windows

Article ID: Q100494

The information in this article applies to:

- Standard Edition of Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

The following article contains the complete contents of the PACKING.LST file distributed with the Standard Edition of Microsoft Visual Basic version 3.0 for Windows.

MORE INFORMATION

PACKING.LST

Disk Contents for Microsoft (R) Visual Basic
for Windows, Standard Edition
Version 3.0

This file lists all the files on the distribution disks provided with this product.

==========

NOTE: Most of the files on the distribution disks are compressed (indicated by an underscore character "_" in the file extension) and must be decompressed before they are used.

The Setup program on Disk 1 decompresses files as it installs them.

Files with a PA_ extension decompress to multiple .ICO files. These files cannot be decompressed manually.

If you need to decompress files manually, you can do so by running setup from the file manager or program manager in the following way:

1. Choose the Run item from the File menu.
2. Type SETUP.EXE /Z [source file] [destination file]

The /Z switch activates an internal decompression system. SETUP.EXE must be run from within Windows.

***** DISK1 ****

msajt110.dl_	data access	windows\system\msajt110.dll
arrows.pa_	icons	vb\icons\arrows\arrows.pak
comm.pa_	icons	vb\icons\comm\comm.pak
computer.pa_	icons	vb\icons\computer\computer.pak
dragdrop.pa_	icons	vb\icons\dragdrop\dragdrop.pak

```
elements.pa_  icons              vb\icons\elements\elements.pak
flags.pa_     icons              vb\icons\flags\flags.pak
industry.pa_  icons              vb\icons\industry\industry.pak
mail.pa_      icons              vb\icons\mail\mail.pak
misc.pa_      icons              vb\icons\misc\misc.pak
office.pa_    icons              vb\icons\office\office.pak
traffic.pa_   icons              vb\icons\traffic\traffic.pak
writing.pa_   icons              vb\icons\writing\writing.pak
vboa300.dl_   ole                windows\system\vboa300.dll
dispcalc.ex_  ole                vb\dispcalc.exe
dispcalc.re_  ole                vb\dispcalc.reg
ole2.re_      ole                windows\system\ole2.reg
readme.tx_    always             vb\readme.txt
calc.fr_      samples            vb\samples\calc\calc.frm
calc.ic_      samples            vb\samples\calc\calc.ico
calc.ma_      samples            vb\samples\calc\calc.mak
calc1.fr_     samples            vb\samples\calc\calc.frx
calldll1.fr_  samples            vb\samples\calldlls\calldlls.frx
calldlls.fr_  samples            vb\samples\calldlls\calldlls.frm
calldlls.ma_  samples            vb\samples\calldlls\calldlls.mak
declares.ba_  samples            vb\samples\calldlls\declares.bas
frmmenus.fr_  samples            vb\samples\calldlls\frmmenus.frm
array.fr_     samples            vb\samples\controls\array.frm
button.fr_    samples            vb\samples\controls\button.frm
button1.fr_   samples            vb\samples\controls\button.frx
check.fr_     samples            vb\samples\controls\check.frm
controls.ma_  samples            vb\samples\controls\controls.mak
listbox.fr_   samples            vb\samples\controls\listbox.frm
main2.fr_     samples            vb\samples\controls\main.frm
multi.fr_     samples            vb\samples\controls\multi.frm
number.fr_    samples            vb\samples\controls\number.frm
scroll.fr_    samples            vb\samples\controls\scroll.frm
wordwrap.fr_  samples            vb\samples\controls\wordwrap.frm
biblio.fr_    samples            vb\samples\datactrl\biblio.frm
biblio1.fr_   samples            vb\samples\datactrl\biblio.frx
biblio.ma_    samples            vb\samples\datactrl\biblio.mak
datactl.ba_   samples            vb\samples\datactrl\datactl.bas
dde.ba_       samples            vb\samples\dde\dde.bas
dde.ma_       samples            vb\samples\dde\dde.mak
execute.fr_   samples            vb\samples\dde\execute.frm
main.fr_      samples            vb\samples\dde\main.frm
alarm.fr_     samples            vb\samples\envir\alarm.frm
alarm.ma_     samples            vb\samples\envir\alarm.mak
alarm1.fr_    samples            vb\samples\envir\alarm.frx
seek.fr_      samples            vb\samples\filectls\seek.frm
winseek.ma_   samples            vb\samples\filectls\winseek.mak
fileproc.ba_  samples            vb\samples\fileio\fileproc.bas
recedit.ba_   samples            vb\samples\fileio\recedit.bas
recedit.fr_   samples            vb\samples\fileio\recedit.frm
recedit.ma_   samples            vb\samples\fileio\recedit.mak
bfly1.bm_     samples            vb\samples\firstapp\bfly1.bmp
bfly2.bm_     samples            vb\samples\firstapp\bfly2.bmp
butterf.fr_   samples            vb\samples\firstapp\butterf.frm
butterf.ma_   samples            vb\samples\firstapp\butterf.mak
butterf1.fr_  samples            vb\samples\firstapp\butterf.frx
picview.fr_   samples            vb\samples\firstapp\picview.frm
picview.ma_   samples            vb\samples\firstapp\picview.mak
```

```
blanker.fr_   samples          vb\samples\graphics\blanker.frm
blanker.ma_   samples          vb\samples\graphics\blanker.mak
blanker1.fr_  samples          vb\samples\graphics\blanker.frx
loan.fr_      samples          vb\samples\grid\loan.frm
loan.ma_      samples          vb\samples\grid\loan.mak
loan1.fr_     samples          vb\samples\grid\loan.frx
filopen.ba_   samples          vb\samples\mdi\filopen.bas
find1.fr_     samples          vb\samples\mdi\find.frm
mdi.fr_       samples          vb\samples\mdi\mdi.frm
mdi1.fr_      samples          vb\samples\mdi\mdi.frx
mdinote.ba_   samples          vb\samples\mdi\mdinote.bas
mdinote.ma_   samples          vb\samples\mdi\mdinote.mak
notepad.fr_   samples          vb\samples\mdi\notepad.frm
about.fr_     samples          vb\samples\menus\about.frm
edit.fr_      samples          vb\samples\menus\edit.frm
edit1.fr_     samples          vb\samples\menus\edit.frx
textedit.ba_  samples          vb\samples\menus\textedit.bas
textedit.ma_  samples          vb\samples\menus\textedit.mak
click.fr_     samples          vb\samples\mouse\click.frm
drag.fr_      samples          vb\samples\mouse\drag.frm
drag1.fr_     samples          vb\samples\mouse\drag.frx
main1.fr_     samples          vb\samples\mouse\main.frm
mouse.ma_     samples          vb\samples\mouse\mouse.mak
scribble.fr_  samples          vb\samples\mouse\scribble.frm
frmmain.fr_   samples          vb\samples\objects\frmmain.frm
multinst.fr_  samples          vb\samples\objects\multinst.frm
multinst.ma_  samples          vb\samples\objects\multinst.mak
objects.ba_   samples          vb\samples\objects\objects.bas
objects.ma_   samples          vb\samples\objects\objects.mak
about1.fr_    samples          vb\samples\ole\about.frm
ole2chld.fr_  samples          vb\samples\ole\ole2chld.frm
ole2demo.ma_  samples          vb\samples\ole\ole2demo.mak
ole2mdi.fr_   samples          vb\samples\ole\ole2mdi.frm
ole2mod1.ba_  samples          vb\samples\ole\ole2mod1.bas
ole2mod2.ba_  samples          vb\samples\ole\ole2mod2.bas
oleauto.ba_   samples          vb\samples\ole\oleauto.bas
oleauto.fr_   samples          vb\samples\ole\oleauto.frm
oleauto.ma_   samples          vb\samples\ole\oleauto.mak
fontdial.fr_  samples          vb\samples\print\fontdial.frm
tccancel.fr_  samples          vb\samples\print\tccancel.frm
timecar1.fr_  samples          vb\samples\print\timecard.frx
timecard.fr_  samples          vb\samples\print\timecard.frm
timecard.ma_  samples          vb\samples\print\timecard.mak
_mssetup.ex_  always           msvb2set.tmp\_mssetup.exe
mscomstf.dl_  always           msvb2set.tmp\mscomstf.dll
mscustom.dl_  always           msvb2set.tmp\mscustom.dll
msdetstf.dl_  always           msvb2set.tmp\msdetstf.dll
msinsstf.dl_  always           msvb2set.tmp\msinsstf.dll
msshlstf.dl_  always           msvb2set.tmp\msshlstf.dll
msuilstf.dl_  always           msvb2set.tmp\msuilstf.dll
packing.ls_   always           vb\packing.lst
setup.ex_     always           msvb2set.tmp\setup.exe
setup.ls_     always           msvb2set.tmp\setup.lst
vbsetup.ex_   always           msvb2set.tmp\vbsetup.exe
vbsetup.in_   always           msvb2set.tmp\vbsetup.ini
message.fr_   samples          vb\setupkit\setup1\message.frm
path.fr_      samples          vb\setupkit\setup1\path.frm
```

```
path1.fr_      samples            vb\setupkit\setup1\path.frx
setup1.ba_     samples            vb\setupkit\setup1\setup1.bas
setup1.fr_     samples            vb\setupkit\setup1\setup1.frm
setup1.gl_     samples            vb\setupkit\setup1\setup1.glb
setup1.ma_     samples            vb\setupkit\setup1\setup1.mak
setup11.fr_    samples            vb\setupkit\setup1\setup1.frx
status.fr_     samples            vb\setupkit\setup1\status.frm
shell.dl_      always             windows\system\shell.dll
ver.dl_        always             windows\system\ver.dll
cbtlib4.dl_    tutorial           vb\vb.cbt\cbtlib4.dll
vb.ex_         visual basic       vb\vb.exe

***** DISK2 ****

biblio.md_     data access        vb\biblio.mdb
msaes110.dl_   data access        windows\system\msaes110.dll
vbdb300.dl_    data access        windows\system\vbdb300.dll
xbs110.dl_     dbase driver       windows\system\xbs110.dll
pdx110.dl_     paradox driver     windows\system\pdx110.dll
btrv110.dl_    btrieve driver     windows\system\btrv110.dll
external.tx_   data access        vb\external.txt
btrieve.tx_    data access        vb\btrieve.txt
perform.tx_    data access        vb\perform.txt
datamgr.ex_    data access        vb\datamgr.exe
datamgr.hl_    data access        vb\datamgr.hlp
msafinx.dl_    visual basic       windows\system\msafinx.dll
msole2.vb_     visual basic       windows\system\msole2.vbx
msolevbx.dl_   visual basic       windows\system\msolevbx.dll
ole2nls.dl_    ole                windows\system\ole2nls.dll
ole2.dl_       ole                windows\system\ole2.dll
aboutbo3.fr_   samples            vb\samples\iconworks\aboutbox.frx
aboutbox.fr_   samples            vb\samples\iconworks\aboutbox.frm
colorpa1.fr_   samples            vb\samples\iconworks\colorpal.frx
colorpal.fr_   samples            vb\samples\iconworks\colorpal.frm
iconedi1.fr_   samples            vb\samples\iconworks\iconedit.frx
iconedit.fr_   samples            vb\samples\iconworks\iconedit.frm
iconwrks.ba_   samples            vb\samples\iconworks\iconwrks.bas
iconwrks.gb_   samples            vb\samples\iconworks\iconwrks.gbl
iconwrks.hl_   samples            vb\samples\iconworks\iconwrks.hlp
iconwrks.ic_   samples            vb\samples\iconworks\iconwrks.ico
iconwrks.ma_   samples            vb\samples\iconworks\iconwrks.mak
screen.ic_     samples            vb\samples\iconworks\screen.ico
toolpal.bm_    samples            vb\samples\iconworks\toolpal.bmp
viewico1.fr_   samples            vb\samples\iconworks\viewicon.frx
viewicon.fr_   samples            vb\samples\iconworks\viewicon.frm
compress.ex_   samples            vb\setupkit\kitfiles\compress.exe
compress.tx_   samples            vb\setupkit\kitfiles\compress.txt
expand.ex_     samples            vb\setupkit\kitfiles\expand.exe
setupa.ex_     samples            vb\setupkit\kitfiles\setup.exe
setupa.ls_     samples            vb\setupkit\kitfiles\setup.lst
setupkit.dl_   samples            vb\setupkit\kitfiles\setupkit.dll
setupwiz.ex_   samples            vb\setupkit\kitfiles\setupwiz.exe
setupwiz.hl_   samples            vb\setupkit\kitfiles\setupwiz.hlp
setupwiz.in_   samples            vb\setupkit\kitfiles\setupwiz.ini
commdlg.dl_    visual basic       windows\system\commdlg.dll
ddeml.dl_      visual basic       windows\system\ddeml.dll
cbt.ex_        tutorial           vb\vb.cbt\cbt.exe
```

```
code.ma_       tutorial           vb\vb.cbt\code.mak
country.ma_    tutorial           vb\vb.cbt\country.mak
form1.fr_      tutorial           vb\vb.cbt\form1.frm
form2.fr_      tutorial           vb\vb.cbt\form2.frm
formchi.fr_    tutorial           vb\vb.cbt\formchi.frm
formchi1.fr_   tutorial           vb\vb.cbt\formchi.frx
mdinpad.fr_    tutorial           vb\vb.cbt\mdinpad.frm
mdinpad1.fr_   tutorial           vb\vb.cbt\mdinpad.frx
payment.fr_    tutorial           vb\vb.cbt\payment.frm
payment.ma_    tutorial           vb\vb.cbt\payment.mak
sweden.fr_     tutorial           vb\vb.cbt\sweden.frm
sweden1.fr_    tutorial           vb\vb.cbt\sweden.frx
vb.le_         tutorial           vb\vb.cbt\vb.les
vbrun300.dl_   visual basic       windows\system\vbrun300.dll
cmdialog.vb_   visual basic       windows\system\cmdialog.vbx
grid.vb_       visual basic       windows\system\grid.vbx

***** DISK3 ****

autoload.ma_   visual basic       vb\autoload.mak
bright.di_     visual basic       vb\bright.dib
constant.tx_   visual basic       vb\constant.txt
datacons.tx_   visual basic       vb\datacons.txt
compobj.dl_    ole                windows\system\compobj.dll
ole2conv.dl_   ole                windows\system\ole2conv.dll
ole2prox.dl_   ole                windows\system\ole2prox.dll
ole2disp.dl_   ole                windows\system\ole2disp.dll
storage.dl_    ole                windows\system\storage.dll
pastel.di_     visual basic       vb\pastel.dib
rainbow.di_    visual basic       vb\rainbow.dib
vb.hl_         visual basic       vb\vb.hlp

Additional reference words: 3.00
KBCategory:
KBSubcategory: RefsDoc
```

PACKING.LST for Professonal Edition of VB 2.0 for Windows

Article ID: Q100631

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic programming system for Windows, version 2.0

SUMMARY

The following article contains the complete contents of the PACKING.LST file distributed with Microsoft Professional Edition of Visual Basic version 2.0 for Windows.

MORE INFORMATION

PACKING.LST

Disk Contents for Microsoft (R) Visual Basic
for Windows, Professional Edition
Version 2.0
(C) Copyright Microsoft Corporation, 1992

This file lists all the files on the distribution disks provided with this product.

==========

NOTE: Most of the files on the distribution disks are compressed (indicated by an underscore character "_" in the file extension) and must be decompressed before they are used.

The Setup program on Disk 1 decompresses files as it installs them.

Files with a PA_ extension decompress to multiple .ICO or .BMP files.

If you need to decompress files manually, you can do so by running setup from the file manager or program manager in the following way:

1. Choose the Run item from the File menu.
2. Type SETUP.EXE /Z [source file] [destination file]

The /Z switch activates an internal decompression system. SETUP.EXE must be run from within Windows.

***** DISK1 ****

assorted.pa_	clip art	vb\bitmaps\assorted\assorted.pak
vbapi.hl_	control dev kit	vb\cdk\vbapi.hlp
hc31.ex_	help compiler	vb\hc\hc31.exe
dragdrop.pa_	icons	vb\icons\dragdrop\dragdrop.pak
elements.pa_	icons	vb\icons\elements\elements.pak
industry.pa_	icons	vb\icons\industry\industry.pak
traffic.pa_	icons	vb\icons\traffic\traffic.pak

```
mcimmp.dr_    visual basic        vb\intro2vb.exemcimmp.dr_
mmp.dl_       clip art            vb\metafile\business\yen.wmfmmp.dl_
oemsetup.in_  data access         vb\odbc\vbodbc.hlpoemsetup.inf
proreadm.tx_  always              vb\readme.txt
_mssetup.ex_  always              msvb2set.tmp\_mssetup.exe
mscomstf.dl_  data access         vb\odbc\mscomstf.dll
mscustom.dl_  always              msvb2set.tmp\mscustom.dll
msdetstf.dl_  data access         vb\odbc\msdetstf.dll
msinsstf.dl_  data access         vb\odbc\msinsstf.dll
msshlstf.dl_  data access         vb\odbc\msshlstf.dll
msuilstf.dl_  data access         vb\odbc\msuilstf.dll
packing.ls_   visual basic        vb\packing.lst
setup.ex_     always              msvb2set.tmp\setup.exe
setup.ls_     always              msvb2set.tmp\setup.lst
vbsetup.ex_   always              msvb2set.tmp\vbsetup.exe
vbsetup.in_   always              msvb2set.tmp\vbsetup.ini
alarm.fr_     samples             vb\samples\envir\alarm.frm
alarm.ma_     samples             vb\samples\envir\alarm.mak
alarm1.fr_    samples             vb\samples\envir\alarm.frx
ver.dl_       always              windows\system\ver.dll
vbknowlg.hl_  knowledgebase       vb\vbknowlg.hlp
cbt.ex_       tutorial            vb\vb.cbt\cbt.exe
vdg.ex_       visual design guide vb\vb.cbt\vdg.exe
vdg.le_       visual design guide vb\vb.cbt\vdg.les
vbrun200.dl_  visual basic        windows\system\vbrun200.dll

***** DISK2 ****

win31wh.h9_   windows api         vb\winapi\win31wh.hlp
                 Split file, part 1 of 3

***** DISK3 ****

win31wh.10_   windows api         vb\winapi\win31wh.hlp
                 Split file, part 2 of 3

***** DISK4 ****

circlock.bm_  clip art            vb\bitmaps\gauge\circlock.bmp
dome.bm_      clip art            vb\bitmaps\gauge\dome.bmp
horz.bm_      clip art            vb\bitmaps\gauge\horz.bmp
horz1.bm_     clip art            vb\bitmaps\gauge\horz1.bmp
semicirc.bm_  clip art            vb\bitmaps\gauge\semicirc.bmp
therbot.bm_   clip art            vb\bitmaps\gauge\therbot.bmp
therlft.bm_   clip art            vb\bitmaps\gauge\therlft.bmp
thermid.bm_   clip art            vb\bitmaps\gauge\thermid.bmp
thermidh.bm_  clip art            vb\bitmaps\gauge\thermidh.bmp
therrt.bm_    clip art            vb\bitmaps\gauge\therrt.bmp
thertop.bm_   clip art            vb\bitmaps\gauge\thertop.bmp
vert.bm_      clip art            vb\bitmaps\gauge\vert.bmp
volbot.bm_    clip art            vb\bitmaps\gauge\volbot.bmp
voltop.bm_    clip art            vb\bitmaps\gauge\voltop.bmp
toolbar.pa_   clip art            vb\bitmaps\toolbar2\toolbar.pak
cdk.tx_       control dev kit     vb\cdk\cdk.txt
circ1.c_      control dev kit     vb\cdk\circ1\circ1.c
circ1.de_     control dev kit     vb\cdk\circ1\circ1.def
circ1.h_      control dev kit     vb\cdk\circ1\circ1.h
```

```
circ1.rc_      control dev kit     vb\cdk\circ1\circ1.rc
circ1.vb_      control dev kit     vb\cdk\circ1\circ1.vbx
circ1cd.bm_    control dev kit     vb\cdk\circ1\circ1cd.bmp
circ1cu.bm_    control dev kit     vb\cdk\circ1\circ1cu.bmp
circ1eu.bm_    control dev kit     vb\cdk\circ1\circ1eu.bmp
circ1mu.bm_    control dev kit     vb\cdk\circ1\circ1mu.bmp
makefil1._     control dev kit     vb\cdk\circ1\makefile
circ2.c_       control dev kit     vb\cdk\circ2\circ2.c
circ2.de_      control dev kit     vb\cdk\circ2\circ2.def
circ2.h_       control dev kit     vb\cdk\circ2\circ2.h
circ2.rc_      control dev kit     vb\cdk\circ2\circ2.rc
circ2.vb_      control dev kit     vb\cdk\circ2\circ2.vbx
circ2cd.bm_    control dev kit     vb\cdk\circ2\circ2cd.bmp
circ2cu.bm_    control dev kit     vb\cdk\circ2\circ2cu.bmp
circ2eu.bm_    control dev kit     vb\cdk\circ2\circ2eu.bmp
circ2mu.bm_    control dev kit     vb\cdk\circ2\circ2mu.bmp
makefil2._     control dev kit     vb\cdk\circ2\makefile
cntr.c_        control dev kit     vb\cdk\cntr\cntr.c
cntr.de_       control dev kit     vb\cdk\cntr\cntr.def
cntr.h_        control dev kit     vb\cdk\cntr\cntr.h
cntr.rc_       control dev kit     vb\cdk\cntr\cntr.rc
cntr.vb_       control dev kit     vb\cdk\cntr\cntr.vbx
cntrcd.bm_     control dev kit     vb\cdk\cntr\cntrcd.bmp
cntrcu.bm_     control dev kit     vb\cdk\cntr\cntrcu.bmp
cntreu.bm_     control dev kit     vb\cdk\cntr\cntreu.bmp
cntrmu.bm_     control dev kit     vb\cdk\cntr\cntrmu.bmp
makefil4._     control dev kit     vb\cdk\cntr\makefile
libentry.as_   control dev kit     vb\cdk\libentry.asm
libentry.ob_   control dev kit     vb\cdk\libentry.obj
makefil5._     control dev kit     vb\cdk\pal\makefile
pal.c_         control dev kit     vb\cdk\pal\pal.c
pal.de_        control dev kit     vb\cdk\pal\pal.def
pal.h_         control dev kit     vb\cdk\pal\pal.h
pal.rc_        control dev kit     vb\cdk\pal\pal.rc
pal.vb_        control dev kit     vb\cdk\pal\pal.vbx
palcd.bm_      control dev kit     vb\cdk\pal\palcd.bmp
palcu.bm_      control dev kit     vb\cdk\pal\palcu.bmp
paleu.bm_      control dev kit     vb\cdk\pal\paleu.bmp
palmu.bm_      control dev kit     vb\cdk\pal\palmu.bmp
makefil7._     control dev kit     vb\cdk\push\makefile
push.c_        control dev kit     vb\cdk\push\push.c
push.de_       control dev kit     vb\cdk\push\push.def
push.h_        control dev kit     vb\cdk\push\push.h
push.rc_       control dev kit     vb\cdk\push\push.rc
push.vb_       control dev kit     vb\cdk\push\push.vbx
pushcd.bm_     control dev kit     vb\cdk\push\pushcd.bmp
pushcu.bm_     control dev kit     vb\cdk\push\pushcu.bmp
pusheu.bm_     control dev kit     vb\cdk\push\pusheu.bmp
pushmu.bm_     control dev kit     vb\cdk\push\pushmu.bmp
pushvb1.h_     control dev kit     vb\cdk\push\pushvb1.h
vbapi.h_       control dev kit     vb\cdk\vbapi.h
vbapi.li_      control dev kit     vb\cdk\vbapi.lib
vbx.rc_        control dev kit     vb\cdk\vbx.rcv
makefil9._     control dev kit     vb\cdk\xlist\makefile
xlist.c_       control dev kit     vb\cdk\xlist\xlist.c
xlist.de_      control dev kit     vb\cdk\xlist\xlist.def
xlist.h_       control dev kit     vb\cdk\xlist\xlist.h
```

xlist.rc_	control dev kit	vb\cdk\xlist\xlist.rc
xlist.vb_	control dev kit	vb\cdk\xlist\xlist.vbx
xlistcd.bm_	control dev kit	vb\cdk\xlist\xlistcd.bmp
xlistcu.bm_	control dev kit	vb\cdk\xlist\xlistcu.bmp
xlisteu.bm_	control dev kit	vb\cdk\xlist\xlisteu.bmp
xlistmu.bm_	control dev kit	vb\cdk\xlist\xlistmu.bmp
xlistvb1.h_	control dev kit	vb\cdk\xlist\xlistvb1.h
arrows.pa_	icons	vb\icons\arrows\arrows.pak
comm.pa_	icons	vb\icons\comm\comm.pak
computer.pa_	icons	vb\icons\computer\computer.pak
flags.pa_	icons	vb\icons\flags\flags.pak
mail.pa_	icons	vb\icons\mail\mail.pak
misc.pa_	icons	vb\icons\misc\misc.pak
office.pa_	icons	vb\icons\office\office.pak
writing.pa_	icons	vb\icons\writing\writing.pak
vbodbca.dl_	data access	vb\bcdll\vbodbca.dll
calc.fr_	samples	vb\samples\calc\calc.frm
calc.ic_	samples	vb\samples\calc\calc.ico
calc.ma_	samples	vb\samples\calc\calc.mak
calc1.fr_	samples	vb\samples\calc\calc.frx
calldll1.fr_	samples	vb\samples\calldlls\calldlls.frx
calldlls.fr_	samples	vb\samples\calldlls\calldlls.frm
calldlls.ma_	samples	vb\samples\calldlls\calldlls.mak
declares.ba_	samples	vb\samples\calldlls\declares.bas
frmmenus.fr_	samples	vb\samples\calldlls\frmmenus.frm
array.fr_	samples	vb\samples\controls\array.frm
button.fr_	samples	vb\samples\controls\button.frm
button1.fr_	samples	vb\samples\controls\button.frx
check.fr_	samples	vb\samples\controls\check.frm
controls.ma_	samples	vb\samples\controls\controls.mak
listbox.fr_	samples	vb\samples\controls\listbox.frm
main2.fr_	samples	vb\samples\controls\main.frm
multi.fr_	samples	vb\samples\controls\multi.frm
number.fr_	samples	vb\samples\controls\number.frm
scroll.fr_	samples	vb\samples\controls\scroll.frm
wordwrap.fr_	samples	vb\samples\controls\wordwrap.frm
dde.ba_	samples	vb\samples\dde\dde.bas
dde.ma_	samples	vb\samples\dde\dde.mak
execute.fr_	samples	vb\samples\dde\execute.frm
main.fr_	samples	vb\samples\dde\main.frm
seek.fr_	samples	vb\samples\filectls\seek.frm
winseek.ma_	samples	vb\samples\filectls\winseek.mak
fileproc.ba_	samples	vb\samples\fileio\fileproc.bas
recedit.ba_	samples	vb\samples\fileio\recedit.bas
recedit.fr_	samples	vb\samples\fileio\recedit.frm
recedit.ma_	samples	vb\samples\fileio\recedit.mak
bfly1.bm_	samples	vb\samples\firstapp\bfly1.bmp
bfly2.bm_	samples	vb\samples\firstapp\bfly2.bmp
butterf.fr_	samples	vb\samples\firstapp\butterf.frm
butterf.ma_	samples	vb\samples\firstapp\butterf.mak
butterf1.fr_	samples	vb\samples\firstapp\butterf.frx
picview.fr_	samples	vb\samples\firstapp\picview.frm
picview.ma_	samples	vb\samples\firstapp\picview.mak
flip.fr_	samples	vb\samples\flip\flip.frm
flip.ma_	samples	vb\samples\flip\flip.mak
form.ic_	samples	vb\samples\flip\form.ico
blanker.fr_	samples	vb\samples\graphics\blanker.frm

```
blanker.ma_   samples          vb\samples\graphics\blanker.mak
blanker1.fr_  samples          vb\samples\graphics\blanker.frx
loan.fr_      samples          vb\samples\grid\loan.frm
loan.ma_      samples          vb\samples\grid\loan.mak
loan1.fr_     samples          vb\samples\grid\loan.frx
aboutdl1.fr_  samples          vb\samples\jigsaw\aboutdlg.frx
aboutdlg.fr_  samples          vb\samples\jigsaw\aboutdlg.frm
global2.ba_   samples          vb\samples\jigsaw\global.bas
jigsaw.fr_    samples          vb\samples\jigsaw\jigsaw.frm
jigsaw.gb_    samples          vb\samples\jigsaw\jigsaw.gbl
jigsaw.ma_    samples          vb\samples\jigsaw\jigsaw.mak
jigsaw1.fr_   samples          vb\samples\jigsaw\jigsaw.frx
jsprocs.ba_   samples          vb\samples\jigsaw\jsprocs.bas
misc20.ic_    samples          vb\samples\jigsaw\misc20.ico
openfile.fr_  samples          vb\samples\jigsaw\openfile.frm
maillst.fr_   samples          vb\samples\mapi\maillst.frm
mailoptf.fr_  samples          vb\samples\mapi\mailoptf.frm
mailsup.ba_   samples          vb\samples\mapi\mailsup.bas
msgview.fr_   samples          vb\samples\mapi\msgview.frm
newmsg.fr_    samples          vb\samples\mapi\newmsg.frm
vbmail.fr_    samples          vb\samples\mapi\vbmail.frm
vbmail.ma_    samples          vb\samples\mapi\vbmail.mak
fileope2.fr_  samples          vb\samples\mdi\fileopen.frm
fileope5.fr_  samples          vb\samples\mdi\fileopen.frx
filopen.ba_   samples          vb\samples\mdi\filopen.bas
find1.fr_     samples          vb\samples\mdi\find.frm
mdi.fr_       samples          vb\samples\mdi\mdi.frm
mdi1.fr_      samples          vb\samples\mdi\mdi.frx
mdinote.ba_   samples          vb\samples\mdi\mdinote.bas
mdinote.ma_   samples          vb\samples\mdi\mdinote.mak
notepad.fr_   samples          vb\samples\mdi\notepad.frm
about.fr_     samples          vb\samples\menus\about.frm
edit.fr_      samples          vb\samples\menus\edit.frm
opensave.fr_  samples          vb\samples\menus\opensave.frm
textedit.ba_  samples          vb\samples\menus\textedit.bas
textedit.ma_  samples          vb\samples\menus\textedit.mak
click.fr_     samples          vb\samples\mouse\click.frm
drag.fr_      samples          vb\samples\mouse\drag.frm
drag1.fr_     samples          vb\samples\mouse\drag.frx
main1.fr_     samples          vb\samples\mouse\main.frm
mouse.ma_     samples          vb\samples\mouse\mouse.mak
scribble.fr_  samples          vb\samples\mouse\scribble.frm
frmmain.fr_   samples          vb\samples\objects\frmmain.frm
multinst.fr_  samples          vb\samples\objects\multinst.frm
multinst.ma_  samples          vb\samples\objects\multinst.mak
objects.ba_   samples          vb\samples\objects\objects.bas
objects.ma_   samples          vb\samples\objects\objects.mak
about1.fr_    samples          vb\samples\ole\about.frm
file.fr_      samples          vb\samples\ole\file.frm
file1.fr_     samples          vb\samples\ole\file.frx
insert.fr_    samples          vb\samples\ole\insert.frm
links.fr_     samples          vb\samples\ole\links.frm
oledemo.ba_   samples          vb\samples\ole\oledemo.bas
oledemo.ma_   samples          vb\samples\ole\oledemo.mak
olemain.fr_   samples          vb\samples\ole\olemain.frm
regview.fr_   samples          vb\samples\ole\regview.frm
regview.ic_   samples          vb\samples\ole\regview.ico
```

```
regview.ma_   samples            vb\samples\ole\regview.mak
regview1.fr_  samples            vb\samples\ole\regview.frx
infofor1.fr_  samples            vb\samples\picclip\infoform.frx
infoform.fr_  samples            vb\samples\picclip\infoform.frm
redtop.fr_    samples            vb\samples\picclip\redtop.frm
redtop.ma_    samples            vb\samples\picclip\redtop.mak
redtop1.fr_   samples            vb\samples\picclip\redtop.frx
fontdial.fr_  samples            vb\samples\print\fontdial.frm
tccancel.fr_  samples            vb\samples\print\tccancel.frm
timecar1.fr_  samples            vb\samples\print\timecard.frx
timecard.fr_  samples            vb\samples\print\timecard.frm
timecard.ma_  samples            vb\samples\print\timecard.mak
cansend.fr_   samples            vb\samples\vbterm\cansend.frm
termset.fr_   samples            vb\samples\vbterm\termset.frm
termset1.fr_  samples            vb\samples\vbterm\termset.frx
vbterm.fr_    samples            vb\samples\vbterm\vbterm.frm
vbterm.gl_    samples            vb\samples\vbterm\vbterm.glo
vbterm.ma_    samples            vb\samples\vbterm\vbterm.mak
vbterm1.fr_   samples            vb\samples\vbterm\vbterm.frx
message.fr_   samples            vb\setupkit\setup1\message.frm
path.fr_      samples            vb\setupkit\setup1\path.frm
path1.fr_     samples            vb\setupkit\setup1\path.frx
setup1.ba_    samples            vb\setupkit\setup1\setup1.bas
setup1.fr_    samples            vb\setupkit\setup1\setup1.frm
setup1.gl_    samples            vb\setupkit\setup1\setup1.glb
setup1.ma_    samples            vb\setupkit\setup1\setup1.mak
setup11.fr_   samples            vb\setupkit\setup1\setup1.frx
status.fr_    samples            vb\setupkit\setup1\status.frm
shell.dl_     always             windows\system\shell.dll
cbtlib4.dl_   tutorial           vb\vb.cbt\cbtlib4.dll
vb.li_        always             windows\system\vb.lic
win31wh.11_   windows api        vb\winapi\win31wh.hlp
                 Split file, part 3 of 3
winhelp.ex_   always             windows\winhelp.exe

***** DISK5 ****

circ3.c_      control dev kit    vb\cdk\circ3\circ3.c
circ3.de_     control dev kit    vb\cdk\circ3\circ3.def
circ3.h_      control dev kit    vb\cdk\circ3\circ3.h
circ3.hl_     control dev kit    vb\cdk\circ3\circ3.hlp
circ3.hp_     control dev kit    vb\cdk\circ3\circ3.hpj
circ3.rc_     control dev kit    vb\cdk\circ3\circ3.rc
circ3.rt_     control dev kit    vb\cdk\circ3\circ3.rtf
circ3.vb_     control dev kit    vb\cdk\circ3\circ3.vbx
circ3cd.bm_   control dev kit    vb\cdk\circ3\circ3cd.bmp
circ3cu.bm_   control dev kit    vb\cdk\circ3\circ3cu.bmp
circ3eu.bm_   control dev kit    vb\cdk\circ3\circ3eu.bmp
circ3mu.bm_   control dev kit    vb\cdk\circ3\circ3mu.bmp
circ3vb1.h_   control dev kit    vb\cdk\circ3\circ3vb1.h
makefil3._    control dev kit    vb\cdk\circ3\makefile
makefil6._    control dev kit    vb\cdk\pix\makefile
pictblt.c_    control dev kit    vb\cdk\pix\pictblt.c
pictblt.h_    control dev kit    vb\cdk\pix\pictblt.h
pictblt.ob_   control dev kit    vb\cdk\pix\pictblt.obj
pix.c_        control dev kit    vb\cdk\pix\pix.c
pix.de_       control dev kit    vb\cdk\pix\pix.def
```

pix.h_	control dev kit	vb\cdk\pix\pix.h
pix.rc_	control dev kit	vb\cdk\pix\pix.rc
pix.vb_	control dev kit	vb\cdk\pix\pix.vbx
pixcd.bm_	control dev kit	vb\cdk\pix\pixcd.bmp
pixcu.bm_	control dev kit	vb\cdk\pix\pixcu.bmp
pixeu.bm_	control dev kit	vb\cdk\pix\pixeu.bmp
pixmu.bm_	control dev kit	vb\cdk\pix\pixmu.bmp
pixvb1.h_	control dev kit	vb\cdk\pix\pixvb1.h
2darrow1.wm_	clip art	vb\metafile\arrows\2darrow1.wmf
2darrow2.wm_	clip art	vb\metafile\arrows\2darrow2.wmf
2darrow3.wm_	clip art	vb\metafile\arrows\2darrow3.wmf
2darrow4.wm_	clip art	vb\metafile\arrows\2darrow4.wmf
3darrow1.wm_	clip art	vb\metafile\arrows\3darrow1.wmf
3darrow2.wm_	clip art	vb\metafile\arrows\3darrow2.wmf
3darrow3.wm_	clip art	vb\metafile\arrows\3darrow3.wmf
3darrow4.wm_	clip art	vb\metafile\arrows\3darrow4.wmf
3darrow5.wm_	clip art	vb\metafile\arrows\3darrow5.wmf
3darrow6.wm_	clip art	vb\metafile\arrows\3darrow6.wmf
3darrow7.wm_	clip art	vb\metafile\arrows\3darrow7.wmf
3dxarrow.wm_	clip art	vb\metafile\arrows\3dxarrow.wmf
3dxcirar.wm_	clip art	vb\metafile\arrows\3dxcirar.wmf
halfarrw.wm_	clip art	vb\metafile\arrows\halfarrw.wmf
hortarrw.wm_	clip art	vb\metafile\arrows\hortarrw.wmf
hozcirar.wm_	clip art	vb\metafile\arrows\hozcirar.wmf
layerarw.wm_	clip art	vb\metafile\arrows\layerarw.wmf
lrgearrw.wm_	clip art	vb\metafile\arrows\lrgearrw.wmf
medarrw1.wm_	clip art	vb\metafile\arrows\medarrw1.wmf
medarrw2.wm_	clip art	vb\metafile\arrows\medarrw2.wmf
multarw1.wm_	clip art	vb\metafile\arrows\multarw1.wmf
multarw2.wm_	clip art	vb\metafile\arrows\multarw2.wmf
multarw3.wm_	clip art	vb\metafile\arrows\multarw3.wmf
multarw4.wm_	clip art	vb\metafile\arrows\multarw4.wmf
smallarw.wm_	clip art	vb\metafile\arrows\smallarw.wmf
tinyarrw.wm_	clip art	vb\metafile\arrows\tinyarrw.wmf
vertarrw.wm_	clip art	vb\metafile\arrows\vertarrw.wmf
vrtcirar.wm_	clip art	vb\metafile\arrows\vrtcirar.wmf
vrtcurar.wm_	clip art	vb\metafile\arrows\vrtcurar.wmf
xarrow.wm_	clip art	vb\metafile\arrows\xarrow.wmf
dbnmp3.od_	data access	vb\odbc\dbnmp3.dll
instcat.sq_	data access	vb\odbc\instcat.sql
odbc.in_	data access	vb\odbc\odbc.inf
odbc.od_	data access	vb\odbc\odbc.dll
odbcadm.ex_	data access	vb\odbc\odbcadm.exe
odbcadm.hl_	data access	vb\odbc\odbcadm.hlp
odbcinst.od_	data access	vb\odbc\odbcinst.dll
odbsetup.ex_	data access	vb\odbc\setup.exe
setupi.od_	data access	vb\odbc\setupi.dll
sqlsetup.od_	data access	vb\odbc\sqlsetup.dll
sqlsrvr.od_	data access	vb\odbc\sqlsrvr.dll
startodb.ex_	data access	vb\odbc\startodb.exe
vbodbc.hl_	data access	vb\odbc\vbodbc.hlp
aboutbo3.fr_	samples	vb\samples\iconworks\aboutbox.frx
aboutbox.fr_	samples	vb\samples\iconworks\aboutbox.frm
colorpa1.fr_	samples	vb\samples\iconworks\colorpal.frx
colorpal.fr_	samples	vb\samples\iconworks\colorpal.frm
iconedi1.fr_	samples	vb\samples\iconworks\iconedit.frx
iconedit.fr_	samples	vb\samples\iconworks\iconedit.frm

```
iconwrks.ba_  samples          vb\samples\iconworks\iconwrks.bas
iconwrks.gb_  samples          vb\samples\iconworks\iconwrks.gbl
iconwrks.hl_  samples          vb\samples\iconworks\iconwrks.hlp
iconwrks.ic_  samples          vb\samples\iconworks\iconwrks.ico
iconwrks.ma_  samples          vb\samples\iconworks\iconwrks.mak
savedlg.fr_   samples          vb\samples\iconworks\savedlg.frm
savedlg1.fr_  samples          vb\samples\iconworks\savedlg.frx
screen.ic_    samples          vb\samples\iconworks\screen.ico
toolpal.bm_   samples          vb\samples\iconworks\toolpal.bmp
viewico1.fr_  samples          vb\samples\iconworks\viewicon.frx
viewicon.fr_  samples          vb\samples\iconworks\viewicon.frm
aboutbo1.fr_  samples          vb\samples\mci\aboutbox.frm
animate.fr_   samples          vb\samples\mci\animate.frm
cd.fr_        samples          vb\samples\mci\cd.frm
global3.ba_   samples          vb\samples\mci\global.bas
mcitest.ba_   samples          vb\samples\mci\mcitest.bas
mcitest.fr_   samples          vb\samples\mci\mcitest.frm
mcitest.ma_   samples          vb\samples\mci\mcitest.mak
mcitest.mi_   samples          vb\samples\mci\mcitest.mid
mcitest.mm_   samples          vb\samples\mci\mcitest.mmm
mcitest.wa_   samples          vb\samples\mci\mcitest.wav
opendlg.fr_   samples          vb\samples\mci\opendlg.frm
wave.fr_      samples          vb\samples\mci\wave.frm
delay.fr_     samples          vb\samples\pen\delay.frm
delay1.fr_    samples          vb\samples\pen\delay.frx
editsubf.fr_  samples          vb\samples\pen\editsubf.frm
gestfrm.fr_   samples          vb\samples\pen\gestfrm.frm
grafpapr.bm_  samples          vb\samples\pen\grafpapr.bmp
inkfrm.fr_    samples          vb\samples\pen\inkfrm.frm
iobfrm.fr_    samples          vb\samples\pen\iobfrm.frm
iobfrm1.fr_   samples          vb\samples\pen\iobfrm.frx
keybrd.fr_    samples          vb\samples\pen\keybrd.frm
keybrd1.fr_   samples          vb\samples\pen\keybrd.frx
penapi.tx_    samples          vb\samples\pen\penapi.txt
penmain.fr_   samples          vb\samples\pen\penmain.frm
penmain1.fr_  samples          vb\samples\pen\penmain.frx
pensmpl.ma_   samples          vb\samples\pen\pensmpl.mak
rcfrm.fr_     samples          vb\samples\pen\rcfrm.frm
rulepapr.bm_  samples          vb\samples\pen\rulepapr.bmp
skbface.bm_   samples          vb\samples\pen\skbface.bmp
transfrm.fr_  samples          vb\samples\pen\transfrm.frm
about2.fr_    samples          vb\samples\savings\about.frm
about3.fr_    samples          vb\samples\savings\about.frx
college.fr_   samples          vb\samples\savings\college.frm
college1.fr_  samples          vb\samples\savings\college.frx
fileope3.fr_  samples          vb\samples\savings\fileopen.frm
fileope4.fr_  samples          vb\samples\savings\fileopen.frx
general.fr_   samples          vb\samples\savings\general.frm
general1.fr_  samples          vb\samples\savings\general.frx
graph.fr_     samples          vb\samples\savings\graph.frm
graph1.fr_    samples          vb\samples\savings\graph.frx
gridfor1.fr_  samples          vb\samples\savings\gridform.frx
gridform.fr_  samples          vb\samples\savings\gridform.frm
mdiform.fr_   samples          vb\samples\savings\mdiform.frm
mdiform1.fr_  samples          vb\samples\savings\mdiform.frx
sample.sa_    samples          vb\samples\savings\sample.sav
savings.ba_   samples          vb\samples\savings\savings.bas
```

```
savings.hl_   samples           vb\samples\savings\savings.hlp
savings.ma_   samples           vb\samples\savings\savings.mak
selform.fr_   samples           vb\samples\savings\selform.frm
aboutbo2.fr_  samples           vb\samples\vbodbc\aboutbox.frm
aboutbo5.fr_  samples           vb\samples\vbodbc\aboutbox.frx
addfield.fr_  samples           vb\samples\vbodbc\addfield.frm
cpystru.fr_   samples           vb\samples\vbodbc\cpystru.frm
databox.fr_   samples           vb\samples\vbodbc\databox.frm
dynagri1.fr_  samples           vb\samples\vbodbc\dynagrid.frx
dynagrid.fr_  samples           vb\samples\vbodbc\dynagrid.frm
dynaset.fr_   samples           vb\samples\vbodbc\dynaset.frm
dynaset1.fr_  samples           vb\samples\vbodbc\dynaset.frx
find.fr_      samples           vb\samples\vbodbc\find.frm
indexadd.fr_  samples           vb\samples\vbodbc\indexadd.frm
join.fr_      samples           vb\samples\vbodbc\join.frm
opendb.fr_    samples           vb\samples\vbodbc\opendb.frm
query.fr_     samples           vb\samples\vbodbc\query.frm
query1.fr_    samples           vb\samples\vbodbc\query.frx
replace.fr_   samples           vb\samples\vbodbc\replace.frm
replace1.fr_  samples           vb\samples\vbodbc\replace.frx
sql.fr_       samples           vb\samples\vbodbc\sql.frm
sql1.fr_      samples           vb\samples\vbodbc\sql.frx
tables.fr_    samples           vb\samples\vbodbc\tables.frm
tables1.fr_   samples           vb\samples\vbodbc\tables.frx
tblstru.fr_   samples           vb\samples\vbodbc\tblstru.frm
vdmdi.fr_     samples           vb\samples\vbodbc\vdmdi.frm
vdmdi1.fr_    samples           vb\samples\vbodbc\vdmdi.frx
visdata.ba_   samples           vb\samples\vbodbc\visdata.bas
visdata.ic_   samples           vb\samples\vbodbc\visdata.ico
visdata.ma_   samples           vb\samples\vbodbc\visdata.mak
zoom.fr_      samples           vb\samples\vbodbc\zoom.frm
compress.ex_  samples           vb\setupkit\kitfiles\compress.exe
compress.tx_  samples           vb\setupkit\kitfiles\compress.txt
expand.ex_    samples           vb\setupkit\kitfiles\expand.exe
setupa.ex_    samples           vb\setupkit\kitfiles\setup.exe
setupa.ls_    samples           vb\setupkit\kitfiles\setup.lst
setupkit.dl_  samples           vb\setupkit\kitfiles\setupkit.dll
commdlg.dl_   visual basic      windows\system\commdlg.dll
ddeml.dl_     visual basic      windows\system\ddeml.dll
fx.dl_        visual basic      windows\system\fx.dll
olecli.dl_    visual basic      windows\system\olecli.dll
code.ma_      tutorial          vb\vb.cbt\code.mak
country.ma_   tutorial          vb\vb.cbt\country.mak
form1.fr_     tutorial          vb\vb.cbt\form1.frm
form2.fr_     tutorial          vb\vb.cbt\form2.frm
formchi.fr_   tutorial          vb\vb.cbt\formchi.frm
mdinpad.fr_   tutorial          vb\vb.cbt\mdinpad.frm
payment.fr_   tutorial          vb\vb.cbt\payment.frm
payment.ma_   tutorial          vb\vb.cbt\payment.mak
sweden.fr_    tutorial          vb\vb.cbt\sweden.frm
vb.le_        tutorial          vb\vb.cbt\vb.les
gsw.ex_       controls          windows\system\gsw.exe
gswdll.dl_    controls          windows\system\gswdll.dll

***** DISK6 ****

bullet.bm_    help compiler     vb\hc\bullet.bmp
```

emdash.bm_	help compiler	vb\hc\emdash.bmp
hc31.er_	help compiler	vb\hc\hc31.err
helpref.hl_	help compiler	vb\hc\helpref.hlp
iconwrks.bm_	help compiler	vb\hc\iconwrks.bmp
iconwrks.hp_	help compiler	vb\hc\iconwrks.hpj
iconwrks.ph_	help compiler	vb\hc\iconwrks.ph
iconwrks.rt_	help compiler	vb\hc\iconwrks.rtf
iwedit.sh_	help compiler	vb\hc\iwedit.shg
mrbc.ex_	help compiler	vb\hc\mrbc.exe
shed.ex_	help compiler	vb\hc\shed.exe
shed.hl_	help compiler	vb\hc\shed.hlp
track.do_	help compiler	vb\hc\track.doc
winhelp.tx_	help compiler	vb\hc\winhelp.txt
3dlrsign.wm_	clip art	vb\metafile\business\3dlrsign.wmf
alphbord.wm_	clip art	vb\metafile\business\alphbord.wmf
alphtrpn.wm_	clip art	vb\metafile\business\alphtrpn.wmf
answmach.wm_	clip art	vb\metafile\business\answmach.wmf
apptbook.wm_	clip art	vb\metafile\business\apptbook.wmf
calcultr.wm_	clip art	vb\metafile\business\calcultr.wmf
calendar.wm_	clip art	vb\metafile\business\calendar.wmf
cent.wm_	clip art	vb\metafile\business\cent.wmf
check.wm_	clip art	vb\metafile\business\check.wmf
clipbord.wm_	clip art	vb\metafile\business\clipbord.wmf
coins.wm_	clip art	vb\metafile\business\coins.wmf
computer.wm_	clip art	vb\metafile\business\computer.wmf
copymach.wm_	clip art	vb\metafile\business\copymach.wmf
deutsch.wm_	clip art	vb\metafile\business\deutsch.wmf
digitals.wm_	clip art	vb\metafile\business\digitals.wmf
digitnum.wm_	clip art	vb\metafile\business\digitnum.wmf
dime.wm_	clip art	vb\metafile\business\dime.wmf
disk35.wm_	clip art	vb\metafile\business\disk35.wmf
disk525.wm_	clip art	vb\metafile\business\disk525.wmf
dollar.wm_	clip art	vb\metafile\business\dollar.wmf
dollars.wm_	clip art	vb\metafile\business\dollars.wmf
envlback.wm_	clip art	vb\metafile\business\envlback.wmf
envlfrnt.wm_	clip art	vb\metafile\business\envlfrnt.wmf
fileclsd.wm_	clip art	vb\metafile\business\fileclsd.wmf
fileopen.wm_	clip art	vb\metafile\business\fileopen.wmf
guilder.wm_	clip art	vb\metafile\business\guilder.wmf
harddisk.wm_	clip art	vb\metafile\business\harddisk.wmf
laptop1.wm_	clip art	vb\metafile\business\laptop1.wmf
laptop2.wm_	clip art	vb\metafile\business\laptop2.wmf
micrchip.wm_	clip art	vb\metafile\business\micrchip.wmf
money.wm_	clip art	vb\metafile\business\money.wmf
moneybag.wm_	clip art	vb\metafile\business\moneybag.wmf
monitor.wm_	clip art	vb\metafile\business\monitor.wmf
monystk1.wm_	clip art	vb\metafile\business\monystk1.wmf
monystk2.wm_	clip art	vb\metafile\business\monystk2.wmf
nickel.wm_	clip art	vb\metafile\business\nickel.wmf
payphone.wm_	clip art	vb\metafile\business\payphone.wmf
pcomputr.wm_	clip art	vb\metafile\business\pcomputr.wmf
penny.wm_	clip art	vb\metafile\business\penny.wmf
peseta.wm_	clip art	vb\metafile\business\peseta.wmf
phone.wm_	clip art	vb\metafile\business\phone.wmf
postcard.wm_	clip art	vb\metafile\business\postcard.wmf
pound.wm_	clip art	vb\metafile\business\pound.wmf
poundbag.wm_	clip art	vb\metafile\business\poundbag.wmf

```
printer.wm_   clip art           vb\metafile\business\printer.wmf
prntout1.wm_  clip art           vb\metafile\business\prntout1.wmf
prntout2.wm_  clip art           vb\metafile\business\prntout2.wmf
prntout3.wm_  clip art           vb\metafile\business\prntout3.wmf
quarter.wm_   clip art           vb\metafile\business\quarter.wmf
rolodex.wm_   clip art           vb\metafile\business\rolodex.wmf
ruble.wm_     clip art           vb\metafile\business\ruble.wmf
satedish.wm_  clip art           vb\metafile\business\satedish.wmf
satelit1.wm_  clip art           vb\metafile\business\satelit1.wmf
satelit2.wm_  clip art           vb\metafile\business\satelit2.wmf
typewrtr.wm_  clip art           vb\metafile\business\typewrtr.wmf
yen.wm_       clip art           vb\metafile\business\yen.wmf
anibuton.vb_  controls           windows\system\anibuton.vbx
cmdialog.vb_  controls           windows\system\cmdialog.vbx
gauge.vb_     controls           windows\system\gauge.vbx
graph.vb_     controls           windows\system\graph.vbx
grid.vb_      controls           windows\system\grid.vbx
keystat.vb_   controls           windows\system\keystat.vbx
mci.vb_       controls           windows\system\mci.vbx
mscomm.vb_    controls           windows\system\mscomm.vbx
msmapi.vb_    controls           windows\system\msmapi.vbx
msmasked.vb_  controls           windows\system\msmasked.vbx
oleclien.vb_  controls           windows\system\oleclien.vbx
pencntrl.vb_  controls           windows\system\pencntrl.vbx
picclip.vb_   controls           windows\system\picclip.vbx
spin.vb_      controls           windows\system\spin.vbx
threed.vb_    controls           windows\system\threed.vbx
win30api.tx_  windows api        vb\winapi\win30api.txt
win31api.hl_  windows api        vb\winapi\win31api.hlp
win31ext.tx_  windows api        vb\winapi\win31ext.txt

***** DISK7 ****

autopro.ma_   controls           vb\autoload.mak
bright.di_    visual basic       vb\bright.dib
demo.ex_      controls           vb\demo.exe
intro2vb.ex_  visual basic       vb\intro2vb.exe
pastel.di_    visual basic       vb\pastel.dib
proconst.tx_  visual basic       vb\constant.txt
rainbow.di_   visual basic       vb\rainbow.dib
samples.tx_   visual basic       vb\samples.txt
ctrlref.hl_   controls           vb\ctrlref.hlp
vb.ex_        visual basic       vb\vb.exe

***** DISK8 ****

vb.hl_        visual basic       vb\vb.hlp

Additional reference words: 2.00
KBCategory:
KBSubcategory: RefsDoc
```

README.TXT for Professional Edition of VB Ver 2.0 for Windows

Article ID: Q100632

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic programming system for Windows, version 2.0

SUMMARY

The following article contains the complete contents of the README.TXT file distributed with Microsoft Professional Edition of Visual Basic version 2.0 for Windows.

MORE INFORMATION

README.TXT

Release Notes for Microsoft (R) Visual Basic (TM) Professional Edition

Version 2.00

This document contains release notes for Microsoft Visual Basic for Windows Professional Edition version 2.0. Information in this document is more current than that in the manuals or online Help.

****
Read Part 1 - Software Installation Information - before installing.
****

Contents

Part	Description
1	Software Installation Information
2	Notes and Tips
3	Notes for Microsoft Visual Basic "Language Reference"
4	Notes for Microsoft Visual Basic "Programmer's Guide"
5	Updated Information
6	Custom Controls
7	Help Compiler

8 ODBC Information

Part 1: Software Installation Information

Before installing Visual Basic 2.0 Professional Edition, you should make backup copies of all the distribution disks. Do not write-protect the distribution disks you use to install Visual Basic. If you do, Visual Basic cannot be successfully installed.

SETUP.EXE is a Windows application; that is, it is run from Windows rather than from the MS-DOS prompt. SETUP.EXE will only run in Windows Standard or Enhanced mode. It will not run in Real mode. You can determine how Windows is configured on your computer by choosing About from the Help menu in the Program Manager.

To install Visual Basic 2.0 Professional, use Program Manager or File Manager to start SETUP.EXE as you would any other Windows-based application. For example, if you are installing from drive A:

- From the Program Manager File menu, choose Run.

- In the Run dialog box, type A:SETUP and choose OK.

-or-

- From the File Manager, double-click on the SETUP.EXE file icon on drive A.

Most of the files on these disks are compressed and must be expanded before they can be used. For Visual Basic to work properly, you must install the files using SETUP.EXE. You cannot simply copy the files to your hard disk.

If you want to install Visual Basic 2.0 Professional Edition in a directory other than where you've installed Visual Basic 1.0, you should specify this on the Installation Location screen. If you install Visual Basic 2.0 Professional Edition in the same directory as Visual Basic 1.0, most of the program, sample, icon, and tutorial files will be updated.

During setup, Visual Basic installs the custom controls GRID.VBX and OLECLIEN.VBX into your Windows system directory. If you have custom controls with the same name as the custom controls included with Visual Basic 2.0, and these controls are not the same GRID.VBX and OLECLIEN.VBX as were distributed with the Microsoft Professional Toolkit for Visual Basic 1.0, then Visual Basic installs GRID.VBX and OLECLIEN.VBX in your Visual Basic directory.

The WINHELP.EXE and COMMDLG.DLL files included with Visual Basic 2.0 are the latest English versions of these files. They will upgrade an older version, regardless of the language (e.g., French and German) of the older version.

IMPORTANT INFORMATION ABOUT ODBC INSTALLATION

After installing Visual Basic Professional, in order to install ODBC

functionality you must run the setup ODBC program from the end of the Visual Basic Professional setup, or from the ODBC setup icon, or from setup in the \VB\ODBC subdirectory. Follow the instructions from the setup program and refer to the ODBC setup documentation in your documentation set. ODBC Setup is a Windows program and should be run from within Windows.

Part 2 : Notes and Tips

Using Command-Line Options

Visual Basic can be started using command line options. The syntax is:

```
VB [[/R[UN] fname] [/C[MD] commandline]]|[/M[AKE] projname[.mak] [exename]]
```

Option	Explanation
/RUN	Runs the application specified by the filename (fname).
/CMD	Allows you to input a command-line argument that you can later use via the Command function. See online Help for more information.
/MAK	Loads the project named in projname and executes the Make .EXE command from the File menu to create an executable file whose name is provided as exename. See the "Microsoft Visual Basic "Programmer's Guide" for more information.

The fname parameter to the /RUN option must have a .MAK extension if it is to be treated as a project file. Any file with a different extension will be loaded as either a form or module into a new project.

Copying Icons to the Clipboard

To copy an icon (.ICO file) from a picture box to the Clipboard, you must set its AutoRedraw property to True, and copy its Image property to the Clipboard by specifying the CF_BITMAP format in the SetData method.

Limit on Size of MultiLine Text Boxes

The Text property of text box with MultiLine = True is limited to approximately 30K of text, while the limit is 32K if MultiLine = False.

Setting the Visible Property of Modal Forms to False

If you set the Visible property of a modal form to False, the form becomes a modeless form. If you need to return a form to a modal state after hiding it, use formid.Show 1 instead of formid.Visible = True.

Shortcut Keys Interrupting the "Learning Microsoft Visual Basic" Lessons

"Learning Microsoft Visual Basic" uses shortcut keys for accessing property settings in the Properties window. These keys may conflict with Shortcut keys you have set for making another Windows application active (set in the Program Manager Item Properties dialog.) Specifically, Ctrl+Shift+H, W, L, T, N, C, and V are used by Learning Visual Basic. If you have these key combinations as Shortcut keys for other Windows-based applications, you may accidentally activate these applications while running "Learning Visual Basic" is running.

Limit on Number of Objects per Application

There is a limit of 512 distinct objects in an application. Visual Basic uses 80 of these for global objects, data types, and standard controls. Thus, your application can have up to 432 form types or control classes. Each custom control class and form type that you create for your application is included in this limit. Each form uses one object and each custom control uses two. Note that custom controls may have multiple control classes in one .VBX file. This corrects the documentation in Appendix D, "Specifications and Limitations" in the "Programmers Guide."

Creating Toolbars or Status Bars with Borders on Top or Bottom

Setting BorderStyle to 1 - Fixed Single creates a border on all four sides of an aligned picture box. If you want to create a toolbar or status bar with borders on only the top or bottom, set BorderStyle to 0 - None. Then, write code in the Picture_Resize event to create a line each time a form is shown or resized. Either place a line control in the picture box and reset the X1, X2, Y1, and Y2 properties, or include the Line statement. For example:

```
Picture1.Line (0, Picture1.Height) - (Picture1.Width, Picture1.Height)
```

OLE Client Custom Control Error Messages

The following error messages correct and add to those contained in the "Language Reference."

31005	Object closed
31006	Can't close
31007	Can't paste
31008	Invalid property value
31009	Object not empty
31010	Property is read-only
31011	Type of object cannot be created
31012	No object name
31013	No document
31014	This action is reserved for future use
31015	Cannot execute object
31016	Server class was not specified before the registration database was accessed
31017	Invalid format on set data or set data text
31018	Server Class is not set
31019	Source document is not set
31020	Source item is not set

Closing Help Files

The code to close a Visual Basic application's Help file when the application concludes is as follows:

```
Declare Function WinHelp Lib "user.exe" (ByVal hWnd As Integer, ByVal
helpfilename$, ByVal hCommand As Integer, ByVal ddata As Long) As Integer

Sub Form_Unload (Cancel As Integer)
    Const HELP_QUIT = 2
    ErrCode% = WinHelp(Form1.hWnd, app.HelpFile, HELP_QUIT, 0)
End Sub
```

Creating Single-Column or Single-Row Grids

To create a single-column or single-row grid where the column or row is not fixed, set the appropriate Col and FixedCol properties at run time.

Naming Conflicts

Do not name any control "Line" or "Timer" as you will be unable to access its properties. The prohibited names conflict with the names of the Line method and Timer function respectively.

Picture formats accepted by DDE client Picture controls

In Windows version 3.1 and above, picture controls acting as clients in a DDE conversation will accept CF_METAFILE, CF_BITMAP, and CF_DIB format graphics. In Windows version 3.0, DDE client Picture controls accept only CF_BITMAP and CF_DIB format graphics.

Implicit Methods for Custom Controls

There are methods that apply to controls and forms which are not exposed via the VBM_METHOD interface. Custom controls have no way of altering, supplementing, or removing these methods when invoked on instances of the custom control. Examples of these methods are PrintForm, Refresh, LinkPoke, LinkRequest, LinkExecute, and SetFocus. The LinkPoke, LinkRequest, and LinkExecute methods apply only to controls with standard DDE properties; SetFocus applies only to controls with MODEL_fFocusOk.

Specifying the Size Property of a Field Object

When specifying the Size property of a field object that you are creating, you should use the length of the corresponding Visual Basic data type. Visual Basic only allows you to create fields using the sizes of the Visual Basic data types. For instance, if you are creating a Currency field, you should create a field of eight bytes. If you are copying one field to another, use the setting of the Type property of the source field to determine the type, and hence the size, of the destination field.

Part 3 : Notes for Microsoft Visual Basic "Language Reference"

Page	Section\Note

82 DateValue Function

Change the last sentence in the second paragraph of the Remarks section as follows:

"For example, in addition to recognizing 12/30/1991 and 12/30/91, DateValue recognizes December 30, 1991 and Dec 30, 1991.

320 Print Method

In the description of expressionlist at the top of the page, the term "text expression" should read "string expression".

386 Shell Function

Change the second sentence in the description of commandstring as follows:

"If the program name in commandstring does not include .BAT, .COM, .EXE, or .PIF extension, .EXE is assumed."

489 Not Operator

See online Help for more current information.

None Me Keyword

The Me keyword is not documented in the "Language Reference." See online Help for complete information.

Part 4: Notes for Microsoft Visual Basic "Programmer's Guide"

Page	Section\Note

4 Visual Basic Documentation

In the second bullet list item, replace "eight" with "seven".

6 Using Online Documentation

In the second sentence, replace "eight" with "seven".

8 Figure 1.2 The Contents Screen

This illustration does not depict the actual Contents screen.

15 Starting Visual Basic

In the second table item, under "Menu equivalent," change to read, "Start command on the Run menu".

19 Setting Properties

In the second paragraph of step 3, change "Clicking the DOWN ARROW key at the right ..." to "Clicking the down arrow at the right ..."

23 Simple Animation

In the Setting column of the table, change "(White)" to "(Black)".

In the paragraph following the table, change "... and the BackColor property to 0 (Black)" to "... and the BackColor property to black"

201 Identifying the Current Mode

In the paragraph at the bottom of the page, the phrase after the semicolon should read, "the unavailable buttons appear dimmed on the toolbar."

210 Using the Calls Dialog

Remove the ")" from the end of step 1.

216 Editing or Deleting a Watch Expression

In the numbered list at the top of the page, remove the "(s)" from the word "expression" in the second step.

220 Assigning Values to Variables and Properties

In the paragraph following the three lines of example statements, change to read, "The first statement alters a property of the currently active form, the second alters a property of the VScroll1 control, and the third assigns a value to a variable."

227 How to Handle Errors

The list of steps is incorrectly numbered. The paragraph now numbered 2 should not be numbered. Remove the number 2 from that paragraph. Then replace the 3 in the following paragraph with 2 and replace 4 in the last paragraph with 3.

229 Exiting an Error-Handling Routine

In the table that describes ways to exit an error-handling routine, make the following changes:

Replace the Resume entry with:

Resume (0) Program execution resumes with the statement that caused the error or the most recently executed call out of the procedure containing the error-handling

routine.

Change the Resume Next entry by removing the period at the end of the sentence and adding "or with the statement immediately following the most recently executed call out of the procedure containing the error-handling routine."

Change the Resume line by removing the period at the end of the sentence and adding, "that must be in the same procedure as the error handler."

234 Change the note at the bottom of the page as follows:

Remove everything after the first sentence. Add the following:

If a Resume statement is executed, control returns to the most recently executed call out of the procedure containing the error handler. If a Resume Next statement is executed, control returns to whatever statement in the procedure containing the error-handling routine immediately follows the most recently executed call out of that procedure. For example, in the Calls list shown in Figure 10.3, if procedure A has an enabled error handler and Procedure B and C don't, an error occurring in Procedure C will be handled by Procedure A's error handler. If that error handler uses a Resume statement, upon exit, the program continues with a call to Procedure B. However, if Procedure A's error handler uses a Resume Next statement, upon exit, the program will continue with whatever statement in Procedure A follows the call to Procedure B. In neither case does the error handler return directly to either the procedure or the statement where the error originally occurred.

420 The Directory List Box

In the code at the bottom of the page, change the first line as follows:

```
GoHigher = 0     ' Initialize for currently expanded directory.
```

421 The File List Box

Change the first paragraph as follows:

"The file list box displays files contained in the directory specified by the Path property at run time. You can display all the files in the current directory on the current drive using the following statement:"

The paragraph that begins, "If you set the System property..." may be misleading. The following additional information is provided to clarify meaning.

The default value for the System and Hidden properties is False. The default value for the Normal, Archive, and ReadOnly properties is True.

When Normal = True, any file that does not have the System or

Hidden attribute is displayed. When Normal = False, you can still display files with ReadOnly and/or Archive attributes by setting the appropriate attribute to True (i.e., ReadOnly = True, Archive = True).

When System = True, any file with the System attribute is displayed unless it also has the Hidden attribute.

When Hidden = True, any file with the Hidden attribute is displayed unless it also has the System attribute.

To display any file that has both Hidden and System among its attributes, both Hidden and System must be True. However, files having either Hidden or System attributes will be displayed as well.

424 Writing Code for the WinSeek Application

In the second paragraph, change the first sentence as follows:

"The WinSeek application resolves this ambiguity by determining if the path of the dirList box is different from the currently highlighted directory."

425 The cmdSearch_Click Procedure

In the sample code shown, change the If statement as follows:

If dirList.Path <> dirList.List (dirList.ListIndex) Then

482 Change the last sentence in the paragraph at the top of the page as follows:

"When the user activates the object (the graph), the server application (MS Graph) is invoked by the client application (Visual Basic), and the object's data is opened for editing."

Part 5: Updated Information

Online Resource Information

The WIN30API.TXT and WIN31EXT.TXT files are ASCII text files containing the functions and constants in the Microsoft Windows 3.0 and 3.1 API, declared in the format used by Microsoft Visual Basic.

They include:

- External procedure declarations for all the Microsoft Windows API functions that can be called from Visual Basic.
- Global constant declarations for all the constants used by the Microsoft Windows API.
- Type declarations for the user-defined types (structures) used by the Microsoft Windows API.

WIN30API.TXT is too large for the Notepad editor supplied with Microsoft Windows, but it can be loaded by Microsoft Word. To use WIN30API.TXT, load it into an editor (such as Microsoft Word) that can handle large files. Copy the declarations you want and paste them into any module in your Visual Basic application.

The file WIN31API.HLP contains the WIN30API.TXT and WIN31EXT.TXT file declarations in help file format. Use this file to easily cut and paste declarations into your Visual Basic 2.0 programs.

NOTE: Some of the Windows API declarations are very long. Some editors will wrap these onto a second line, and will copy them as multiple lines rather than as a single line. Declarations in Visual Basic cannot span lines, so if you paste these as multiple lines Visual Basic reports an error. If this happens, you can either adjust the margins in the editor before copying, or remove the line break after pasting.

You can place the declarations that you copy from the WIN31API.HLP file in the Declarations section of any form or module. You can also place the constant declarations anywhere in any form or module if you remove the Global keyword.

Once you have pasted the declaration for a Windows API routine (as well as any associated constant and type declarations) into your application, you can call that routine as you would call any Visual Basic procedure.

WARNING: Visual Basic cannot verify the data you pass to Microsoft Windows API routines. Calling a Microsoft Windows API routine with an invalid argument can result in unpredictable behavior - your application, Visual Basic, or Windows could crash or hang. When experimenting with Windows API routines, save your work often.

The WIN31WH.HLP file contains complete reference information for the functions, messages, and data structures in the Microsoft Windows 3.1 API. This is the same file that is shipped with the Microsoft Windows Software Development Kit. Open it by double-clicking the WIN SDK Help icon in the Visual Basic group in the Windows Program Manager.

WIN31WH.HLP provides detailed information about each of the functions, messages, and data structures in the Microsoft Windows API.

Installing Online Resource Files

WINSDK.HLP, APIXREF.HLP, and WINAPI.TXT are old versions of Windows 3.0 API support files that were included with the Visual Basic 1.0 Professional Toolkit. These files are not deleted when you install Visual Basic 2.0, but you may want to delete them yourself. The installed files WIN31WH.HLP, WIN30API.TXT, WIN31EXT.TXT, and WIN31API.HLP are replacements for the older files.

Part 6: Custom Control Reference

Installing Custom Controls

Visual Basic 2.0 installs custom controls in the Windows system directory. If you previously installed the Visual Basic 1.0 Professional Toolkit, you may have old versions of controls in the \VB\VBX directory. You should remove any older versions of controls you may have on your machine.

VB.LIC

Please make sure that the copy of VB.LIC in the \windows\system directory is the only one on your computer. Otherwise, you may encounter problems loading the new custom controls.

If an older VB.LIC is being used, or you have a missing or incorrectly installed VB.LIC file, the following message appears when you try to add the custom control file to your project:

> "License file for custom control not found. You do not have an appropriate license to use this custom control in the design environment."

Animated Button (ANIBUTON.VBX) Update

"Value" is the default value of the control.

Unlike the standard command button, when you press [Enter] on an animated command button, a Click event is NOT generated. You can use the KeyPress event to detect a click if necessary.

Communications Control (MSCOMM.VBX) Update

The port address and interrupt address can be changed from the Windows Control Panel.

The following defined constants for the Handshaking property have been added to the CONSTANT.TXT file:

```
Global Const MSCOMM_HANDSHAKE_NONE = 0
Global Const MSCOMM_HANDSHAKE_XONXOFF = 1
Global Const MSCOMM_HANDSHAKE_RTS = 2
Global Const MSCOMM_HANDSHAKE_RTSXONXOFF = 3
```

InBufferCount property: This property is Read/Write at run time.

SThreshold property: The MSCOMM_EV_SEND event is only fired once, when the number of characters crosses the SThreshold. For example, if SThreshold equals five the MSCOMM_EV_SEND event occurs only when the number of characters drops from five to four in the output queue. If there are never more than SThreshold characters in the output queue the event is never fired.

OnComm Event: The MSCOMM_ER_RXOVER only gets set when the receive queue overflows. It will not get set after a Chr$(26), EOF.

PortOpen property: If either the DTREnable or RTSEnable properties are set

to True before the port is opened, the properties are set to False when the port is closed. Otherwise, the DTR and RTS lines remain in their previous state.

The DSR_EVENT is only fired when DSR goes from -1 to 0.

Graph (GRAPH.VBX) Update

The Graph control includes the Palette property. Refer to online Help.

MAPI (Messaging API) Update

The RESOLVE_NAME action resets RecipType.

Masked Edit Control (MSMASKED.VBX) Update

The Text property is not available at design time.

Multimedia Control (MCI.VBX) Update

All references to the DisplayhWnd property should be hWndDisplay.

DeviceType property should refer to "Videodisc" not "Videodisk" as a device.

TimeFormat property should include

2	MCI_FORMAT_MSF	Minutes, seconds, and frames are packed into a four-byte integer. From least significant byte to most significant byte, the individual data values follow:

Minutes (least significant byte)
Seconds
Frames
Unused (most significant byte)

Pen Controls (PENCNTRL.VBX) Update

The error constants are incorrect on pages 278, 280, 287, 288, and 297. The correct values are defined in online Help.

The Ink On Bitmap and On-Screen Keyboard controls refer to CONSTANT.TXT. The correct file for constant definitions is PENAPI.TXT in the SAMPLES\PEN directory.

For the SKB_Change event of the On-Screen Keyboard Control, there is one additional possible value for the ChangeCode parameter:

SKN_TERMINATED	The on-screen keyboard has been closed.

This constant is defined in the file PENAPI.TXT.

The Distribution Note at the beginning of the chapter "Pen Edit Controls" should say the custom control file is named PENCNTRL.VBX.

To facilitate the interaction between Microsoft Windows For Pen API and Visual Basic, we have created two new APIs in the PENCNTRL.VBX file:

CPointerToVBType(ByVal lpSrc As Long, vbDest As Any, ByVal cNum By Integer)

- Copies cNum number of bytes from a memory location pointed to by lpSrc and places them in the vbDest memory location.

VBTypeToCPointer(vbSrc As Any, ByVal lpDest As Long, ByVal cNum By Integer)

- This reverses the process, copying from Visual Basic memory to a location specified by lpDest.

These two functions are used when dealing with RcResult or when pen data is required to be accessed by a Visual Basic program.

Shipping PENWIN.DLL with Your Application

PENWIN.DLL is a fully redistributable component of Windows for Pen Computing. Because applications will seek to leverage the Pen API - Visual Basic controls in particular - PENWIN.DLL can be shipped with your application. There are some considerations to keep in mind when shipping PENWIN.DLL with your application:

1. PENWIN.DLL functions ONLY under Windows 3.1. It WILL NOT WORK with Windows 3.0 because it functions only as an installable device driver (a feature not present in Windows 3.0).

2. As with other redistributable components (such as COMMDLG.DLL and the OLE libraries), it is the responsibility of the application vendor to determine whether PENWIN.DLL has already been installed (there is a GetSystemMetrics() call for this) and to ensure that the version of PENWIN.DLL with the latest version stamping is the one that is running. These issues are the same for all redistributable components, and further information is contained in the Windows SDK.

3. Unlike some of the other redistributable components, if your application installs PENWIN.DLL for the first time, or replaces the current version with a later one, Windows will have to be restarted. As an installable driver PENWIN.DLL can be loaded only at Windows boot time. Restarting Windows can be accomplished via an ExitWindows() call or by simply prompting the user to do so.

   NOTE: To install PENWIN.DLL on a Windows 3.1 system follow the directions listed under the "Pen Sample" in the file SAMPLES.TXT in the SAMPLES subdirectory.

4. PENWIN.DLL may be in either the \WINDOWS or the \WINDOWS\SYSTEM

directory. The default will be \WINDOWS, but since Windows for Pen Computing is an OEM product, Microsoft cannot completely control where PENWIN.DLL is located on a particular machine.

To get a good feel for the Pen Windows controls, you are encouraged to use and experiment with the Pen sample application (PENSMPL.MAK).

Spin Control (SPIN.VBX) Update

Removed TabIndex property.

3D Command Button (THREED.VBX) Update

Under certain video drivers, the 3D Command button will not print when performing a PrintForm operation. This problem occurs when printing from the design environment or at run-time. If you are distributing an application that causes 3D Command buttons to be printed you may want to implement an alternative method of printing a form as outlined in KnowledgeBase article Q84066, "How to Print Entire VB Form and Control the Printed Size." This article can be found in the online KnowledgeBase provided with the Visual Basic 2.0 Professional Edition.

Part 7: Help Compiler

HC31.EXE and Using Protected Mode Memory

Version 3.10.504 of the Windows Help Compiler (HC31.EXE) requires protected- mode memory. A number of system configurations support protected- mode memory by providing DMPI or VCPI servers. The following configurations provide protected-mode memory access to the Help Compiler, although not all configurations have been fully tested.

* Microsoft Windows version 3.0 or 3.1
  The best way to access protected mode memory is to run the Help Compiler in an MS-DOS session under Windows 3.1 enhanced or standard modes. You can also compile in an MS-DOS session under Windows 3.0 enhanced mode.

* EMM386 under MS-DOS
  The Help Compiler runs under EMM386 (if the noems option is not used and enough EMS memory is allocated to EMM386).

* 386Max
  The Help Compiler runs as a DMPI client with version 6.01 of 386Max (the version shipped with C7 or its equivalent). The Help Compiler may also work with some earlier versions of 386Max as a VCPI client if enough EMS memory is configured.

* QEMM
  Under version 6.02 of QEMM, the Help Compiler runs as a DPMI client. It should also work under some earlier versions of QEMM as a VCPI client, provided it is configured with enough EMS memory.

* EMS Memory Requirements

You need enough EMS memory to hold the Help Compiler and dynamic data being compiled. The exact amount of memory needed depends heavily on the size of the help file you want to compile. Between 1MB and 2MB should meet most needs.

Part 8: ODBC Information

If you use Windows for Workgroups and the ODBC object layer with SQL Sever installed on a Novell NetWare LAN, you will not be able to access SQL Server from ODBC. The Network Integration Kit (NIK) will resolve this problem. For details please contact Microsoft Product Support at the numbers listed in the Visual Basic Help file.

The Database object now supports the QueryTimeout property. Refer to the online help.

For information on distributing Visual Basic ODBC applications, refer to the online Help contents screen.

Additional reference words: 2.00
KBCategory:
KBSubcategory: RefsDoc

PACKING.LST for Professional Edition of VB 3.0 for Windows

Article ID: Q100633

The information in this article applies to:

- The Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The following article contains the complete contents of the PACKING.LST file distributed with Microsoft Professional Edition of Visual Basic version 3.0 for Windows.

MORE INFORMATION

PACKING.LST

Disk Contents for Microsoft (R) Visual Basic for Windows,
Professional Edition Version 3.0

This file lists all the files on the distribution disks provided with this product.

==========

NOTE: Most of the files on the distribution disks are compressed (indicated by an underscore character "_" in the file extension) and must be decompressed before they are used.

The Setup program on Disk 1 decompresses files as it installs them.

Files with a PA_ extension decompress to multiple .ICO files. These files cannot be decompressed manually.

If you need to decompress files manually, you can do so by running setup from the file manager or program manager in the following way:

1. Choose the Run item from the File menu.
2. Type SETUP.EXE /Z [source file] [destination file]

The /Z switch activates an internal decompression system. SETUP.EXE must be run from within Windows.

***** DISK1 ****

assorted.pa_	clip art	vb\bitmaps\assorted\assorted.pak
circlock.bm_	clip art	vb\bitmaps\gauge\circlock.bmp
dome.bm_	clip art	vb\bitmaps\gauge\dome.bmp
horz.bm_	clip art	vb\bitmaps\gauge\horz.bmp
horz1.bm_	clip art	vb\bitmaps\gauge\horz1.bmp
semicirc.bm_	clip art	vb\bitmaps\gauge\semicirc.bmp
therbot.bm_	clip art	vb\bitmaps\gauge\therbot.bmp

therlft.bm_	clip art	vb\bitmaps\gauge\therlft.bmp
thermid.bm_	clip art	vb\bitmaps\gauge\thermid.bmp
thermidh.bm_	clip art	vb\bitmaps\gauge\thermidh.bmp
therrt.bm_	clip art	vb\bitmaps\gauge\therrt.bmp
thertop.bm_	clip art	vb\bitmaps\gauge\thertop.bmp
vert.bm_	clip art	vb\bitmaps\gauge\vert.bmp
volbot.bm_	clip art	vb\bitmaps\gauge\volbot.bmp
voltop.bm_	clip art	vb\bitmaps\gauge\voltop.bmp
outbmps.pa_	clip art	vb\bitmaps\outbmps\outbmps.pak
toolbar.pa_	clip art	vb\bitmaps\toolbar3\toolbar.pak
circ1.c_	control dev kit	vb\cdk\circ1\circ1.c
circ1.de_	control dev kit	vb\cdk\circ1\circ1.def
circ1.h_	control dev kit	vb\cdk\circ1\circ1.h
circ1.rc_	control dev kit	vb\cdk\circ1\circ1.rc
circ1.vb_	control dev kit	vb\cdk\circ1\circ1.vbx
circ1cd.bm_	control dev kit	vb\cdk\circ1\circ1cd.bmp
circ1cu.bm_	control dev kit	vb\cdk\circ1\circ1cu.bmp
circ1eu.bm_	control dev kit	vb\cdk\circ1\circ1eu.bmp
circ1mu.bm_	control dev kit	vb\cdk\circ1\circ1mu.bmp
makefil1._	control dev kit	vb\cdk\circ1\makefile
circ2.c_	control dev kit	vb\cdk\circ2\circ2.c
circ2.de_	control dev kit	vb\cdk\circ2\circ2.def
circ2.h_	control dev kit	vb\cdk\circ2\circ2.h
circ2.rc_	control dev kit	vb\cdk\circ2\circ2.rc
circ2.vb_	control dev kit	vb\cdk\circ2\circ2.vbx
circ2cd.bm_	control dev kit	vb\cdk\circ2\circ2cd.bmp
circ2cu.bm_	control dev kit	vb\cdk\circ2\circ2cu.bmp
circ2eu.bm_	control dev kit	vb\cdk\circ2\circ2eu.bmp
circ2mu.bm_	control dev kit	vb\cdk\circ2\circ2mu.bmp
makefil2._	control dev kit	vb\cdk\circ2\makefile
cntr.c_	control dev kit	vb\cdk\cntr\cntr.c
cntr.de_	control dev kit	vb\cdk\cntr\cntr.def
cntr.h_	control dev kit	vb\cdk\cntr\cntr.h
cntr.rc_	control dev kit	vb\cdk\cntr\cntr.rc
cntr.vb_	control dev kit	vb\cdk\cntr\cntr.vbx
cntrcd.bm_	control dev kit	vb\cdk\cntr\cntrcd.bmp
cntrcu.bm_	control dev kit	vb\cdk\cntr\cntrcu.bmp
cntreu.bm_	control dev kit	vb\cdk\cntr\cntreu.bmp
cntrmu.bm_	control dev kit	vb\cdk\cntr\cntrmu.bmp
makefil4._	control dev kit	vb\cdk\cntr\makefile
makefil5._	control dev kit	vb\cdk\pal\makefile
pal.c_	control dev kit	vb\cdk\pal\pal.c
pal.de_	control dev kit	vb\cdk\pal\pal.def
pal.h_	control dev kit	vb\cdk\pal\pal.h
pal.rc_	control dev kit	vb\cdk\pal\pal.rc
pal.vb_	control dev kit	vb\cdk\pal\pal.vbx
palcd.bm_	control dev kit	vb\cdk\pal\palcd.bmp
palcu.bm_	control dev kit	vb\cdk\pal\palcu.bmp
paleu.bm_	control dev kit	vb\cdk\pal\paleu.bmp
palmu.bm_	control dev kit	vb\cdk\pal\palmu.bmp
makefil7._	control dev kit	vb\cdk\push\makefile
push.c_	control dev kit	vb\cdk\push\push.c
push.de_	control dev kit	vb\cdk\push\push.def
push.h_	control dev kit	vb\cdk\push\push.h
push.rc_	control dev kit	vb\cdk\push\push.rc
push.vb_	control dev kit	vb\cdk\push\push.vbx
pushcd.bm_	control dev kit	vb\cdk\push\pushcd.bmp

pushcu.bm_	control dev kit	vb\cdk\push\pushcu.bmp
pusheu.bm_	control dev kit	vb\cdk\push\pusheu.bmp
pushmu.bm_	control dev kit	vb\cdk\push\pushmu.bmp
pushvb1.h_	control dev kit	vb\cdk\push\pushvb1.h
vbapi.hl_	control dev kit	vb\cdk\vbapi.hlp
hc31.ex_	help compiler	vb\hc\hc31.exe
arrows.pa_	icons	vb\icons\arrows\arrows.pak
comm.pa_	icons	vb\icons\comm\comm.pak
computer.pa_	icons	vb\icons\computer\computer.pak
dragdrop.pa_	icons	vb\icons\dragdrop\dragdrop.pak
elements.pa_	icons	vb\icons\elements\elements.pak
flags.pa_	icons	vb\icons\flags\flags.pak
industry.pa_	icons	vb\icons\industry\industry.pak
mail.pa_	icons	vb\icons\mail\mail.pak
misc.pa_	icons	vb\icons\misc\misc.pak
office.pa_	icons	vb\icons\office\office.pak
traffic.pa_	icons	vb\icons\traffic\traffic.pak
writing.pa_	icons	vb\icons\writing\writing.pak
mcimmp.dr_	never	mcimmp.dr_
mmp.dl_	never	mmp.dl_
oemsetup.in_	never	oemsetup.inf
vboa300.dl_	ole	windows\system\vboa300.dll
dispcalc.ex_	ole	vb\dispcalc.exe
dispcalc.re_	ole	vb\dispcalc.reg
proreadm.tx_	always	vb\readme.txt
mssetup.ex	always	msvb2set.tmp_mssetup.exe
mscomstf.dl_	data access	vb\odbc\mscomstf.dll
mscustom.dl_	always	msvb2set.tmp\mscustom.dll
msdetstf.dl_	data access	vb\odbc\msdetstf.dll
msinsstf.dl_	data access	vb\odbc\msinsstf.dll
msshlstf.dl_	data access	vb\odbc\msshlstf.dll
msuilstf.dl_	data access	vb\odbc\msuilstf.dll
packing.ls_	visual basic	vb\packing.lst
setup.ex_	always	msvb2set.tmp\setup.exe
setup.ls_	always	msvb2set.tmp\setup.lst
vbsetup.ex_	always	msvb2set.tmp\vbsetup.exe
vbsetup.in_	always	msvb2set.tmp\vbsetup.ini
calc.fr_	samples	vb\samples\calc\calc.frm
calc.ic_	samples	vb\samples\calc\calc.ico
calc.ma_	samples	vb\samples\calc\calc.mak
calc1.fr_	samples	vb\samples\calc\calc.frx
calldll1.fr_	samples	vb\samples\calldlls\calldlls.frx
calldlls.fr_	samples	vb\samples\calldlls\calldlls.frm
calldlls.ma_	samples	vb\samples\calldlls\calldlls.mak
declares.ba_	samples	vb\samples\calldlls\declares.bas
frmmenus.fr_	samples	vb\samples\calldlls\frmmenus.frm
array.fr_	samples	vb\samples\controls\array.frm
button.fr_	samples	vb\samples\controls\button.frm
button1.fr_	samples	vb\samples\controls\button.frx
check.fr_	samples	vb\samples\controls\check.frm
controls.ma_	samples	vb\samples\controls\controls.mak
listbox.fr_	samples	vb\samples\controls\listbox.frm
main2.fr_	samples	vb\samples\controls\main.frm
multi.fr_	samples	vb\samples\controls\multi.frm
number.fr_	samples	vb\samples\controls\number.frm
scroll.fr_	samples	vb\samples\controls\scroll.frm
wordwrap.fr_	samples	vb\samples\controls\wordwrap.frm

```
biblio.fr_    samples          vb\samples\datactrl\biblio.frm
biblio1.fr_   samples          vb\samples\datactrl\biblio.frx
biblio.ma_    samples          vb\samples\datactrl\biblio.mak
datactl.ba_   samples          vb\samples\datactrl\datactl.bas
dde.ba_       samples          vb\samples\dde\dde.bas
dde.ma_       samples          vb\samples\dde\dde.mak
execute.fr_   samples          vb\samples\dde\execute.frm
main.fr_      samples          vb\samples\dde\main.frm
alarm.fr_     samples          vb\samples\envir\alarm.frm
alarm.ma_     samples          vb\samples\envir\alarm.mak
alarm1.fr_    samples          vb\samples\envir\alarm.frx
seek.fr_      samples          vb\samples\filectls\seek.frm
winseek.ma_   samples          vb\samples\filectls\winseek.mak
fileproc.ba_  samples          vb\samples\fileio\fileproc.bas
recedit.ba_   samples          vb\samples\fileio\recedit.bas
recedit.fr_   samples          vb\samples\fileio\recedit.frm
recedit.ma_   samples          vb\samples\fileio\recedit.mak
bfly1.bm_     samples          vb\samples\firstapp\bfly1.bmp
bfly2.bm_     samples          vb\samples\firstapp\bfly2.bmp
butterf.fr_   samples          vb\samples\firstapp\butterf.frm
butterf.ma_   samples          vb\samples\firstapp\butterf.mak
butterf1.fr_  samples          vb\samples\firstapp\butterf.frx
picview.fr_   samples          vb\samples\firstapp\picview.frm
picview.ma_   samples          vb\samples\firstapp\picview.mak
blanker.fr_   samples          vb\samples\graphics\blanker.frm
blanker.ma_   samples          vb\samples\graphics\blanker.mak
blanker1.fr_  samples          vb\samples\graphics\blanker.frx
loan.fr_      samples          vb\samples\grid\loan.frm
loan.ma_      samples          vb\samples\grid\loan.mak
loan1.fr_     samples          vb\samples\grid\loan.frx
filopen.ba_   samples          vb\samples\mdi\filopen.bas
find1.fr_     samples          vb\samples\mdi\find.frm
mdi.fr_       samples          vb\samples\mdi\mdi.frm
mdi1.fr_      samples          vb\samples\mdi\mdi.frx
mdinote.ba_   samples          vb\samples\mdi\mdinote.bas
mdinote.ma_   samples          vb\samples\mdi\mdinote.mak
notepad.fr_   samples          vb\samples\mdi\notepad.frm
about.fr_     samples          vb\samples\menus\about.frm
edit.fr_      samples          vb\samples\menus\edit.frm
textedit.ba_  samples          vb\samples\menus\textedit.bas
textedit.ma_  samples          vb\samples\menus\textedit.mak
click.fr_     samples          vb\samples\mouse\click.frm
drag.fr_      samples          vb\samples\mouse\drag.frm
drag1.fr_     samples          vb\samples\mouse\drag.frx
main1.fr_     samples          vb\samples\mouse\main.frm
mouse.ma_     samples          vb\samples\mouse\mouse.mak
scribble.fr_  samples          vb\samples\mouse\scribble.frm
frmmain.fr_   samples          vb\samples\objects\frmmain.frm
multinst.fr_  samples          vb\samples\objects\multinst.frm
multinst.ma_  samples          vb\samples\objects\multinst.mak
objects.ba_   samples          vb\samples\objects\objects.bas
objects.ma_   samples          vb\samples\objects\objects.mak
about1.fr_    samples          vb\samples\ole\about.frm
ole2chld.fr_  samples          vb\samples\ole\ole2chld.frm
ole2demo.ma_  samples          vb\samples\ole\ole2demo.mak
ole2mdi.fr_   samples          vb\samples\ole\ole2mdi.frm
ole2mod1.ba_  samples          vb\samples\ole\ole2mod1.bas
```

```
ole2mod2.ba_  samples             vb\samples\ole\ole2mod2.bas
oleauto.ba_   samples             vb\samples\ole\oleauto.bas
oleauto.fr_   samples             vb\samples\ole\oleauto.frm
oleauto.ma_   samples             vb\samples\ole\oleauto.mak
infofor1.fr_  samples             vb\samples\picclip\infoform.frx
infoform.fr_  samples             vb\samples\picclip\infoform.frm
redtop.fr_    samples             vb\samples\picclip\redtop.frm
redtop.ma_    samples             vb\samples\picclip\redtop.mak
redtop1.fr_   samples             vb\samples\picclip\redtop.frx
fontdial.fr_  samples             vb\samples\print\fontdial.frm
tccancel.fr_  samples             vb\samples\print\tccancel.frm
timecar1.fr_  samples             vb\samples\print\timecard.frx
timecard.fr_  samples             vb\samples\print\timecard.frm
timecard.ma_  samples             vb\samples\print\timecard.mak
cansend.fr_   samples             vb\samples\vbterm\cansend.frm
termset.fr_   samples             vb\samples\vbterm\termset.frm
termset1.fr_  samples             vb\samples\vbterm\termset.frx
vbterm.fr_    samples             vb\samples\vbterm\vbterm.frm
vbterm.gl_    samples             vb\samples\vbterm\vbterm.glo
vbterm.ma_    samples             vb\samples\vbterm\vbterm.mak
vbterm1.fr_   samples             vb\samples\vbterm\vbterm.frx
dialer.ma_    samples             vb\samples\vbterm\dialer.mak
dialer.fr_    samples             vb\samples\vbterm\dialer.frm
message.fr_   samples             vb\setupkit\setup1\message.frm
path.fr_      samples             vb\setupkit\setup1\path.frm
path1.fr_     samples             vb\setupkit\setup1\path.frx
setup1.ba_    samples             vb\setupkit\setup1\setup1.bas
setup1.fr_    samples             vb\setupkit\setup1\setup1.frm
setup1.gl_    samples             vb\setupkit\setup1\setup1.glb
setup1.ma_    samples             vb\setupkit\setup1\setup1.mak
setup11.fr_   samples             vb\setupkit\setup1\setup1.frx
status.fr_    samples             vb\setupkit\setup1\status.frm
shell.dl_     always              windows\system\shell.dll
ver.dl_       always              windows\system\ver.dll
vbknowlg.hl_  knowledgebase       vb\vbknowlg.hlp
cbt.ex_       tutorial            vb\vb.cbt\cbt.exe
cbtlib4.dl_   tutorial            vb\vb.cbt\cbtlib4.dll
vdg.ex_       visual design guide vb\vb.cbt\vdg.exe
vdg.le_       visual design guide vb\vb.cbt\vdg.les
vb.li_        always              windows\system\vb.lic

***** DISK2 ****

win31wh.h1_   windows api         vb\winapi\win31wh.hlp
 Split file, part 1 of 2

***** DISK3 ****

cdk.tx_       control dev kit     vb\cdk\cdk.txt
tn001.tx_     control dev kit     vb\cdk\tn001.txt
tn002.tx_     control dev kit     vb\cdk\tn002.txt
libentry.as_  control dev kit     vb\cdk\libentry.asm
libentry.ob_  control dev kit     vb\cdk\libentry.obj
makefil6._    control dev kit     vb\cdk\pix\makefile
pictblt.c_    control dev kit     vb\cdk\pix\pictblt.c
pictblt.h_    control dev kit     vb\cdk\pix\pictblt.h
pix.c_        control dev kit     vb\cdk\pix\pix.c
```

pix.de_	control dev kit		vb\cdk\pix\pix.def
pix.h_	control dev kit		vb\cdk\pix\pix.h
pix.rc_	control dev kit		vb\cdk\pix\pix.rc
pix.vb_	control dev kit		vb\cdk\pix\pix.vbx
pixcd.bm_	control dev kit		vb\cdk\pix\pixcd.bmp
pixcu.bm_	control dev kit		vb\cdk\pix\pixcu.bmp
pixeu.bm_	control dev kit		vb\cdk\pix\pixeu.bmp
pixmu.bm_	control dev kit		vb\cdk\pix\pixmu.bmp
pixvb1.h_	control dev kit		vb\cdk\pix\pixvb1.h
vbapi.h_	control dev kit		vb\cdk\vbapi.h
vbapi.li_	control dev kit		vb\cdk\vbapi.lib
vbx.rc_	control dev kit		vb\cdk\vbx.rcv
wps.ex_	control dev kit		vb\cdk\wps.exe
makefil9._	control dev kit		vb\cdk\xlist\makefile
xlist.c_	control dev kit		vb\cdk\xlist\xlist.c
xlist.de_	control dev kit		vb\cdk\xlist\xlist.def
xlist.h_	control dev kit		vb\cdk\xlist\xlist.h
xlist.rc_	control dev kit		vb\cdk\xlist\xlist.rc
xlist.vb_	control dev kit		vb\cdk\xlist\xlist.vbx
xlistcd.bm_	control dev kit		vb\cdk\xlist\xlistcd.bmp
xlistcu.bm_	control dev kit		vb\cdk\xlist\xlistcu.bmp
xlisteu.bm_	control dev kit		vb\cdk\xlist\xlisteu.bmp
xlistmu.bm_	control dev kit		vb\cdk\xlist\xlistmu.bmp
xlistvb1.h_	control dev kit		vb\cdk\xlist\xlistvb1.h
maillst.fr_	samples		vb\samples\mapi\maillst.frm
maillst1.fr_	samples		vb\samples\mapi\maillst.frx
mailoptf.fr_	samples		vb\samples\mapi\mailoptf.frm
mailsup.ba_	samples		vb\samples\mapi\mailsup.bas
msgview.fr_	samples		vb\samples\mapi\msgview.frm
msgview1.fr_	samples		vb\samples\mapi\msgview.frx
newmsg.fr_	samples		vb\samples\mapi\newmsg.frm
vbmail.fr_	samples		vb\samples\mapi\vbmail.frm
vbmail1.fr_	samples		vb\samples\mapi\vbmail.frx
vbmail.ma_	samples		vb\samples\mapi\vbmail.mak
firsttab.bm_	samples		vb\samples\msout\firsttab.bmp
lasttab.bm_	samples		vb\samples\msout\lasttab.bmp
midtab.bm_	samples		vb\samples\msout\midtab.bmp
phone.fr_	samples		vb\samples\msout\phone.frm
phone1.fr_	samples		vb\samples\msout\phone.frx
phone.ma_	samples		vb\samples\msout\phone.mak
phone.md_	samples		vb\samples\msout\phone.mdb
delay.fr_	samples		vb\samples\pen\delay.frm
delay1.fr_	samples		vb\samples\pen\delay.frx
editsubf.fr_	samples		vb\samples\pen\editsubf.frm
gestfrm.fr_	samples		vb\samples\pen\gestfrm.frm
grafpapr.bm_	samples		vb\samples\pen\grafpapr.bmp
inkfrm.fr_	samples		vb\samples\pen\inkfrm.frm
iobfrm.fr_	samples		vb\samples\pen\iobfrm.frm
iobfrm1.fr_	samples		vb\samples\pen\iobfrm.frx
keybrd.fr_	samples		vb\samples\pen\keybrd.frm
keybrd1.fr_	samples		vb\samples\pen\keybrd.frx
penapi.tx_	samples		vb\samples\pen\penapi.txt
penmain.fr_	samples		vb\samples\pen\penmain.frm
penmain1.fr_	samples		vb\samples\pen\penmain.frx
pensmpl.ma_	samples		vb\samples\pen\pensmpl.mak
rcfrm.fr_	samples		vb\samples\pen\rcfrm.frm
rulepapr.bm_	samples		vb\samples\pen\rulepapr.bmp

```
skbface.bm_   samples            vb\samples\pen\skbface.bmp
transfrm.fr_  samples            vb\samples\pen\transfrm.frm
win31wh.h2_   windows api        vb\winapi\win31wh.hlp
 Split file, part 2 of 2

***** DISK4 ****

biblio.md_    data access        vb\biblio.mdb
circ3.c_      control dev kit    vb\cdk\circ3\circ3.c
circ3.de_     control dev kit    vb\cdk\circ3\circ3.def
circ3.h_      control dev kit    vb\cdk\circ3\circ3.h
circ3.hl_     control dev kit    vb\cdk\circ3\circ3.hlp
circ3.hp_     control dev kit    vb\cdk\circ3\circ3.hpj
circ3.rc_     control dev kit    vb\cdk\circ3\circ3.rc
circ3.rt_     control dev kit    vb\cdk\circ3\circ3.rtf
circ3.vb_     control dev kit    vb\cdk\circ3\circ3.vbx
circ3cd.bm_   control dev kit    vb\cdk\circ3\circ3cd.bmp
circ3cu.bm_   control dev kit    vb\cdk\circ3\circ3cu.bmp
circ3eu.bm_   control dev kit    vb\cdk\circ3\circ3eu.bmp
circ3mu.bm_   control dev kit    vb\cdk\circ3\circ3mu.bmp
circ3vb1.h_   control dev kit    vb\cdk\circ3\circ3vb1.h
circ3vb2.h_   control dev kit    vb\cdk\circ3\circ3vb2.h
makefil3._    control dev kit    vb\cdk\circ3\makefile
xbs110.dl_    dbase driver       windows\system\xbs110.dll
pdx110.dl_    paradox driver     windows\system\pdx110.dll
btrv110.dl_   btrieve driver     windows\system\btrv110.dll
external.tx_  data access        vb\external.txt
btrieve.tx_   data access        vb\btrieve.txt
perform.tx_   data access        vb\perform.txt
datamgr.ex_   data access        vb\datamgr.exe
datamgr.hl_   data access        vb\datamgr.hlp
2darrow1.wm_  clip art           vb\metafile\arrows\2darrow1.wmf
2darrow2.wm_  clip art           vb\metafile\arrows\2darrow2.wmf
2darrow3.wm_  clip art           vb\metafile\arrows\2darrow3.wmf
2darrow4.wm_  clip art           vb\metafile\arrows\2darrow4.wmf
3darrow1.wm_  clip art           vb\metafile\arrows\3darrow1.wmf
3darrow2.wm_  clip art           vb\metafile\arrows\3darrow2.wmf
3darrow3.wm_  clip art           vb\metafile\arrows\3darrow3.wmf
3darrow4.wm_  clip art           vb\metafile\arrows\3darrow4.wmf
3darrow5.wm_  clip art           vb\metafile\arrows\3darrow5.wmf
3darrow6.wm_  clip art           vb\metafile\arrows\3darrow6.wmf
3darrow7.wm_  clip art           vb\metafile\arrows\3darrow7.wmf
3dxarrow.wm_  clip art           vb\metafile\arrows\3dxarrow.wmf
3dxcirar.wm_  clip art           vb\metafile\arrows\3dxcirar.wmf
halfarrw.wm_  clip art           vb\metafile\arrows\halfarrw.wmf
hortarrw.wm_  clip art           vb\metafile\arrows\hortarrw.wmf
hozcirar.wm_  clip art           vb\metafile\arrows\hozcirar.wmf
layerarw.wm_  clip art           vb\metafile\arrows\layerarw.wmf
lrgearrw.wm_  clip art           vb\metafile\arrows\lrgearrw.wmf
medarrw1.wm_  clip art           vb\metafile\arrows\medarrw1.wmf
medarrw2.wm_  clip art           vb\metafile\arrows\medarrw2.wmf
multarw1.wm_  clip art           vb\metafile\arrows\multarw1.wmf
multarw2.wm_  clip art           vb\metafile\arrows\multarw2.wmf
multarw3.wm_  clip art           vb\metafile\arrows\multarw3.wmf
multarw4.wm_  clip art           vb\metafile\arrows\multarw4.wmf
smallarw.wm_  clip art           vb\metafile\arrows\smallarw.wmf
tinyarrw.wm_  clip art           vb\metafile\arrows\tinyarrw.wmf
```

vertarrw.wm_	clip art	vb\metafile\arrows\vertarrw.wmf
vrtcirar.wm_	clip art	vb\metafile\arrows\vrtcirar.wmf
vrtcurar.wm_	clip art	vb\metafile\arrows\vrtcurar.wmf
xarrow.wm_	clip art	vb\metafile\arrows\xarrow.wmf
dbnmp3.od_	sql server driver	vb\odbc\dbnmp3.dl_
instcat.sq_	sql server driver	vb\odbc\instcat.sql
drvssrvr.hl_	sql server driver	vb\odbc\drvssrvr.hl_
sqlsrvr.od_	sql server driver	vb\odbc\sqlsrvr.dl_
aboutbo3.fr_	samples	vb\samples\iconworks\aboutbox.frx
aboutbox.fr_	samples	vb\samples\iconworks\aboutbox.frm
colorpa1.fr_	samples	vb\samples\iconworks\colorpal.frx
colorpal.fr_	samples	vb\samples\iconworks\colorpal.frm
iconedi1.fr_	samples	vb\samples\iconworks\iconedit.frx
iconedit.fr_	samples	vb\samples\iconworks\iconedit.frm
iconwrks.ba_	samples	vb\samples\iconworks\iconwrks.bas
iconwrks.gb_	samples	vb\samples\iconworks\iconwrks.gbl
iconwrks.hl_	samples	vb\samples\iconworks\iconwrks.hlp
iconwrks.ic_	samples	vb\samples\iconworks\iconwrks.ico
iconwrks.ma_	samples	vb\samples\iconworks\iconwrks.mak
screen.ic_	samples	vb\samples\iconworks\screen.ico
toolpal.bm_	samples	vb\samples\iconworks\toolpal.bmp
viewico1.fr_	samples	vb\samples\iconworks\viewicon.frx
viewicon.fr_	samples	vb\samples\iconworks\viewicon.frm
aboutbo1.fr_	samples	vb\samples\mci\aboutbox.frm
aboutbo6.fr_	samples	vb\samples\mci\aboutbox.frx
animate.fr_	samples	vb\samples\mci\animate.frm
cd.fr_	samples	vb\samples\mci\cd.frm
cd1.fr_	samples	vb\samples\mci\cd.frx
global3.ba_	samples	vb\samples\mci\global.bas
mcitest.ba_	samples	vb\samples\mci\mcitest.bas
mcitest.fr_	samples	vb\samples\mci\mcitest.frm
mcitest1.fr_	samples	vb\samples\mci\mcitest.frx
mcitest.ma_	samples	vb\samples\mci\mcitest.mak
mcitest.mi_	samples	vb\samples\mci\mcitest.mid
mcitest.mm_	samples	vb\samples\mci\mcitest.mmm
mcitest.wa_	samples	vb\samples\mci\mcitest.wav
opendlg.fr_	samples	vb\samples\mci\opendlg.frm
wave.fr_	samples	vb\samples\mci\wave.frm
aboutbo2.fr_	samples	vb\samples\visdata\aboutbox.frm
aboutbo5.fr_	samples	vb\samples\visdata\aboutbox.frx
addfield.fr_	samples	vb\samples\visdata\addfield.frm
attach.fr_	samples	vb\samples\visdata\attach.frm
cpystru.fr_	samples	vb\samples\visdata\cpystru.frm
databox.fr_	samples	vb\samples\visdata\databox.frm
dataform.fr_	samples	vb\samples\visdata\dataform.frm
datafor1.fr_	samples	vb\samples\visdata\dataform.frx
dynagri1.fr_	samples	vb\samples\visdata\dynagrid.frx
dynagrid.fr_	samples	vb\samples\visdata\dynagrid.frm
dynaset.fr_	samples	vb\samples\visdata\dynaset.frm
dynaset1.fr_	samples	vb\samples\visdata\dynaset.frx
find.fr_	samples	vb\samples\visdata\find.frm
indexadd.fr_	samples	vb\samples\visdata\indexadd.frm
join.fr_	samples	vb\samples\visdata\join.frm
opendb.fr_	samples	vb\samples\visdata\opendb.frm
query.fr_	samples	vb\samples\visdata\query.frm
query1.fr_	samples	vb\samples\visdata\query.frx
replace.fr_	samples	vb\samples\visdata\replace.frm

```
replace1.fr_  samples           vb\samples\visdata\replace.frx
seek4.fr_     samples           vb\samples\visdata\seek.frm
sql.fr_       samples           vb\samples\visdata\sql.frm
sql1.fr_      samples           vb\samples\visdata\sql.frx
tables.fr_    samples           vb\samples\visdata\tables.frm
tables1.fr_   samples           vb\samples\visdata\tables.frx
tblstru.fr_   samples           vb\samples\visdata\tblstru.frm
tableobj.fr_  samples           vb\samples\visdata\tableobj.frm
tableob1.fr_  samples           vb\samples\visdata\tableobj.frx
vdmdi.fr_     samples           vb\samples\visdata\vdmdi.frm
vdmdi1.fr_    samples           vb\samples\visdata\vdmdi.frx
visdata.ba_   samples           vb\samples\visdata\visdata.bas
visdata.ic_   samples           vb\samples\visdata\visdata.ico
visdata.ma_   samples           vb\samples\visdata\visdata.mak
zoom.fr_      samples           vb\samples\visdata\zoom.frm
compress.ex_  samples           vb\setupkit\kitfiles\compress.exe
compress.tx_  samples           vb\setupkit\kitfiles\compress.txt
expand.ex_    samples           vb\setupkit\kitfiles\expand.exe
setupa.ex_    samples           vb\setupkit\kitfiles\setup.exe
setupa.ls_    samples           vb\setupkit\kitfiles\setup.lst
setupkit.dl_  samples           vb\setupkit\kitfiles\setupkit.dll
setupwiz.ex_  samples           vb\setupkit\kitfiles\setupwiz.exe
setupwiz.hl_  samples           vb\setupkit\kitfiles\setupwiz.hlp
setupwiz.in_  samples           vb\setupkit\kitfiles\setupwiz.ini
code.ma_      tutorial          vb\vb.cbt\code.mak
country.ma_   tutorial          vb\vb.cbt\country.mak
form1.fr_     tutorial          vb\vb.cbt\form1.frm
form2.fr_     tutorial          vb\vb.cbt\form2.frm
formchi.fr_   tutorial          vb\vb.cbt\formchi.frm
formchi1.fr_  tutorial          vb\vb.cbt\formchi.frx
mdinpad.fr_   tutorial          vb\vb.cbt\mdinpad.frm
mdinpad1.fr_  tutorial          vb\vb.cbt\mdinpad.frx
payment.fr_   tutorial          vb\vb.cbt\payment.frm
payment.ma_   tutorial          vb\vb.cbt\payment.mak
sweden.fr_    tutorial          vb\vb.cbt\sweden.frm
sweden1.fr_   tutorial          vb\vb.cbt\sweden.frx
vb.le_        tutorial          vb\vb.cbt\vb.les
gsw.ex_       controls          windows\system\gsw.exe
gswdll.dl_    controls          windows\system\gswdll.dll
winhelp.ex_   always            windows\winhelp.exe

***** DISK5 ****

bullet.bm_    help compiler     vb\hc\bullet.bmp
emdash.bm_    help compiler     vb\hc\emdash.bmp
hc31.er_      help compiler     vb\hc\hc31.err
helpref.hl_   help compiler     vb\hc\helpref.hlp
iconwrks.bm_  help compiler     vb\hc\iconwrks.bmp
iconwrks.hp_  help compiler     vb\hc\iconwrks.hpj
iconwrks.ph_  help compiler     vb\hc\iconwrks.ph
iconwrks.rt_  help compiler     vb\hc\iconwrks.rtf
iwedit.sh_    help compiler     vb\hc\iwedit.shg
mrbc.ex_      help compiler     vb\hc\mrbc.exe
shed.ex_      help compiler     vb\hc\shed.exe
shed.hl_      help compiler     vb\hc\shed.hlp
track.do_     help compiler     vb\hc\track.doc
winhelp.tx_   help compiler     vb\hc\winhelp.txt
```

3dlrsign.wm_	clip art	vb\metafile\business\3dlrsign.wmf
alphbord.wm_	clip art	vb\metafile\business\alphbord.wmf
alphtrpn.wm_	clip art	vb\metafile\business\alphtrpn.wmf
answmach.wm_	clip art	vb\metafile\business\answmach.wmf
apptbook.wm_	clip art	vb\metafile\business\apptbook.wmf
calcultr.wm_	clip art	vb\metafile\business\calcultr.wmf
calendar.wm_	clip art	vb\metafile\business\calendar.wmf
cent.wm_	clip art	vb\metafile\business\cent.wmf
check.wm_	clip art	vb\metafile\business\check.wmf
clipbord.wm_	clip art	vb\metafile\business\clipbord.wmf
coins.wm_	clip art	vb\metafile\business\coins.wmf
computer.wm_	clip art	vb\metafile\business\computer.wmf
copymach.wm_	clip art	vb\metafile\business\copymach.wmf
deutsch.wm_	clip art	vb\metafile\business\deutsch.wmf
digitals.wm_	clip art	vb\metafile\business\digitals.wmf
digitnum.wm_	clip art	vb\metafile\business\digitnum.wmf
dime.wm_	clip art	vb\metafile\business\dime.wmf
disk35.wm_	clip art	vb\metafile\business\disk35.wmf
disk525.wm_	clip art	vb\metafile\business\disk525.wmf
dollar.wm_	clip art	vb\metafile\business\dollar.wmf
dollars.wm_	clip art	vb\metafile\business\dollars.wmf
envlback.wm_	clip art	vb\metafile\business\envlback.wmf
envlfrnt.wm_	clip art	vb\metafile\business\envlfrnt.wmf
fileclsd.wm_	clip art	vb\metafile\business\fileclsd.wmf
fileopen.wm_	clip art	vb\metafile\business\fileopen.wmf
guilder.wm_	clip art	vb\metafile\business\guilder.wmf
harddisk.wm_	clip art	vb\metafile\business\harddisk.wmf
laptop1.wm_	clip art	vb\metafile\business\laptop1.wmf
laptop2.wm_	clip art	vb\metafile\business\laptop2.wmf
micrchip.wm_	clip art	vb\metafile\business\micrchip.wmf
money.wm_	clip art	vb\metafile\business\money.wmf
moneybag.wm_	clip art	vb\metafile\business\moneybag.wmf
monitor.wm_	clip art	vb\metafile\business\monitor.wmf
monystk1.wm_	clip art	vb\metafile\business\monystk1.wmf
monystk2.wm_	clip art	vb\metafile\business\monystk2.wmf
nickel.wm_	clip art	vb\metafile\business\nickel.wmf
payphone.wm_	clip art	vb\metafile\business\payphone.wmf
pcomputr.wm_	clip art	vb\metafile\business\pcomputr.wmf
penny.wm_	clip art	vb\metafile\business\penny.wmf
peseta.wm_	clip art	vb\metafile\business\peseta.wmf
phone.wm_	clip art	vb\metafile\business\phone.wmf
postcard.wm_	clip art	vb\metafile\business\postcard.wmf
pound.wm_	clip art	vb\metafile\business\pound.wmf
poundbag.wm_	clip art	vb\metafile\business\poundbag.wmf
printer.wm_	clip art	vb\metafile\business\printer.wmf
prntout1.wm_	clip art	vb\metafile\business\prntout1.wmf
prntout2.wm_	clip art	vb\metafile\business\prntout2.wmf
prntout3.wm_	clip art	vb\metafile\business\prntout3.wmf
quarter.wm_	clip art	vb\metafile\business\quarter.wmf
rolodex.wm_	clip art	vb\metafile\business\rolodex.wmf
ruble.wm_	clip art	vb\metafile\business\ruble.wmf
satedish.wm_	clip art	vb\metafile\business\satedish.wmf
satelit1.wm_	clip art	vb\metafile\business\satelit1.wmf
satelit2.wm_	clip art	vb\metafile\business\satelit2.wmf
typewrtr.wm_	clip art	vb\metafile\business\typewrtr.wmf
yen.wm_	clip art	vb\metafile\business\yen.wmf
odbc.in_	data access	vb\odbc\odbc.inf

```
odbc.od_       data access         vb\odbc\odbc.dl_
odbcadm.ex_    data access         vb\odbc\odbcadm.ex_
odbcinst.od_   data access         vb\odbc\odbcinst.dll
odbsetup.ex_   data access         vb\odbc\setup.exe
odb_mssu.ex_   data access         vb\odbc\_mssetup.exe
odbsetup.ls_   data access         vb\odbc\setup.lst
odbcstp.ex_    data access         vb\odbc\odbcstp.exe
commdlg.od_    data access         vb\odbc\commdlg.dll
ctl3d.od_      data access         vb\odbc\ctl3d.dll
lzexpand.dl_   data access         vb\odbc\lzexpand.dll
ver.od_        data access         vb\odbc\ver.dll
ora6win.od_    oracle driver       vb\odbc\ora6win.dl_
oracle.tx_     oracle driver       vb\odbc\oracle.tx_
orasetup.od_   oracle driver       vb\odbc\orasetup.dl_
sqora.od_      oracle driver       vb\odbc\sqora.dl_
drvoracl.hl_   oracle driver       vb\odbc\drvoracl.hl_
odbcinst.hl_   data access         vb\odbc\odbcinst.hl_
mscpxlt.od_    data access         vb\odbc\mscpxlt.dl_
10070220.cp_   data access         vb\odbc\10070220.cp_
10070437.cp_   data access         vb\odbc\10070437.cp_
10070850.cp_   data access         vb\odbc\10070850.cp_
10070860.cp_   data access         vb\odbc\10070860.cp_
10070861.cp_   data access         vb\odbc\10070861.cp_
10070863.cp_   data access         vb\odbc\10070863.cp_
10070865.cp_   data access         vb\odbc\10070865.cp_
msole2.vb_     controls            windows\system\msole2.vbx
crystal.vb_    controls            windows\system\crystal.vbx
gauge.vb_      controls            windows\system\gauge.vbx
keystat.vb_    controls            windows\system\keystat.vbx
mci.vb_        controls            windows\system\mci.vbx
mscomm.vb_     controls            windows\system\mscomm.vbx
msmapi.vb_     controls            windows\system\msmapi.vbx
msmasked.vb_   controls            windows\system\msmasked.vbx
msoutlin.vb_   controls            windows\system\msoutlin.vbx
picclip.vb_    controls            windows\system\picclip.vbx
spin.vb_       controls            windows\system\spin.vbx
win30api.tx_   windows api         vb\winapi\win30api.txt
win31api.hl_   windows api         vb\winapi\win31api.hlp
win31ext.tx_   windows api         vb\winapi\win31ext.txt
winmmsys.tx_   windows api         vb\winapi\winmmsys.txt

***** DISK6 ****

msafinx.dl_    visual basic        windows\system\msafinx.dll
compobj.dl_    ole                 windows\system\compobj.dll
ole2nls.dl_    ole                 windows\system\ole2nls.dll
ole2.dl_       ole                 windows\system\ole2.dll
ole2.re_       ole                 windows\system\ole2.reg
ole2conv.dl_   ole                 windows\system\ole2conv.dll
ole2prox.dl_   ole                 windows\system\ole2prox.dll
ole2disp.dl_   ole                 windows\system\ole2disp.dll
storage.dl_    ole                 windows\system\storage.dll
crxlate.dl_    report writer       windows\system\crxlate.dll
ctl3d.cr_      report writer       windows\system\ctl3d.dll
p3conv.dl_     report writer       windows\system\p3conv.dll
p3dib.dl_      report writer       windows\system\p3dib.dll
p3file.dl_     report writer       windows\system\p3file.dll
```

```
p3info.dl_    report writer       windows\system\p3info.dll
pdbjet.dl_    report writer       windows\system\pdbjet.dll
pdctjet.dl_   report writer       windows\system\pdctjet.dll
pdirjet.dl_   report writer       windows\system\pdirjet.dll
pdsodbc.dl_   report writer       windows\system\pdsodbc.dll
msabc110.dl_  report writer       windows\system\msabc110.dll
commdlg.dl_   visual basic        windows\system\commdlg.dll
ddeml.dl_     visual basic        windows\system\ddeml.dll
vbrun300.dl_  visual basic        windows\system\vbrun300.dll
cmdialog.vb_  visual basic        windows\system\cmdialog.vbx
crpe.dl_      controls            windows\system\crpe.dll
grid.vb_      visual basic        windows\system\grid.vbx

***** DISK7 ****

autopro.ma_   controls            vb\autoload.mak
bright.di_    visual basic        vb\bright.dib
datacons.tx_  visual basic        vb\datacons.txt
msajt110.dl_  data access         windows\system\msajt110.dll
msaes110.dl_  data access         windows\system\msaes110.dll
vbdb300.dl_   data access         windows\system\vbdb300.dll
demo.ex_      controls            vb\demo.exe
pastel.di_    visual basic        vb\pastel.dib
proconst.tx_  visual basic        vb\constant.txt
rainbow.di_   visual basic        vb\rainbow.dib
samples.tx_   visual basic        vb\samples.txt
ctrlref.hl_   controls            vb\ctrlref.hlp
vb.ex_        visual basic        vb\vb.exe

***** DISK8 ****

a_to_b.md_    report writer       vb\report\a_to_b.mdb
crystal.md_   report writer       vb\report\crystal.mdb
empdata.md_   report writer       vb\report\empdata.mdb
crw.ex_       report writer       vb\report\crw.exe
crw.hl_       report writer       vb\report\crw.hlp
crw.ne_       report writer       vb\report\crw.net
labels.tx_    report writer       vb\report\labels.txt
crvbxsam.ma_  report writer       vb\report\crvbxsam.mak
crvbxsam.fr_  report writer       vb\report\crvbxsam.frm
form2a.fr_    report writer       vb\report\form2.frm
form3.fr_     report writer       vb\report\form3.frm
vbxmdb.rp_    report writer       vb\report\vbxmdb.rpt

***** DISK9 ****

msolevbx.dl_  controls            windows\system\msolevbx.dll
vb.hl_        visual basic        vb\vb.hlp
anibuton.vb_  controls            windows\system\anibuton.vbx
graph.vb_     controls            windows\system\graph.vbx
pencntrl.vb_  controls            windows\system\pencntrl.vbx
threed.vb_    controls            windows\system\threed.vbx

Additional reference words: 3.00
KBCategory:
KBSubcategory: RefsDoc
```

Developer Services Offers Solution Provider Packages

Article ID: Q100781

The information in this article applies to:

- Microsoft Visual Basic for Windows, version 3.0

SUMMARY

Solution Provider Services is a new package that customers can buy to get technical help. This package is sold through Developer Services.

MORE INFORMATION

For more information on the Solution Provider packages or to purchase the package, call Developer Services at 1-800-227-4679 and ask to speak to someone about the "Solution Provider" packages.

Customers who already have the Solution Provider package can use it by calling 1-800-227-4679 followed by extension 11700 and then their five-digit member number for technical support.

Additional reference words: 3.00
KBCategory:
KBSubcategory: RefsProd

How to Get Entire VB KB in 2 Help Files with Full-Text Search

Article ID: Q105541

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 2.0 and 3.0

SUMMARY

You can get the complete Microsoft Visual Basic for Windows Knowledge Base in two help files with (VBKB_FT.EXE) or without (VBKB.EXE) full-text search. These help files will be updated regularly. The latest versions were created in February 1994.

- VBKB_FT.EXE is a collection of indexed Help files and dynamic link libraries (.DLL files) that give you the complete Microsoft Visual Basic for Windows Knowledge Base in two Help files with full-text search.

- VBKB.EXE is the same collection of articles without full-text search.

MORE INFORMATION

Contents of VBKB_FT.EXE with Full-Text Search S14447

- README.TXT

- VB_BUGS.HLP (latest versions of the articles that discuss bugs, fixes, or updates to Visual Basic for Windows)

- VB_BUGS.IND (index required for VB_BUGS.HLP for full-text search)

- VB_TIPS.HLP (latest versions of the articles that give tips and techniques for Visual Basic for Windows)

- VB_TIPS.IND (index required for VB_TIPS.HLP for full-text search)

- FTENGINE.DLL (required for full-text search -- put in WINDOWS directory)

- FTUI.DLL (required for full-text search -- put in WINDOWS directory)

- MVAPI.DLL (required for full-text search -- put in WINDOWS directory)

Contents of VBKB.EXE without Full-Text Search S14347

- README.TXT

- VB_BUGS.HLP (latest versions of the articles that discuss bugs, fixes, or updates to Visual Basic for Windows)

- VB_TIPS.HLP (latest versions of the articles that give tips and techniques for Visual Basic for Windows)

MORE INFORMATION
================

There are over 600 categorized articles in the Visual Basic for Windows collection. The two help files that hold these articles have been placed in a self-extracting file that you can download from several different places (listed below). Choose to download either VBKB_FT.EXE (the full-text search version) or VBKB.EXE (the version without full-text search).

The Help files in VBKB_FT.EXE have an additional Find button that allows full-text search. The Help files in VBKB.EXE do not have the Find button and do not allow full-text search. The technical content in VBKB.EXE is identical to that in VBKB_FT.EXE. VBKB.EXE is less than a megabyte in size while VBKB_FT.EXE is approximately 2.5 megabytes. VBKB_FT.EXE is larger because it includes index and .DLL files needed for full-text search.

To obtain the Help files, download VBKB.EXE or VBKB_FT.EXE. Then run it in an empty directory to extract the files.

- Download VBKB_FT.EXE -- if you want to use full-text search to query the Microsoft Visual Basic Knowledge Base. The Help files in this package include a Find button that allows you to search the Microsoft Knowledge Base for any word you choose.

- Download VBKB.EXE if you want a smaller package and don't need full-text search. The Help files in this package have only the Search button, which allows you to search for article Q numbers (the number that identifies each Microsoft Knowledge Base article).

Where to Find VBKB.EXE and VBKB_FT.EXE

Download either VBKB.EXE or VBKB_FT.EXE (both are self-extracting files) from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for and download VBKB.EXE
  -or-
  Search for and download VBKB_FT.EXE

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download VBKB.EXE or VBKB_FT.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get VBKB.EXE
  -or-
  Get VBKB_FT.EXE

After downloading either VBKB.EXE or VBKB_FT.EXE, run it to extract the files it contains.

Send Comments and Corrections to y-KBFeed@Microsoft.Com

If you have corrections, comments, or feedback on any of the articles in the Visual Basic for Windows collection, please send the Q number of the article along with your comments in a personal electronic mail message to y-KBFeed@Microsoft.Com on the Internet. You can do this in CompuServe Mail by putting the following on the TO: line of your message:

   >Internet: y-KBFeed@Microsoft.Com

In the text, please ask that your message be given to the Visual Basic Knowledge Base Lead (KBL). Please send mail to the same address if you have feedback on how we could improve the Microsoft Knowledge Base or if you would like to contribute articles.

Additional reference words: 2.00 3.00 softlib on-line
KBCategory:
KBSubcategory: RefsProd

How to Write C DLLs and Call Them from Visual Basic

Article ID: Q106553

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article outlines how to use DLLs with Visual Basic. It covers the following issues:

Section A

1.0 What Is a DLL
1.1 Why Use a DLL
1.2 Anatomy of a DLL
1.3 DLL Memory Management Issues
1.4 Building a DLL Using Visual C++
1.5 Example C DLL

Section B

2.0 Calling DLLs from Visual Basic
2.1 DLL Parameters
2.2 Trouble Shooting
2.3 Example Visual Basic Calling Program

MORE INFORMATION

SECTION A

1.0 What Is a DLL

DLLs (Dynamic Link Libraries) are an important aspect of Windows. A DLL contains functions that your executable program can call during execution. In other words, a DLL is a library of functions that your program can link with dynamically.

A link can be static or dynamic. Static links don't change. All the address information needed by your program to access the library function is fixed when the executable file is created and remains unchanged during execution.

Dynamic links are created as needed. When your program needs a function that is not in the executable file, Windows loads the dynamic link library (the DLL), making all of its functions available to your application. At that time, Windows resolves the address of each function and dynamically

links it to your application.

All Custom controls used in Visual Basic are DLLs. The only difference is that they require special handling in terms of messages received from Visual Basic.

1.1 Why Use DLLs

Here are four reasons why you might want to use a DLL:

- Access to C Run-Time Functions:

  The C run-time library has many useful functions that would not be available to Visual Basic programmers were it not for DLLs. For example, the _dos_getdiskfree function allows you to calculate the total amount of disk space and the free disk space available on a drive.

- Access to Windows API (Application Programming Interface) Functions that Require Callback Routines:

  Some Windows API functions require a callback function. A callback function is a function that Windows will call while executing the API call. An example of this sort of function is EnumTaskWindows, which will give the handle of all windows that are owned by a particular task.

- Speed:

  C is a fully compiled language that works at a level that is fairly close to native machine code. This means that the execution of programs that are well written in C will be fast.

- Load on Use:

  Code and data from a DLL are loaded only when needed. A DLL can be organized such that only required parts are loaded as opposed to the entire DLL. This reduces the amount of memory required and the time taken to load.

1.2 Anatomy of a DLL

Every DLL must contain a LibMain function and should contain a Windows Exit Procedure (WEP) in addition to the exported functions that can be called by an executable program.

- LibMain:

  A DLL must contain the LibMain function. The LibMain function is called by the system to initialize the DLL. LibMain is called only once -- when the first program that requires the DLL is loaded. The following are the parameters passed to LibMain:

  - HANDLE : Handle to the instance of the DLL.
  - WORD : Library's data segment.
  - WORD : Heap size.
  - LPSTR : Command line parameters.

- WEP:

  The WEP (Windows Exit Procedure) performs cleanup for a DLL before the library is unloaded. Although a WEP function was required for every DLL in previous versions of the Windows operating system, for version 3.1 it is optional. A WEP should be included in the module definition file (.DEF) in Visual C, for example:

  ```
  EXPORTS
     WEP
  ```

- Exported Functions:

  These are the functions you want to call from your DLL. They are denoted by _export. _export is used for backward compatibility. All the functions you want to call must also be listed in the (.DEF) file of your DLL.

1.3 DLL Memory management issues

Use the large memory model.

C stores all variables defined as static or global (defined outside of a function) in the program's heap space, and C stores all other variables on the stack.

In the small and medium model, all pointers are near by default. This means that the data is accessed by 16-bit offsets to either the data segment (DS) register, or the stack segment (SS) register. Unfortunately, the compiler has no way of knowing whether the offset is from the DS or the SS. In most programs this would not be a problem because the DS and SS point to the same segment. A DLL, however, is a special case.

A DLL has its own data segment but shares its stack with the calling program. This means that the DS and the SS do not point to the same location. The easiest solution to this problem is to build the DLL in the large memory model where all variables are referenced by a 32-bit value.

Why Allocate Memory Dynamically?

Allocating memory dynamically is a Windows-friendly technique. Declaring large arrays of data takes up space in either your program's stack, which is limited to 64K, or you program's Data Segment, which wastes disk space and Windows memory. It is better to ask Windows for the memory when you need it, and then free it when you have finished.

Allocating Memory

In Windows, you can dynamically allocate two types of memory, local and global. Local memory is limited to 64K, and in the case of a DLL, local memory is shared with the program that called the DLL. Global memory is all of the memory available to Windows after it has loaded.

Local memory is allocated and managed using the LocalAlloc, LocalLock LocalUnlock, and LocalFree functions -- as in this example:

```
char* pszBuffer;
....
pszBuffer = (char *) LocalAlloc (LPTR, 20);
...
LocalFree (pszBuffer);
```

It is faster to allocate local memory than it is to allocate global memory. But allocations from the local heap are limited to 64K, which must be shared amongst all programs that are calling the DLL. It is best to use local memory when small, short lived blocks of memory are required.

Global memory is allocated and managed using the GlobalAlloc, GlobalLock GlobalUnlock, and GlobalFree functions -- as in this example:

```
HGLOBAL hglb;
char* pszBuffer;

hglb = GlobalAlloc (GHND, 2048);
   // GHND allocates the memory as moveable and
   // initialized to 0
   // 2048 is the amount of memory to be allocated...
pszBuffer = GlobalLock (hglb);
...
GlobalUnlock (hglb);
GlobalFree (hglb);
```

The GlobalAlloc function allocates memory in multiples of 4K.

If you want to share memory allocated in the DLL with other programs, you should allocate it using the GMEM_SHARED flag. If you want to share the memory through DDE, you must allocate it by using the GMEM_DDESHARE flag.

Be Careful When Storing Data in Static Variables

If you try to store data in a DLL using global or static variables, don't be surprised if these values have changed when you next call your DLL. The data stored in this way will be common to all applications that access this DLL. No matter how many applications use a DLL, there is only one instance of the DLL. The best way to get around this is to return structures from the DLL and pass them in again when they are needed.

File Handles

It is not possible to share file handles between applications or DLLs. Each application has its own file-handle table. For two applications to use the same file using a DLL, they must both open the file individually.

1.4 Building a DLL Using Visual C++

Here are the steps necessary to build a DLL using Visual C++:

1. Start Visual C++.

2. Create a new project by choosing New from the Project menu. Select the following options:

   - Set the Project Type to "Windows dynamic-link library (.DLL)"
   - Clear the "Use Microsoft Foundation Classes" check box.

   You can also set or view these options later by choosing Project from the Options menu.

3. Add your existing .C and .DEF files to the project by using the dialog box that comes up when you choose Edit from the Project menu. Or enter your code directly in the Visual C++ editing window. (See the .C and .DEF example code listed below.)

4. From the Project menu, choose the Build <yourname>.DLL option.

1.5 Example C DLL

The following DLL contains the GetDiskInfo function, which can be called from Visual Basic. It will return the disk space available, the current drive name and the volume name.

C Code Example, DISKINFO.C:

```
#include <windows.h>
#include <dos.h>

int CALLBACK LibMain (HANDLE hInstance, WORD wDataSeg, WORD wHeapSize,
LPSTR lpszCmdLine)
{
   if (wHeapSize > 0)
      UnlockData (0);  //Unlocks the data segment of the library.
   return 1;
}

void __export CALLBACK GetDiskInfo (char *cDrive, char *szVolumeName,
unsigned long *ulFreeSpace)
{
   unsigned drive;
   struct _diskfree_t driveinfo;
   struct _find_t c_file;

   _dos_getdrive (&drive);
   _dos_getdiskfree( drive, &driveinfo );

   if (!_dos_findfirst( "*.*", _A_VOLID, &c_file ))
      wsprintf( szVolumeName, "%s", c_file.name);
   else
      wsprintf ( szVolumeName, "NO LABEL");

   *cDrive = drive + 'A' -1;

   *ulFreeSpace = (unsigned long) driveinfo.avail_clusters * (unsigned
      long) driveinfo.sectors_per_cluster * (unsigned long)
```

```
      driveinfo.bytes_per_sector;
}
```

Use the following DISKINFO.DEF file in Visual C++:

```
   LIBRARY  diskinfo
   DESCRIPTION 'GetDiskInfo Can be called from Visual Basic'
   EXETYPE WINDOWS 3.1
   CODE PRELOAD MOVEABLE DISCARDABLE
   DATA PRELOAD MOVEABLE SINGLE
   HEAPSIZE    4096
   EXPORTS
      GetDiskInfo @1
```

NOTE: The LIBRARY name in the .DEF file must be the same as the DLL file name, or else Visual Basic will give you "Error in loading DLL." For example, create the file DISKINFO.DLL using the LIBRARY DISKINFO statement in the .DEF file above.

SECTION B

2.0 Calling DLLs from Visual Basic

In Visual Basic, all functions, including DLL functions, that you want to call must first be declared by using the Declare statement. You can declare your functions in the declarations section of a Form or a Module. If you declare a DLL procedure or function in a Form, it is private to that Form. To make it public, you must declare it in a Module. The following is an example Declare statement:

```
   Declare Sub getdiskinfo Lib "c:\somepath\diskinfo.dll"
      (ByVal mydrive As String, ByVal myvolume As String, free As Long)
```

You must enter the entire Declare statement as one, single line. This particular Declare statement declares the user-defined procedure GETDISKINFO located in user-created DISKINFO.DLL file.

Once you declare the function, you can call and use the function just as you would call and use a Visual Basic function.

2.1 DLL Parameters

Because DLLs are typically written in C, DLLs can use a wide variety of parameters not directly supported by Visual Basic. As a result, when passing parameters, he programmer has to find the appropriate data type to pass.

Passing Arguments by Value or by Reference

By default, Visual Basic passes all arguments by reference. (When passing by reference, Visual Basic supplies a 32-bit far address.) However, many DLL functions expect an argument to be passed by value. This can be achieved by placing the ByVal keyword in front of the argument declaration.

The following sections show you how to convert parameters to Visual Basic.

8- to 16-Bit Numeric Parameters

Pass 8- to 16-bit numeric parameters (int, short, unsigned int, unsigned short, BOOL, and WORD) as Integer.

32-bit Numeric Parameters

Pass 32-bit numeric parameters (long, unsigned long, and DWORD) as LONG.

32-Bit Signed Integer Parameters

Pass 32-bit signed integer parameters as Currency or Double.

Object Handles

All handles are unique 16-bit integer values associated with a Window and are passed by value, so pass these parameters as Integer.

Strings

Strings include the LPSTR and LPBYTE data types (pointer to characters or pointer to unsigned characters). Pass these parameters as (ByVal paramname As String). DLL functions cannot return Visual Basic strings. They do sometimes return LPSTRs, which can be copied into Visual Basic strings by using API functions.

To pass Visual Basic strings directly, pass them as (param As String).

NOTE: Visual Basic strings require special handling, so don't pass strings directly unless the DLL explicitly requires it.

Pointers to Numeric Values

Pass pointers to numeric values by simply not using the ByVal keyword.

Structures

If the Visual Basic user-defined type matches the structure expected by the DLL, the structure can be passed by reference.

NOTE: Structures cannot be passed by value.

Pointers to Arrays

Pass the first element of the array by reference.

Pointers to functions

Visual Basic does not support callback functions, so DLL functions that have pointers to functions cannot be used with Visual Basic.

Null Pointers

If a DLL expects a Null pointer, pass it as (ByVal paramname As Any). You can use &0 or &0H as the value of paramname.

2.2 Trouble Shooting

Below are solutions to some problems you may encounter.

System Resources Keep Getting Lower After the DLL Is Called

If your DLL is using GDI objects, you must remember to free them after using them. This may not be obvious in Visual Basic, but when using the Windows SDK (software development kit) if you create a GDI object (for example, CreateBrushIndirect), you must delete it by using DeleteObject later on.

Bad DLL Calling Convention Error

This error is often caused by incorrectly omitting or including the ByVal keyword from the Declare statement. This error can also be caused if the wrong parameters are passed.

Error in loading DLL

This error occurs when you call a dynamic-link library procedure and the file specified in the procedure's Declare statement cannot be loaded. You can use the Microsoft Windows API function LoadLibrary to find out more specific information about why a DLL fails to load.

General Protection (GP) Fault

GP faults occur when your program writes to a block of memory that doesn't belong to it. The two most likely reasons for this are:

- You overstepped an array boundary. C does not check that the array subscript you are writing to is valid. Therefore, you can easily write to memory you don't own.

- You are using a pointer to a memory location that you have freed. The best option is to assign NULL to all pointers after you free their memory.

A GP fault can also occur when an incorrect variable type is passed to the DLL function.

2.3 Example Visual Basic Calling Program

There are two parts to calling a DLL in a Visual Basic program. First you declare the function, and then you use it in event code.

Here is an example of a Declare statement. The Declare statement should be put in a module or in a form's General Declarations section.

```
   ' Enter the following Declare as one, single line:
   Declare Sub getdiskinfo Lib "c:\dllartic\diskinfo.dll"
      (ByVal mydrive As String, ByVal myvolume As String, free As Long)
```

Specify ByVal statements exactly as shown, or else a GP fault may occur.

Once the function is declared, you can use it in event code. The following example uses a function from the DLL in the Command1 Click event code:

```
Sub Command1_Click ()
   Dim drive As String * 1
   Dim volume As String * 20
   Dim free As Long
   Call getdiskinfo(drive, volume, free)
   Text1.Text = drive
   Text2.Text = volume
   Text3.Text = Str$(free)
End Sub
```

Additional reference words: 3.00
KBCategory:
KBSubCategory: APrgOther RefsDoc

LONG: VB Pro 3.0 SAMPLES.TXT: Descriptions of Sample Programs

Article ID: Q107990

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY
=======

The following article contains the complete contents of the SAMPLES.TXT file distributed with the Professional Edition of Visual Basic version 3.0 for Windows. SAMPLES.TXT describes the sample programs shipped with this product.

MORE INFORMATION
================

SAMPLES.TXT

Samples Notes for Microsoft (R) Visual Basic (R)
Professional Edition

Version 3.00

========
Contents
========

Sample	Description
1	MAPI
2	Multimedia
3	Outline
4	Pen
5	PicClip
6	ODBC
7	Communications

===============
1: MAPI Sample
===============

Program example using the MSMAPI.VBX controls, MapiMessages and MapiSession controls.

NOTE: To run this sample, you need a MAPI-compliant messaging system such as the one contained in Microsoft Windows for Workgroups.

BRIEF DESCRIPTION:
This sample program illustrates the use of the MAPI controls by

sending and receiving electronic mail.

BACKGROUND:
The MapiSession control establishes a MAPI session and signs off from a MAPI session. The MapiMessages control allows you to perform a variety of messaging systems functions after a messaging session has been established. These functions include accessing, downloading, and sending messages, displaying the details and address book dialog boxes, accessing data attachments, resolving recipient names during addressing, and performing compose, reply, reply-all, forward, and deleting actions on messages.

FILES:
   MAILLST.FRM.......Form for displaying the list of mail messages.
   MAILLST.FRX.......The binary portion of the mail messages form.
   MAILOPTF.FRM......Form for setting options.
   MAILSUP.BAS.......Module of support routines for the example.
   MSGVIEW.FRM.......Form in which a viewed message is displayed.
   MSGVIEW.FRX.......The binary portion of the viewed messages form.
   NEWMSG.FRM........Form for creating a new message.
   VBMAIL.FRM........The main form.
   VBMAIL.FRX........The binary portion of the main form.
   VBMAIL.MAK........The MAKE file for the project.

TO RUN:
After starting the Visual Basic environment (VB.EXE), you can load files in this sample program by choosing Open Project from the File menu and selecting the VBMAIL.MAK file.

==========
2: Multimedia Sample
==========

This sample program illustrates the use of some of the device types supported by the Media Control Interface (MCI). The four device types used in this program are:

Device Type	Description
MMMovie	Plays Multimedia movie files (*.MMM).
CDAudio	Plays audio discs from the CD-ROM drive.
Sequencer	Plays MIDI sequencer sound files (*.MID).
WaveAudio	Plays digitized waveform sound files (*.WAV).

HARDWARE REQUIREMENTS:
In order to access a CD-ROM drive or a sound card, the hardware, along with any supporting device drivers, should be installed and configured for the machine. The following table lists the hardware required for the MCI device types used in this program:

Device Type	Hardware
MMMovie	No extra hardware. *See software requirements.
CDAudio	CD-ROM drive.
Sequencer	Sound card (e.g., Sound Blaster Pro)
WaveAudio	Sound card.

SOFTWARE REQUIREMENTS:
In order to run the MCI application, Microsoft Windows with Multimedia Extensions 1.0, or Windows with Multimedia must be installed. The Multimedia Extensions include device drivers for the different types of MCI devices.

To access a specific MCI device type, the corresponding MCI device driver must be installed. The following table lists the device drivers required for the MCI device types used in this program:

Device Type	Device Driver	
MMMovie	MCIMMP.DRV	*See note below.
CDAudio	MCICDA.DRV	
Sequencer	MCISEQ.DRV	
WaveAudio	MCIWAVE.DRV	

The [mci] section of the SYSTEM.INI file contains a list of the installed MCI device types. This is what your file would look like if all 4 of the above device types were installed:

```
[mci]
CDAudio=mcicda.drv
WaveAudio=mciwave.drv
Sequencer=mciseq.drv
MMMovie=mcimmp.drv
```

BACKGROUND:
The MCI control provides a high-level interface for using multimedia devices. Using the property settings of a control, you can determine the settings and the actions of a device.

The MCI control is also smart enough to know what actions are relevant for the current state of a device. For example, if you click a "Pause" button while playing a movie, the "Play" button is automatically re-enabled.

MMMovie NOTE:
If you are unable to run an animation movie (*.MMM file), make sure the MCIMMP.DRV and MMP.DLL files are installed on your machine. To install the animation driver from Windows 3.1, run the "Drivers" applet from the Control Panel, insert Disk 1 of the Professional Edition. Select "Unlisted Driver," and accept "A:\" as the directory. You will then be ready to use the animation features of the MCI control.

FILES:

ABOUTBOX.FRM	The dialog box for the "About Box."
ABOUTBOX.FRX	The binary portion of the "About Box" form.
ANIMATE.FRM	Form for playing Multimedia Movie files.
CD.FRM	The form for playing compact disc audio.
CD.FRX	The binary portion of the compact disc form.
GLOBAL.BAS	Global data types and declarations.
MCITEST.BAS	Global Subs and Functions.
MCITEST.FRM	The main form.
MCITEST.FRX	The binary portion of the main form.

```
   MCITEST.MAK........The make file for the project.
   MCITEST.MID........Sample MIDI sequencer file.
   MCITEST.MMM........Sample Multimedia Movie file.
   MCITEST.WAV........Sample waveform file.
   OPENDLG.FRM........Form for holding the common dialog control.
   WAVE.FRM...........Form for playing waveform and MIDI files.

TO RUN:
In the Visual Basic environment (VB.EXE), you can load the files
in this sample program by choosing Open Project from the File
menu, and selecting the MCITEST.MAK file.

Two additional files appear in the Project Window:

MCI.VBX............MCI control.
CMDIALOG.VBX.......Common Dialog control.

==================
3:  Outline Sample
==================
Program example using the MSOUTLIN.VBX control.

BRIEF DESCRIPTION:
This sample program illustrates the use of the Outline control
with an address book database application.

BACKGROUND:
The outline control allows users to graphically present hierarchically
structured data in a variety of different ways.

FILES:
   PHONE.FRM.........The main form.
   PHONE.FRX.........Binary data for the main form.
   PHONE.MAK.........The MAKE file for the project.
   PHONE.MDB.........Database file.
   PHONE.LDB.........Database file.
   FIRSTTAB.BMP......Bitmap for the project.
   LASTTAB.BMP.......Bitmap for the project.
   MIDTAB.BMP........Bitmap for the project.

TO RUN:
After starting the Visual Basic environment (VB.EXE), you can load
files in this sample program by choosing Open Project from the File
menu and select the PHONE.MAK file.

==============
4:  Pen Sample
==============

NOTE:  To run this sample you must have Windows for Pen Computing,
       or install the PENWIN.DLL file. See below.

BRIEF DESCRIPTION of PEN SAMPLE PROGRAM:
This sample program illustrates many uses of the Pen controls:

  * the new On-Screen Keyboard control
  * inking and displaying bitmaps on the new InkOnBitmap control
```

  * handwriting recognition in both delayed and non-delayed mode
  * recognition of gestures
  * transfer of ink data between controls
  * manipulation of the Recognition Context data structure
  * and more.

The file PENAPI.TXT contains function and constant declarations
for the entire set of Windows for Pen Computing API.

FILES:
   DELAY.FRM.........Form for demonstrating Delayed Recognition.
   DELAY.FRX.........The binary for DELAY.FRM.
   EDITSUBF.FRM......Support Form for Gesture demonstration.
   GESTFRM.FRM.......Form for demonstrating Custom Gestures.
   INKFRM.FRM........Form for showing transfer of Ink data.
   IOBFRM.FRM........Form demonstrating the Ink On Bitmap control.
   IOBFRM.FRX........The binary for IOBFRM.FRM.
   KEYBRD.FRM........Form for the On-Screen Keyboard Button demo.
   KEYBRD.FRX........The binary for KEYBRD.FRM.
   PENAPI.TXT........Pen function declarations and constants.
   PENMAIN.FRM.......Main form.
   PENMAIN.FRX.......The binary for MAIN.FRM.
   PENSMPL.MAK.......The MAKE file for the project.
   RCFRM.FRM.........Form demonstrating the Recognition Context.
   GRAFPAPR.BMP......Bitmap for creating graph paper effect.
   RULEPAPR.BMP......Bitmap for creating ruled paper effect.
   SKBFACE.BMP.......Bitmap of default keyboard on SKB Button.
   TRANSFRM.FRM......Form for demonstrating transfer of ink data.

TO RUN:
After starting the Visual Basic environment (VB.EXE), you can load
files in this sample program by choosing Open Project from the File
menu and select the PENSMPL.MAK file.

Installing PENWIN.DLL:
This procedure will result in a system that will let you run
applications that contain Visual Basic BEdit, HEdit, InkOnBitmap,
and SKB Button controls and call the Windows for Pen Computing APIs.
You will not be able to perform handwriting recognition or draw ink
on the screen. Microsoft Windows 3.1 is required.

SYSTEM.INI Changes
The following items must be added or changed in your SYSTEM.INI
file so that the pen extensions will work. NOTE: Back up your old
SYSTEM.INI file before proceeding.

1. In the "[boot]" section:
      Add "penwindows" to the list of drivers after "drivers=".
      For example:
        drivers = mmsystem.dll penwindows

2. In the "[drivers]" section:
      Add a new item "penwindows" and set it equal to the path to
      PENWIN.DLL.
      For example:
        penwindows = C:\WINDOWS\PENWIN.DLL

3. Restart Windows.

When Windows is restarted, PENWIN.DLL will be loaded as an installed driver, and you will be able to run applications containing Visual Basic BEdit, HEdit, InkOnBitmap, and SKB Button controls and call the Windows for Pen Computing APIs.

==========
5: PicClip Sample
==========

Program example using the PICCLIP.VBX control.

BRIEF DESCRIPTION:
This sample program illustrates one of the many possible uses of the Picture Clipping control.

BACKGROUND:
This sample application uses the PicClip control to spin a top.

FILES:
   INFOFORM.FRM......The information form.
   INFOFORM.FRX......The binary data for the information form.
   REDTOP.FRM........The main form.
   REDTOP.FRX........The binary data for the main form.
   REDTOP.MAK........The MAKE file for the project.

TO RUN:
After starting the Visual Basic environment (VB.EXE), you can load files in this sample program by choosing Open Project from the File menu and selecting the REDTOP.MAK file.

========
6: ODBC
========

Program example using ODBC and the VT (Virtual Table) object layer.

NOTE: To access ODBC data sources with this sample, you must first install ODBC using the ODBC setup program provided with Visual Basic Professional Edition.

BRIEF DESCRIPTION:
This sample program illustrates various programming techniques used to access data through the VT layer built into the Visual Basic Professional Edition. It behaves like a general purpose database utility capable of the following functions:

   1. Table Creation
   2. Table Modification (adding and deleting fields and indexes)
   3. Data Browsing/Modifying one record at a time using a dynaset or a table.
   4. Data Browsing via the Grid control (non-updatable)
   5. Data Browsing/Modifying via the new Data Control
   6. Data Export to Tab Delimited text file
   7. Direct SQL Statement execution for any SQL supported functions such as Insert, Update, Delete, Drop, Create,

Dump, etc.
8. AdHoc Query tool that helps users unfamiliar with SQL create complex queries with where clauses, joins, order by and group by expressions while limiting output to selected columns
9. Transaction Processing
10. Copying table structures and data to same or different server
11. Support of Microsoft Access, Dbase 3, Dbase 4, FoxPro 2.0, Paradox 3.x, Btrieve, SQL and Oracle data, both DDL and DML.

The code contains comments to help explain the use of the various methods in the data access layer. Code and forms may be copied from this application to other applications with minimal modification.

ODBC BACKGROUND:
ODBC (Open Database Connectivity) is a standard adopted by multiple vendors designed to enable users to connect to any data source with a single application. This is achieved through a layered approach including:

1. Programming Layer-embedded functions in the development tool which in this case is Visual Basic Professional.
2. Driver Manager-the basic ODBC library that routes calls to the appropriate driver.
3. Data Driver - the library of functions that acts upon a specific database backend such as SQL Server, Xbase, Excel, etc. (note that SQL Server is the first of many drivers to become available for ODBC)

These layers work together to enable data access from any source for which an ODBC driver exists. The sample application will work, without modification, on any new level one ODBC driver that becomes available. With multiple drivers, connections may be made to different data sources from the same application at the same time enabling seamless data access from disparate data sources.

FILES:
```
   ABOUTBOX.FRM......Standard "About box" for the application.
   ABOUTBOX.FRX......Icon for the "About Box".
   ADDFIELD.FRM......Form to add fields to Tables.
   CPYSTRU.FRM.......Form to copy Table structures.
   DATABOX.FRM.......General purpose list form.
   DYNAGRID.FRM......Form used to display data in a Grid control.
   DYNAGRID.FRX......Icon for DYNAGRID.FRM.
   DYNASET.FRM.......Form to display data in single record mode.
   DYNASET.FRX.......Icon for DYNASET.FRM
   FIND.FRM.........Form used to find records in a Dynaset.
   INDEXADD.FRM......Form used to add indexes to Tables.
   JOIN.FRM.........Form used to add Joins to the Query Builder.
   OPENDB.FRM........Form used to open a database.
   QUERY.FRM.........Form used to build Queries.
   QUERY.FRX.........Icon for QUERY.FRM.
   REPLACE.FRM.......Form to perform global replaces on a Table.
   REPLACE.FRX.......Icon for REPLACE.FRM.
   SEEK.FRM.........Form used to get input for Seek function on
                    Table form.
```

```
   SQL.FRM..........Form to enter and execute SQL statements.
   SQL.FRX..........Icon for SQL.FRM.
   TABLES.FRM.......Form used to display table lists.
   TABLES.FRX.......Icon for TABLES.FRM.
   TABLEOBJ.FRM.....Form used to display data in Table object
   TABLEOBJ.FRX.....Icon for TABLEOBJ.FRM
   TBLSTRU.FRM......Form to display and modify table structures.
   VDMDI.FRM........Main MDI form for the application.
   VDMDI.FRX........Icon for VDMDI.FRM.
   VISDATA.BAS......Support functions for the application.
   VISDATA.ICO......Icon for the applicaiton.
   VISDATA.MAK......Make file for applicaiton.
   ZOOM.FRM.........Form to zoom in on character data in the
                    dynaset forms.
```

TO RUN:
After starting the Visual Basic environment (VB.EXE), you can load files in this sample program by choosing Open Project from the File menu and selecting the VISDATA.MAK file in the SAMPLES\VISDATA directory.

To open a local database, choose the type of database and a file open common dialog will be provided with the file type set to the requested data file type.

If you choose ODBC from the File/Open menu, the next dialog you will see is the Open Database form. Because you probably have no servers entered, you will need to enter a name for an existing SQL server on your network. If you already know the user ID and password, you can add them as well. The Database name is optional. Once you have entered this data, select 'Okay' and you should be able to log on to the server. You may get some more dialogs in the process. Answer any questions you can and ask the SQL administrator for help if you run into problems or don't know some of the parameters.

Once a database is open, double-click a table name to open the table in the selected mode (Single Record or Table View). Use the Query Builder to create dynasets with selected data from one or more tables at a time. If "Use Data Control" or "Use Grid" is chosen, a dynaset will be created. However, the following objects will be created only under the following circumstances when the "No Data Control" option is selected:

Data Type	Feature Chosen	Object Type Created
Microsoft Access	Table Open	Table
Microsoft Access	Query Open	Dynaset
Microsoft Access	Execute SQL	Dynaset
ISAM	Table Open	Table
ISAM	Execute SQL	Dynaset
ODBC	Any	Dynaset

The table is always updatable and the dynaset will be updatable in most cases except on ODBC with no unique index, certain multiple table joins, and other SQL select statements such as count(*), max(), and so on.

7: Communications Samples

VBTERM Sample

VBTERM is a terminal emulation program example using MSCOMM.VBX.

BRIEF DESCRIPTION:
This sample program illustrates how to use the communications control with a serial port.

BACKGROUND:
MSCOMM.VBX allows you to open a serial port, change its settings, send and receive data through the port, and monitor and set many of the different data lines. It's dual-method access allows for both polling and event driven communications.

FILES:

- CANSEND.FRM.....Dialog box used during file transfer.
- TERMSET.FRM.....Form used to change the serial port settings.
- TERMSET.FRX.....Binary data for TERMSET.FRM.
- VBTERM.FRM.....The main form.
- VBTERM.FRX.....Binary data for VBTERM.FRM.
- VBTERM.GLO.....Global declarations.
- VBTERM.MAK.....The MAKE file for the project.

TO RUN:
After starting the Visual Basic environment (VB.EXE), you can load files in this sample program by choosing Open Project from the File menu and select the VBTERM.MAK file.

Dialer Sample

DIALER is a program example demonstrating how to dial out using MSCOMM.VBX with a modem.

FILES:

- DIALER.FRM.....The main form.
- DIALER.MAK.....The MAKE file for the project.

TO RUN:
After starting the Visual Basic environment (VB.EXE), you can load files in this sample program by choosing Open Project from the File menu and select the DIALER.MAK file.

Additional reference words: 3.00
KBCategory: Refs
KBSubcategory: RefsDoc

DOC: WinHelp Declaration Incorrect in Windows Ver 3.1 API Ref

Article ID: Q108036

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article corrects a documentation error for the WinHelp function call as described in the Windows version 3.1 API Reference help file that shipped with Microsoft Visual Basic version 3.0 for Windows.

MORE INFORMATION

The declaration is incorrectly shown as:

```
Declare Function WinHelp Lib "User" (ByVal hWnd As Integer,
                        ByVal lpHelpFile As String,
                        ByVal wCommand As Integer,
                        dwData As Any) As Integer
```

The correct declaration is as follows:

```
Declare Function WinHelp Lib "User" (ByVal hWnd As Integer,
                        ByVal lpHelpFile As String,
                        ByVal wCommand As Integer,
                        ByVal dwData As Any) As Integer
```

NOTE: Each Declare statement must be entered as one, single line.

Notice that the "ByVal" keyword was omitted from the last parameter in the online reference. This means that the function is passing the last parameter "dwData" by reference. It needs to be passed by value.

The most common error that occurs when using the incorrect declaration is a message box stating "Help topic does not exist."

Additional reference words: 3.00
KBCategory: Refs
KBSubcategory: RefsDoc

LONG: List of Trappable Errors for Visual Basic 3.0

Article ID: Q108340

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article lists error codes and messages for the errors that you can trap at run time by using the On Error statement and the Err function.

MORE INFORMATION

When running VB.EXE, you can follow these steps to get more information on a specific error.

1. Stop your program if it is currently running.

2. Press the F8 key to go from design mode to break mode.

3. Press Ctrl+B to activate the debug window.

4. Enter the following statement and wait for the error message to appear:

   Error <error-number>

5. Press the F1 key.

Trappable Errors

```
   3 Return without GoSub
   5 Illegal function call
   6 Overflow
   7 Out of memory
   9 Subscript out of range
  10 Duplicate definition
  11 Division by zero
  13 Type mismatch
  14 Out of string space
  16 String formula too complex
  17 Can't continue
  19 No Resume
  20 Resume without error
  28 Out of stack space
  35 Sub or Function not defined
  48 Error in loading DLL
  49 Bad DLL calling convention
  51 Internal error
  52 Bad file name or number
```

```
 53 File not found
 54 Bad file mode
 55 File already open
 57 Device I/O error
 58 File already exists
 59 Bad record length
 61 Disk full
 62 Input past end of file
 63 Bad record number
 64 Bad file name
 67 Too many files
 68 Device unavailable
 70 Permission denied
 71 Disk not ready
 74 Can't rename with different drive
 75 Path/File access error
 76 Path not found
 91 Object variable not Set
 92 For loop not initialized
 93 Invalid pattern string
 94 Invalid use of Null
 95 Cannot destroy active form instance
260 No timer available
280 DDE channel not fully closed; awaiting response from foreign
    application
281 No More DDE channels
282 No foreign application responded to a DDE initiate
283 Multiple applications responded to a DDE initiate
284 DDE channel locked
285 Foreign application won't perform DDE method or operation
286 Timeout while waiting for DDE response
287 User pressed Escape key during DDE operation
288 Destination is busy
289 Data not provided in DDE operation
290 Data in wrong format
291 Foreign application quit
292 DDE conversation closed or changed
293 DDE Method invoked with no channel open
294 Invalid DDE Link format
295 Message queue filled; DDE message lost
296 PasteLink already performed on this control
297 Can't set LinkMode; invalid LinkTopic
298 DDE requires ddeml.dll
320 Can't use character device names in file names: ' '
321 Invalid file format
340 Control array element ' ' doesn't exist
341 Invalid control array index
342 Not enough room to allocate control array ' '
343 Object not an array
344 Must specify index for object array
345 Reached limit: cannot create any more controls for this form
360 Object already loaded
361 Can't load or unload this object
362 Can't unload controls created at design time
363 Custom control ' ' not found
364 Object was unloaded
365 Unable to unload within this context
```

366 No MDI Form available to load
380 Invalid property value
381 Invalid property array index
382 ' ' property cannot be set at run time
383 ' ' property is read-only
384 A form can't be moved or sized while minimized or maximized
385 Must specify index when using property array
386 ' ' property not available at run time
387 ' ' property can't be set on this control
388 Can't set Visible property from a parent menu
389 Invalid key
390 No Defined Value
391 Name not available
392 MDI child forms cannot be hidden
393 ' ' property cannot be read at run time
394 ' ' property is write-only
395 Can't use separator bar as menu name
400 Form already displayed; can't show modally
401 Can't show non-modal form when modal form is displayed
402 Must close or hide topmost modal form first
403 MDI forms cannot be shown modally
404 MDI child forms cannot be shown modally
420 Invalid object reference
421 Method not applicable for this object
422 Property ' ' not found
423 Property or control ' ' not found
424 Object required
425 Invalid object use
426 Only one MDI Form allowed
427 Invalid object type; Menu control required
428 Popup menu must have at least one submenu
429 OLE Automation server cannot create object
430 Class does not support OLE Automation
431 OLE Automation server cannot load file
432 OLE Automation file or object name syntax error
433 OLE Automation object does not exist
434 Access to OLE Automation object denied
435 OLE initialization error
436 OLE Automation method returned unsupported type
437 OLE Automation method did not return a value
438 OLE Automation no such property or method
439 OLE Automation argument type mismatch
440 OLE Automation error.
441 Error loading VBOA300.DLL
442 OLE Automation Lbound or Ubound on non Array value
443 OLE Automation Object does not have a default value
444 Method not applicable in this context
460 Invalid Clipboard format
461 Specified format doesn't match format of data
480 Can't create AutoRedraw image
481 Invalid picture
482 Printer error
520 Can't empty Clipboard
521 Can't open Clipboard
600 Set value not allowed on collections
601 Get value not allowed on collections
602 General ODBC error: ' '

```
 603 ODBC - SQLAllocEnv failure
 604 ODBC - SQLAllocConnect failure
 605 OpenDatabase - invalid connect string
 606 ODBC - SQLConnect failure ' '
 607 Access attempted on unopened DataBase
 608 ODBC - SQLFreeConnect error
 609 ODBC - GetDriverFunctions failure
 610 ODBC - SQLAllocStmt failure
 611 ODBC - SQLTables (TableDefs.Refresh) failure: ' '
 612 ODBC - SQLBindCol failure
 613 ODBC - SQLFetch failure: ' '
 614 ODBC - SQLColumns (Fields.Refresh) failure: ' '
 615 ODBC - SQLStatistics (Indexes.Refresh) failure: ' '
 616 Table exists - append not allowed
 617 No fields defined - cannot append table
 618 ODBC - SQLNumResultCols (CreateDynaset) failure: ' '
 619 ODBC - SQLDescibeCol (CreateDynaset) failure: ' '
 620 Dynaset is open - CreateDynaset method not allowed
 621 Row-returning SQL is illegal in ExecuteSQL method
 622 CommitTrans/Rollback illegal - Transactions not support
 623 Name not found in this collection
 624 Unable to Build Data Type Table
 625 Data type of field ' ' not supported by target database
 626 Attempt to Move past EOF
 627 Dynaset is not updatable or Edit method has not been invoked
 628 ' ' Dynaset method illegal - no scrollable cursor support
 629 Warning:   (ODBC - SQLSetConnectOption failure)
 630 Property is read-only
 631 Zero rows affected by Update method
 632 Update illegal without previous Edit or AddNew method
 633 Append illegal - Field is part of a TableDefs collection
 634 Property value only valid when Field is part of a Dynaset
 635 Cannot set the property of an object which is part of a Database
     object
 636 Set field value illegal without previous Edit or AddNew method
 637 Append illegal - Index is part of a TableDefs collection
 638 Access attempted on unopened Dynaset
 639 Field type is illegal
 640 Field size illegal for specified Field Type
 641   illegal - no current record
 642 Reserved parameter must be FALSE
 643 Property Not Found
 644 ODBC - SQLConfigDataSource error ' '
 645 ODBC Driver does not support exclusive access to Dynasets
 646 GetChunk: Offset/Size argument combination illegal
 647 Delete method requires a name argument
 648 Data access objects require VBDB300.DLL
2420 Syntax error in number
2421 Syntax error in date
2422 Syntax error in string
2423 Invalid use of '.', '!', or '()'.
2424 Unknown name
2425 Unknown function name
2426 Function isn't available in expressions
2427 Object has no value
2428 Invalid arguments used with domain function
2429 In operator without ()
```

2430 Between operator without And
2431 Syntax error
2432 Syntax error
2433 Syntax error
2434 Syntax error
2435 Extra)
2436 Missing),], or
2437 Invalid use of vertical bars
2438 Syntax error
2439 Wrong number of arguments used with function
2440 IIF function without ()
2442 Invalid use of parentheses
2443 Invalid use of Is operator
2445 Expression too complex
2446 Out of memory during calculation
2447 Invalid use of '.', '!', or '()'.
2448 Can't set value.
2449 Invalid method in expression.
2450 Invalid reference to form ' '.
2451 Invalid reference to report ' '.
2452 Invalid reference to Parent property.
2453 Invalid reference to control ' '.
2454 Invalid reference to '! '.
2455 Invalid reference to property ' '.
2456 Invalid form number reference.
2457 Invalid report number reference.
2458 Invalid control number reference.
2459 Can't refer to Parent property in Design view.
2460 Can't refer to Dynaset property in Design view.
2461 Invalid section reference.
2462 Invalid section number reference.
2463 Invalid group level reference.
2464 Invalid group level number reference.
2465 Invalid reference to field ' '.
2466 Invalid reference to Dynaset property.
2467 Object referred to in expression no longer exists.
2468 Invalid argument used with DatePart, DateAdd or DateDiff function.
2469 1 in validation rule: '|2'.
2470   in validation rule.
2471   in query.
2472   in linked master field.
2473 1 in '|2' expression.
2474 No control is active.
2475 No form is active.
2476 No report is active.
2477 Invalid subclass ' ' referred to in TypeOf function.
3000 Reserved error ( ); there is no message for this error.
3001 Invalid argument.
3002 Couldn't start session.
3003 Couldn't start transaction; too many transactions already nested.
3004 Couldn't find database ' '.
3005 ' ' isn't a valid database name.
3006 Database ' ' is exclusively locked.
3007 Couldn't open database ' '.
3008 Table ' ' is exclusively locked.
3009 Couldn't lock table ' '; currently in use.
3010 Table ' ' already exists.

3011 Couldn't find object ' '.
3012 Object ' ' already exists.
3013 Couldn't rename installable ISAM file.
3014 Can't open any more tables.
3015 ' ' isn't an index in this table.
3016 Field won't fit in record.
3017 Field length is too long.
3018 Couldn't find field ' '.
3019 Operation invalid without a current index.
3020 Update without AddNew or Edit.
3021 No current record.
3022 Can't have duplicate key; index changes were unsuccessful.
3023 AddNew or Edit already used.
3024 Couldn't find file ' '.
3025 Can't open any more files.
3026 Not enough space on disk.
3027 Couldn't update; database is read-only.
3028 Couldn't initialize data access because file 'SYSTEM.MDA' couldn't
be opened.
3029 Not a valid account name or password.
3030 ' ' isn't a valid account name.
3031 Not a valid password.
3032 Can't delete account.
3033 No permission for ' '.
3034 Commit or Rollback without BeginTrans.
3035 Out of memory.
3036 Database has reached maximum size.
3037 Can't open any more tables or queries.
3038 Out of memory.
3039 Couldn't create index; too many indexes already defined.
3040 Disk I/O error during read.
3041 Incompatible database version.
3042 Out of MS-DOS file handles.
3043 Disk or network error.
3044 ' ' isn't a valid path.
3045 Couldn't use ' '; file already in use.
3046 Couldn't save; currently locked by another user.
3047 Record is too large.
3048 Can't open any more databases.
3049 ' ' is corrupted or isn't a Microsoft Access database.
3050 Couldn't lock file; SHARE.EXE hasn't been loaded.
3051 Couldn't open file ' '.
3052 MS-DOS file sharing lock count exceeded. You need to increase the
number of locks installed with SHARE.EXE.
3053 Too many client tasks.
3054 Too many Memo or Long Binary fields.
3055 Not a valid file name.
3056 Couldn't repair this database.
3057 Operation not supported on attached tables.
3058 Can't have Null value in index.
3059 Operation canceled by user.
3060 Wrong data type for parameter ' '.
3061 1 parameters were expected, but only |2 were supplied.
3062 Duplicate output alias ' '.
3063 Duplicate output destination ' '.
3064 Can't open action query ' '.
3065 Can't execute a non-action query.

3066 Query must have at least one output field.
3067 Query input must contain at least one table or query.
3068 Not a valid alias name.
3069 Can't have action query ' ' as an input.
3070 Can't bind name ' '.
3071 Can't evaluate expression.
3073 Operation must use an updatable query.
3074 Can't repeat table name ' ' in FROM clause.
3075 1 in query expression '|2'.
3076   in criteria expression.
3077   in expression.
3078 Couldn't find input table or query ' '.
3079 Ambiguous field reference ' '.
3080 Joined table ' ' not listed in FROM clause.
3081 Can't join more than one table with the same name ( ).
3082 JOIN operation ' ' refers to a non-joined table.
3083 Can't use internal report query.
3084 Can't insert into action query.
3085 Undefined function ' ' in expression.
3086 Couldn't delete from specified tables.
3087 Too many expressions in GROUP BY clause.
3088 Too many expressions in ORDER BY clause.
3089 Too many expressions in DISTINCT output.
3090 Resultant table may not have more than one Counter field.
3091 HAVING clause ( ) without grouping or aggregation.
3092 Can't use HAVING clause in TRANSFORM statement.
3093 ORDER BY clause ( ) conflicts with DISTINCT.
3094 ORDER BY clause ( ) conflicts with GROUP BY clause.
3095 Can't have aggregate function in expression ( ).
3096 Can't have aggregate function in WHERE clause ( ).
3097 Can't have aggregate function in ORDER BY clause ( ).
3098 Can't have aggregate function in GROUP BY clause ( ).
3099 Can't have aggregate function in JOIN operation ( ).
3100 Can't set field ' ' in join key to Null.
3101 Join is broken by value(s) in fields ' '.
3102 Circular reference caused by ' '.
3103 Circular reference caused by alias ' ' in query definition's SELECT list.
3104 Can't specify Fixed Column Heading ' ' in a crosstab query more than once.
3105 Missing destination field name in SELECT INTO statement ( ).
3106 Missing destination field name in UPDATE statement ( ).
3107 Couldn't insert; no insert permission for table or query ' '.
3108 Couldn't replace; no replace permission for table or query ' '.
3109 Couldn't delete; no delete permission for table or query ' '.
3110 Couldn't read definitions; no read definitions permission for table or query ' '.
3111 Couldn't create; no create permission for table or query ' '.
3112 Couldn't read; no read permission for table or query ' '.
3113 Can't update ' '; field not updatable.
3114 Can't include Memo or Long Binary when you select unique values ( ).
3115 Can't have Memo or Long Binary in aggregate argument ( ).
3116 Can't have Memo or Long Binary in criteria ( ) for aggregate function.
3117 Can't sort on Memo or Long Binary ( ).
3118 Can't join on Memo or Long Binary ( ).

3119 Can't group on Memo or Long Binary ( ).
3120 Can't group on fields selected with '*' ( ).
3121 Can't group on fields selected with '*'.
3122 ' ' not part of aggregate function or grouping.
3123 Can't use '*' in crosstab query.
3124 Can't input from internal report query ( ).
3125 ' ' isn't a valid name.
3126 Invalid bracketing of name ' '.
3127 INSERT INTO statement contains unknown field name ' '.
3128 Must specify tables to delete from.
3129 Invalid SQL statement; expected 'DELETE', 'INSERT', 'PROCEDURE',
'SELECT', or 'UPDATE'.
3130 Syntax error in DELETE statement.
3131 Syntax error in FROM clause.
3132 Syntax error in GROUP BY clause.
3133 Syntax error in HAVING clause.
3134 Syntax error in INSERT statement.
3135 Syntax error in JOIN operation.
3136 Syntax error in LEVEL clause.
3137 Missing semicolon (;) at end of SQL statement.
3138 Syntax error in ORDER BY clause.
3139 Syntax error in PARAMETER clause.
3140 Syntax error in PROCEDURE clause.
3141 Syntax error in SELECT statement.
3142 Characters found after end of SQL statement.
3143 Syntax error in TRANSFORM statement.
3144 Syntax error in UPDATE statement.
3145 Syntax error in WHERE clause.
3146 ODBC--call failed.
3147 ODBC--data buffer overflow.
3148 ODBC--connection failed.
3149 ODBC--incorrect DLL.
3150 ODBC--missing DLL.
3151 ODBC--connection to ' ' failed.
3152 ODBC--incorrect driver version ' 1'; expected version '|2'.
3153 ODBC--incorrect server version ' 1'; expected version '|2'.
3154 ODBC--couldn't find DLL ' '.
3155 ODBC--insert failed.
3156 ODBC--delete failed.
3157 ODBC--update failed.
3158 Couldn't save record; currently locked by another user.
3159 Not a valid bookmark.
3160 Table isn't open.
3161 Couldn't decrypt file.
3162 Null is invalid.
3163 Couldn't insert or paste; data too long for field.
3164 Couldn't update field.
3165 Couldn't open .INF file.
3166 Missing memo file.
3167 Record is deleted.
3168 Invalid .INF file.
3169 Illegal type in expression.
3170 Couldn't find installable ISAM.
3171 Couldn't find net path or user name.
3172 Couldn't open PARADOX.NET.
3173 Couldn't open table 'MSysAccounts' in SYSTEM.MDA.
3174 Couldn't open table 'MSysGroups' in SYSTEM.MDA.

3175 Date is out of range or is in an invalid format.
3176 Couldn't open file ' '.
3177 Not a valid table name.
3178 Out of memory.
3179 Encountered unexpected end of file.
3180 Couldn't write to file ' '.
3181 Invalid range.
3182 Invalid file format.
3183 Not enough space on temporary disk.
3184 Couldn't execute query; couldn't find linked table.
3185 SELECT INTO remote database tried to produce too many fields.
3186 Couldn't save; currently locked by user ' 2' on machine '|1'.
3187 Couldn't read; currently locked by user ' 2' on machine '|1'.
3188 Couldn't update; currently locked by another session on this
machine.
3189 Table ' 1' is exclusively locked by user '|3' on machine '|2'.
3190 Too many fields defined.
3191 Can't define field more than once.
3192 Couldn't find output table ' '.
3193 (unknown)
3194 (unknown)
3195 (expression)
3196 Couldn't use ' '; database already in use.
3197 Data has changed; operation stopped.
3198 Couldn't start session. Too many sessions already active.
3199 Couldn't find reference.
3200 Can't delete or change record. Since related records exist in table
' ', referential integrity rules would be violated.
3201 Can't add or change record. Referential integrity rules require
a related record in table ' '.
3202 Couldn't save; currently locked by another user.
3203 Can't specify subquery in expression ( ).
3204 Database already exists.
3205 Too many crosstab column headers ( ).
3206 Can't create a relationship between a field and itself.
3207 Operation not supported on Paradox table with no primary key.
3208 Invalid Deleted entry in [dBASE ISAM] section in INI file.
3209 Invalid Stats entry in [dBASE ISAM] section in INI file.
3210 Connect string too long.
3211 Couldn't lock table ' '; currently in use.
3212 Couldn't lock table ' 1'; currently in use by user '|3' on machine
'|2'.
3213 Invalid Date entry in [dBASE ISAM] section in INI file.
3214 Invalid Mark entry in [dBASE ISAM] section in INI file.
3215 Too many Btrieve tasks.
3216 Parameter ' ' specified where a table name is required.
3217 Parameter ' ' specified where a database name is required.
3218 Couldn't update; currently locked.
3219 Can't perform operation; it is illegal.
3220 Wrong Paradox sort sequence.
3221 Invalid entries in [Btrieve ISAM] section in WIN.INI.
3222 Query can't contain a Database parameter.
3223 ' ' isn't a valid parameter name.
3224 Btrieve--data dictionary is corrupted.
3225 Encountered record locking deadlock while performing Btrieve
operation.
3226 Errors encountered while using the Btrieve DLL.

3227 Invalid Century entry in [dBASE ISAM] section in INI file.
3228 Invalid CollatingSequence entry in [Paradox ISAM] section in INI file.
3229 Btrieve--can't change field.
3230 Out-of-date Paradox lock file.
3231 ODBC--field would be too long; data truncated.
3232 ODBC--couldn't create table.
3233 ODBC--incorrect driver version.
3234 ODBC--remote query timeout expired.
3235 ODBC--data type not supported on server.
3236 ODBC--encountered unexpected Null value.
3237 ODBC--unexpected type.
3238 ODBC--data out of range.
3239 Too many active users.
3240 Btrieve--missing WBTRCALL.DLL.
3241 Btrieve--out of resources.
3242 Invalid reference in SELECT statement.
3243 None of the import field names match fields in the appended table.
3244 Can't import password-protected spreadsheet.
3245 Couldn't parse field names from first row of import table.
3246 Operation not supported in transactions.
3247 ODBC--linked table definition has changed.
3248 Invalid NetworkAccess entry in INI file.
3249 Invalid PageTimeout entry in INI file.
3250 Couldn't build key.
3251 Feature not available.
3252 Illegal reentrancy during query execution.
3254 ODBC--Can't lock all records.
3255 ODBC--Can't change connect string parameter.
3256 Index file not found.
3257 Syntax error in WITH OWNERACCESS OPTION declaration.
3258 Query contains ambiguous (outer) joins.
3259 Invalid field data type.
3260 Couldn't update; currently locked by user ' 2' on machine '|1'.
3261
3262
3263 Invalid database object.
3264 No fields defined - cannot append table.
3265 Name not found in this collection.
3266 Append illegal - Field is part of a TableDefs collection.
3267 Property value only valid when Field is part of a recordset.
3268 Cannot set the property of an object which is part of a Database object.
3269 Append illegal - Index is part of a TableDefs collection.
3270 Property not found.
3271 Invalid property value.
3272 Object is not an array.
3273 Method not applicable for this object.
3274 External table isn't in the expected format.
3275 Unexpected error from external database driver ( ).
3276 Invalid database ID.
3277 Can't have more than 10 fields in an index.
3278 Database engine has not been initialized.
3279 Database engine has already been initialized.
3280 Can't delete a field that is part of an index.
3281 Can't delete an index that is used in a relationship.
3282 Can't perform operation on a nontable.

3283 Primary key already exists.
3284 Index already exists.
3285 Invalid index definition.
3286 Invalid type for Memo field.
3287 Can't create index on Memo field or Long Binary field.
3288 Invalid ODBC driver.
3289 Paradox: No primary index.
3290 Syntax error.
3291 Syntax error in CREATE TABLE statement.
3292 Syntax error in CREATE INDEX statement.
3293 Syntax error in column definition.
3294 Syntax error in ALTER TABLE statement.
3295 Syntax error in DROP INDEX statement.
3296 Syntax error in DROP statement.
3297 Operation not supported in version 1.1
3298 Couldn't import. No records found or all records contained errors.
3299 Several tables exist with that name; please specify owner, as in 'owner.table'.

Additional reference words: 3.00
KBCategory:
KBSubcategory: RefsDoc

LONG: VB 3.0 EXTERNAL.TXT: Using External Database Tables

Article ID: Q108422

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The following article contains the complete contents of the EXTERNAL.TXT file distributed with the Standard and Professional Editions of Visual Basic version 3.0 for Windows. EXTERNAL.TXT describes how to access external database tables using the data control.

MORE INFORMATION

EXTERNAL.TXT

Release Notes for Microsoft (R) Visual Basic (TM) Standard Edition

Version 3.00

This document contains release notes for Microsoft Visual Basic for Windows Standard Edition version 3.0. Information in this document is more current than that in the manuals or online Help.

Visual Basic Standard Edition users can access external database tables using the data control. Professional Edition users should read Appendix C, "Accessing External Databases," in the "Data Access Guide," "Professional Features," Book 2.

How to Use This Document

To view EXTERNAL.TXT on screen in Windows Notepad, maximize the Notepad window.

To print EXTERNAL.TXT, open it in Windows Write, Microsoft Word, or another word processor. Then select the entire document and format the text in 10-point Courier before printing.

Contents

Part	Description
1	Accessing External Databases

2 Opening External Tables

3 Achieving Optimal Performance with External Tables

Part 1: Accessing External Databases

The Standard Edition of Visual Basic can open any of the supported external databases. The following database formats are currently supported:

- Microsoft Access / Visual Basic (databases other than the open database)
- Btrieve (with data definition files FILE.DDF and FIELD.DDF)
- dBASE III and dBASE IV
- FoxPro and FoxPro for Windows
- Paradox

Note

Visual Basic can read and write Microsoft Access databases since it shares a common database engine with Microsoft Access. References to "Visual Basic" databases include those databases created or manipulated by Microsoft Access.

Pros and Cons of Accessing External Databases

You have two choices when using data from external sources.

You can directly access the external database table, or you can import the data into your Visual Basic database. It would make sense to directly access the table if it is already on an established external database which is being actively updated. In this case the mechanisms already set up to update, manage and share the data can remain in place, but your application will have to deal with the extra overhead involved in fetching the external data.

External tables can be used in most respects like any other table in your Visual Basic database while they are being used by other applications sharing the same host server. You can also combine operations that include external data from external tables with data stored in the local database. If you choose to import data from external tables into Visual Basic tables, this can be accomplished through Microsoft Access, or you can also use a Visual Basic application. Visual Basic is especially adept at reading ASCII-delimited files using the INPUT # statement.

The next sections discuss how to access external tables. Note that you can have an existing Microsoft Access or Visual Basic database that already has tables attached to it. Attached tables have linkage information built into the database that permits Visual Basic to access the data as if it were a part of your database. In this case, Visual Basic can extract data from these attached tables without any extra work on your part.

Tips for Using External Tables

When you use an external database table, consider the following tips:

- Before your application uses any data access objects, you'll need to provide Visual Basic with the location of the .INI file that contains initialization parameters for each of the external databases you expect to use. You can do this using the SetDataAccessOption statement. For more information, search Help for "SetDataAccessOption."

- The initialization file must contain a section that includes necessary external database setup information. See the section "Initialization File Details" below for a listing of the .INI file settings needed to connect to each of the supported databases.

Note

If you do not have the correct entries (as described below) in your VB.INI or Appname.INI file you will trigger the trappable error "Cannot find installable ISAM."

* If you access an external table from a Microsoft Access database, you may need to supply a password. You can do this by using the SetDefaultWorkspace statement. For example, the following code indicates the user name is "Chrissy" and the password is "HighIQ."

```
SetDefaultWorkspace "Chrissy", "HighIQ"
```

For more details, see SetDefaultWorkspace in the Language Reference or in Help.

* If you access an external table from Btrieve, Paradox, or an SQL database, you may need to supply a password. Note that this password is different from a Microsoft Access user password; it's the password set in the external database. The database password is supplied in the Connect property of the data control using the PWD identifier. For example, the following Connect string includes a password:

```
"Paradox;PWD=mypword;"
```

* To access an external table on a network, you must connect to the network and have access to the database file or directory. If your network redirector supports it, and you want Visual Basic to automatically connect to the appropriate file server each time you open an external table, specify the fully-qualified network path name for the file in the DatabaseName, Connect and RecordSourceName properties. There is no mechanism to provide a network share password. In cases where a password is required to gain access to a network share, you'll have to connect to the share, provide a password, and pre-assign a drive letter to the share before starting your program.

For example, if you use a Microsoft LAN Manager network, you might enter the following path to connect to a remote dBASE file:

```
\\server\share\datadir\author.dbf
```

To provide this path when opening a remote dBASE III table, use this code:

```
' Assume we want to attach a dBASE III table called AUTHOR
' on the \\SERVER\SHARE\DATADIR server.
'
Data1.Databasename = "\\SERVER\SHARE\DATADIR"
Data1.RecordSource = "AUTHOR"
Data1.Connect = "dBASE III"
Data1.Refresh
```

As long as the dBASE file is not moved, the data will be available to your application. Generally, the syntax for attaching other types of external files is similar to the technique shown above.

* When defining external tables, only Paradox will support primary key definitions. Paradox tables require primary keys.

* Although you can use an attached table as if it were a Microsoft Access table, there are special considerations. For information about working with attached tables, see "Using Attached External Tables" later in this document.

* When you work with multiple external database tables, occasionally you may find the Updatable property is False. Generally, this is due to the complexity of the query. To be able to consistently update external tables, you may find it easier to access them in simpler queries.

* When Visual Basic manipulates external databases, it creates temporary indexes for the queries being performed on the workstation's hard disk - even if the database is on an external (networked) device. Temporary space is allocated from the directory indicated by the TEMP environment string variable, which usually points to the \WINDOWS\TEMP directory. If your system has not established a TEMP environment variable, if it points to an invalid path or if your workstation does not have sufficient space for these temporary indexes, your application may behave unpredictably as Windows and Visual Basic run out of resource space. The amount of space needed is a function of the size of the external table and can vary from a few thousand bytes to several megabytes.

* When deleting records from dBASE or FoxPro databases, the records may reappear when the table is closed and reopened. To tell Visual Basic not to fetch deleted records, set the DELETED parameter in the .INI file to "On" (the default).

Part 2: Opening External Tables

The method for opening each of the external databases is roughly the same. Subsequent sections in this appendix deal with the individual characteristics for each of the supported external database formats. When using the data control to directly open external tables, you will need to either work interactively with Visual Basic's Properties window at design time to set the individual properties for the data control, or use code in your application to make the settings.

Accessing Paradox Tables

Visual Basic can access external tables from Paradox versions 3.0 and 3.5. If you provide the correct password, Visual Basic can open encrypted Paradox tables. If you open an external Paradox table, you can extract and update data even if others are using it in Paradox.

When opening external Paradox database tables directly, you'll also need to specify the name of the directory (not a filename) as the DatabaseName property of the data control and the name of the table file in the RecordSource property. For example, to open a Paradox file "Author.DB" and use the name "ParaAuthor" to reference it as a table object, use the following code:

```
data1.Connect = "Paradox;"          ' Specify database type
data1.DatabaseName = "C:\Paradox"   ' Point to directory
data1.RecordSource = "Author"       ' Name database table file
data1.Refresh
While Not data1.RecordSet.EOF
 Print data1.RecordSet(0)           ' Dump field(0) to the form
 data1.RecordSet.MoveNext           ' for all records
Wend
```

Important

Paradox stores important information about a table's primary key in an index (.PX) file. If you access a Paradox table that has a primary key, Visual Basic needs the .PX file to open the external table. If you delete or move this file, you won't be able to open the external table. If you attach a Paradox table that doesn't have a primary key, you cannot update data in the table using Visual Basic. To be able to update the table, define a primary key in Paradox.

Paradox to Microsoft Access Data-Type Conversions

When you access an external Paradox table, Visual Basic translates Paradox data types into the corresponding Visual Basic data types. The following table lists the data-type conversions.

Paradox data type	Microsoft Access data type
Alphanumeric	Text
Currency	Number (FieldSize property set to Double)
Date	Date/Time
Number	Number (FieldSize property set to Double)
Short number	Number (FieldSize property set to Integer)

Accessing dBASE and FoxPro Files

Visual Basic can directly open external .DBF files in dBASE III, dBASE IV, or FoxPro version 2.0 or 2.5 format. If you directly open a dBASE or FoxPro table file, you can view and update data, even if others are using it with dBASE or FoxPro. If you access a dBASE or FoxPro file, you can also tell Visual Basic to use one or more index files (.NDX or .MDX for dBASE; .IDX or .CDX for FoxPro) to improve performance.

For dBASE and FoxPro databases, Visual Basic keeps track of the table indexes in a special information (.INF) file. When you use Visual Basic to update the data in your .DBF file, Visual Basic also updates the index files to reflect your changes. The .INF file is created for you when you use Visual Basic to create a new index for a dBASE or FoxPro table, or you can create them yourself with a text editor.

The format for the .INF files is as follows:

```
TableName.INF contains:
NDX1=<Index 1 Filename>.NDX
NDX2=<Index 2 Filename>.NDX
NDXn=<Index n Filename>.NDX
```

For example, an .INF file for the Authors table would be AUTHORS.INF and it might contain:

```
NDX1=CityIndx.NDX
NDX2=NameIndx.NDX
```

Place these index and .INF files in the same directory as the other dBASE III files. FoxPro and dBASE databases are not maintained in a single file but in a disk directory which contains separate data, index, and other support files. When opening external FoxPro and dBASE database tables directly, you'll also need to specify the name of the directory (not a filename) as the DatabaseName property in the data control and the name of the table file in the RecordSource property. For example, to open a FoxPro version 2.5 file "Author.DBF", use this code:

```
data1.Connect = "FoxPro 2.5;"           ' Specify database type
data1.DatabaseName = "C:\FoxPro"        ' Point to directory
data1.RecordSource = "Author"           ' Name database table file
data1.Refresh
While Not data1.RecordSet.EOF
    Print data1.RecordSet(0)            ' Dump field(0) to the form
    data1.RecordSet.MoveNext            ' for all records
Wend
```

FoxPro and dBASE Memo fields are located in separate files. These files cannot be located or moved outside of the directory containing the table files. FoxPro and dBASE database systems do not physically delete records but merely mark them for deletion at a later time. You must PACK the .DBF file (using your own utilities) to remove these records from the .DBF files. The CompactDatabase function will not affect attached tables. If you use the .INI file setting DELETED = ON (in the [dBASE ISAM] section), Visual Basic filters out deleted records so that they do not appear in recordsets. With DELETED=OFF, all records are included in the recordsets you create, including deleted records. This allows dBASE and FoxPro users to undelete records. In this case, when you access a dBASE or FoxPro table, Visual Basic builds a Dynaset from the records. When you delete a record, Visual Basic deselects the record in the Dynaset and marks the record as deleted in the .DBF file. If you refresh the Dynaset or reopen the table the records will still be present.

Important
=========

If you access a .DBF file and associate an index file (.NDX or .MDX for dBASE or .IDX; .CDX for FoxPro), Visual Basic needs the index file to open the attached table. If you delete or move index files or the information (.INF) file, you won't be able to open the external table. Additionally, if you use dBASE or FoxPro to update data in a .DBF file that you have accessed from your Visual Basic Database, you must also update any dBASE or FoxPro indexes associated with the .DBF file. If the index files are not current when Visual Basic tries to use them, the results of your queries are unpredictable.

dBASE and FoxPro to Microsoft Access Data-Type Conversions

When you access a dBASE or FoxPro file, Visual Basic translates dBASE and FoxPro data types into the corresponding Microsoft Access data types. The following table lists the data-type conversions.

dBASE data type	Microsoft Access data type
Character	Text
Date	Date/Time
General (FoxPro only)	OLE
Logical	Yes/No
Memo	Memo
Numeric, Float	Number (FieldSize property set to Double)

Accessing Btrieve Tables

Using Visual Basic, you can directly open in Btrieve 5.1x format. To use Btrieve tables, you must have the data definition files FILE.DDF and FIELD.DDF, which tell Visual Basic the structure of your tables. These files are created by Xtrieve* or by another .DDF file-building program. If you delete or move these files or your data files, you won't be able to open an attached Btrieve table. For more information on using Btrieve with Visual Basic, see the text file BTRIEVE.TXT in your Visual Basic directory.

When accessing Btrieve database tables, you'll need to specify the name of the Btrieve data file (.DDF) as the DatabaseName (DATABASE= in the Connect property) and the name of the table file in the SourceTableName property. In this case the Btrieve file name may have no bearing on the name of the table. The correct file names are stored in the FILE.DDF file.

For example, to open a Btrieve file "FILE.DDF" to reference the Btrieve database table "Author", use this code:

```
data1.Connect = "Btrieve;"           ' Specify database type
data1.DatabaseName = "C:\Btrieve\FILE.DDF"  ' Point to database file
data1.RecordSource = "Author"        ' Name database table file
data1.Refresh

While Not data1.RecordSet.EOF
   Print data1.RecordSet(0)     ' Dump field(0) to the form
   data1.RecordSet.MoveNext     ' for all records
Wend
```

WIN.INI Initialization File Settings

The Btrieve driver uses the [BTRIEVE] section of the WIN.INI file (not VB.INI) when it accesses Btrieve files. After you install Visual Basic and specify that you want to access Btrieve files, the WIN.INI file contains the following default settings:

```
[BTRIEVE]
Options=/m:64 /p:4096 /b:16 /f:20 /l:40 /n:12 /t:c:\VB3\BTRIEVE.TRN
```

The following table gives a brief description of each switch. You should consult your Btrieve documentation and BTRIEVE.TXT supplied with Visual Basic for a definitive and current listing of these settings. If you install another application that modifies these settings from the values shown, Visual Basic may not function normally.

Switch	Definition
/m	Memory size
/p	Page size
/b	Pre-image buffer size
/f	Open files
/l	Multiple locks
/n	Files in a transaction
/t	Transaction file name. (Must be visible to all Btrieve users.)

NOTE

To use Btrieve data, you must have the Btrieve for Windows dynamic-link library (WBTRCALL.DLL), which is not provided with Visual Basic. This file is available with Novell* Btrieve for Windows, Novell NetWare* SQL, and other Windows-based products that use Btrieve. If you expect to share a Btrieve database, you will need to make sure that the path given for the transaction file (as specified above) is visible on the net to all users of the database. Generally, this file is placed on a common server that all users have access to. The default setting for this parameter does not take this into account.

Btrieve to Microsoft Access Data-Type Conversions

When you access a Btrieve table, Visual Basic translates Btrieve data types into the corresponding Microsoft Access data types.

The following table lists the data-type conversions.

Btrieve data type	Microsoft Access data type
Date, time	Date/Time
Float or bfloat (4-byte)	Number (FieldSize property set to Single)
Float or bfloat (8-byte), decimal, numeric	Number (FieldSize property set to Double)
Integer (1-, 2-, or 4-byte)	Number (FieldSize property set to Byte, Integer, or Long Integer)
Logical	Yes/No
Lvar	OLE Object

Money	Currency
Note	Memo
String, lstring, zstring	Text

Part 3: Achieving Optimal Performance with External Tables

Although you can use external tables as if they're regular Microsoft Access tables, it's important to keep in mind that they aren't actually in your Visual Basic database. Each time you view data in an external table, Visual Basic has to retrieve records from another file. This can take time, especially if the external table is on a network.

If you're using an external table on a network, follow these guidelines for best results:

* View only the data you need. Don't page up and down unnecessarily in the data. Avoid jumping to the last record in a large table unless you want to add new records to the table.

* Use queries to limit the number of records that you fetch. This way, Visual Basic can transfer less data over the network.

* In queries that involve external tables, avoid using functions in query criteria. In particular, avoid using aggregate functions, such as DSum, anywhere in your queries. When you use an aggregate function, Visual Basic retrieves all of the data in the external table in order to execute the query.

* If you often add records to an external table, add the records to a Microsoft Access-format table and use an action query to append all added records in one operation. This saves time because Visual Basic won't have to retrieve all the records in the external table.

* Remember that other users may be trying to use an external table at the same time you are. When a Visual Basic Database is on a network, you should avoid locking records longer than necessary.

NOTE

If the information stored in the attached table link properties changes (for example, the database file is moved or a password is changed), you won't be able to open the attached table. To specify current information, delete the outdated link and attach the table again.

Initialization File Details

When Visual Basic is installed, you can install as many of the external database drivers as you want. For those drivers that are installed, an associated .INI file entry is made. Shown below are the default settings for all supported external database drivers. In some cases, these .INI file settings are discussed earlier in the specific driver sections. When you ship your application, it will be necessary to create an initialization file that has the correct .INI settings for the drivers you want to support.

NOTE

To determine the number of retries on commit locks, Visual Basic uses
the following formula for the actual retry count:

Count = LockRetry * CommitLockRetry

VB.INI or <Appname>.INI Default Settings

```
[Options]
SystemDB=C:\MYPATH\SYSTEM.MDA   ; Access SYSTEM.MDA for use only if
            ; Microsoft Access is being used
[ISAM]
PageTimeout=5                   ;500 ms - non-read-locked page timeout
MaxBufferSize=128               ;128K
LockRetry=20                    ;20 - retries on Read/Write locks
CommitLockRetry=20              ;20 - retries on Commit locks
ReadAheadPages=16               ;16 pages

[Installable ISAMs]
Paradox 3.X=C:\VB\pdx110.DLL    ;Path of the Paradox driver
FoxPro  2.0=C:\VB\xbs110.DLL    ;Path of the FoxPro 2.0 driver
FoxPro  2.5=C:\VB\xbs110.DLL    ;Path of the FoxPro 2.5 driver
dBASE III=C:\VB\xbs110.DLL      ;Path of the dBASE III driver
dBASE IV=C:\VB\xbs110.DLL       ;Path of the dBASE IV driver
Btrieve=C:\VB\btrv110.DLL       ;Path of the Btrieve driver

[Paradox ISAM]
PageTimeout=600                 ;60 seconds
ParadoxUserName=Joe User        ;Name displayed when lock
                                ; conflicts occur
ParadoxNetPath=P:\PDOXDB\       ;Path to the PARADOX.NET file
CollatingSequence=Ascii         ;Collating sequence of your files
            ;  (Ascii, International, Norwegian-Danish,
            ;  or Swedish-Finnish)

[BTrieve ISAM]
PageTimeout=600                 ;60 seconds
[dBase ISAM]
PageTimeout=600                 ;60 seconds CollatingSequence=Ascii
            ;Collating sequence
            ;(Ascii or International)
Century=Off ;Use of four-digit dates
            ;(On or Off)
Date=American                   ;Date format: correlates to
            ;the SET DATE command in dBase
Mark=47     ;Decimal value of the ascii
            ;mark character:correlates to the
            ;SET MARK command in dBase
Deleted=ON  ;Show and operate on deleted records
            ;Deleted=On: never operate
            ;on deleted records
```

WIN.INI

The following line must appear in WIN.INI (located in your Windows directory) if you intend to use external Btrieve tables. The details of this entry are discussed in the Btrieve section earlier in this appendix.

```
[BTRIEVE]
Options= /m:64 /P:4096 /b:16 /f:20 /l:40 /n:12 /t:c:\VB\BTRIEVE.TRN
```

Additional reference words: 3.00
KBCategory: Refs
KBSubcategory: RefsDoc

VB 3.0 CONSTANT.TXT Gives Values for Named Constants

Article ID: Q108468

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The CONSTANT.TXT file is distributed with the Standard and Professional Editions of Visual Basic version 3.0 for Windows.

CONSTANT.TXT is the global constant file for Visual Basic. The CONSTANT.TXT file provides an electronic copy of named constants mentioned in the Visual Basic manuals and sample programs.

For convenient programming access to all global constants, you can load the CONSTANT.TXT file into a code module in Visual Basic.

If you are updating a Visual Basic application written with an older version, you should replace your global constants with the constants in the CONSTANT.TXT file.

MORE INFORMATION

Constants for Both Standard and Professional Editions

The CONSTANT.TXT file defines named Const constants for the following topics. All of these apply to both the Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0:

Action Property
Actions
Align (picture box)
Alignment
Arrange Method for MDI Forms
AutoActivate modes
BorderStyle (form)
BorderStyle (Shape and Line)
Button parameter masks
Check Value
Clipboard formats
CMDIALOG.VBX
ColAlignment, FixedAlignment Properties
Color Dialog Flags
Colors
Common Dialog Control
Data control
DisplayTypes
Drag (controls)

DragMode
DragOver
DrawMode
DrawStyle
Editmode property values
Enumerated Types
ErrNum (LinkError)
Error Constants
Error event Response arguments
Event Parameters
File Open/Save Dialog Flags
FillStyle
Fillstyle Property
Fonts Dialog Flags
Function Parameters
General
Grid
Help Constants
Key Codes
LinkMode (forms and controls)
MiscFlag Bits
MousePointer
MsgBox parameters
MsgBox return values
OLE Client Control
OLEType
OLETypeAllowed
Options property values
Printer Dialog Flags
Properties
QueryUnload
ScaleMode
ScrollBar
SetAttr, Dir, GetAttr functions
Shape
Shift parameter masks
Show parameters
SizeModes
Special Verb Values
System Colors
Update Event Constants
UpdateOptions
Validate event Action arguments
Variant VarType tags
VerbFlag Bit Masks
WindowState
ZOrder Method

Constants for Professional Edition

The Professional Edition includes the following controls:

1. 3-D Controls (Frame/Panel/Option/Check/Command/Group Push)
2. Animated Button
3. Gauge Control
4. Graph Control Section

5. Key Status Control
6. Spin Button
7. MCI Control (Multimedia)
8. Masked Edit Control
9. Comm Control
10. Outline Control

The CONSTANT.TXT file defines the values of named Const constants for the following topics which apply to the Professional Edition of Microsoft Visual Basic for Windows:

3D Controls
Action
Alignment (Check Box)
Alignment (Frame)
Alignment (Option Button)
Alignment (Panel)
Animated Button
Area Chart Options
Autosize (Command Button)
Autosize (Panel)
Autosize (Ribbon Button)
Bar Chart Options
BevelInner (Panel)
BevelOuter (Panel)
Click Filter property
ClipMode
Colors
Comm Control
Cycle property
Data Arrays
Draw Mode
Error code constants
ERROR CONSTANT DECLARATIONS (MAPI CONTROLS)
Event constants
FloodType (Panel)
Font3D (Panel, Command Button, Option Button, Check Box, Frame)
Gantt Chart Options
GAUGE
Graph Control
Graph Types
Grids
Handshaking
HLC Chart Options
Key Status Control
Line Styles
Line/Polar Chart Options
MAPI MESSAGE CONTROL CONSTANTS
Masked Edit Control
MCI Control (Multimedia)
MISCELLANEOUS GLOBAL CONSTANT DECLARATIONS (MAPI CONTROLS)
Mode Property
NotifyValue Property
Orientation Property
Outline
Outline Control Error Constants
Patterns

PicDrawMode Property
PictureDnChange (Ribbon Button)
PictureType
Pie Chart Options
Print Options
RecordMode Porperty
ShadowColor (Panel, Frame)
ShadowStyle (Frame)
SpecialOp Property
Spin Button
SpinOrientation
Statistics
Style
Style Property
Symbols
TextPosition Property
TimeFormat Property

REFERENCES
==========

 - Online Help file VB.HLP
 - "Microsoft Visual Basic Version 3.0 for Windows: Programmer's Guide"
 - "Microsoft Visual Basic Version 3.0 for Windows: Language Reference"

Additional reference words: 3.00
KBCategory: Refs
KBSubcategory: RefsDoc

VB 3.0 DATACONS.TXT: Const Constant Values for Data Access

Article ID: Q108469

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The following article contains the complete contents of the DATACONS.TXT file distributed with the Standard and Professional Editions of Visual Basic version 3.0 for Windows. DATACONS.TXT gives the values of named constants used for data access in Visual Basic.

The DATACONS.TXT file is required as a supplement to the following manual, because this manual gives the names of the constants without specifying all the values:

- Microsoft Visual Basic 3.0: Professional Features Book 2: Data Access Guide

MORE INFORMATION

DATACONS.TXT

```
'
' Data Access constants
'

' Option argument values (CreateDynaset, etc)
Global Const DB_DENYWRITE = &H1
Global Const DB_DENYREAD = &H2
Global Const DB_READONLY = &H4
Global Const DB_APPENDONLY = &H8
Global Const DB_INCONSISTENT = &H10
Global Const DB_CONSISTENT = &H20
Global Const DB_SQLPASSTHROUGH = &H40

' SetDataAccessOption
Global Const DB_OPTIONINIPATH = 1

' Field Attributes
Global Const DB_FIXEDFIELD = &H1
Global Const DB_VARIABLEFIELD = &H2
Global Const DB_AUTOINCRFIELD = &H10
Global Const DB_UPDATABLEFIELD = &H20

' Field Data Types
Global Const DB_BOOLEAN = 1
Global Const DB_BYTE = 2
```

```
Global Const DB_INTEGER = 3
Global Const DB_LONG = 4
Global Const DB_CURRENCY = 5
Global Const DB_SINGLE = 6
Global Const DB_DOUBLE = 7
Global Const DB_DATE = 8
Global Const DB_TEXT = 10
Global Const DB_LONGBINARY = 11
Global Const DB_MEMO = 12

' TableDef Attributes
Global Const DB_ATTACHEXCLUSIVE = &H00010000
Global Const DB_ATTACHSAVEPWD = &H00020000
Global Const DB_SYSTEMOBJECT = &H80000002
Global Const DB_ATTACHEDTABLE = &H40000000
Global Const DB_ATTACHEDODBC = &H20000000

' ListTables TableType
Global Const DB_TABLE = 1
Global Const DB_QUERYDEF = 5

' ListTables Attributes (for QueryDefs)
Global Const DB_QACTION = &HF0
Global Const DB_QCROSSTAB = &H10
Global Const DB_QDELETE = &H20
Global Const DB_QUPDATE = &H30
Global Const DB_QAPPEND = &H40
Global Const DB_QMAKETABLE = &H50

' ListIndexes IndexAttributes values
Global Const DB_UNIQUE = 1
Global Const DB_PRIMARY = 2
Global Const DB_PROHIBITNULL= 4
Global Const DB_IGNORENULL= 8
' ListIndexes FieldAttributes value
Global Const DB_DESCENDING = 1  'For each field in Index

' CreateDatabase and CompactDatabase Language constants
Global Const DB_LANG_GENERAL = ";LANGID=0x0809;CP=1252;COUNTRY=0"
Global Const DB_LANG_SPANISH = ";LANGID=0x040A;CP=1252;COUNTRY=0"
Global Const DB_LANG_DUTCH = ";LANGID=0x0413;CP=1252;COUNTRY=0"
'VB3 and Access 1.1 Databases:
Global Const DB_LANG_SWEDFIN = ";LANGID=0x040C;CP=1252;COUNTRY=0"
'VB3 and Access 1.1 Databases:
Global Const DB_LANG_NORWDAN = ";LANGID=0x0414;CP=1252;COUNTRY=0"
'VB3 and Access 1.1 Databases:
Global Const DB_LANG_ICELANDIC = ";LANGID=0x040F;CP=1252;COUNTRY=0"
'Access 1.0 Databases only:
Global Const DB_LANG_NORDIC = ";LANGID=0x041D;CP=1252;COUNTRY=0"

' CreateDatabase and CompactDatabase options
Global Const DB_VERSION10 = 1        ' Microsoft Access Version 1.0
Global Const DB_ENCRYPT = 2          ' Make database encrypted.
Global Const DB_DECRYPT = 4          ' Decrypt database while compacting.

'Collating order values
```

```
Global Const DB_SORTGENERAL = 256    ' Sort by EFGPI rules (English,
French,
                                     ' German,Portuguese, Italian)
Global Const DB_SORTSPANISH = 258    ' Sort by Spanish rules
Global Const DB_SORTDUTCH = 259      ' Sort by Dutch rules
Global Const DB_SORTSWEDFIN = 260    ' Sort by Swedish, Finnish rules
Global Const DB_SORTNORWDAN = 261    ' Sort by Norwegian, Danish rules
Global Const DB_SORTICELANDIC = 262  ' Sort by Icelandic rules
Global Const DB_SORTPDXINTL = 4096   ' Sort by Paradox international rules
Global Const DB_SORTPDXSWE = 4097    ' Sort by Paradox Swedish, Finnish
                                     ' rules
Global Const DB_SORTPDXNOR = 4098    ' Sort by Paradox Norwegian, Danish
                                     ' rules
Global Const DB_SORTUNDEFINED = -1   ' Sort rules are undefined or unknown


Additional reference words: 3.00
KBCategory: Refs
KBSubcategory: RefsDoc
```

Category Keywords for All Visual Basic KB Articles

Article ID: Q108753

The information in this article applies to:

- Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Each article in the Visual Basic for Windows collection contains at least one keyword (called a KBSubcategory keyword) that places the article in an appropriate category. This article lists all the KBSubcategory keywords.

MORE INFORMATION

Category & Subcategory Description	KBSubcategory Keyword
Setup / Installation (Setins)	Setins
Environment-specific Issues (Envt)	
VB Design Environment	EnvtDes
Run-Time Environment	EnvtRun
Programming (Prg)	
Visual Basic Forms and Controls	
Standard Controls / Forms	PrgCtrlsStd
Custom Controls	PrgCtrlsCus
Third-Party Controls	PrgCtrlsThird
Optimization	
Memory Management	PrgOptMemMgt
General Optimization Tips	PrgOptTips
General VB Programming	PrgOther
Advanced programming (APrg)	
Network	APrgNet
Windows Programming (APIs / DLLs)	
Printing	APrgPrint
Graphics	APrgGrap
Windowing	APrgWindow
INI Files	APrgINI
Other API / DLL Programming	APrgOther
Data Access	
ODBC	APrgDataODBC
IISAM	APrgDataIISAM
Access	APrgDataAcc
General Database Programming	APrgDataOther
3rd Party DLL's	APrgThirdDLL

Inter-Application Programmability (IAP)

OLE	IAPOLE
DDE	IAPDDE
3rd Party Interoperability	IAPThird

Tools (Tls)

Setup Toolkit / Wizard	TlsSetWiz
Control Development Kit (CDK)	TlsCDK
Help Compiler (HC)	TlsHC

References (Refs)

Documentation / Help File Fixes	RefsDoc
Product Information	RefsProd
Third-Party Information	RefsThird
PSS-Only Information	RefsPSS

Using Keywords to Query the KB

At Microsoft, we use the subcategory keywords to organize the articles for Help files and for the FastTips Catalog. You can use them to query the Microsoft Knowledge Base for Visual Basic articles that apply to that category or subcategory. For example, you can find all the general database programming articles by querying on the following words in the Microsoft Knowledge Base:

```
visual and basic and APrgDataOther
```

Use the asterisk (*) wildcard to find articles that fall into the general categories or into an intermediate subcategory. The first element in each keyword is the category. For example, to find all the articles that apply to Visual Basic Forms and Controls regardless of whether they are standard, custom, or third-party controls, use the following words to query the Microsoft Knowledge Base:

```
visual and basic and PrgCtrls*
```

To find all advanced programming articles, query on these words:

```
visual and basic and APrg*
```

Add KBSubcategory Keyword to Each Article

When contributing an article to the Visual Basic Knowledge Base, add the appropriate KBSubcategory keyword to the bottom of the article on the KBSubcategory line. Each article in the Visual Basic for Windows collection contains the following section at the bottom of the article:

```
Additional reference words:
KBCategory:
KBSubcategory: <keyword>
```

An article usually has only one subcategory keyword, but it may have more.

If you are interested in contributing, please obtain the guidelines by

querying on the following words in the Microsoft Knowledge Base:

   visual and basic and kbguide and kbartwrite

Additional reference words: 3.00 dskbguide subcatkey
KBCategory:
KBSubcategory: RefsPSS

LONG: How to Call Windows API from VB - General Guidelines

Article ID: Q110219

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 2.0 and 3.0

SUMMARY

This article gives general guidelines and examples to introduce you to the process of calling Windows API functions from a Visual Basic application. The examples used in this article are discussed individually in other articles in the Microsoft Knowledge Base. They are repeated here as examples for those new to the process of calling Windows API functions.

One of the most powerful features of Microsoft Visual Basic is the Declare statement, which allows you, the Visual Basic programmer, to call the routines in any Dynamic Link Library (DLL). Microsoft Windows is itself a collection of DLLs, so Visual Basic can call almost any of the functions in the Microsoft Windows Application Programming Interface (API). By calling these routines you can perform tricks that are impossible in Visual Basic alone.

MORE INFORMATION

The Windows API can appear daunting at first. You need to approach it with a sense of adventure: for a Visual Basic programmer, the Windows API is a huge unexplored jungle of over five hundred functions. Fortunately, Visual Basic takes care of so many details for you that you will never have to learn anything about most of these functions. But some of them do things that are very hard to do in Visual Basic alone. And a few of them allow you to do things in your Visual Basic application that you can't do any other way. This article is your guide to the API jungle.

Backups Are Crucial

As with any good adventure, there are risks as well. When calling the Windows API, you may declare a function incorrectly or pass it the wrong values. As a result, you may get a general protection (GP) fault or an Unexpected Application Error (UAE). Fortunately, insurance for this adventure is cheap: always save your work before you run it. Keep backups for every version.

Additional Resources You Might Want

While reading this article and trying out the examples, you may find it helpful to keep the Visual Basic Programmer's Guide handy. Chapter 24, "Calling Procedures in DLLs" explains details only touched on here.

To go beyond this article, you need to get documentation for the Windows API. The Professional Edition of Visual Basic includes this information in two Help files (WIN31WH.HLP and WIN31API.HLP) and a text file (WINAPI.TXT) the complete list of Visual Basic DLL procedure, constant, and user-defined type declarations for the Windows API. You can search for the declaration you want in the WIN31API.HLP Help file; then copy and paste them into your code. Alternatively, you can copy the declarations from the WINAPI.TXT file. You'll find all three files in the \VB\WINAPI directory

Two-Step Process

There are two steps to using a DLL procedure in Visual Basic. First you declare it once. Then you call it as many times as it is needed. The remainder of this article provides a number of examples you can use to test this two-step process.

Declaring DLL Routines

Most DLL routines, including those in the Windows API, are documented using notation from the C programming language. This is only natural, as most DLLs are written in C. However, this poses something of a challenge for the intrepid Visual Basic programmer who wants to call these routines.

In order to translate the syntax of a typical API routine into a Visual Basic Declare statement, you have to understand something about how both C and Visual Basic pass their arguments. The usual way for C to pass numeric arguments is "by value": a copy of the value of the argument is passed to the routine.

Sometimes C arguments are pointers, and these arguments are said to be passed "by reference." Passing an argument by reference allows the called routine to modify the argument and return it. C strings and arrays are always passed by reference.

Visual Basic, on the other hand, usually passes all of its arguments by reference. In effect, when you pass arguments to a Visual Basic procedure you are actually passing "far" (32 bit) pointers to those values. In order to pass arguments to a C routine that expects its arguments to be passed by value, you have to use the ByVal keyword with the argument in the Declaration statement.

Obviously, if a DLL routine is expecting an argument to be passed by value and you pass a pointer instead, it is not going to behave as you expect. Likewise, if the routine is expecting a pointer to a value and you pass the value itself, the routine is going to attempt to access the wrong memory location and probably cause a GP fault or UAE. So be careful.

One added wrinkle to this is that Visual Basic strings do not use the same format as C strings. Visual Basic has overloaded the ByVal keyword to mean "pass a C string" when it is used with a string argument in a declare statement.

C argument types and their equivalent declarations in Visual Basic:

If the argument is	Declare it as

standard C string (LPSTR, char far *)	ByVal S$
Visual Basic string (see note)	S$
integer (WORD, HANDLE, int)	ByVal I%
pointer to an integer (LPINT, int far *)	I%
long (DWORD, unsigned long)	ByVal L&
pointer to a long (LPDWORD, LPLONG, DWORD far *)	L&
standard C array (A[ ])	base type array (no ByVal)
Visual Basic array (see note)	A()
struct (typedef)	S As Struct

NOTE: You will never pass a Visual Basic string or array to a DLL routine unless the DLL was written specifically for use with Visual Basic. Visual Basic strings and arrays are represented in memory by "descriptors" (not pointers), which are useless to DLL routines that were not written with Visual Basic in mind.

There is one more complication to this, however. Some Windows functions take a 32 bit argument that sometimes is a "far" (32 bit) pointer to something and sometimes is just a 32 bit value. The fourth argument in the SendMessage function is like this. If you are going to call it with just a pointer or with just a value, you can declare it appropriately. For example, you could declare SendMessage to take a pointer to a string:

```
' Enter the following Declare statement as one, single line:
Declare Function SendMessage Lib "user"
   (ByVal hWnd%, ByVal msg%, ByVal wp%, ByVal lp$) As Long
```

Or you could declare it to take a 32 bit value:

```
' Enter the following Declare statement as one, single line:
Declare Function SendMessage Lib "user"
   (ByVal hWnd%, ByVal msg%, ByVal wp%, ByVal lp&) As Long
```

Notice the fourth argument is declared ByVal lp$ in the first example and ByVal lp& in the second.

However, what if you want to call it with both kinds of arguments in the same program? If you declare it one way and call it another, you will get an error message from Visual Basic. The solution is to declare the argument As Any:

```
' Enter the following Declare statement as one, single line:
Declare Function SendMessage Lib "user"
   (ByVal hWnd%, ByVal msg%, ByVal wp%, lp As Any) As Long
```

The Any "data type" tells Visual Basic not to do any type checking for that argument. So now you can pass it anything as long as it is what the function is expecting. If an argument is declared As Any, you must specify whether the argument is passed by value or not -- when you actually call the function. You do this by using ByVal for strings and for arguments that should be passed by value, and omitting ByVal for arguments that should be passed by reference. Use the appropriate entry from the second column in the table shown above as the argument when you call the function.

For example, if you have declared the fourth argument in SendMessage As Any, you can pass a string in that argument:

```
buflen& = SendMessage(txthWnd, EM_GETLINE, lineNum%, ByVal buf$)
```

Notice you use the ByVal keyword in the call. This tells Visual Basic that you want to pass a standard C string. If you don't include the ByVal, you will pass a Visual Basic string descriptor, which is not something SendMessage knows how to handle.

You can also pass an array:

```
dummy& = SendMessage(any_hWnd%, EM_SETTABSTOPS, NumCol%, ColSizes(1))
```

Notice you do not use ByVal in this case because you want to pass a pointer (specifically a pointer to the indicated element in the array -- all subsequent array elements are packed into memory after it).

You can pass a long integer value as the fourth argument:

```
dummy& = SendMessage(txthWnd, EM_LINESCROLL, 0, ByVal ScrollAmount&)
```

Note the use of ByVal here. You want to pass the value itself, rather than a pointer to the value. It's very important that you pass a Long integer for this argument. If you pass a normal Integer Visual Basic will not convert it into Long.

If you're careful to match up what you're passing with what the routine expects, you should have no trouble calling the Windows API to get Visual Basic to do what you want as demonstrated in the examples that comprise the remainder of this article.

Scoping Out the System

One of the nice things about Windows is that it insulates you from a lot of the details of the system. You can print to the printer without knowing what kind it is; you can display things on the screen without knowing its resolution. However, there may be times when your application needs to know key information about the system. For example, you may want your application to perform different calculations depending on whether the system has a math coprocessor or not.

Fortunately, Windows provides several functions that you can use to obtain this kind of information. For example the GetWinFlags API function can give you a lot of information.

Place this declaration in the declarations section of a form or module, or in the global module:

```
Declare Function GetWinFlags Lib "kernel" () As Long
```

As functions in the Windows API go, this one is very simple. It is found in the Windows "kernel" DLL. It takes no arguments (hence the empty parentheses in the declaration) and returns a Long integer. This Long will have bits, or flags, set to indicate certain facts about the system. Here are some of the flags:

```
Const WF_CPU286 = &H2&
```

```
Const WF_CPU386 = &H4&
Const WF_CPU486 = &H8&
Const WF_STANDARD = &H10&
Const WF_ENHANCED = &H20&
Const WF_80x87 = &H400&
```

Place the constants in the Declarations section of the form or module where you declare the GetWinFlags function.

Now you can call GetWinFlags and use the And operator with these constants to test the value returned. For example:

```
Dim WinFlags As Long
WinFlags = GetWinFlags()
If WinFlags And WF_ENHANCED Then
   Print "Windows Enhanced Mode ";
Else
   Print "Windows Standard Mode ";
End If
If WinFlags And WF_CPU486 Then Print "on a 486"
If WinFlags And WF_CPU386 Then Print "on a 386"
If WinFlags And WF_CPU286 Then Print "on a 286"
If WinFlags And WF_80x87 Then Print "Math coprocessor available"
```

There's one important fact about the system that this function does not provide: the version of Windows. You can obtain that information with the GetVersion function:

```
Declare Function GetVersion Lib "Kernel" () as Long
```

This returns a Long integer containing the version numbers of MS-DOS and Windows. Here's the code that extracts the version information:

```
Dim Ver As Long, WinVer As Long, DosVer As Long
Ver = GetVersion()
WinVer = Ver And &HFFFF
Print "Windows " + Format$(WinVer Mod 256) + "." + Format$(WinVer \ 256)
DosVer = Ver \ &H10000
Print "MS-DOS " + Format$(DosVer \ 256) + "." + Format$(DosVer Mod 256)
```

GetSystemMetrics is another Windows function that provides useful system information. You declare it like this:

```
Declare Function GetSystemMetrics Lib "User" (ByVal nIndex%) As Integer
```

This function is located in the "User" DLL. It takes one argument: an integer indicating which item of system information you want it to return. This argument, like most arguments to Windows API functions, is passed by value. Because Visual Basic usually passes arguments by reference, you have to include the ByVal keyword to specify the argument should be passed by value. This is very important. Forgetting ByVal when it is needed or including it when it isn't often leads to problems.

GetSystemMetrics provides a potpourri of information. For example, you can use it to find out if a mouse is installed in the system with code like this:

```
Const SM_MOUSEPRESENT = 19
If GetSystemMetrics(SM_MOUSEPRESENT) Then Print "Mouse installed"
```

Some other useful information provided by GetSystemMetrics is the size of the arrow bitmaps used by standard horizontal and vertical scroll bars. This is important because the size of these bitmaps varies with the resolution of the display and the display driver installed. When you create an application that uses a horizontal scroll bar control, you usually give it a fixed height; likewise, when you create a form with a vertical scroll bar control, you usually give it a fixed width. You fix these sizes based on what looks good on your system. Unfortunately, what looks good on your display can look strange on a display that has a different resolution. If you are writing applications that need to look good on a variety of display resolutions, you need to write code that can determine the standard size of scroll bars on the current display and dynamically resize your scroll bar controls to match. You need to write code like this:

```
Const SM_CXVSCROLL = 2
Const SM_CYHSCROLL = 3
ScaleMode = 3 'Pixels
VScroll1.Width = GetSystemMetrics(SM_CXVSCROLL)
HScroll1.Height = GetSystemMetrics(SM_CYHSCROLL)
```

Notice that the values returned by the GetSystemMetrics function are always in pixels, so you need to set the ScaleMode of the form to 3 (pixels) before setting the sizes of the scroll bars.

There are a lot of other system values you can obtain using GetSystemMetrics, but not all of them are useful to Visual Basic programmers. Here are a few of the interesting ones:

```
Const SM_CXSCREEN = 0   'Width of screen in pixels
Const SM_CYSCREEN = 1   'Height of screen in pixels
Const SM_CYCAPTION = 4  'Height of form titlebar in pixels
Const SM_CXICON = 11    'Width of icon in pixels
Const SM_CYICON = 12    'Height of icon in pixels
Const SM_CXCURSOR = 13  'Width of mousepointer in pixels
Const SM_CYCURSOR = 14  'Height of mousepointer in pixels
Const SM_CYMENU = 15    'Height of top menu bar in pixels
```

Yet another function that provides system information is GetDeviceCaps. This function returns information about a particular device in the system, such as the printer or the display. Like many of the functions you will see in this article, the declaration for GetDeviceCaps is too long to fit on one line, but you must type it all on one, single line:

```
' Enter the following Declare statement as one, single line:
Declare Function GetDeviceCaps Lib "GDI" (ByVal hDC%, ByVal nIndex%)
   As Integer
```

GetDeviceCaps is found in the "GDI" DLL and it takes two arguments. The first allows you to specify the device for which you want information. When calling the function from Visual Basic, supply either the hDC property of a form or the hDC property of the Printer object. The second argument specifies the device information you want to get. There are a lot of possible values for this second argument, but only a couple of them are very interesting to the Visual Basic programmer. For example, you can find

out how many colors the screen or printer supports:

```
Const PLANES = 14
Const BITSPIXEL = 12
Dim Cols As Long
Cols = GetDeviceCaps(hDC, PLANES) * 2 ^ GetDeviceCaps(hDC, BITSPIXEL)
```

The number of colors a device supports is the product of the number of color planes it has and the number of bits per pixel in each plane. Because each bit can represent two colors, you have to raise 2 to the power of the number of bits per pixel, and then multiply that by the number of color planes, to get the total number of colors that the device can display.

Some Useful Tricks

That's enough poking about in the system. Here are some useful tricks. If you have used the Shell function in Visual Basic, you have probably discovered that it will only run files that have the extension .EXE, .COM, .PIF, or .BAT. But you can double-click almost any file in the File Manager, and Windows does the right thing. For example, if it is a .TXT file, Windows starts Notepad. How would you add this kind of functionality to your own applications?

Windows stores the association between data files and their related application (such as the association between a .TXT file and the NOTEPAD application) in the WIN.INI file in the Extensions section. Windows also provides a function called GetProfileString that reads the WIN.INI file for you. Here is the Declare for GetProfileString:

```
' Enter the following Declare statement on one, single line:
Declare Function GetProfileString Lib "Kernel"
   (ByVal Sname$, ByVal Kname$, ByVal Def$, ByVal Ret$, ByVal Size%)
   As Integer
```

GetProfileString searches the WIN.INI section specified in the first argument for the key specified in the second argument and returns the value for that key in the third argument. The fourth argument provides the length of the string passed as the third argument.

Therefore, once you know the extension of a file, you can use GetProfileString to find the parent application for files with that extension. Here's a function that does that:

```
Function FindApp (Ext As String) As String
   ' Find the parent app for a file with the given extension
   Dim Sname As String, Ret As String, Default As String
   Ret = String$(255, 0)
   Default = Ret
   Sname = "Extensions"
   nSize = GetProfileString(Sname, Ext, Default, Ret, Len(Ret))
   If Left$(Ret, 1) <> Chr$(0) Then
      FindApp = Mid$(Ret, 1, InStr(Ret, "^") - 1)
   End If
End Function
```

GetProfileString is an example of a Windows function that returns a string

by modifying one of its arguments. To use these kinds of functions, you must create a string and fill it with something (character code 0 in the example above) before you call the function. This is because Windows cannot enlarge strings the way Visual Basic can, so whenever you pass a string to Windows you must ensure that it is long enough to hold the largest possible string that Windows might return.

You can add new values to WIN.INI using the WriteProfileString function:

```
' Enter the following Declare statement on one, single line:
Declare Function WriteProfileString Lib "Kernel"
   (ByVal Sname$, ByVal Kname$, ByVal Set$) As Integer
```

This function searches the WIN.INI file for the section specified in the first argument and the key specified in the second argument. Then it replaces the key value with the value specified in the third argument. If the key is not found, it adds the key and its value to the specified section. If it does not find the section, it adds that to WIN.INI as well.

Some applications use their own private .INI files rather than using WIN.INI -- Visual Basic has its own VB.INI, for example. You can use the functions GetPrivateProfileString and WritePrivateProfileString to manipulate other .INI files:

```
' Enter each of the following Declare statements on one, single line:
Declare Function GetPrivateProfileString Lib "Kernel"
   (ByVal Sname$, ByVal Kname$, ByVal Def$, ByVal Ret$, ByVal Size%,
   ByVal Fname$) As Integer
Declare Function WritePrivateProfileString Lib "Kernel"
   (ByVal Sname$, ByVal Kname$, ByVal Set$, ByVal Fname$) As Integer
```

These work exactly like GetProfileString and WriteProfileString, except they have one additional argument that specifies the path and filename of the .INI file.

Now, where should you put that custom .INI file? One obvious place is the Windows directory. But people name that directory all sorts of things: \WINDOWS or \WIN3 or who knows what. It might not even be at the root level. How can you find it? Well, Windows knows where this directory is, and it provides a function to tell you:

```
' Enter the following Declare statement on one, single line:
Declare Function GetWindowsDirectory Lib "User"
   (ByVal P$, ByVal S%) As Integer
```

The first argument is a string that the function will fill with the path to the Windows directory; the second argument is the length of this string. Again, because you are passing a string to be filled by Windows, you must make sure it is large enough to accommodate whatever string Windows might provide. In this case, the Windows Reference warns that it should be at least 144 characters. On the other hand, 144 characters is the worst case so in most cases there will be a lot of unused characters that you will need to trim off. The GetWindowsDirectory function returns an integer value that indicates the actual length of the returned string. So here's some fancy code that calls the function and trims the returned string all in one line:

```
Dim WinPath As String
WinPath = String$(145, Chr$(0))
WinPath = Left$(WinPath, GetWindowsDirectory(WinPath, Len(WinPath)))
```

In addition, there is usually a \SYSTEM directory within the windows directory. Once again, that could be called anything, and once again Windows provides a function to find it: GetSystemDirectory.

This function is declared in exactly the same way as GetWindowsDirectory and can be called in the same way, so substitute it in the declaration and code above and try it out.

Another sensible place to put that .INI file is in the same directory as the application. It should be trivial to figure out what directory a running Visual Basic application is stored in, but it's not. This information isn't provided through the Command$ system variable, unfortunately. And even CurDir$ isn't reliable because the application could have been run using a full path without changing the current directory to the application's directory. Windows API calls to the GetModuleHandle and GetModuleFileName functions give you what you need. Here are the declarations:

```
' Enter each of the following Declare statement on one, single line:
Declare Function GetModuleHandle Lib "Kernel"
   (ByVal FileName$) As Integer
Declare Function GetModuleFileName Lib "Kernel"
   (ByVal hModule%, ByVal FileName$, ByVal nSize%) As Integer
```

GetModuleHandle takes the filename of a running program and returns a "module handle." All you need to know about the module handle is that it is an integer and you pass it to the GetModuleFileName function.

GetModuleFilename takes three arguments; the first is the module handle returned by GetModuleHandle. The second is a string that the function fills with the complete path and filename of the program specified by the module handle. The third is the size of this string. The value returned by GetModuleFilename is the length of the path that it placed in the string you passed.

Using these two functions, obtaining the path to a running program is easy:

```
Dim hMod As Integer, Path As String
hMod = GetModuleHandle%("MyApp.EXE")
Path = String$(145, Chr$(0))
Path = Left$(Path, GetModuleFileName%(hMod, Path, Len(Path))
```

Notice that you are again passing a large string and using the value returned by the function to trim the string down to size with Left$.

Another trick you can perform with GetModuleHandle is limiting your application to a single instance. Normally, Windows allows you to run multiple instances (copies) of the same program. Most of the time this is a handy feature, but sometimes it can cause problems. If your program uses data
files, having more than one instance of the program accessing those files at the same time can leave the files in an inconsistent state. You could write the program so that it works correctly even if there are multiple

copies running, but that's a lot of work and sometimes it's not even possible. An easier method is to just ensure that only one copy of the program can be run. Thanks to GetModuleHandle and another Windows function, GetModuleUsage, this is easy:

```
Declare Function GetModuleUsage Lib "Kernel" (ByVal hModule%) As Integer
```

GetModuleUsage returns how many instances of the specified program exist. The program is specified by passing GetModuleUsage a module handle, which is what GetModuleHandle returns. Putting code like this in the Form_Load event for your startup form (or in your Sub Main if you don't have a startup form) ensures that only a single instance of your application can be run:

```
If GetModuleUsage(GetModuleHandle("YOURAPP.EXE")) > 1 Then
   MsgBox "This program is already loaded!", 16
   End
End If
```

GetModuleHandle and GetModuleUsage work for DLLs as well as ordinary executable files, so you could use this technique to find out how many Visual Basic executables are running by using GetModuleUsage with the module handle for VBRUN100.DLL.

Sending Messages to Controls

Windows is built around messages. The Windows system sends messages to applications. You see these messages in your Visual Basic program as events. In addition, applications send messages to each other (this is the basis for DDE), and applications even send messages to themselves.

You can get in on the action. By sending messages to controls, you can get them to do things that would otherwise be impossible, such as setting tab stops in list boxes and getting a single line of text from a multi-line text box. You can also do things by sending a single message that would take a lot more Basic code to accomplish, such as emptying a list box. You send messages with SendMessage function:

```
' Enter the following Declare statement as one, single line:
Declare Function SendMessage Lib "User"
   (ByVal hWnd%, ByVal msg%, ByVal wp%, lp As Any) As Long
```

The first argument identifies the recipient of the message. Windows uses "handles" to keeps track of everything it uses. Handles are integer ID numbers that Windows assigns to things -- like the module handles used earlier to refer to programs.

Controls are just another kind of window as far as Windows is concerned. Controls are identified by their window handle, or hWnd. To send a message to a control you need its hWnd.

Visual Basic provides the hWnd for a form through the hWnd property. Unfortunately, there is no such property for any of the controls. The controls are windows and do have hWnds, but Visual Basic doesn't provide them for you. So you have to resort to some subterfuge. Windows has a function called GetFocus that will return the hWnd for the window that has

the focus:

```
Declare Function GetFocus Lib "user" () As Integer
```

And we can give the focus to any control using the Visual Basic SetFocus method. So to get the hWnd for a control, use code similar to this:

```
AnyControl.SetFocus
control_hWnd = GetFocus()
```

The second argument in the SendMessage function is the message number. All of the message numbers are some offset from the WM_USER message, which has the value 1024 (&H400 in hexadecimal notation). The complete list of message numbers is included in the WINAPI.TXT file.

The last two arguments in SendMessage supply additional information for a particular message. What they contain varies from message to message. Notice that the last argument was declared As Any. This is different from the way the SendMessage function is declared in the WINAPI.TXT file. It allows you to pass any data type as the fourth argument.

The Long integer value that SendMessage returns depends on what message you sent. Sometimes you send a message to tell a control to do something, and the return value is zero if the control could perform the action and non-zero if it could not. Sometimes you send a message to a control to find out something about that control, and in those cases the return value is the information you requested. And sometimes the return value means nothing at all.

Try out SendMessage by starting with something simple: emptying list boxes. If you've spent much time programming with list boxes, you are probably annoyed that there is no simple way to empty a list. Instead of telling the list box to simply empty itself, you have to loop through all the entries and use the RemoveItem method on each one. But there's a better way.

Windows provides a message (LB_RESETCONTENT) that you can send to a list box to make it empty itself in one step.

```
Const WM_USER = &H400
Const LB_RESETCONTENT = WM_USER + 5
```

Here is a procedure that uses this message to empty any list box:

```
Sub ClearListBox(Ctrl As Control)
   Ctrl.SetFocus
   dummy& = SendMessage(GetFocus(), LB_RESETCONTENT, 0, ByVal 0&)
End Sub
```

The RESETCONTENT message needs no additional information, so the last two arguments to SendMessage are zero. Notice that the last argument is ByVal 0& (zero followed by an ampersand character). The ampersand is very important; it ensures that a long (32 bit) zero is passed. There is no useful information returned when this message is sent, so the SendMessage function is assigned to a dummy variable.

You can also empty combo box lists in the same way; just use this constant for the message:

```
Const CB_RESETCONTENT = WM_USER+11
```

While on the topic of lists, there's an easy way to find strings in a list. You can loop through all the items in the list, but why bother when the LB_FINDSTRING message allows you to find a string in a list box with a single function call? When you send the LB_FINDSTRING message, the SendMessage function returns the index of the first item in the list that matches the string you specified (so obviously you should only use this message with sorted list boxes).

```
Const LB_FINDSTRING = WM_USER + 16
Dim itemNum As Long
itemNum = SendMessage(GetFocus(), LB_FINDSTRING, -1, ByVal "Visual")
Print "Windows is item: "; Format$(itemNum)
```

This finds the first list item that begins with "Visual" and returns its index in the list. It will match even if there are additional characters following the specified string, so the example above would match "Visual Basic" if it was the first string in the list beginning with "Visual." Again, this technique works as well with combo boxes as it works with list boxes. Just use the message:

```
Const CB_FINDSTRING = WM_USER + 12
```

And, speaking of combo boxes, here's a neat trick for a dropdown list combo box (a combo box with the Style property set to 2). This code drops the list automatically when the combo box gets the focus:

```
Sub Combo1_GotFocus ()
   Const CB_SHOWDROPDOWN = WM_USER + 15
   Dummy& =  SendMessage(GetFocus(), CB_SHOWDROPDOWN, 1, ByVal 0&)
End Sub
```

Sending messages in the GotFocus event for a control is a good technique because it allows you to avoid explicitly setting the focus to a control to get its hWnd.

Another useful message is LB_GETTOPINDEX. When you send this message to a list box, the SendMessage function returns the index of the first visible item in the list. This is valuable if the list has been scrolled and you want to determine which items are actually visible in the list (in a DragDrop event, for example).

```
Const LB_GETTOPINDEX = WM_USER+15
FirstItem& = SendMessage(GetFocus(), LB_GETTOPINDEX, 0, ByVal 0&)
```

You can also send a message to scroll a list box to make any item the first visible item in the list:

```
Const LB_SETTOPINDEX = WM_USER+24
Success& = SendMessage(GetFocus(), LB_SETTOPINDEX, item%, ByVal 0&)
```

You could combine this message with the LB_FINDSTRING message to scroll the list box so that the list item found by LB_FINDSTRING is at the top of the list.

One especially valuable use of SendMessage is to set the tabstops in a list or
text box. You have probably discovered that list boxes and multi-line text boxes handle tabs automatically, so if you assign text that contains tabs (character code 9) the columns line up automatically. Unfortunately, Visual Basic gives you no way to adjust where the columns fall -- except by sending a message. For a list box, the message is:

```
Const LB_SETTABSTOPS = WM_USER + 19
```

When you send this message, you must supply an array of integers that specify the new tab positions. (These positions are specified in terms of characters; when the list box or text box contains a proportional font, these are "average" characters.) This array is the fourth argument to the SendMessage function; the number of elements in the array is the third argument. Notice that to pass an array to a Windows function, you actually pass the first element of the array:

```
ReDim tabs(3) As Integer
tabs(1) = 10
tabs(2) = 50
tabs(3) = 90
List.SetFocus
dummy& = SendMessage(GetFocus(), LB_SETTABSTOPS, 0, ByVal 0&)
dummy& = SendMessage(GetFocus(), LB_SETTABSTOPS, 3, tabs(1))
```

The first call to SendMessage clears any existing tabstops; the second sets three tabstops as specified in the array.

You can set the tabstops in a multi-line text box as well; just send the message EM_SETTABSTOPS:

```
Const EM_SETTABSTOPS = WM_USER + 27
```

This won't work for a single-line text box. Speaking of multi-line text boxes, there are several useful messages you can send to a multi-line text box to get information that Visual Basic does not provide. For example, you can send the EM_GETLINECOUNT message to get the number of lines text in a multi-line text box:

```
Const EM_GETLINECOUNT = WM_USER+10
lineCount& = SendMessage(GetFocus(), EM_GETLINECOUNT, 0, ByVal 0&)
```

You can obtain the text contained in any line of a multi-line text box with the message:

```
Const EM_GETLINE = WM_USER + 20
```

When sending this message, you have to provide the number of the line you want to retrieve in the third argument to SendMessage, and the string to be filled with the contents of the line as the fourth argument. There's one odd thing about this string. Normally, when you pass a string to a Windows function you also supply the size of the string as an argument. However, the usual place for that information is in the third argument, and that is already being used to specify which line you want retrieved. So you have to place the length of the string in the first two bytes of the string, using code like this:

```
   Dim LineNum As Integer, linelength As Integer, buf As String
   'Set linelength to some reasonable value
   buf = String$(linelength, chr$(0))
   buf = Chr$(linelength Mod 256) + Chr$(linelength \ 256) + Buf
   ' Enter the following two lines as one, single line:
   buf = Left$(buf,
      SendMessage(GetFocus(), EM_GETLINE, lineNum, ByVal buf))
```

Another handy message for multi-line text boxes is EM_LINESCROLL, which allows you to scroll them horizontally and vertically. You specify the amount to scroll in the fourth argument of the SendMessage function: place the number of characters to scroll horizontally in the high word (by multiplying by 65536) and the number of lines to scroll vertically in the low order word. For example:

```
   Sub ScrollIt (ctl As Control, chars As Integer, lines As Integer)
      Const EM_LINESCROLL = WM_USER + 6
      Dim scroll As Long
      scroll = chars * 65536 + lines
      ctl.SetFocus
      dummy& = SendMessage(GetFocus(), EM_LINESCROLL, 0, ByVal scroll)
   End Sub
```

This is a relative scroll: if you use the value 2 the text box will scroll down by two lines; if you use the value -65536 the text box will scroll left by one character.

Another feature that Visual Basic does not directly support is a way to restrict the number of characters that can be entered in a text box. You can do this by responding to the various Key events, but there is an easier way: send the EM_LIMITTEXT message to the text box:

```
   Sub Text1_GotFocus()
      Const EM_LIMITTEXT = WM_USER+21
      dummy& = SendMessage(GetFocus(), EM_LIMITTEXT, numChars, ByVal 0&)
   End Sub
```

Here the third argument specifies the maximum number of characters the text box will accept. If you want to set it back to normal, send EM_LIMITTEXT with that argument set to zero. You can also restrict the number of characters accepted by a combo box by sending the combo box the message CB_LIMITTEXT:

```
   Const CB_LIMITTEXT = WM_USER+1
```

One last trick: turn a text box into a password control. Windows provides automatic support for text boxes that display asterisks (or some other character) instead of the actual characters the user types. To take advantage of this support, set a style bit in the text box. Normally you set style bits when you create a control, but you can't do that because Visual Basic creates the control for you.

Fortunately, Windows allows you to set that bit after the control is created (this is one of the few style bits that you can change after a control is created). The functions that get and set the style information for a window are as follows:

```
' Enter each of the following Declare statements as one, single line:
Declare Function GetWindowLong Lib "User"
   (ByVal hWnd%, ByVal nIndex%) As Long
Declare Function SetWindowLong Lib "User"
   (ByVal hWnd%, ByVal nIndex%, ByVal NewLong&) As Long
```

To set the password style bit, call GetWindowLong to get the style information, use the Or operator to set the bit, and then call SetWindowLong to store the new style. Once again, do this in the GotFocus event so you don't have to worry about using SetFocus to get the hWnd:

```
Sub Text1_GotFocus ()
   Const ES_PASSWORD = &H20
   Const EM_SETPASSWORD = 1052
   Const GWL_STYLE = -16
   Const Asterisk = 42
   Dim TxthWnd As Integer, WindowLong As Long
   TxthWnd = GetFocus()
   WindowLong = GetWindowLong(TxthWnd, GWL_STYLE)
   WindowLong = WindowLong Or ES_PASSWORD
   WindowLong = SetWindowLong(TxthWnd, GWL_STYLE, WindowLong)
   WindowLong = SendMessage(TxthWnd, EM_SETPASSWORD, Asterisk, ByVal 0&)
End Sub
```

You can define the character you want displayed in place of the actual characters the user types by sending the EM_SETPASSWORD message to the control. This example sets the password character to asterisks. If you want to use a different character, supply a different ANSI character code when you send the EM_SETPASSWORD message.

There are some limitations to this password functionality. For one thing, the characters in the text box are not stored as asterisks; they are just displayed that way. This is good, because it allows your code to easily check what the user typed. But it is also bad, because any user can select the contents of the text box, copy it, and paste it somewhere else and see the actual characters that were typed in the text box. Whether you consider this a flaw depends on whether you expect the text box containing a password to sit around on a screen where some malicious users can copy it. Usually, this is not a problem. But it is something to keep in mind.

Exploring on Your Own

The Windows API is like an enormous hidden world that is just waiting to be explored. The API documentation is the treasure map, and like the maps in all good adventure stories, it requires some translation to be useful. And you aren't limited to just the Windows API. Almost any DLL contains functions that you might find useful. For example, many spell checkers are implemented as DLLs; if you know how to declare and call the functions in one of these DLLs, you can add spell-checking to your Visual Basic programs (assuming that you obey the copyright restrictions for the DLL, of course).

Finding out how to declare and call the DLL functions can be tough sometimes, however. You'll learn to keep your eyes open for anything that looks like API documentation. Who knows what treasure you'll discover inside some obscure DLL? And as new versions of Windows appear, the

treasure will only increase. The Multimedia Extensions for Microsoft Windows are just a collection of DLLs, after all. With the right hardware, think of how much fun you'll have calling those from your Visual Basic programs.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: APrgOther RefsProd

LONG: Microsoft Consulting Services Naming Conventions for VB

Article ID: Q110264

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

It is a good idea to establish naming conventions for your Visual Basic code. This article gives you the naming conventions used by Microsoft Consulting Services (MCS).

This document is a superset of the Visual Basic coding conventions found in the Visual Basic "Programmer's Guide."

NOTE: The third-party controls mentioned in this article are manufactured by vendors independent of Microsoft. Microsoft makes no warranty, implied or otherwise, regarding these controls' performance or reliability.

MORE INFORMATION

Naming conventions help Visual Basic programmers:

- Standardize the structure, coding style and logic of an application.
- Create precise, readable, and unambiguous source code.
- Be consistent with other language conventions (most importantly, the Visual Basic Programmers Guide and standard Windows C Hungarian notation).
- Be efficient from a string size and labor standpoint, thus allowing a greater opportunity for longer and fuller object names.
- Define the minimal requirements necessary to do the above.

Setting Environment Options

Use Option Explicit. Declare all variables to save programming time by reducing the number of bugs caused by typos (for example, aUserNameTmp vs. sUserNameTmp vs. sUserNameTemp). In the Environment Options dialog, set Require Variable Declaration to Yes. The Option Explicit statement requires you to declare all the variables in your Visual Basic program.

Save Files as ASCII Text. Save form (.FRM) and module (.BAS) files as ASCII text to facilitate the use of version control systems and minimize the damage that can be caused by disk corruption. In addition, you can:

- Use your own editor
- Use automated tools, such as grep
- Create code generation or CASE tools for Visual Basic
- Perform external analysis of your Visual Basic code

To have Visual Basic always save files as ASCII text, from the Environment Options dialog, set the Default Save As Format option to Text.

Object Naming Conventions for Standard Objects

The following tables define the MCS standard object name prefixes. These prefixes are consistent with those documented in the Visual Basic version 3.0 Programmers Guide.

Prefix	Object Type	Example
ani	Animation button	aniMailBox
bed	Pen Bedit	bedFirstName
cbo	Combo box and drop down list box	cboEnglish
chk	Checkbox	chkReadOnly
clp	Picture clip	clpToolbar
cmd (3d)	Command button (3D)	cmdOk (cmd3dOk)
com	Communications	comFax
ctr	Control (when specific type unknown)	ctrCurrent
dat	Data control	datBiblio
dir	Directory list box	dirSource
dlg	Common dialog control	dlgFileOpen
drv	Drive list box	drvTarget
fil	File list box	filSource
frm	Form	frmEntry
fra (3d)	Frame (3d)	fraStyle (fra3dStyle)
gau	Gauge	gauStatus
gpb	Group push button	gpbChannel
gra	Graph	graRevenue
grd	Grid	grdPrices
hed	Pen Hedit	hedSignature
hsb	Horizontal scroll bar	hsbVolume
img	Image	imgIcon
ink	Pen Ink	inkMap
key	Keyboard key status	keyCaps
lbl	Label	lblHelpMessage
lin	Line	linVertical
lst	List box	lstPolicyCodes
mdi	MDI child form	mdiNote
mpm	MAPI message	mpmSentMessage
mps	MAPI session	mpsSession
mci	MCI	mciVideo
mnu	Menu	mnuFileOpen
opt (3d)	Option Button (3d)	optRed (opt3dRed)
ole	OLE control	oleWorksheet
out	Outline control	outOrgChart
pic	Picture	picVGA
pnl3d	3d Panel	pnl3d
rpt	Report control	rptQtr1Earnings
shp	Shape controls	shpCircle
spn	Spin control	spnPages
txt	Text Box	txtLastName
tmr	Timer	tmrAlarm
vsb	Vertical scroll bar	vsbRate

Object Naming Convention for Database Objects

Prefix	Object Type	Example
db	ODBC Database	dbAccounts
ds	ODBC Dynaset object	dsSalesByRegion
fdc	Field collection	fdcCustomer
fd	Field object	fdAddress
ix	Index object	ixAge
ixc	Index collection	ixcNewAge
qd	QueryDef object	qdSalesByRegion
qry (suffix)	Query (see NOTE)	SalesByRegionQry
ss	Snapshot object	ssForecast
tb	Table object	tbCustomer
td	TableDef object	tdCustomers

NOTE: Using a suffix for queries allows each query to be sorted with its associated table in Microsoft Access dialogs (Add Table, List Tables Snapshot).

Menu Naming Conventions

Applications frequently use an abundance of menu controls. As a result, you need a different set of naming conventions for these controls. Menu control prefixes should be extended beyond the initial mnu label by adding an additional prefix for each level of nesting, with the final menu caption at the end of the name string. For example:

Menu Caption Sequence	Menu Handler Name
Help.Contents	mnuHelpContents
File.Open	mnuFileOpen
Format.Character	mnuFormatCharacter
File.Send.Fax	mnuFileSendFax
File.Send.Email	mnuFileSendEmail

When this convention is used, all members of a particular menu group are listed next to each other in the object drop-down list boxes (in the code window and property window). In addition, the menu control names clearly document the menu items to which they are attached.

Naming Conventions for Other Controls

For new controls not listed above, try to come up with a unique three character prefix. However, it is more important to be clear than to stick to three characters.

For derivative controls, such as an enhanced list box, extend the prefixes above so that there is no confusion over which control is really being used. A lower-case abbreviation for the manufacturer would also typically be added to the prefix. For example, a control instance created from the Visual Basic Professional 3D frame could uses a prefix of fra3d to avoid confusion over which control is really being used. A command button from MicroHelp could use cmdm to differentiate it from the standard command button (cmd).

Third Party Controls

Each third party control used in an application should be listed in the application's overview comment section, providing the prefix used for the control, the full name of the control, and the name of the software vendor:

Prefix	Control Type	Vendor
cmdm	Command Button	MicroHelp

Variable and Routine Naming

Variable and function names have the following structure:

<prefix><body><qualifier><suffix>

Part	Description	Example
<prefix>	Describes the use and scope of the variable.	iGetRecordNext
<body>	Describes the variable.	iGetNameFirst
<qualifier>	Denotes a derivative of the variable.	iGetNameLast
<suffix>	The optional Visual Basic type character.	iGetRecordNext%

Prefixes:

The following tables define variable and function name prefixes that are based on Hungarian C notation for Windows. These prefixes should be used with all variables and function names. Use of old Basic suffixes (such as %, &, #, etc.) are discouraged.

Variable and Function Name Prefixes:

Prefix	Converged	Variable Use	Data Type	Suffix
b	bln	Boolean	Integer	%
c	cur	Currency - 64 bits	Currency	@
d	dbl	Double - 64 bit signed quantity	Double	#
dt	dat	Date and Time	Variant	
e	err	Error		
f	sng	Float/Single - 32 bit signed floating point	Single	!
h		Handle	Integer	%
i		Index	Integer	%
l	lng	Long - 32 bit signed quantity	Long	&
n	int	Number/Counter	Integer	%
s	str	String	String	$
u		Unsigned - 16 bit unsigned quantity	Long	&
	udt	User-defined type		
vnt	vnt	Variant	Variant	
a		Array		

NOTE: the values in the Converged column represent efforts to pull together the naming standards for Visual Basic, Visual Basic for Applications, and Access Basic. It is likely that these prefixes will become Microsoft standards at some point in the near future.

Scope and Usage Prefixes:

Prefix	Description
g	Global
m	Local to module or form
st	Static variable
(no prefix)	Non-static variable, prefix local to procedure
v	Variable passed by value (local to a routine)
r	Variable passed by reference (local to a routine)

Hungarian notation is as valuable in Visual Basic as it is in C. Although the Visual Basic type suffixes do indicate a variable's data type, they do not explain what a variable or function is used for, or how it can be accessed. Here are some examples:

iSend - Represents a count of the number of messages sent
bSend - A Boolean flag defining the success of the last Send operation
hSend - A Handle to the Comm interface

Each of these variable names tell a programmer something very different. This information is lost when the variable name is reduced to Send%. Scope prefixes such as g and m also help reduce the problem of name contention especially in multi-developer projects.

Hungarian notation is also widely used by Windows C programmers and constantly referenced in Microsoft product documentation and in industry programming books. Additionally, the bond between C programmers and programmers who use Visual Basic will become much stronger as the Visual C++ development system gains momentum. This transition will result in many Visual Basic programmers moving to C for the first time and many programmers moving frequently back and forth between both environments.

The Body of Variable and Routine Names

The body of a variable or routine name should use mixed case and should be as long as necessary to describe its purpose. In addition, function names should begin with a verb, such as InitNameArray or CloseDialog.

For frequently used or long terms, standard abbreviations are recommended to help keep name lengths reasonable. In general, variable names greater than 32 characters can be difficult to read on VGA displays.

When using abbreviations, make sure they are consistent throughout the entire application. Randomly switching between Cnt and Count within a project will lead to unnecessary confusion.

Qualifiers on Variable and Routine Names

Related variables and routines are often used to manage and manipulate a common object. In these cases, use standard qualifiers to label the derivative variables and routines. Although putting the qualifier after the body of the name might seem a little awkward (as in sGetNameFirst, sGetNameLast instead of sGetFirstName, sGetLastName), this practice will help order these names together in the Visual Basic editor routine lists, making the logic and structure of the application easier to understand.

The following table defines common qualifiers and their standard meaning:

Qualifier	Description (follows Body)
First	First element of a set.
Last	Last element of a set.
Next	Next element in a set.
Prev	Previous element in a set.
Cur	Current element in a set.
Min	Minimum value in a set.
Max	Maximum value in a set.
Save	Used to preserve another variable that must be reset later.
Tmp	A "scratch" variable whose scope is highly localized within the code. The value of a Tmp variable is usually only valid across a set of contiguous statements within a single procedure.
Src	Source. Frequently used in comparison and transfer routines.
Dst	Destination. Often used in conjunction with Source.

User Defined Types

Declare user defined types in all caps with _TYPE appended to the end of the symbol name. For example:

```
Type CUSTOMER_TYPE
   sName As String
   sState As String * 2
   lID as Long
End Type
```

When declaring an instance variable of a user defined type, add a prefix to the variable name to reference the type. For example:

```
Dim custNew as CUSTOMER_TYPE
```

Naming Constants

The body of constant names should be UPPER_CASE with underscores (_) between words. Although standard Visual Basic constants do not include Hungarian information, prefixes like i, s, g, and m can be very useful in understanding the value and scope of a constant. For constant names, follow the same rules as variables. For Example:

```
mnUSER_LIST_MAX   ' Max entry limit for User list (integer value,
                  ' local to module)
gsNEW_LINE        ' New Line character string (global to entire
                  ' application)
```

Variant Data Type

If you know that a variable will always store data of a particular type, Visual Basic can handle that data more efficiently if you declare a variable of that type.

However, the variant data type can be extremely useful when working with databases, messages, DDE, or OLE. Many databases allow NULL as a valid value for a field. Your code needs to distinguish between NULL, 0 (zero), and "" (empty string). Many times, these types of operations can use a generic service routine that does not need to know the type of data it receives to process or pass on the data.

For example:

```
Sub ConvertNulls(rvntOrg As Variant, rvntSub As Variant)
   ' If rvntOrg = Null, replace the Null with rvntSub
   If IsNull(rvntOrg) Then rvntOrg = rvntSub
End Sub
```

The are some drawbacks, however, to using variants. Code statements that use variants can sometimes be ambiguous to the programmer.

For example:

```
vnt1 = "10.01" : vnt2 = 11 : vnt3 = "11" : vnt4 = "x4"
vntResult = vnt1 + vnt2  ' Does vntResult = 21.01 or 10.0111?
vntResult = vnt2 + vnt1  ' Does vntResult = 21.01 or 1110.01?
vntResult = vnt1 + vnt3  ' Does vntResult = 21.01 or 10.0111?
vntResult = vnt3 + vnt1  ' Does vntResult = 21.01 or 1110.01?
vntResult = vnt2 + vnt4  ' Does vntResult = 11x4 or ERROR?
vntResult = vnt3 + vnt4  ' Does vntResult = 11x4 or ERROR?
```

The above examples would be much less ambiguous and easier to read, debug, and maintain if the Visual Basic type conversion routines were used instead.

For Example:

```
iVar1 = 5 + val(sVar2)   ' use this (explicit conversion)
vntVar1 = 5 + vntVar2    ' not this (implicit conversion)
```

Commenting Your Code

All procedures and functions should begin with a brief comment describing the functional characteristics of the routine (what it does). This description should not describe the implementation details (how it does it) because these often change over time, resulting in unnecessary comment maintenance work, or worse yet, erroneous comments. The code itself and any necessary in-line or local comments will describe the implementation.

Parameters passed to a routine should be described when their functions are not obvious and when the routine expects the parameters to be in a specific range. Function return values and global variables that are changed by the

routine (especially through reference parameters) must also be described at the beginning of each routine.

Routine header comment blocks should look like this (see the next section "Formatting Your Code" for an example):

Section	Comment Description
Purpose	What the routine does (not how).
Inputs	Each non-obvious parameter on a separate line with in-line comments
Assumes	List of each non-obvious external variable, control, open file, and so on.
Returns	Explanation of value returned for functions.
Effects	List of each effected external variable, control, file, and so on and the affect it has (only if this is not obvious)

Every non-trivial variable declaration should include an in-line comment describing the use of the variable being declared.

Variables, controls, and routines should be named clearly enough that in-line commenting is only needed for complex or non-intuitive implementation details.

An overview description of the application, enumerating primary data objects, routines, algorithms, dialogs, database and file system dependencies, and so on should be included at the start of the .BAS module that contains the project's Visual Basic generic constant declarations.

NOTE: The Project window inherently describes the list of files in a project, so this overview section only needs to provide information on the most important files and modules, or the files the Project window doesn't list, such as initialization (.INI) or database files.

Formatting Your Code

Because many programmers still use VGA displays, screen real estate must be conserved as much as possible, while still allowing code formatting to reflect logic structure and nesting.

Standard, tab-based, block nesting indentations should be two to four spaces. More than four spaces is unnecessary and can cause statements to be hidden or accidentally truncated. Less than two spaces does not sufficiently show logic nesting. In the Microsoft Knowledge Base, we use a three-space indent. Use the Environment Options dialog to set the default tab width.

The functional overview comment of a routine should be indented one space. The highest level statements that follow the overview comment should be indented one tab, with each nested block indented an additional tab.

For example:

```
'*********************************************************************
'Purpose:   Locate first occurrence of a specified user in UserList array.
```

```
'Inputs:    rasUserList():  the list of users to be searched
'           rsTargetUser:   the name of the user to search for
'Returns:   the index of the first occurrence of the rsTargetUser
'           in the rasUserList array. If target user not found, return -1.
'**************************************************************************
'Enter the next two lines as one, single line:
Function iFindUser (rasUserList() As String, rsTargetUser as String)
   As Integer
   Dim i As Integer           ' loop counter
   Dim bFound As Integer      ' target found flag
   iFindUser = -1
   i = 0
   While i <= Ubound(rasUserList) and Not bFound
      If rasUserList(i) = rsTargetUser Then
         bFound = True
         iFindUser = i
      End If
   Wend
End Function
```

Variables and non-generic constants should be grouped by function rather than by being split off into isolated areas or special files. Visual Basic generic constants such as HOURGLASS should be grouped in a single module (VB_STD.BAS) to keep them separate from application-specific declarations.

Operators

Always use an ampersand (&) when concatenating strings, and use the plus sign (+) when working with numerical values. Using a plus sign (+) with nonnumerical values, may cause problems when operating on two variants.

For example:

```
   vntVar1 = "10.01"
   vntVar2 = 11
   vntResult = vntVar1 + vntVar2         ' vntResult =  21.01
   vntResult = vntVar1 & vntVar2         ' vntResult = 10.0111
```

Scope

Variables should always be defined with the smallest scope possible. Global variables can create enormously complex state machines and make the logic of an application extremely difficult to understand. Global variables also make the reuse and maintenance of your code much more difficult.

Variables in Visual Basic can have the following scope:

Scope	Variable Declared In:	Visibility
Procedure-level	Event procedure, sub, or function	Visible in the procedure in which it is declared
Form-level,	Declarations section of a form	Visible in every

Module-level	or code module (.FRM, .BAS)	procedure in the form or code module
Global	Declarations section of a code module (.BAS, using Global keyword)	Always visible

In a Visual Basic application, only use global variables when there is no other convenient way to share data between forms. You may want to consider storing information in a control's Tag property, which can be accessed globally using the form.object.property syntax.

If you must use global variables, it is good practice to declare all of them in a single module and group them by function. Give the module a meaningful name that indicates its purpose, such as GLOBAL.BAS.

With the exception of global variables (which should not be passed), procedures and functions should only operate on objects that are passed to them. Global variables that are used in routines should be identified in the general comment area at the beginning of the routine. In addition, pass arguments to subs and functions using ByVal, unless you explicitly want to change the value of the passed argument.

Write modular code whenever possible. For example, if your application displays a dialog box, put all the controls and code required to perform the dialog's task in a single form. This helps to keep the application's code organized into useful components and minimizes its runtime overhead.

Third Party Controls

NOTE: The products discussed below are manufactured by vendors independent of Microsoft. Microsoft makes no warranty, implied or otherwise, regarding these products' performance or reliability.

The following table lists standard third party vendor name prefix characters to be used with control prefixes:

Vendor	Abbv
MicroHelp (VBTools)	m
Pioneer Software	p
Crescent Software	c
Sheridan Software	s
Other (Misc)	o

The following table lists standard third party control prefixes:

Control Type	Control Name	Abbr	Vendor	Example	VBX File Name
Alarm	Alarm	almm	MicroHelp	almmAlarm	MHTI200.VBX
Animate	Animate	anim	MicroHelp	animAnimate	MHTI200.VBX
Callback	Callback	calm	MicroHelp	calmCallback	MHAD200.VBX
Combo Box	DB_Combo	cbop	Pioneer	cbopComboBox	QEVBDBF.VBX
Combo Box	SSCombo	cbos	Sheridan	cbosComboBox	SS3D2.VBX
Check Box	DB_Check	chkp	Pioneer	chkpCheckBox	QEVBDBF.VBX

Chart	Chart	chtm	MicroHelp	chtmChart	MHGR200.VBX
Clock	Clock	clkm	MicroHelp	clkmClock	MHTI200.VBX
Button	Command Button	cmdm	MicroHelp	cmdmCommandButton	MHEN200.VBX
Button	DB_Command	cmdp	Pioneer	cmdpCommandButton	QEVBDBF.VBX
Button (Group)	Command Button (multiple)	cmgm	MicroHelp	cmgmBtton	MHGR200.VBX
Button	Command Button (icon)	cmim	MicroHelp	cmimCommandButton	MHEN200.VBX
CardDeck	CardDeck	crdm	MicroHelp	crdmCard	MHGR200.VBX
Dice	Dice	dicm	MicroHelp	dicmDice	MHGR200.VBX
List Box (Dir)	SSDir	dirs	Sheridan	dirsDirList	SS3D2.VBX
List Box (Drv)	SSDrive	drvs	Sheridan	drvsDriveList	SS3D2.VBX
List Box (File)	File List	film	MicroHelp	filmFileList	MHEN200.VBX
List Box (File)	SSFile	fils	Sheridan	filsFileList	SS3D2.VBX
Flip	Flip	flpm	MicroHelp	flpmButton	MHEN200.VBX
Scroll Bar	Form Scroll	fsrm	MicroHelp	fsrmFormScroll	???
Gauge	Gauge	gagm	MicroHelp	gagmGauge	MHGR200.VBX
Graph	Graph	gpho	Other	gphoGraph	XYGRAPH.VBX
Grid	Q_Grid	grdp	Pioneer	grdpGrid	QEVBDBF.VBX
Scroll Bar	Horizontal Scroll Bar	hsbm	MicroHelp	hsbmScroll	MHEN200.VBX
Scroll Bar	DB_HScroll	hsbp	Pioneer	hsbpScroll	QEVBDBF.VBX
Graph	Histo	hstm	MicroHelp	hstmHistograph	MHGR200.VBX
Invisible	Invisible	invm	MicroHelp	invmInvisible	MHGR200.VBX
List Box	Icon Tag	itgm	MicroHelp	itgmListBox	MHAD200.VBX
Key State	Key State	kstm	MicroHelp	kstmKeyState	MHTI200.VBX
Label	Label (3d)	lblm	MicroHelp	lblmLabel	MHEN200.VBX
Line	Line	linm	MicroHelp	linmLine	MHGR200.VBX
List Box	DB_List	lstp	Pioneer	lstpListBox	QEVBDBF.VBX
List Box	SSList	lsts	Sheridan	lstsListBox	SS3D2.VBX
MDI Child	MDI Control	mdcm	MicroHelp	mdcmMDIChild	???
Menu	SSMenu	mnus	Sheridan	mnusMenu	SS3D3.VBX
Marque	Marque	mrqm	MicroHelp	mrqmMarque	MHTI200.VB
Picture	OddPic	odpm	MicroHelp	odpmPicture	MHGR200.VBX
Picture	Picture	picm	MicroHelp	picmPicture	MHGR200.VBX
Picture	DB_Picture	picp	Pioneer	picpPicture	QEVBDBF.VBX
Property Vwr	Property Viewer	pvrm	MicroHelp	pvrmPropertyViewer	MHPR200.VBX
Option (Group)	DB_RadioGroup	radp	Pioneer	radqRadioGroup	QEVBDBF.VBX
Slider	Slider	sldm	MicroHelp	sldmSlider	MHGR200.VBX
Button (Spin)	Spinner	spnm	MicroHelp	spnmSpinner	MHEN200.VBX
Spreadsheet	Spreadsheet	sprm	MicroHelp	sprmSpreadsheet	MHAD200.VBX
Picture	Stretcher	strm	MicroHelp	strmStretcher	MHAD200.VBX
Screen Saver	Screen Saver	svrm	MicroHelp	svrmSaver	MHTI200.VBX
Switcher	Switcher	swtm	MicroHelp	swtmSwitcher	???
List Box	Tag	tagm	MicroHelp	tagmListBox	MHEN200.VBX
Timer	Timer	tmrm	MicroHelp	tmrmTimer	MHTI200.VBX
ToolBar	ToolBar	tolm	MicroHelp	tolmToolBar	MHAD200.VBX
List Box	Tree	trem	MicroHelp	tremTree	MHEN200.VBX
Input Box	Input (Text)	txtm	MicroHelp	inpmText	MHEN200.VBX
Input Box	DB_Text	txtp	Pioneer	txtpText	QEVBDBF.VBX
Scroll Bar	Vertical Scroll Bar	vsbm	MicroHelp	vsbmScroll	MHEN200.VBX
Scroll Bar	DB_VScroll	vsbp	Pioneer	vsbpScroll	QEVBDBF.VBX

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: PrgOther RefsDoc

How to Add Items into Control Menu Box of Visual Basic Form

Article ID: Q110498

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

To add items into the Control-menu box of a Visual Basic Form, you can use the AppendMenu API (application programming interface) function in Windows. However, Visual Basic cannot directly detect any events for the added menu item. To catch the message for the added menu item, you can use a subclass control. You can write subclass controls using Microsoft C, but not using Visual Basic. Alternatively, you can obtain subclass controls from third-party programs such as SpyWorks from Desaware.

The Control-menu box, found in the upper-left corner of a Visual Basic form, is also known as the System-menu box in other products for Windows. The default Control-menu box contains the following nine entries including separators:

```
Restore
Move
Size
Minimize
Maximize
-----------------------
Close            Alt+F4
-----------------------
Switch to...   Ctrl+Esc
```

MORE INFORMATION

In Windows programming terms, subclassing is the process of creating a message handling procedure and intercepting messages for a given window, handling any messages you choose, and passing the rest to the window's original message handler.

The subclass procedure is a message filter that performs nondefault processing for a few key messages, and passes other messages to a default window procedure using the CallWindowProc API function. The CallWindowProc function passes a message to the Windows system, which in turns sends the message to the target window procedure. The target window procedure cannot be called directly by the subclass procedure because the target procedure (in this case a window procedure) is exported.

How to Contact Desaware

NOTE: Desaware products are manufactured independent of Microsoft.

Microsoft makes no warranty, implied or otherwise, regarding these products' performance or reliability.

Desaware
5 Town & Country Village #790
San Jose, CA 95128
Contact: Gabriel Appleman (213) 943-3305
          Dan Appleman (408) 377-4770
Fax: (408) 371-3530

The Desaware company offers the following products:

- Custom Control Factory -- An interactive development tool for creating custom controls including animated pushbuttons, multistate buttons, enhanced buttons, check boxes, and option button controls for Windows applications.

- CCF-Cursors -- Provides you with complete control over cursors (mouse pointers) in Visual Basic applications. Create your own cursors or convert icons to cursors, and much more. Includes over 50 cursors.

- SpyWorks-VB -- An advanced development tool for use with Visual Basic. SpyWorks contains subclass controls.

This information is subject to change.

Additional reference words: 3.00
KBCategory: APrg Refs
KBSubcategory: APrgOther RefsThird

DOCERR: GetPrivateProfileString Declaration Incorrect in API

Article ID: Q110826

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article corrects a documentation error for the GetPrivateProfileString function call as described in the Windows version 3.1 API Reference help file that shipped with Microsoft Visual Basic version 3.0 for Windows.

MORE INFORMATION

The declaration is incorrectly shown as:

```
Declare Function GetPrivateProfileString Lib "Kernel"
                                 (ByVal lpApplicationName As String,
                                  lpKeyName As Any,
                                  ByVal lpDefault As String,
                                  ByVal lpReturnedString As String,
                                  ByVal nSize As Integer,
                                  ByVal lpFileName As String) As Integer
```

The correct declaration is as follows:

```
Declare Function GetPrivateProfileString Lib "Kernel"
                                 (ByVal lpApplicationName As String,
                                  ByVal lpKeyName As Any,
                                  ByVal lpDefault As String,
                                  ByVal lpReturnedString As String,
                                  ByVal nSize As Integer,
                                  ByVal lpFileName As String) As Integer
```

NOTE: Each Declare statement must be entered as one, single line.

Notice that the "ByVal" keyword was omitted from the second parameter in the online reference. This means that the function is passing the second parameter (lpKeyName) by reference. It needs to be passed by value.

The most common problem that occurs when using the incorrect declaration is that when the function is called, it returns a copy of "lpdefault" in the "lpReturnedString" parameter instead of the actual value referenced by KeyName.

Additional reference words: 3.00
KBCategory: Refs
KBSubcategory: RefsDoc

Fixlist for Visual Basic for Windows as of 14-Feb-1994

Article ID: Q111475

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY
=======

This single fixlist article lists the fixed bugs in Visual Basic for Windows. Each of these fixed bugs is completely described in an article in the Microsoft Knowledge Base. The titles and article identification numbers (Q numbers) are listed below. For more information on any one of these fixed bugs, please see the complete article in the Microsoft Knowledge Base.

If you prefer, you can have any of these articles faxed to you by using Microsoft FastTips. To use this service, call (800) 963-4300 and follow the prompts. When prompted to enter the Item ID, enter the Q number (without the Q) to have the fixed bug article faxed to you.

A similar article lists the unfixed bugs. To get both these lists, query on the following word in the Microsoft Knowledge Base:

   kblist

==========
FIXED PROBLEMS IN VISUAL BASIC VERSIONS 1.0, 2.0, AND 3.0 FOR WINDOWS
==========

FIX: VB Debug.Print in MouseMove Event Causes MouseMove Event
Article ID: Q72679

FIX: Overflow in VB Drawing Circle Segment w/ Radius of Zero
Article ID: Q73280

FIX: UAE When Place More than 64K in VB List Box or Combo Box
Article ID: Q73374

FIX: Pull-Down on Drive Box Disabled When Change Width of Box
Article ID: Q73809

FIX: UAE/GPF Changing MS-DOS Win Display If VB at Breakpoint
Article ID: Q74193

FIX: Overflow Error If Print Long String to Form or Printer
Article ID: Q74517

FIX: Control Overlaid by 2nd Control Won't Refresh If Moved
Article ID: Q74519

FIX: Open Project Dialog Misbehaves If Project Dir Deleted

Article ID: Q75519

FIX: Text Not Highlighted When Copy Immediate Win to Clipboard
Article ID: Q75762

FIX: Undocumented Separator Property of a VB Menu Item
Article ID: Q76550

FIX: Can't Have Menu with No Caption Bar/Buttons/Control Box
Article ID: Q76553

FIX: ControlBox Property False Disables Focus w/ Keys in Menus
Article ID: Q76556

FIX: StretchBlt() Gives UAE/GPF with 256-Color Video Drivers
Article ID: Q77314

FIX: Visual Basic List Box Won't Open if Resized at Run Time
Article ID: Q79030

FIX: Text Too Narrow with Italic Fonts in Visual Basic Labels
Article ID: Q79117

FIX: SendKeys Causes Erratic Mouse Behavior on IBM PS/2
Article ID: Q79603

FIX: DDE from Excel to VB Ver 1.0 Uses Up Windows GDI Heap
Article ID: Q80440

FIX: File Not Loaded If No Extension in Load Picture Dialog
Article ID: Q80643

FIX: Panel Custom Control Caption Not Dimmed When Disabled
Article ID: Q80868

FIX: Graph Custom Control Incompatible w/ HP II Series Printer
Article ID: Q80912

FIX: Animated Button Custom Control: Caption May Be Truncated
Article ID: Q81223

FIX: Graph Control's Negative Values Plot As Positive
Article ID: Q81451

FIX: Gauge: Incomplete Paint with Max-Min Difference > 100
Article ID: Q81462

FIX: Grid: Changing Font Properties Resets ColWidth, RowHeight
Article ID: Q81463

FIX: VB Graph Custom Control: BottomTitle Text May Disappear
Article ID: Q81950

FIX: Outline Transparent in 3D Button When Outline=False
Article ID: Q82160

FIX: Graph Custom Control: LabelText May Overlap

Article ID: Q82874

FIX: Graph Custom Control Legends May Print Incorrectly
Article ID: Q82875

FIX: Grid Cell Border May Not Display with Some BackColors
Article ID: Q83759

FIX: Omitting Year for DateValue May Give Unexpected Results
Article ID: Q84115

FIX: Toolkit 3-D Option & Check Controls Don't Repaint in 3.1
Article ID: Q84475

FIX: Toolkit Setup Routine Causes Out of String Space Error
Article ID: Q85155

FIX: Grid Custom Control RemoveItem Does Not Update RowHeight
Article ID: Q85436

FIX: GP Fault or UAE When Unload Form in DragOver Event
Article ID: Q93233

FIX: UAE/GPF Occurs If EXE Uses Variable Length String in Type
Article ID: Q93256

FIX: UAE/GPF When Use Static Array in Event Procedure After F5
Article ID: Q93257

FIX: UAE/GPF When VB Control Name Identical to Property Name
Article ID: Q93424

FIX: UAE/GPF When Square Brackets '[]' Around MSGBOX Function
Article ID: Q93425

FIX: GPF/UAE When Converting String > 32K to Double Precision
Article ID: Q93435

FIX: VB Painting Problem Occurs When Low on System Resources
Article ID: Q93436

FIX: Result Differs When Comparing Single w/ Double Precision
Article ID: Q93437

FIX: GPF/UAE When Closing DDE Application from the Task List
Article ID: Q94166

FIX: GPF/UAE w/ Stop Command in Event Procedure & Deleted Sub
Article ID: Q94167

FIX: GPF When Pasting 8 Bit .DIB File into Anibutton Control
Article ID: Q94168

FIX: VB MCITEST CD Player Sample Displays Incorrect Track
Article ID: Q94185

FIX: GPF/UAE After Undoing Edit of Option Explicit Statement

Article ID: Q94216

FIX: GPF/UAE When Assign NULL to VBM_GETPROPERTY of type HLSTD
Article ID: Q94217

FIX: Using Graphics Method on DB Objects May Cause GPF/UAE
Article ID: Q94242

FIX: Adding Watch Point in VB May Cause UAE in Windows 3.0
Article ID: Q94243

FIX: GPF/UAE When Large Tag w/ MultiSelect of 30+ Controls
Article ID: Q94244

FIX: Setting Add Watch May Cause Your Program to Reset
Article ID: Q94290

FIX: Setting Add Watch May Cause GP Fault or UAE
Article ID: Q94292

FIX: Painting Problems When FontItalic Set True for Text Box
Article ID: Q94293

FIX: Grid Control Paints Incorrectly When Press PGUP or PGDN
Article ID: Q94296

FIX: GPF/UAE When New Project Loaded After Large Previous Proj
Article ID: Q94351

FIX: No Out of Memory Error Generated with Text Box > 32K
Article ID: Q94698

FIX: Attempting to Refresh Null TableDef Field Causes GP Fault
Article ID: Q94773

FIX: GPF When Using 8514 Driver with Long String in Text Box
Article ID: Q94774

FIX: Changing Decimal Separator Causes Load Errors for Form
Article ID: Q94776

FIX: GPF When Making .EXE File If Forms Saved as Binary
Article ID: Q94892

FIX: Bad .MAK File Prevents Display of Make EXE File Dialog
Article ID: Q94939

FIX Large Sub or Function or Module Can Cause GP Fault or UAE
Article ID: Q95285

FIX: GPF/UAE When Create or Use Huge Array w/ Large Elements
Article ID: Q95290

FIX: Error Message: Timeout While Waiting for DDE Response
Article ID: Q95428

FIX: FixedCols Can Cause Paint Problem with Grid Control

Article ID: Q95429

FIX: Problems Calling DoEvents from a Scroll Bar Change Event
Article ID: Q95498

FIX: MAPI: GPF When Attempt to Download 923 or More Messages
Article ID: Q95501

FIX: Extra Chars in Masked Edit Cause Empty InvalidText Box
Article ID: Q95508

FIX: Text Box/Mask Edit in Select Mode If MsgBox in LostFocus
Article ID: Q95509

FIX: Focus Rectangle Remains When Grid Loses Focus
Article ID: Q95514

FIX: GPF When Erase User-Defined Array of Variable Strings
Article ID: Q95525

FIX: Loading Proj Gives Err: Custom control 'Graph' not found
Article ID: Q95590

FIX: GPF When Making EXE If Declare Is Missing Lib & DLL Name
Article ID: Q95829

FIX: Resizing MDIForm with UI Does Not Update Height & Width
Article ID: Q96097

FIX: Scroll Bar Thumb Doesn't Do Change Event as It Should
Article ID: Q96798

FIX: Can't Open ODBCADM.HLP Err Msg During Data Access Setup
Article ID: Q97083

FIX: No Menu Event with Maximized MDI Child
Article ID: Q97135

FIX: Mouse Misbehaves After Changing Graph Visible Property
Article ID: Q97588

FIX: OLE Client: Copying Linked Object Gives Err: Can't Paste
Article ID: Q97619

FIX: GPF/UAE with Huge Array Size as Multiple of 64K Bytes
Article ID: Q98990

FIX: Erase Won't Clear Contents of Huge Fixed Array as Variant
Article ID: Q99457

FIX: VB 2.0 Prof Demo Causes Error: Invalid File Format
Article ID: Q100611

FIX: Repaint Prob Adding Graphical Control as Child of Graph
Article ID: Q102606

FIX: GPF with Long Formulas in Crystal Reports Custom Control

Article ID: Q108658

FIX: Double-Click Still Maximizes/Restores If MaxButton=False
Article ID: Q110309

Additional reference words: kblist
KBCategory:
KBSubCategory: RefsProd

Buglist for Visual Basic for Windows as of 14-Feb-1994

Article ID: Q111476

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 1.0, 2.0, and 3.0

SUMMARY
=======

This single buglist article lists the unfixed bugs in Visual Basic for Windows. Each of these bugs is completely described in an article in the Microsoft Knowledge Base. The titles and article identification numbers (Q numbers) are listed below. For more information on any of these bugs, please see the complete article in the Microsoft Knowledge Base.

If you prefer, you can have any of these articles faxed to you by using Microsoft FastTips. To use this service, call (800) 963-4300 and follow the prompts. When prompted to enter the Item ID, enter the Q number (without the Q) to have the bug article faxed to you.

A similar article lists the fixed bugs. To get both these lists, query on the following word in the Microsoft Knowledge Base:

   kblist

==========
UNFIXED BUGS IN VISUAL BASIC VERSIONS 1.0, 2.0, AND 3.0 FOR WINDOWS
==========

BUG: TABs Paste Incorrectly as | to VB.EXE's Immediate Window
Article ID: Q73700

BUG: Scroll Box Flashing Not Updated If Bar Resized w/ Focus
Article ID: Q73839

BUG: [Character May Sort Incorrectly in List or Combo Box
Article ID: Q74132

BUG: Can Click in Code Window Without Activating it in VB.EXE
Article ID: Q74194

BUG: Pressing ESC or CTRL+BREAK Makes Mouse Pointer Disappear
Article ID: Q74409

BUG: No Beep When Click Form and the Menu Design Window Is Up
Article ID: Q74518

BUG: Incorrectly Accessing System Menu of Hidden Form
Article ID: Q74564

BUG: ExtFloodFill Won't Fill Over QBColors If AutoRedraw=True
Article ID: Q75640

BUG: Duplicate Procedure Name Alters Original Capitalization
Article ID: Q76514

BUG: No Option Button Active (Dotted) in Frame
Article ID: Q76520

BUG: Italic and Large Fonts Display Poorly in Text Boxes
Article ID: Q76555

BUG: Dir List Box Does Not Give Error 68 Device Unavailable
Article ID: Q76628

BUG: FormName Not in Correct Order After Out of Memory Error
Article ID: Q76983

BUG: LinkTimeOut of -1 Waits Only 6553.5 Secs Before Time Out
Article ID: Q77243

BUG: DateSerial Does Not Give Error for Invalid Month or Day
Article ID: Q77393

BUG: Incorrect Focus Shift for Disabled Control in Break Mode
Article ID: Q77734

BUG: Extra Click Event if Double-Click When Mouse Button Down
Article ID: Q77738

BUG: CTRL+LEFT/RIGHT ARROW Behaves Differently When Edit/Type
Article ID: Q77928

BUG: ToolBox Picture Control Bitmap Too Small on EGA
Article ID: Q78132

BUG: Using Nonstandard Icons Can Cause UAE/GP Fault/Hang
Article ID: Q78380

BUG: Right Mouse Button Causes Remote Control Menus
Article ID: Q78773

BUG: Multiline Text Box Contents Not Gray When Enabled=False
Article ID: Q78892

BUG: Visual Basic Code Window Hides Split View if Resized
Article ID: Q79057

BUG: Invalid outside Sub Error When Copy or Paste to General
Article ID: Q79240

BUG: Resetting ListIndex Property Generates Click Event
Article ID: Q79241

BUG: Some Property Values May Be Incorrect in Maximized Form
Article ID: Q79242

BUG: Option Button w/ Focus Selected When Click Form Caption
Article ID: Q79602

BUG: Click Event May Fail to Occur in Cascading Menu
Article ID: Q80023

BUG: TAB Character Can Incorrectly Cause KeyUp/KeyDown Events
Article ID: Q80286

BUG: MDI Child CTRL+INSERT in Properties List Causes UAE/GPF
Article ID: Q80777

BUG: No Resources Causes Failed to Open Graphics Server Error
Article ID: Q80780

BUG: Gauge Custom Control: No Error for Illegal NeedleWidth
Article ID: Q80905

BUG: MDI Child Left/Top Property Wrong in Properties Bar
Article ID: Q80907
BUG: MDI Child Control: Large Height/Width Value Not Accepted
Article ID: Q80908

BUG: Grid Custom Control: Scroll Bars Displayed Unnecessarily
Article ID: Q80967

BUG: Gauge Custom Control: Valid NeedleWidth Range 1 to 32767
Article ID: Q81187

BUG: 3-D Panel Control Doesn't Resize to Key Status Control
Article ID: Q81449

BUG: Vertical Linear Gauge Loses Upper Border's Bottom Pixels
Article ID: Q81460

BUG: InnerBottom/InnerRight Defines Gauge Fill Area Badly
Article ID: Q81461

BUG: Graph: ExtraData May Not Say: Invalid Property Value
Article ID: Q81472

BUG: 3D Command Button Shows Outline when Outline = False
Article ID: Q81951

BUG: Scroll Control: UAE/GPF If Drag Method in GotFocus Event
Article ID: Q81955

BUG: Grid: No Error Changing FixedAlignment on Non-Fixed Col
Article ID: Q81998

BUG: Graph: AutoInc Increments ThisPoint Instead of ThisSet
Article ID: Q81999

BUG: Animated Button: 8 Pt. Roman/Mdrn Fonts Don't Underline
Article ID: Q82004

BUG: Graph Axis Titles Don't Switch on Horizontal Bar Graphs
Article ID: Q83463

BUG: Menu Can Cover Top of MDI Child Control If BorderStyle=0
Article ID: Q83858

BUG: VB Graph Custom Control: SeeThru Paints Incorrectly
Article ID: Q84236

BUG: Must Call API to Print Color Text on Color Printer in VB
Article ID: Q84269

BUG: Some Controls Not Printed with PrintForm in Windows 3.1
Article ID: Q84471

BUG: THREED.VBX: Command/Group Push Buttons Show Invalid File
Article ID: Q84553

BUG: Common Dialog Custom Controls Don't Display Printer Fonts
Article ID: Q84839

BUG: MDI Child Control Skips Index with Control Array
Article ID: Q87765

BUG: Generic / Text Only Printer Driver Prints 66 Lines
Article ID: Q87767

BUG: Illegal function call / Division By Zero Errors
Article ID: Q94778

BUG: Stack Fault When Move Sets Tiny Width in 2-Item Combo Box
Article ID: Q95197

BUG: GPF/UAE If Multi-Select Controls w/ No Common Properties
Article ID: Q95430

BUG: Type Mismatch Error If Use VAL Function on Big Hex Value
Article ID: Q95431

BUG: Stack Fault May Occur If Trapping Divide By Zero
Article ID: Q95499

BUG: GPF When Close Form That Contains a Single MCI Control
Article ID: Q95500

BUG: Neg ScaleHeight Resizes Control When Form Saved as ASCII
Article ID: Q95513

BUG: Stack Fault When Move Makes Combo Box Width Too Small
Article ID: Q95830

BUG: Unable to Edit LinkNotify Event If Control Has Long Name
Article ID: Q97027

BUG: ODBC Getchunk Method on Non-Memo Field Causes GPF/UAE
Article ID: Q97082

BUG: OLE DataText Prop Doesn't Free Memory When Object Closed
Article ID: Q97136

BUG: Changing Default Printer Doesn't Effect Printer.Fonts
Article ID: Q99705

BUG: Wrong Menu Click Event After Hiding Menu
Article ID: Q99872

BUG: MaskedEdit MaxLength Reset to 64 When Mask=""
Article ID: Q99873

BUG: Overflow Error When CurrentX Or CurrentY Greater Than 32K
Article ID: Q100190

BUG: VB Pro Setup Fails to Correctly Associate .HLP Files
Article ID: Q100191

BUG: Out of Memory Error on Show Next from Debug Menu
Article ID: Q100192

BUG: 3D Button Loses 256-Color Palette When Load 2nd Bitmap
Article ID: Q100193

BUG: Grid Control Repaints When Another Form Is Made Active
Article ID: Q100195

BUG: Unload in 3D GroupPush Button Causes GP Fault
Article ID: Q100327

BUG: Referencing Data Object Gives Error: Object not an Array
Article ID: Q100367

BUG: GPF in Some Video Drivers When Load RLE Bitmaps > 20K
Article ID: Q100610

BUG: Font3D Property Set Incorrectly in THREED.VBX Controls
Article ID: Q100612

BUG: Data Access Setup Can Give Incorrect Error Message
Article ID: Q100613

BUG: Ref to NPV / IRR / MIRR Gives Undefined Functions Error
Article ID: Q101245

BUG: Incorrect Result When Multiple Aggregate Functions in SQL
Article ID: Q101256

BUG: Incorrect Behavior in MaskedEdit BorderStyle Property
Article ID: Q101257

BUG: Problems Printing Projects to HPLJ4
Article ID: Q101379

BUG: ALT+MINUS SIGN Does Not Work with Maximized MDI Forms
Article ID: Q101380

BUG: GP Fault When Opening Menu Design Window in VB.EXE
Article ID: Q101381

BUG: VB Dynasets Incorrectly Bypass Defaults on SQL Server
Article ID: Q101522

BUG: Bad Result If Multiple Aggregate Functions in SQL Stmt
Article ID: Q101553

BUG: Out of Memory w/ Var Named ClientLeft/Top/Width/Height
Article ID: Q102069

BUG: Setup Wizard Error: Sharing Violation Reading Drive C:
Article ID: Q102478

BUG: Domain Functions Available Only Within SQL Statement
Article ID: Q102479

BUG: Can't Load Custom Control DLL: PICCLIP.VBX in Windows 3.0
Article ID: Q102649

BUG: Out of Memory w/ MSOLE2.VBX When SHARE.EXE Not Loaded
Article ID: Q103438

BUG: Invalid Argument Err on Execute Method w/ SQL Passthrough
Article ID: Q103976

BUG: GPF in VB.EXE at 0038:3B6F w/ Compile-Time Error & Set
Article ID: Q105140

BUG: Error 13 (Type Mismatch) & Error 3061 w/ SQL Queries
Article ID: Q105171

BUG: Overflow in VB version 3.0 ICONWRKS Sample Program
Article ID: Q105808

BUG: VB Printer.Width/Height Values Incorrect for Plotter
Article ID: Q106495

BUG: VB Setup Files Modified or Corrupted, Using \WINDOWS Path
Article ID: Q106496

BUG: Name Not Found in This Collection When Deleting Member
Article ID: Q107362

BUG: Incorrect VB Error When Delete Index on Open Table
Article ID: Q107363

BUG: First Item Can Disappear in Outline Control Style 0 or 2
Article ID: Q108659

BUG: Out of Memory Error When Adding 35-50 Pen Controls
Article ID: Q110989

Additional reference words: 1.00 2.00 3.00 kblist
KBCategory:
KBSubcategory: RefsProd

PRB: Insufficient Disk Space Error When Setup Copies Files

Article ID: Q74648

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

If you receive an "Insufficient disk space" error message when running Visual Basic's Setup program, it may be caused by using Windows with a temporary Windows swap file instead of the permanent Windows swap file.

MORE INFORMATION

Pages 520 through 529 in the "Microsoft Windows User's Guide" version 3.0 manual discuss Windows swap files. Permanent swap files are contiguous so that your disk does not contain files in fragmented pieces, which may happen if you are using temporary swap files. Temporary Windows swap files may grow in size, which may cause the "Insufficient disk space" error during the execution of Visual Basic's Setup program. However, permanent Windows swap files will not change in size, so using permanent Windows swap files may help to avoid the "Insufficient disk space" error.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: Setins

Example of Client-Server DDE Between Visual Basic Applications

Article ID: Q74861

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article outlines the steps necessary to initiate dynamic data exchange (DDE) between a Microsoft Visual Basic destination application and a Visual Basic source application.

This article demonstrates how to:

- Create a Visual Basic application to function as a DDE source.
- Create a Visual Basic application to function as a DDE destination.
- Initiate a manual DDE link (information updated upon request from the destination) between the destination application and the source application.
- Use LinkRequest to update information in the destination application from information in the source application.
- Initiate a automatic DDE link (information updated automatically from source to destination) between the destination application and the source application.
- Use LinkPoke to send information from the destination application to the source application.
- Change the LinkMode property between automatic and manual.

MORE INFORMATION

This information is included with the Help file provided with Microsoft Professional Toolkit for Visual Basic version 1.0, Microsoft Visual Basic version 2.0, and Microsoft Visual Basic version 3.0.

A destination application sends commands through DDE to the source application to establish a link. Through DDE, the source provides data to the destination at the request of the destination or accepts information at the request of the destination.

Step-by-Step Example

The steps below show how to establish a DDE conversation between two Visual Basic applications.

STEP ONE: Create the Source Application in Visual Basic

1. Start a new project in Visual Basic. Form1 is created by default.

2. Change the Caption property of Form1 to Source.

3. Change the Form1 LinkMode property to 1 - Source.

4. Put a Text Box (Text1) on Form1.

5. Save the form and project with the name SOURCE.

6. From the File menu, choose Make EXE File. In the Make EXE File dialog box, choose OK to accept SOURCE.EXE as the name of the EXE file.

STEP TWO: Create the Destination Application in Visual Basic
---

1. From the File menu, choose New Project. Form1 is created by default.

2. Change the Caption property of Form1 to Destination.

3. Add the following controls to Form1, and give them the properties indicated:

Default Name	Caption	Name
Text1	(Not applicable)	Text1
Option1	Manual Link	ManualLink
Option2	Automatic Link	AutomaticLink
Command1	Poke	Poke
Command2	Request	Request

4. Add the following code to the General Declaration section of Form1:

```
Const AUTOMATIC= 1
Const MANUAL = 2
Const NONE = 0

'(NOTE:  For Visual Basic version 1.0, also add the following
'        constants:
'Const True = -1
'Const False = 0
```

5. Add the following code to the Load event procedure of Form1:

```
Sub Form_Load ()
   'This procedure will start the VB source application.

   z% = Shell("SOURCE", 1)

   z% = DoEvents()   'Causes Windows to finish processing Shell command.

   Text1.LinkMode = NONE   'Clears DDE link if it already exists.

   Text1.LinkTopic = "Source|Form1"   'Sets up link with VB source.
   Text1.LinkItem = "Text1"           'Set link to text box on source.
   Text1.LinkMode = MANUAL            'Establish a manual DDE link.
   ManualLink.Value = TRUE            'Sets appropriate option button.
End Sub
```

6. Add the following code to the Click event procedure of ManualLink:

```
   Sub ManualLink_Click ()
      Request.Visible = TRUE    'Make request button valid.
      Text1.LinkMode = NONE     'Clear DDE Link.
      Text1.LinkMode = MANUAL   'Reestablish new LinkMode.
   End Sub
```

7. Add the following code to the Clink event procedure of AutomaticLink:

```
   Sub AutomaticLink_Click ()
      Request.Visible = FALSE   'No need for button with automatic link.
      Text1.LinkMode = NONE     'Clear DDE Link.

      Text1.LinkMode = AUTOMATIC   'Reestablish new LinkMode.
   End Sub
```

8. Add the following code to the Click event procedure of Request:

```
   Sub Request_Click ()
      'With a manual DDE link, this button will be visible, and when
      'selected it will request an update of information from the source
      'application to the destination application.
      Text1.LinkRequest
   End Sub
```

9. Add the following code to the Click event procedure of Poke:

```
   Sub Poke_Click ()
      'With any DDE link, this button will be visible, and when it's
      'selected, will poke information from the destination application
      'into the source application.
      Text1.LinkPoke
   End Sub
```

STEP THREE: Run the Visual Basic Destination Application

Choose one of these options:

- Run the Visual Basic destination application from the VB.EXE environment by skipping to step 4 below.
- Save the application. Then create an .EXE file and run it from Windows by beginning with step 1 below.

1. From the File menu, choose Save, and save the form and project with the name DEST.

2. From the File menu, choose Make EXE File, and give it the name DEST.EXE.

3. Exit the Visual Basic environment (VB.EXE).

4. Run the application from Windows if it's an .EXE file or from the VB.EXE environment.

5. Form1 of the destination application will load and the source application will automatically start.

STEP FOUR: Experiment with DDE Between Visual Basic Applications
---

1. Try typing some text into the source application's text box. Then click the Request button. The text appears in the destination application's text box.

2. Click the Automatic Link button and then type some more text into the source application's text box. The text is automatically updated in the destination application's text box.

3. Type some text into the destination application's text box. Then click the Poke button to send the text to the source application's text box.

For additional information on dynamic data exchange (DDE) between Visual Basic and other Windows-based applications, query on the following words in the Microsoft Knowledge Base:

   DDE and Visual Basic

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: IAPDDE

DDE Example Between Visual Basic and Word for Windows

Article ID: Q74862

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article outlines the steps necessary to initiate dynamic data exchange (DDE) between a Microsoft Visual Basic application and a Microsoft Word for Windows (WINWORD.EXE) document at run time.

This article demonstrates how to:

- Prepare a Word for Windows document for active DDE.
- Initiate a manual DDE link (information updated upon request from the destination) between the Visual Basic application (the destination) and the document loaded into Word for Windows (the source).
- Use LinkRequest to update information in the Visual Basic destination based on information contained in the Word for Windows source.
- Initiate a automatic DDE link (information updated automatically from source to destination) between the Visual Basic destination and the Word for Windows source.
- Use LinkPoke to send information from the Visual Basic destination to the Word for Windows source.
- Change the LinkMode property between automatic and manual.

MORE INFORMATION

This information is included with the Help file provided with Microsoft Professional Toolkit for Visual Basic version 1.0, Microsoft Visual Basic version 2.0, and Microsoft Visual Basic version 3.0.

A destination application sends commands through DDE to the source application to establish a link. Through DDE, the source provides data to the destination at the request of the destination or accepts information at the request of the destination.

Example Showing How to Establish a DDE Conversation

The steps below give an example of how to establish a DDE conversation between a Visual Basic application and a document loaded into Word for Windows (WINWORD.EXE).

STEP ONE: Create the Source Document in Word for Windows

1. Start Microsoft Word for Windows. Document1 is created by default.

NOTE: The Tip of the Day option in Microsoft Word version 6.0 for Windows must be turned off in order for this to work.

2. From the Window menu, choose Arrange All. This removes maximization if the document was maximized. Note that the title at the top of the WINWORD.EXE main title bar is now:

   Microsoft Word

   instead of:

   Microsoft Word - Document1

3. Press CTRL+SHIFT+END to select to the end of the document.

4. From the Insert menu (or the Edit menu in Microsoft Word version 6.0), choose Bookmark. Under Bookmark Name, type:

   DDE_Link

   Press the ENTER key. This sets a bookmark for the entire document. This bookmark functions as the LinkItem in the DDE conversation.

5. From the File menu, choose Save As, and save the document with the name SOURCE.DOC.

   NOTE: SOURCE.DOC must be saved to the working directory used by the Visual Basic application or the DDE won't work. Specifying the path in the SHELL statement in Visual Basic will not work.

6. Exit from Word for Windows. For this particular example to function correctly, WINWORD.EXE must not be loaded and running.

STEP TWO: Create the Destination Application in Visual Basic
---

1. Start Visual Basic. Form1 is created by default.

2. Create the following controls on Form1, giving the controls the properties shown in the table:

Default Name	Caption	Name
Text1	(Not applicable)	Text1
Option1	Manual Link	ManualLink
Option2	Automatic Link	AutomaticLink
Command1	Poke	Poke
Command2	Request	Request

3. Add the following code to the General Declaration section of Form1:

```
Const AUTOMATIC = 1
Const MANUAL = 2
Const NONE = 0
```

4. Add the following code to the Load event procedure of Form1:

```
Sub Form_Load ()
   'This procedure starts WINWORD.EXE, loads the document that was
   'created earlier, and prepares for DDE by creating a bookmark to
   'the whole document. This bookmark is necessary because it
   'functions as the LinkItem for the source in the DDE conversation.

   z% = Shell("WinWord Source.Doc",1)

   z% = DoEvents ()  'Process Windows events to ensure that
                     'WINWORD.EXE is executed before any attempt is
                     'made to perform DDE with it.

   Text1.LinkMode = NONE              'Clears DDE link if it exists.
   Text1.LinkTopic = "WinWord|Source" 'Sets up link with WINWORD.EXE.
   Text1.LinkItem = "DDE_Link"        'Set link to bookmark on document.
   Text1.LinkMode = MANUAL            'Establish a manual DDE link.
   ManualLink.Value = TRUE
End Sub
```

5. Add the following code to the Click event procedure of the Manual Link button:

```
Sub ManualLink_Click ()
   Request.Visible = TRUE  'Make request button valid.
   Text1.LinkMode = NONE   'Clear DDE Link.
   Text1.LinkMode = MANUAL 'Reestablish new LinkMode.
End Sub
```

6. Add the following code to the Click event procedure of the Automatic Link button:

```
Sub AutomaticLink_Click ()
   Request.Visible = FALSE     'No need for button with automatic link.
   Text1.LinkMode = NONE       'Clear DDE Link.
   Text1.LinkMode = AUTOMATIC  'Reestablish new LinkMode.
End Sub
```

7. Add the following code to the Click event procedure of the Request button:

```
Sub Request_Click ()
   'With a manual DDE link this button is visible. Clicking this button
   'requests an update of information from the source application to the
   'destination application.
   Text1.LinkRequest
End Sub
```

8. Add the following code to the Click event procedure of the Poke button:

```
Sub Poke_Click ()
   'With any DDE link, this button is visible. Clicking this button
   'pokes information from the destination application into the source
   'application.
   Text1.LinkPoke
End Sub
```

STEP THREE: Try it out

Now, you have two choices. You can run the Visual Basic destination application from the Visual Basic VB.EXE environment by skipping to step 4 below, or you can save the application, create an .EXE file, and run that from Windows by beginning with step 1 below.

1. From the File menu, choose Save, and save the form and project with the name DEST.

2. From the File menu, choose Make EXE File with the name DEST.EXE.

3. Exit from the Visual Basic environment (VB.EXE).

4. Run the application. Run an .EXE file from Windows, or if you're in the Visual Basic environment, from the Run menu, choose Start.

   Form1 of the Visual Basic destination application will be loaded, and Word for Windows will automatically start and load SOURCE.DOC.

5. Make sure the main title bar in WINWORD.EXE reads "Microsoft Word," not "Microsoft Word - SOURCE.DOC." If the title bar is not correct, choose Arrange All from the Window menu.

STEP FOUR: Experiment with DDE Between Visual Basic and Word for Windows
---

1. Try typing some text into the document in Word for Windows. Then click the Request button. The text appears in the text box.

2. Click the Automatic Link button. Then type some more text into the document in Word for Windows. The text is automatically updated in the Visual Basic text box.

3. Type some text in the text box in the Visual Basic application. Then click the Poke button. The text goes to the Word for Windows document.

Note that if in the WINWORD.EXE document, you delete the total contents of the bookmark, the bookmark is also deleted. Any attempt to perform DDE with this WINWORD.EXE session after deleting the bookmark causes this error:

   Foreign applications won't perform DDE method or operation.

If this happens, you must recreate the bookmark in the document in Word for Windows before performing any further DDE operations.

In Visual Basic version 1.0, you need to add the following two global constants to the form's general declarations section:

```
CONST TRUE = -1
CONST FALSE = NOT TRUE
```

Additional reference words: 1.00 2.00 3.00 winword
KBCategory:
KBSubcategory: IAPDDE

DDE from Visual Basic for Windows to Excel for Windows

Article ID: Q75089

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Excel for Windows, version 5.0

SUMMARY

This article describes how to initiate a dynamic data exchange (DDE) conversation between a Visual Basic destination application and a Microsoft Excel source application.

This article demonstrates how to:

- Prepare a Microsoft Excel for Windows document for active DDE.
- Initiate a manual DDE link (information updated upon request from the destination) between Visual Basic (the destination) and Excel (the source).
- Use the LinkRequest method to update information in Visual Basic (the destination) based on information contained in Excel (the source).
- Initiate a automatic DDE link (information updated automatically from source to destination) between Visual Basic (the destination) and Excel (the source).
- Use the LinkPoke method to send information from Visual Basic (the destination) to Excel (the source).
- Change the LinkMode property between automatic and manual.

MORE INFORMATION

A destination application sends commands through DDE to the source application to establish a link. Through DDE, the source provides data to the destination at the request of the destination or accepts information at the request of the destination.

The procedure below is as an example showing how to establish a DDE conversation between Visual Basic and Excel for Windows.

STEP ONE: Create the Source Spreadsheet in Excel

1. Start Excel. A document (spreadsheet) with Sheet1 as the title is created by default.

2. From the File menu, choose Save As, and save the document (spreadsheet) naming it SOURCE.XLS

4. Exit Excel. For this example to function properly, Excel must not be loaded and running.

STEP TWO: Create the Destination Application in Visual Basic

The destination is the application that performs the link operations. It prompts the source to send information or informs the source that information is being sent to it.

1. Start Visual Basic (VB.EXE). Form1 is created by default.

2. Add the following controls to Form1, and give them the properties indicated:

Default Name	Caption	Name
Text1	(not applicable)	Text1
Option1	Manual Link	ManualLink
Option2	Automatic Link	AutomaticLink
Command1	Poke	Poke
Command2	Request	Request

3. Add the following code to the general Declaration section of Form1:

```
Const AUTOMATIC = 1
Const MANUAL = 2
Const NONE = 0
```

4. Add the following code to the Load event procedure of Form1:

```
Sub Form_Load ()
   'This procedure starts Excel and loads SOURCE.XLS, the
   'spreadsheet created above.
   Dim ErrorTries As Integer
   ErrorTries = 0
   On Error GoTo errorhandler

   z% = Shell("c:\EXCEL\excel SOURCE.XLS", 1)

   DoEvents         'Process Windows events to ensure that
                    'Excel executes before making any attempt
                    'to perform DDE.

   Text1.LinkMode = NONE    'Clear DDE link if it already exists.
   'Text1.LinkTopic = "Excel|source.xls"
   'Set up link with Excel:

   Text1.LinkTopic = "Excel|C:\VB3\[SOURCE.XLS]Sheet1"

   Text1.LinkItem = "R1C1"  'Set link to first cell on spreadsheet.
   Text1.LinkMode = MANUAL  'Establish a manual DDE link.
   ManualLink.Value = True
   Exit Sub

errorhandler:
   If Err = 282 And ErrorTries < 15 Then
      ErrorTries = ErrorTries + 1
      DoEvents
```

```
      Resume
   Else
      Error Err
   End If

End Sub
```

5. Add the following code to the Click event procedure of the Manual Link button:

```
Sub ManualLink_Click ()
   Request.Visible = TRUE    'Make request button valid.
   Text1.LinkMode = NONE     'Clear DDE Link.
   Text1.LinkMode = MANUAL   'Reestablish new LinkMode.
End Sub
```

6. Add the following code to the Click event procedure of the Automatic Link button:

```
Sub AutomaticLink_Click ()
   Request.Visible = FALSE     'No need for button with automatic link.
   Text1.LinkMode = NONE       'Clear DDE Link.
   Text1.LinkMode = AUTOMATIC  'Reestablish new LinkMode.
End Sub
```

7. Add the following code to the Click event procedure of the Request button:

```
Sub Request_Click ()
   'With a manual DDE link this button will be visible and when
   'selected it will request an update of information from the source
   'application to the destination application.
    Text1.LinkRequest
End Sub
```

8. Add the following code to the Click event procedure of the Poke button:

```
Sub Poke_Click ()
   'With any DDE link this button will be visible and when selected
   'it will poke information from the destination application to the
   'source application.
   Text1.LinkPoke
End Sub
```

STEP THREE: Run the Visual Basic Destination Application

You have two choices:

- Run the Visual Basic destination application from the Visual Basic environment by skipping to step 4 below.
- Save the application. Then create an .EXE file, and run it from Windows by beginning with step 1 below.

1. From the Visual Basic File menu, choose Save, and save the Form and Project naming both DEST.

2. From the File menu, choose Make EXE File. Name it DEST.EXE.

3. Exit from Visual Basic.

4. Run the application from Windows if an .EXE file or from the Visual Basic environment.

5. Form1 of the destination application will be loaded and Excel will automatically start with the document SOURCE.XLS loaded.

6. Make sure the main title bar in Excel reads "Microsoft Excel," not "Microsoft Excel - SOURCE.XLS." If the title bar is incorrect, choose Arrange All from the Window menu.

STEP FOUR: Experiment with DDE between Visual Basic and Excel
---

1. Try typing some text in R1C1 in the spreadsheet. Then click the Request button. The text appears in the text box.

   Be sure to press the ENTER key after entering text into an Excel cell before clicking the Request button in the Visual Basic program. If you don't, a "Timeout while waiting for DDE response" error message will display because of the TEXT1.LINKREQUEST statement. This occurs because while entering text into a cell, Excel is in a polling loop for data entry. No real data is transferred to the cell until you press ENTER. Therefore, Visual Basic continues to request the data from the cell, but Excel does not pay attention to the request until it exits the polling loop, which results in the DDE time-out message.

2. Choose the Automatic Link button and then type some more text in R1C1 of the spreadsheet. The text is automatically updated in the Visual Basic text box.

3. Type some text in the text box in the Visual Basic application and choose the Poke button. The text is sent to R1C1 in the Excel spreadsheet.

Note: If you have the Ignore Remote Requests option selected in the Excel Workspace dialog box, you will not be able to establish DDE from Visual Basic. Make sure the Ignore Remote Requests option isn't selected.

For Visual Basic version 1.0 add the following constants to the general declarations of the form:

```
CONST TRUE = -1
CONST FALSE = NOT TRUE
```

For more information on DDE between Visual Basic and other Windows-based applications, query on the following words in the Microsoft Knowledge Base:

   DDE and Visual Basic

Additional reference words: 1.00 2.00 3.00 4.00
KBCategory:
KBSubcategory: IAPDDE

Using DDE Between Visual Basic and Q+E for Windows

Article ID: Q75090

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article describes how to initiate a Dynamic Data Exchange (DDE) conversation between a Microsoft Visual Basic for Windows destination application and a Pioneer Software Q+E for Windows source application. (Q+E is a database query tool.)

This article demonstrates how to:

1. Prepare a Q+E database file for active DDE.

2. Initiate a manual DDE link (information updated upon request from the destination) between Visual Basic for Windows (the destination) and Q+E (the source).

3. Use LinkRequest to update information in Visual Basic for Windows (the destination) based on information contained in Q+E (the source).

4. Initiate a automatic DDE link (information updated automatically from source to destination) between Visual Basic for Windows (the destination) and Q+E (the source).

5. Use LinkPoke to send information from Visual Basic for Windows (the destination) to Q+E (the source).

6. Change the LinkMode property between Automatic and Manual.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

A destination application sends commands through DDE to the source application to establish a link. Through DDE, the source provides data to the destination at the request of the destination or accepts information at the request of the destination.

The following steps serve as a example of how to establish a DDE conversation between Visual Basic for Windows and Q+E.

First, generate a Q+E database file to act as the source.

1. Create a database (.DBF) file (see the Q+E manuals for the

procedure). For this example, you will use one of the default files, ADDR.DBF, that is provided with Microsoft Excel for Windows.

2. If Q+E is already running, exit Q+E. For this example to work properly, Q+E must not be loaded and running.

Next, create the destination application in Visual Basic for Windows.

The destination is the application that performs the link operations. It prompts the source to send information or informs the source that information is being sent.

1. Start Visual Basic for Windows. Form1 will be created by default.

2. Create the following controls with the following properties on Form1:

Default Name	Caption	Name
Text1	(not applicable)	Text1
Option1	Manual Link	ManualLink
Option2	Automatic Link	AutomaticLink
Command1	Poke	Poke
Command2	Request	Request

(In Visual Basic version 1.0 for Windows, set the CtlName Property for the above objects instead of the Name property.)

3. Add the following code to the General Declaration section of Form1:

```
Const AUTOMATIC = 1
Const MANUAL = 2
Const NONE = 0

'  Const TRUE = -1  ' In Visual Basic 1.0 for Windows uncomment
'  Const FALSE = 0  ' these two lines.
```

4. Add the following code to the Load event procedure of Form1:

```
Sub Form_Load ()        ' This procedure will start Q+E and load the
                        ' file "ADDR.DBF".
    z% = Shell("C:\EXCEL\QE C:\EXCEL\QE\ADDR.DBF",1)
    z% = DoEvents ()                ' Process Windows events. This
                                    ' ensures that Q+E will be
                                    ' executed before any attempt is
                                    ' made to perform DDE with it.
    Text1.LinkMode = NONE           ' Clears DDE link if it already
                                    ' exists.
    Text1.LinkTopic = "QE|QUERY1"   ' Sets up link with Q+E.
    Text1.LinkItem = "R1C1"         ' Set link to first cell on
                                    ' spreadsheet.
    Text1.LinkMode = MANUAL         ' Establish a manual DDE link.
    ManualLink.Value = TRUE
End Sub
```

5. Add the following code to the Click event procedure of the Manual Link button:

```
Sub ManualLink_Click ()
        Request.Visible = TRUE       ' Make request button valid.
        Text1.LinkMode = NONE        ' Clear DDE Link.
        Text1.LinkMode = MANUAL      ' Reestablish new LinkMode.
End Sub
```

6. Add the following code to the Click event procedure of the AutomaticLink button:

```
Sub HotLink_Click ()
        Request.Visible = FALSE  ' No need for button with automatic link.
        Text1.LinkMode = NONE    ' Clear DDE Link.
        Text1.LinkMode = AUTOMATIC        ' Reestablish new LinkMode.
End Sub
```

7. Add the following code to the Click event procedure of the Request button:

```
Sub Request_Click ()
    ' With a manual DDE link this button will be visible and when
    ' selected it will request an update of information from the source
    ' application to the destination application.
    Text1.LinkRequest
End Sub
```

8. Add the following code to the Click event procedure of the Poke button:

```
Sub Poke_Click ()
    ' With any DDE link this button will be visible and when selected
    ' it will poke information from the destination application to the
    ' source application.
    Text1.LinkPoke
End Sub
```

You can now run the Visual Basic for Windows destination application from the Visual Basic for Windows environment (skip to step 4) or you can save the application and create an .EXE file and run that from Windows (continue to step 1):

1. From the File menu, save the Form and Project using the name CLIENT.

2. From the File menu, choose Make an EXE File, and name it CLIENT.EXE.

3. Exit Visual Basic for Windows.

4. Run the application (from Windows if an .EXE file, or from the Run menu if from the Visual Basic for Windows environment). Form1 of the destination application will be loaded and Q+E will automatically be started with the database file ADDR.DBF loaded.

5. Make sure that the main title bar in Q+E reads "Q + E," NOT "Q + E - ADDR.DBF." If the title bar is incorrect, then from the Window menu of Q+E, choose Arrange All.

You can now experiment with DDE between Visual Basic for Windows and Q+E for Windows:

1. Try typing some text in R1C1 (the cell that holds the name "Tyler") in the Q+E spreadsheet and then choose the Request button. The text will appear in the Visual Basic for Windows text box.

2. Choose the Automatic Link button and then type some more text in R1C1 of the Q+E spreadsheet. The text is automatically updated in the Visual Basic for Windows text box.

3. Type some text in the text box in the Visual Basic for Windows application and choose the Poke button. The text is sent to R1C1 in the Q+E spreadsheet.

Note that if you do not have the Allow Editing option checked on the Edit menu in Q+E, you will not be able to change the contents of the Q+E spreadsheet. This may prevent some DDE operations. For example, attempting to LinkPoke to Q+E from Visual Basic for Windows when the Allow Editing option is not chosen will cause the program to crash and result in a "Foreign application won't perform DDE method or operation" error message. Attempting to change the contents of the spreadsheet from Q+E will result in a "Use the allow editing command before making changes" error message. From the Edit menu of Q+E, choose Allow Editing to enable this option. When viewed from the Edit menu, Allow Editing should have a check mark next to it when enabled.

You can also establish DDE between applications at design time. For more information, see page 356 of the "Microsoft Visual Basic: Programmer's Guide" version 1.0 manual, or Chapter 20 of the "Microsoft Visual Basic Programmer's Guide" version 2.0 manual.

For additional information on DDE between Microsoft Visual Basic for Windows and other Windows applications query on the following words in the Microsoft Knowledge Base:

   DDE and Visual Basic

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: IAPDDE

DDE Example Between Visual Basic and Windows Program Manager

Article ID: Q76551

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Windows, version 3.0 and 3.1

SUMMARY

This article demonstrates how to send dynamic data exchange (DDE) interface commands to the Microsoft Windows Program Manager from Microsoft Visual Basic for Windows using DDE.

The interface commands available through DDE with the Windows Program Manager are as follows:

```
CreateGroup(GroupName,GroupPath)
ShowGroup(GroupName,ShowCommand)
AddItem(CommandLine,Name,IconPath,IconIndex,XPos,YPos)
DeleteGroup(GroupName)
ExitProgman(bSaveState)
```

A full explanation of the above commands can be found in Chapter 22, pages 19-22 of the "Microsoft Windows Software Development Kit Guide to Programming" version 3.0 manual.

An application can also obtain a list of Windows groups from the Windows Program Manager by issuing a LinkRequest to the "PROGMAN" item.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The following program demonstrates how to use four of the five Windows Program Manager DDE interface commands and the one DDE request:

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Create the following controls with the given properties on Form1:

Object	Name	Caption
TextBox	Text1	
Button	Command1	Make
Button	Command2	Delete
Button	Command3	Request

(In Visual Basic version 1.0 for Window set the CtlName Property for the above objects instead of the Name property.)

3. Add the following code to the Command1_Click event:

```
Sub Command1_Click ()
  Text1.LinkTopic = "ProgMan|Progman"
  Text1.LinkMode = 2              ' Establish manual link.

  Text1.LinkExecute "[CreateGroup(Test Group)]"
          ' Make a group in Windows Program Manager.

  Text1.LinkExecute "[AddItem(c:\vb\vb.exe, Visual Basic)]"
          ' Add an item to that group.

  Text1.LinkExecute "[ShowGroup(Test Group, 7)]"
          ' Iconize the group and focus to VB application.

  On Error Resume Next  ' Disconnecting link with Windows Program
  Text1.LinkMode = 0    ' Manager causes an error in Windows 3.0.
             ' This is a known problem with Windows Program Manager.
End Sub
```

4. Add the following code to the Command2_Click event:

```
Sub Command2_Click ()
  Text1.LinkTopic = "ProgMan|Progman"
  Text1.LinkMode = 2          ' Establish manual link.

  Text1.LinkExecute "[DeleteGroup(Test Group)]"
          ' Delete the group and all items within it.

  On Error Resume Next  ' Disconnecting link with Windows Program
  Text1.LinkMode = 0    ' Manager causes an error in Windows 3.0.
              ' This is a known problem with Windows Program Manager.
End Sub
```

5. Add the following code to the Command3_Click event:

```
Sub Command3_Click ()
  Text1.LinkTopic = "ProgMan|Progman"
  Text1.LinkItem = "PROGMAN"
  Text1.LinkMode = 2      ' Establish manual link.
  Text1.LinkRequest       ' Get a list of the groups.

  On Error Resume Next  ' Disconnecting link with Windows Program
  Text1.LinkMode = 0    ' Manager causes an error in Windows 3.0.
             ' This is a known problem with Windows Program Manager.
End Sub
```

5. Press the F5 key to run the program.

6. Choose the Make button, then choose the Delete button. Note the result.

7. Choose the Request button. This will put a list of the groups

in the Windows Program Manager to be placed in the text box. The individual items are delimited by a carriage return plus linefeed.

As noted in the Windows Software Development Kit (SDK) manual mentioned above, the ExitProgman() command will only work if Windows Program Manager is NOT the shell (the startup program when you start Windows).

For a more comprehensive explanation of the CreateGroup, ShowGroup, AddItem, DeleteGroup, and ExitProgman commands, query on the following words in the Microsoft Knowledge Base:

DDE and CreateGroup

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: IAPDDE

Visual Basic and DDE/OLE with Other Windows Applications

Article ID: Q76562

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Microsoft Visual Basic for Windows can link to a number of Windows applications through dynamic data exchange (DDE). Visual Basic can also, through the addition of custom controls, link to other Windows applications through object linking and embedding (OLE). Custom controls for OLE support are provided with Microsoft Professional Toolkit for Visual Basic, available from Microsoft End User Sales and Service or from your nearest dealer of Microsoft products.

MORE INFORMATION

Visual Basic has built-in support for DDE. Visual Basic can link and share information with any other Windows application that also supports DDE.

Additional articles in this Knowledge Base discuss exactly how to establish a DDE link between Visual Basic and the following applications:

- Another Visual Basic application
- Microsoft Word for Windows
- Microsoft Excel for Windows
- Q+E (shipped with Microsoft Excel)

To locate these articles, query on the following words:

```
Visual and Basic and DDE
```

A Visual Basic application can also use OLE to link with any other Windows application that supports OLE.

OLE controls are not built into Visual Basic itself, but are readily available through the Microsoft Professional Toolkit for Visual Basic, available from Microsoft End User Sales and Service or your nearest Microsoft dealer.

A more challenging approach to obtain OLE support is to write your own custom control. With the Visual Basic Control Development Kit (CDK), along with either the Microsoft Windows Software Development Kit (SDK) and Microsoft C or Microsoft QuickC for Windows, you can create a custom control that supports OLE and add it to your Visual Basic application. The Visual Basic CDK is shipped as part of Microsoft

Professional Toolkit for Microsoft Visual Basic version 1.0 for Windows.

Below is a list of applications for Microsoft Windows and their abilities to support DDE and/or OLE.

Product	Version	Supports DDE	Supports OLE
Bookshelf	1.0	No	Yes
Money	1.0	No	Yes
Publisher	1.0	No	Yes
Visual Basic	1.0	Yes	No and Yes*
Excel	3.0	Yes	Yes
PowerPoint	2.0	No	Yes
Project	1.0	No	No
Word	1.0	Yes	No
Word	2.0	Yes	Yes
Works	2.0	No	No

* Not built into Visual Basic itself, but is available through Microsoft Professional Toolkit for Visual Basic or through another Visual Basic custom control.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: IAPOLE IAPDDE

PRB: Workaround for Not Enough Memory to Load Tutorial Error

Article ID: Q78000

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

Under any one of the conditions listed in the CAUSE section, If you try to run the Visual Basic tutorial, a message box will tell you, "Not Enough Memory To Load Tutorial."

CAUSE

- The Visual Basic tutorial is not actually installed.
- The current directory is not pointing to the location of VB.EXE.
- The VB.LES file is corrupt.

RESOLUTION

You can verify that the current directory is pointing to the location of VB.EXE by clicking the Visual Basic Icon in the Program Manager and choosing File Properties from the Program Manager Menu. The Working Directory option should specify the correct location of VB.EXE.

The subdirectory \VB\VB.CBT\ contains files for the Visual Basic tutorial. If the file VB.LES has been modified or replaced by another file, the tutorial cannot be run and two erroneous dialog boxes will open. The messages displayed in these dialog boxes are incorrect and should be ignored.

The first dialog box has the title "Visual Basic Tutorial" and displays the message "Out of memory". Choosing the OK button will clear this box and another one will open.

The second dialog box is titled "Microsoft Visual Basic." It displays the message "Not enough memory to load tutorial." Choose the OK button to clear this box.

To correct this problem, reinstall Visual Basic so that the VB.LES file is replaced by the correct file. Note that to reinstall Visual Basic correctly, you must first delete all files from the previous installation. Remember to save all of your program files (*.FRM, *.MAK, and so on) before deleting the previous installation.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: Setins

VB CDK VBAPI.LIB Contains CodeView Information

Article ID: Q78211

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY
=======

The Microsoft Visual Basic Control Development Kit (CDK) provides a library of Visual Basic API functions, VBAPI.LIB, which contains Microsoft CodeView information. This CodeView information may not be usable by non-Microsoft languages. The Visual Basic CDK was included with Microsoft Professional Toolkit for Microsoft Visual Basic programming system version 1.0 for Windows. And the CodeView information is provided with Visual Basic version 2.0.

A version of VBAPI.LIB without Microsoft CodeView information is available in the Software/Data Library and can be found by searching on the word VBAPI, the Q number of this article, or S13227. VBAPI was archived using the PKware file-compression utility.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: TlsCDK

How to Subclass a VB Form Using VB CDK Custom Control

Article ID: Q78398

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

In Windows programming terms, subclassing is the process of creating a message handling procedure and intercepting messages for a given window, handling any messages you choose, and passing the rest to the window's original message handler.

The subclass procedure is basically a message filter that performs non-default processing for a few key messages, and passes other messages to a default window procedure using CallWindowProc(). The CallWindowProc() function passes a message to the Windows system, which in turns sends the message to the target window procedure. The target window procedure cannot be called directly by the subclass procedure because the target procedure (in this case a window procedure) is exported.

Below is a simple example of how to subclass a Visual Basic form by writing a custom control using the Visual Basic Control Development Kit (CDK). The Visual Basic CDK is shipped as part of Microsoft Professional Toolkit for Microsoft Visual Basic version 1.0 for Windows and as part of the Professional Edition of Microsoft Visual Basic versions 2.0 and 3.0 for Windows.

MORE INFORMATION

The following code example demonstrates how to subclass a form from a custom control using the Visual Basic Custom CDK.

This example is developed using the CIRCLE.C program example from the CIRCLE1 project supplied with the CDK package. Only the file(s) that have changed from this project are included, and it is assumed that you have the additional CDK files as well as a C compiler capable of creating a Windows 3.0 compatible dynamic link library (DLL).

The basic idea for subclassing is to examine the window structure for a window directly using the GetWindowLong function to determine the address of the original window procedure. You can then change the address of the target window's window procedure to the address of your subclass procedure using SetWindowLong. In your subclass window procedure, you handle the messages you wish and use CallWindowProc to pass along other messages to the original window procedure.

```
//=================== CIRCLE1 ==================
// CIRCLE.C
// An example of subclassing a Visual Basic Form
//==============================================

#define  NOCOMM
#include <windows.h>

#include <vbapi.h>
#include "circle.h"

//declare the subclass procedure
LONG FAR PASCAL _export SbClsProc(HWND,USHORT,USHORT,LONG);

//far pointer to the default procedure
FARPROC lpfnOldProc = (FARPROC) NULL ;

//get the controls parent handle(form1)
HWND    hParent ;

//-------------------------------------------------------------
// Circle Control Procedure
//-------------------------------------------------------------
LONG FAR PASCAL _export CircleCtlProc (HCTL hctl, HWND hwnd,
     USHORT msg, USHORT wp, LONG lp)
{
   LONG lResult ;
   switch (msg)
   {
      case WM_CREATE:
         switch (VBGetMode())
         {
            //this will only be processed during run mode
            case MODE_RUN:
            {
               hParent = GetParent (hwnd) ;
               //get the address instance to normal proc
               lpfnOldProc = (FARPROC) GetWindowLong
                              (hParent, GWL_WNDPROC) ;
               //reset the address instance to the new proc
               SetWindowLong (hParent,
                        GWL_WNDPROC, (LONG) SbClsProc) ;
            }
            break ;
         }
         break ;
   }
   // call the default VB proc
   lResult = VBDefControlProc(hctl, hwnd, msg, wp, lp);
   return lResult;

}

LONG FAR PASCAL _export SbClsProc (HWND hwnd, USHORT msg,
     USHORT wp, LONG lp)
{
   switch (msg)
```

```
   {
      case WM_SIZE:
      {
      //place size event here for example...
      }
      break;
      case WM_DESTROY:
         SetWindowLong (hwnd, GWL_WNDPROC,
                        (LONG) lpfnOldProc) ;
      break ;
   }
   // call CircleCtlProc to process any other messages
   return (CallWindowProc(lpfnOldProc, hwnd, msg, wp, lp));
}

;========================================================
;Circle.def - module definition file for CIRCLE3.VBX control
;========================================================
LIBRARY         CIRCLE
EXETYPE         WINDOWS
DESCRIPTION     'Visual Basic Circle Custom Control'

CODE            MOVEABLE
DATA            MOVEABLE SINGLE

HEAPSIZE        1024

EXPORTS
           WEP       @1 RESIDENTNAME
           SbClsProc @2
;--------------------------------------------------------

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: TlsCDK
```

VB CDK Custom Property Name Cannot Start with Numeric Value

Article ID: Q78399

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The Property Name (npszName) field in the PROPINFO structure for the Visual Basic Control Development Kit (CDK) cannot start with a numeric value.

This information needs to be added to page 143 of the "Microsoft Visual Basic: Control Development Guide" shipped with Microsoft Professional Toolkit for Visual Basic 1.0 for Windows, or page 132 of the "Microsoft Visual Basic: Control Development Guide" shipped with the earlier CDK add-on for Microsoft Visual Basic.

MORE INFORMATION

When a control property starts with a numeric value, Visual Basic will generate the binding/syntax checking error "Expected: end-of-statement." However, the property works correctly in the Visual Basic design mode from the Properties bar (or the Properties window in version 2.0)

Steps to Reproduce Problem

1. Rebuild the Circle3 example provided with the CDK after changing the PROPINFO Property_FlashColor structure in CIRCLE3.H to the following:

   ```
   PROPINFO Property_FlashColor =
   {
   "2FlashColor", DT_COLOR | PF_fGetData | PF_fSetData |
   PF_fSaveData | PF_fEditable, OFFSETIN(CIRCLE,
   FlashColor)
   } ;
   ```

2. While in Visual Basic development environment (VB.EXE) with the Circle3 control loaded, assign the 2FlashColor property a value:

   ```
   Circle1.2FlashColor = 2
   ```

3. Press F5 to generate the "Expected: end-of-statement" error message. The text "FlashColor" will be selected for the syntax error.

Additional reference words: 1.00 2.00
KBCategory:

KBSubcategory: TlsCDK

PRB: SETUP.EXE Error: Insufficient Disk Space on: C:\WINDOWS

Article ID: Q78961

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SYMPTOMS

Visual Basic displays the following message during setup if there is less 382K of space in version 2.0 or less than 330K of space in version 1.0 available to Windows on the drive where Windows resides -- which may be different from the drive where you are installing Visual Basic.

```
   Error - Insufficient disk space on: C:\WINDOWS
```

STATUS

This behavior is by design.

MORE INFORMATION

At first, SETUP.EXE for Visual Basic copies the files VBSETUP.EXE and VBRUN100.DLL into the Windows subdirectory. If there is not enough space on the drive where Windows resides (such as in C:\WINDOWS), Visual Basic will display the error.

This is the disk space available to Windows just before setup. This may differ from the amount of space reported at the MS-DOS command prompt outside of Windows because of temporary files that Windows creates during operation.

VBSETUP.EXE is deleted when setup is completed. VBRUN100.DLL is copied over to the Visual Basic subdirectory, but is not deleted from the Windows subdirectory.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: Setins

Call VBSetErrorMessage() In Response to VBM_ Messages Only

Article ID: Q80403

The information in this article applies to:

- Microsoft Visual Basic Control Development Kit (CDK) for Microsoft Visual Basic Programming system for Windows, version 1.0
- Microsoft Professional Toolkit for Microsoft Visual Basic, version 1.0
- Professional Edition of Microsoft Visual Basic for Windows, version 2.0

SYMPTOMS

The Visual Basic Control Development Kit (CDK) API function VBSetErrorMessage() operates correctly only when called in response to a VBM_ message, such as VBM_SETPROPERTY.

STATUS

This behavior is by design.

MORE INFORMATION

The VBSetErrorMessage() function can be called from a custom control in response to a VBM_ message to pass an error number and message back to Visual Basic. When execution returns to Visual Basic, a trappable run-time error will occur, with the error number and message specified in the call to VBSetErrorMessage.

The VBSetErrorMessage routine works only in response to messages that originate from Visual Basic itself (VBM_ messages). Visual Basic responds to the return code for VBM_ messages, and in turn sets the error condition in the program. If the return code for a VBM_ message is True, Visual Basic will generate an error condition. For other messages (non VBM_ messages), Visual Basic must pass along the return code to the originator of the message (usually Windows); therefore, Visual Basic will not generate an error condition for these messages.

Reference(s):

"Microsoft Visual Basic: Control Development Guide," (c) 1992, page 117 (shipped with Professional Toolkit)

"Microsoft Visual Basic: Control Development Guide," (c) 1991, page 108 (part no. 20666)

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: TlsCDK

Getting Program Manager Group Names into Combo Box in VB

Article ID: Q80410

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

To get a list of group names in the Windows 3.0 Program Manager, you can call the Windows API GetPrivateProfileString function from a Visual Basic program. This article describes a method of using the Windows API GetPrivateProfileString function to get all the group names from Program Manager and place them into a Visual Basic combo box.

MORE INFORMATION

Windows initialization (.INI) files contain information that defines your Windows environment. Examples of Windows initialization files are WIN.INI and SYSTEM.INI, which are commonly found in the C:\WINDOWS subdirectory. Windows and Windows applications can use the information stored in these files to configure themselves to meet your needs and preferences. For a description of initialization files, read the WININI.TXT file that comes with Microsoft Windows 3.0.

An initialization file is composed of at least an application name and a key name. The contents of Windows initialization files have the following format:

```
[Application name]
keyname=value
```

The GetPrivateProfile family of API functions are used to retrieve information from any initialization file that you specify.

To declare this API function within your program, include the following Declare statement in the global module or the general Declarations section of a Visual Basic form. The entire Declare statement must be on one, single line.

```
Declare Function GetPrivateProfileString% Lib "Kernel"
  (ByVal lpAppName$, ByVal lpKeyName$, ByVal lpDefault$,
   ByVal lpReturnedString$, ByVal nSize%, ByVal lpFileName$)
```

The formal arguments to these functions are described as follows:

Argument	Description
lpAppName$	Name of a Windows application that appears in the

	.INI file.
lpKeyName$	Key name that appears in the .INI file.
lpFileName$	Points to a string that names the .INI file. If lpFileName does not contain a path to the file, Windows searches for the file in the Windows directory.
lpDefault$	Specifies the default value for the given key if the key cannot be found in the .INI file.
lpReturnedString$	Specifies the buffer that receives the character string.
nSize%	Specifies the maximum number of characters (including the last null character) to be copied to the buffer.

Code Example

To get the group names from Program Manager into a combo box, do the following:

1. Start Visual Basic or from the File menu, select New Project (ALT, F, N) if Visual Basic is already running. Form1 will be created by default.

2. Add a combo box (Combo1) to Form1.

3. Within the global Declarations section of Form1, add the following Windows API function declaration. Note that the Declare statement below must appear on a single line.

```
Declare Function GetPrivateProfileString% Lib "kernel"
  (ByVal lpAppName$, ByVal lpKeyName$,ByVal
   lpDefault$,ByVal lpReturnString$,ByVal nSize%,
   ByVal lpFileName$)
```

4. Within the Form_Load event procedure for Form1, add the following code:

```
Sub Form_Load()
    ' This is the name of the group in the PROGMAN.INI file
    lpAppName$ = "Groups"

    ' All group names start with Group: Group1, Group2, etc.
    lpKeyName$ = "Group"

    ' If no group found return value in lpDefault$
    lpDefault$ = ""

    ' Initialize string
    lpReturnString$ = Space$(128)
    Size% = Len(lpReturnString$)
```

```
      ' This is the path and name the PROGMAN.INI file.
      lpFileName$ = "c:\windows\progman.ini"

      Valid% = 1
      i% = 0

      While (Valid%)

        i% = i% + 1

        ' The following three lines must be typed on a single line
        Valid% = GetPrivateProfileString(lpAppName$, lpKeyName$
               + LTrim$(Str$(i%)), lpDefault$, lpReturnString$,
               Size%, lpFileName$)

        ' Discard the trailing spaces and null character.
        group$ = Left$(lpReturnString$, Valid%)

        ' check to see if string was returned. Change arguments
        ' passed to the Mid$ statement to change what is displayed in combo
        ' box. By setting number to 15 this strips c:\windows\
        ' and .GRP
        ' The following 2 lines must be on one line
        If Valid% > 0 Then combo1.AddItem Mid(group$, 12,
             Len(group$) - 15)
      Wend

      ' Set text of combo box to first item in list
      combo1.listindex = 0

   End Sub
```

5. From the Run menu, choose Start (ALT, R, S). The combo box will contain the filenames (without the extension) of the group (.GRP extension) files in the Windows directory. The group name conforms to the MS-DOS filename convention; it is limited to eight characters.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: IAPDDE

VB DDE to Excel with Embedded TAB Can Truncate String in Excel

Article ID: Q82157

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

If you send strings containing TAB characters in a dynamic data exchange (DDE) conversation from Microsoft Visual Basic for Windows to Microsoft Excel, the string may be truncated in Excel if you specify a specific row and column in the Visual Basic for Windows LinkItem property. If you do not specify a column in the LinkItem property but only specify a specific row, your string will be parsed by Excel, and each TAB will cause the characters following the TAB to be entered into the following cell in Excel.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

This behavior occurs when the following is true:

- A string that you are trying to send to Excel through DDE contains an embedded TAB.
- You set your LinkItem property to a specific Excel cell (both row and column, such as R1C1, meaning row 1 column 1).

The attempted conversation will result in a truncated string. For example, if you pass the following string to Excel

```
"The cow jumped" + Chr$(9) + "over the moon"
```

and if the two conditions above are true, the only thing you will see on the Excel side is "The cow jumped". The rest of the string will be lost.

The following code example passes strings to Excel from a list box with TAB-delimited columns. Run the program twice, and uncomment the LinkItem line to see the different behavior.

Steps to Reproduce Behavior

1. Run Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Put a text box on the form (Form1), and change the Name (change CtlName in Visual Basic version 1.0 for Windows) property to "ddebox".

3. Put a list box (List1) and a command button (Command1) on Form1.

4. Add the following code to the Form_Load procedure:

```
Sub Form_Load ()
  Form1.Show
  ' Add items to list box with TABs embedded.
  List1.AddItem "hey" + Chr$(9) + "is"
  List1.AddItem "for" + Chr$(9) + "horses"
End Sub
```

5. Add the following code to the Command1_Click event procedure:

```
Sub Command1_Click ()
    Const NONE = 0, COLD = 2     ' Define constants.

    If ddebox.LinkMode = NONE Then
        Z% = Shell("Excel", 4)   ' Start Excel.
       ' Set link topic.
        ddebox.LinkTopic = "Excel|Sheet1"
        ddebox.LinkItem = ""     ' Set link item.
        ddebox.LinkMode = COLD   ' Set link mode.
    End If

    ' Loop through all items in list box:
    For i% = 0 To List1.ListCount - 1
      Row$ = Format$(i% + 1)         ' Format row variable.
      ' ddebox.LinkItem = "R"+Row$   ' Take out comment to send entire
                                     ' string.
      ' Comment next line when uncommenting above line.
      ddebox.LinkItem = "R" + Row$ + "C1"   ' This statement truncates
                                            ' string in Excel.
      ddebox.text = List1.list(i%) ' Assign text box to list box string.
      ddebox.LinkPoke     ' Send the string to Excel.
    Next

    ddebox.LinkMode = NONE
End Sub
```

For best results, make sure Excel is not running before you start the program. When you start the program, notice the list box has the strings added to it during the form Load event. If you choose the command button to initialize the DDE conversation with the program typed in exactly as shown, the following will appear in Excel:

```
hey   ' This will be in cell A1.
for   ' This will be in cell A2.
```

If you change the assignment statement of the LinkItem of the ddebox from

```
   ddebox.LinkItem = "R" + Row$ + "C1"
```

to

```
   ddebox.LinkItem = "R"+ Row$
```

notice that the entire string is passed to Excel with the following results:

```
hey       is      ' These words will be in A1 and B1.
for        horses ' These words will be in A2 and B2.
```

The reason for this behavior is that Excel uses TABs as its delimiter. You can use this method to send multiple items to Excel, placing them in their own cells if desired. If that is not the desired result, you will have to make sure you compensate for the lost parts of the string.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: IAPDDE

VB Example of Using DDE LinkExecute to Word for Windows 2.0

Article ID: Q82879

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0
- Microsoft Word for Windows, versions 2.0 and 6.0

SUMMARY

This article demonstrates how to send a LinkExecute event to Microsoft Word for Windows from Microsoft Visual Basic for Windows using dynamic data exchange (DDE).

The commands available through DDE with Word for Windows are as follows:

- Any Macro in Word for Windows
- Any embedded WordBasic command built into Word for Windows

A full explanation of the above commands can be found in Word for Windows online Help under the topic "WordBasic."

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The following example program demonstrates how to:

- Automatically start Word for Windows
- Automatically send text typed in a Visual Basic for Windows text box to the Word for Windows document
- Print the Word for Windows document to the selected printer.

1. Start Visual Basic for Windows, or from the File menu, choose New Project (press ALT, F, N) if Visual Basic for Windows is already running. Form1 is created by default.

2. Create the following controls with the given properties on Form1:

Object	Name	Caption
TextBox	Text1	
Button	Command1	Start Word
Button	Command2	Link
Button	Command3	Send Text
Button	Command4	Print

(In Visual Basic version 1.0 for Windows set the CtlName Property

for the above objects instead of the Name property.)

3. Add the following code to the Command1_Click event:

```
Sub Command1_Click ()
   x = Shell("winword.exe", 7) ' Start Word for Windows minimized
                               ' without the focus.
   x = DoEvents()              ' This gives WinWord time to load.
End Sub
```

4. Add the following code to the Command2_Click event procedure:

```
Sub Command2_Click ()
   text1.LinkMode = 0 ' Clears DDE link if it already exists.
   text1.LinkTopic = "WinWord|document1" ' Set up link w/ WINWORD.EXE.

   text1.LinkMode = 2       ' Establish a manual DDE link.
   text1.linktimeout = 60   ' Set the time for a response to 6 seconds.
   ' If a DDETIMEOUT occurs increase the Text1.Linktimeout.

   ' Enter the following two lines as one, single line:
   text1.LinkExecute
      "[InsertBookmark .Name="+Chr$(34)+"Test"+Chr$(34)+"]"

   ' NOTE: the space is necessary as shown before .Name in the above
   ' LinkExecute statement.

   ' For Microsoft Word version 6.0, use the following instead and
   ' enter the two lines as one single line -- after removing the
   ' single quotation mark from the start of both lines:
   ' text1.LinkExecute
   '    "[EditBookmark .Name ="+Chr$(34)+"Test"+Chr$(34)+"]"

   text1.LinkItem = "Test"      ' Set link to a bookmark on document.
End Sub
```

5. Add the following code to the Command3_Click event procedure:

```
Sub Command3_Click ()
   text1.LinkPoke  ' Sends the contents of the text box.
End Sub
```

6. Add the following code to the Command4_Click event procedure:

```
Sub Command4_Click ()
   text1.LinkExecute "[FilePrintDefault]"   ' Prints the doc with the
                                            ' default printer settings.
End Sub
```

7. Press the F5 key to run the program.

8. Choose the Start Word button.

9. Choose the Link button. This will establish a DDE conversation with Word's Document1 and create a bookmark called Test using LinkExecute and the embedded InsertBookmark WordBasic command. It will then set the LinkItem to this newly created bookmark in Document1.

10. Type some text in the text box and choose the Send Text command
    button to send the contents of the text box to Word for Windows.

11. Choose the Print button to print the document in Word for Windows.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: IAPDDE

VB CDK: Example of Subclassing a Visual Basic Form

Article ID: Q83806

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The subclass procedure is a message filter that performs non-default processing for a few key messages, and passes other messages to a control's default window procedure using CallWindowProc. The CallWindowProc function passes a message to Windows, which in turn sends the message to the target window procedure. The target window procedure cannot be called directly by the subclass procedure because the target procedure is exported.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The following code example demonstrates how to subclass a form using the Microsoft Visual Basic for Windows Custom Control Development Kit (CDK).

This example is developed using the CIRCLE.C source file from the CIRCLE1 project supplied with the CDK package. Only the file(s) that have changed from this project are included, and it is assumed that you have the additional CDK files.

```
  //=================== CIRCLE1 ==============================
  // CIRCLE.C
  // An example of subclassing a Visual Basic for Windows Form
  //==========================================================

#define  NOCOMM
#include <windows.h>

#include <vbapi.h>
#include "circle.h"

// Declare the subclass procedure.
LONG FAR PASCAL _export SbClsProc(HWND,USHORT,USHORT,LONG);

// Far pointer to the default procedure.
FARPROC lpfnOldProc = (FARPROC) NULL ;

// Get the controls parent handle(form1).
```

```
HWND    hParent ;

//-------------------------------------------------------
// Circle Control Procedure
//-------------------------------------------------------
LONG FAR PASCAL _export CircleCtlProc (HCTL hctl, HWND hwnd,
     USHORT msg, USHORT wp, LONG lp)
{
   LONG lResult ;
   switch (msg)
   {
      case WM_CREATE:
         switch (VBGetMode())
         {
            // This will only be processed during run mode.
            case MODE_RUN:
            {
               hParent = GetParent (hwnd) ;
               // Get the address instance to normal proc.
               lpfnOldProc = (FARPROC) GetWindowLong
                              (hParent, GWL_WNDPROC) ;
               // Reset the address instance to the new proc.
               SetWindowLong (hParent,
                        GWL_WNDPROC, (LONG) SbClsProc) ;
            }
            break ;
         }
         break ;
   }
   // Call the default VB for Windows proc.
   lResult = VBDefControlProc(hctl, hwnd, msg, wp, lp);
   return lResult;

}

LONG FAR PASCAL _export SbClsProc (HWND hwnd, USHORT msg,
     USHORT wp, LONG lp)
{
   switch (msg)
   {
      case WM_SIZE:
      {
      // Place size event here for example...
      }
      break;
      case WM_DESTROY:
         SetWindowLong (hwnd, GWL_WNDPROC,
                         (LONG) lpfnOldProc) ;
      break ;
   }
   // Call CircleCtlProc to process any other messages.
   return (CallWindowProc(lpfnOldProc, hwnd, msg, wp, lp));
}

;============================================================
;Circle.def - module definition file for CIRCLE3.VBX control
;============================================================
```

```
LIBRARY         CIRCLE
EXETYPE         WINDOWS
DESCRIPTION     'Visual Basic Circle Custom Control'

CODE            MOVEABLE
DATA            MOVEABLE SINGLE

HEAPSIZE        1024

EXPORTS
           WEP       @1 RESIDENTNAME
           SbClsProc @2
;-------------------------------------------------------
Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: TlsCDK
```

VB Example of Using DDE to Run a Word 2.0 for Windows Macro

Article ID: Q85857

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article demonstrates how to send a LinkExecute command to Word for Windows version 2.0 from Visual Basic using dynamic data exchange (DDE) to run a macro.

MORE INFORMATION

The following example program demonstrates how to automatically start Word for Windows and execute a WinWord macro called MyMacro.

Steps to Create Example Program

1. Run Visual Basic, or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Create the following controls on Form1 with the following properties:

Object	CtrlName	Caption
TextBox	Text1	
Button	Command1	Start Word
Button	Command2	MyMacro

3. Add the following code to the Command1 Click event:

```
Sub Command1_Click ()
    x = Shell("winword.exe", 7) 'Start Word for Windows
                                'minimized without the focus
 End Sub
```

4. Add the following code to the Command2 Click event:

```
Sub Command2_Click ()
    Text1.LinkMode = 0 'Clears DDE link if it already exists.
    Text1.LinkTopic = "Winword|document1"   'Sets up link with
                                            'WINWORD.EXE.
    Text1.LinkMode = 2        'Establish a cold DDE link.
    Text1.LinkTimeout = 60    'Set the time for a response to 6 seconds;
```

```
                                'if a DDETIMEOUT occurs, increase the
                                'Text1.LinkTimeout
' Enter the following two lines as one, single line:
Text1.LinkExecute
   "[ToolsMacro .Name ="+Chr$(34)+"MyMacro"+Chr$(34)+",.Run]"
    '(Note that the space is necessary as shown before .Name in the
    ' above LinkExecute statement.)
End Sub
```

5. Create a macro called MyMacro in WinWord that inserts "hello world" in the document:

   a. Switch to WinWord.

   b. From the Tools menu, choose Macro.

   c. Type "MyMacro" in the Macro Name field. Choose the Edit button.

   d. Type the following:

```
      Insert "Hello World"
```

   e. From the File menu, choose Close. At the "Do you want to keep the changes to Global: MyMacro?" prompt, choose Yes (this will save the newly created MyMacro macro).

   f. From the File menu, choose Exit. At the "Do you want to save the global glossary and command changes?" prompt, choose Yes. (The MyMacro macro has been added to the WinWord NORMAL.DOT file.)

6. Press F5 to run the program.

7. Choose the Start Word button.

8. Choose the MyMacro button. This will establish a DDE conversation with Word Document1 and execute the MyMacro macro.

9. Switch to WinWord to verify that the Document1 contains "Hello World," confirming that the MyMacro macro has been run (the CTRL+HOME key combination will move the cursor back to the beginning of the document).

Additional reference words: 1.00 2.00 3.00 winword
KBCategory:
KBSubcategory: IAPDDE

How to Use a Linked Paintbrush Object with OLECLIEN.VBX

Article ID: Q86776

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The following example program demonstrates how to use the Visual Basic OLE Client (OLECLIEN.VBX) custom control to create a linked Paintbrush object.

The following OLEClient property settings are required to create a Paintbrush Object Linking and Embedding (OLE) object:

   Class   - "PBrush"

   SourceDoc - The full path of a bitmap file to use (for example, c:\windows\arches.bmp).

   SourceItem - A string containing the pixel coordinates of the part of the bitmap to display. These coordinates should be in the format "x1 y1 x2 y2".

This information applies to the OLECLIEN.VBX custom control in Visual Basic.

Note that Windows version 3.0 Paintbrush does not support OLE; you must have Windows version 3.1 in order to use this example.

MORE INFORMATION

The following program demonstrates how to create a linked Paintbrush object in Visual Basic using the OLECLIEN.VBX custom control.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the OLECLIEN.VBX custom control file. The OLE Client tool appears in the Toolbox.

3. Place a command button and an OLEClient control on Form1.

4. Enter the following code:

```
Sub Command1_Click()
   OLEClient1.Class = "PBrush"
```

```
      OLEClient1.Protocol = "StdFileEditing"
      OLEClient1.SourceDoc = "c:\windows\arches.bmp"

      '  The SourceItem for Paintbrush is the coordinates of
      '  of an object image in bitmap - "x1 y1 x2 y2".
      OLEClient1.SourceItem = "0 0 121 159"

      OLEClient1.ServerType = 0  ' Linked.
      OLEClient1.Action = 1      ' CreateFromFile.
      Command1.Enabled = 0
   End Sub

   Sub OleClient1_DblClick ()
      OLEClient1.Action = 7  ' Activate (open for editing).
   End Sub

   Sub Form_Unload (Cancel As Integer)
      OLEClient1.Action = 9  ' Close (terminate connection).
   End Sub
```

5. Press F5 to run the program. Click the command button to create the OLE object. Double-clicking the OLEClient control will start Paintbrush for you to edit the OLE object.

Reference(s):

"Microsoft Professional Toolkit for Visual Basic: Custom Control Reference" Pages 196-232

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: IAPOLE

How to Obtain a Listing of Classes for OLE Client Control

Article ID: Q87001

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

Below is an example of how to obtain a list of the Object Linking and Embedding (OLE) class properties for the OLE Client control in Visual Basic. This example is based on the ServerAcceptFormats example on page 214 in the "Microsoft Professional Toolkit for Visual Basic: Custom Control Reference" for version 1.0.

This example gets the information from the REG.DAT file in your Windows directory. It uses the ServerClasses property to return a listing of the classes to a list box. The Class property is discussed on pages 198-201 and 207 of the "Microsoft Professional Toolkit for Visual Basic: Custom Control Reference" for version 1.0. The ServerClasses property is discussed on pages 201 and 217 in the same manual.

Note that the CtlName property in Visual Basic version 1.0 has been changed to the Name property in Visual Basic version 2.0.

MORE INFORMATION

This example uses a single form with two list boxes, two labels, and one OLE Client control. One list box should have a CtlName (or Name) of Identifier, and the other list box should have a CtlName (or Name) of FileType. Each label is placed above a list box, with the captions of Identifier and File Type, respectively.

There are three event procedures (Form_Load, Identifier_Click, and FileType) and one procedure, located in the general section of Form1, called Fillitems(S$).

The example results in two lists. The available OLE classes are listed in the Identifier list box, and the Class File Types are listed in the File Type list box.

When you click a certain class in the Identifier list box, the associated class display is highlighted in the second Identifier-Display list box.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File. In the Files box, select the OLECLIENT.VBX custom control file. The OLE Client tool will appear in the Toolbox.

3. Add two label boxes (Label1 and Label2) and two list boxes (List1 and List2) to Form1. Position Label1 above List1, and Label2 above List2.

4. Change the Control Name of List1 to Identifier, and change the Caption of Label1 to Identifier.

5. Change the Control Name of List2 to FileType, and change the Caption of Label2 to FileType.

6. Add the following code to the Form_Load event procedure:

```
Sub Form_Load ()
   Dim I As Integer
   ' Fill the Identifier and FileType list boxes
   For I = 0 To OLEClient1.ServerClassCount - 1
      Identifier.AddItem OLEClient1.ServerClasses(I)
      FileType.AddItem OLEClient1.ServerClassesDisplay(I)
   Next I
End Sub
```

7. Add the following code to the Identifier_Click event procedure after you have changed the control name in step 4 above:

```
Sub Identifier_Click ()
   ' When user selects a Class, highlight the associated ClassDisplay.
   FileType.ListIndex = Identifier.ListIndex
   ' Display information associated with the selected class.
   FillItems (OLEClient1.ServerClasses(Identifier.ListIndex))
End Sub
```

8. Add the following code to the FileType_Click event procedure after you have changed the control name in step 4:

```
Sub FileType_Click ()
   ' When user selects a ClassDisplay, highlight the associated Class.
   Identifier.ListIndex = FileType.ListIndex
End Sub
```

9. Add the following code to the (general) section of the form's Code window under Object:

```
Sub FillItems (S$)
   Dim I As Integer
   ' Set the ServerClass.
   OLEClient1.ServerClass = S$
End Sub
```

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: IAPOLE

Visual Basic 3.0 Setup & Installation Questions & Answers

Article ID: Q92546

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, version 3.0

---

1. Q. I am having difficulty with the ODBC Setup and Installation. Is there any information on how this should be done for various databases?

   A. There are several help files and text files that deal specifically with ODBC setup and connections issues. You can find a list of these and other information files by following three steps:

      1. Open the Visual Basic help file.
      2. Choose the Contents button.
      3. Select "Other Information sources."

2. Q. The setup program for Microsoft Visual Basic version 3.0 for Windows takes from 15 to 30 minutes to finish. Is this normal?

   A. No, this is not normal behavior for Visual Basic setup. We are aware of one configuration that displays this symptom. The problem is with an SCSI (scuzzy) driver (ASPIDOS) loaded in high memory. If you load this driver in low memory, there is no problem.

3. Q. I successfully installed Microsoft Visual Basic version 3.0 for Windows with no error messages, but all the help file icons in the VB group in Program Manager are gray MS-DOS icons. When I choose these icons, I get an error message that says:

      Cannot Run Program. There is no application associated with this file. Choose Associate from the File menu to create an association.

      Why does this happen?

   A. This is a known problem with The Setup program in the Professional edition of Visual Basic version 3.0 for Windows. The Setup program adds the following problem line to the extensions section of the WIN.INI file if no association for .HLP files currently exists:

      ```
      HLP=D:\WINDOWS\SETUPWIZ.INI ^.HLP
      ```

      To fix the problem replace the line with this line:

      ```
      HLP=WINHELP.EXE ^.HLP
      ```

      For more information on this problem, please see Microsoft Knowledge Base article Q100191.

Additional reference words: 3.00 ivrfax fasttips

KBCategory:
KBSubcategory: Setins

Visual Basic 3.0 Programming Questions & Answers

Article ID: Q92550

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, version 3.0

---

1. Q. I use the picture control to group other controls. However when I select the picture control, the other controls do not remain on top of the picture control. How can I correct this problem?

   A. This problem occurs if you place the controls on the form in the same place as the picture control but not in the picture control. itself. To group the controls in a picture control, you must first select the Picture control and then draw the desired control within the Picture control. For more information, please see Chapter 3 of the "Microsoft Visual Basic version 3.0 Programmer's Guide."

2. Q. How can I make calls from Visual Basic to the functions in the Windows Application Programming Interface (API) or other dynamic link libraries (DLLs)?

   A. To call a subroutine or function from one of the Windows APIs or any other DLL, you need to first provide a Declare statement for that subroutine or function in your Visual Basic application. The exact syntax for the declaration for each Windows API function can be found in the WIN31API.HLP help file included with the Professional Edition of Visual Basic. For more information, please see Chapter 24 of the "Microsoft Visual Basic version 3.0 Programmer's Guide."

3. Q. Is there a reference available that lists the correct Visual Basic declarations for the Multimedia API functions?

   A. Yes, the file is called WINMMSYS.TXT. It comes with the Professional edition of Visual Basic. You can find it in the \VB\WINAPI directory.

4. Q. Is there a reference available that lists the correct Visual Basic declarations for the Windows for Workgroups API functions?

   A. No, at this time such a file is not available from Microsoft. However, you can obtain a copy of the Windows for Workgroups SDK from the WINEXT forum on CompuServe.

5. Q. I followed the examples in the manuals and in the help file on how to use Domain functions such as DSum and DCount, but I keep receiving this error:

      Reference to undefined function or array.

      Why?

   A. The examples provided for the Domain Aggregate functions are

incorrect. These functions must be used within an SQL Statement just as SQL Aggregate functions such as Sum and Count are used. Please look at the SQL Aggregate examples to see how to use these functions within an SQL Statement. For more information, query on the following words in the Microsoft Knowledge Base:

DOMAIN and FUNCTION and SQL

6. Q. I want to sort the records referenced by the Data Control in my application. I tried to use the Index Property as described in the example in the manual and in the help file, but I receive the following error message:

   Property 'Index' not found

   Why?

   A. The examples provided in the Index Property are incorrect. The Index property does not apply to the Data Control. To sort the records referenced by the Data Control, use the ORDER BY Clause within an SQL Statement in the RecordSource property of the Data Control.

7. Q. Is there a better way than the Print Form method to print Forms and Controls in a program?

   A. Yes, it is possible to print forms and/or controls and specify the printed size by using various Windows API function calls. This process is documented in Microsoft Knowledge Base article Q85978. You can also find this article in the top 10 Microsoft Knowledge Base articles that are in the Visual Basic help file. To view these articles, select "Technical Support" from the Contents screen in the Visual Basic help file. Then select "Knowledge Base Articles on Visual Basic."

Additional reference words: 3.00 ivrfax fasttips
KBCategory:
KBSubcategory: PrgCtrlsStd APrgOther TlsCDK

How to Establish a Network DDE Link Using Visual Basic

Article ID: Q93160

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article demonstrates how to establish a network Dynamic Data Exchange (DDE) link between two computers running Microsoft Windows for Workgroups.

MORE INFORMATION

Under DDE, a destination (or client) application sends commands through DDE to the source (or server) application to establish a link. Through DDE, the source provides data to the destination at the request of the destination or accepts information at the request of the destination. When you use DDE with Windows version 3.0 or 3.1 based applications, the source and destination applications are both located on the same computer.

When you use Network DDE with Windows for Workgroups based applications, DDE functions exactly the same way as standard DDE except that the source and destination applications are located on different computers.

Before establishing a network DDE link, you must first establish a network DDE share for the conversation by calling the API NDdeShareAdd() function located in the NDDEAPI.DLL file. Here is the Visual Basic declaration:

```
' Enter the following as one, single line:
Declare Function NDdeShareAdd Lib "NDDEAPI.DLL" (Server As Any, ByVal Level
   As Integer, ShareInfo As NDDESHAREINFO, ByVal nSize As Long) As Integer
```

Enter the entire statement as a single line. The first parameter is always a 0 and is passed with ByVal 0& from Visual Basic. The second parameter is always 2. The next parameter is a filled ShareInfo structure (given below). The last parameter is the size of the ShareInfo structure.

Here is The structure of the NDDESHAREINFO structure:

```
   Type NDDESHAREINFO
      szShareName As String * MAX_NDDESHARENAME_PLUSONE
      lpszTargetApp As Long    'LPSTR lpszTargetApp
      lpszTargetTopic As Long  'LPSTR lpszTargetTopic
      lpbPassword1 As Long     'LPBYTE lpbPassword1
      cbPassword1 As Long      'DWORD  cbPassword1;

      dwPermissions1 As Long   'DWORD  dwPermissions1;
      lpbPassword2 As Long     'LPBYTE lpbPassword2;
      cbPassword2 As Long      'DWORD  cbPassword2;
```

```
      dwPermissions2  As Long  'DWORD  dwPermissions2;
      lpszItem As Long         'LPSTR  lpszItem;
      cAddItems As Long        'LONG   cAddItems;
      lpNDdeShareItemInfo As Long
   End Type
```

The following table describes each field of the NDDESHAREINFO type:

Field Name	Purpose
szShareName	Name of the share to add.
lpszTargetApp	Pointer to null-terminated string containing the service or application name.
lpszTargetTopic	Pointer to null-terminated string holding the topic name
lpbPassword1	Pointer to the read-only password -- uppercase, null-terminated string. If null, pass null string, not zero.
cbPassword1	Length of read-only password
dwPermissions1	Full access password
cbPassword2	Length of the full access password
dwPermissions2	Permissions allowed by the full access password

Here are the permissions allowed for dwPermissions:

Name	Value	Function
NDDEACCESS_REQUEST	&H1	Allows LinkRequest
NDDEACCESS_ADVISE	&H2	Allows LinkAdvise
NDDEACCESS_POKE	&H4	Allows LinkPoke
NDDEACCESS_EXECUTE	&H8	Allows LinkExecute
NDDEACCESS_START_APP	&H10	Starts source application on connect

Here are the possible return values from NDdeShareAdd():

Name	Value	Meaning
NDDE_NO_ERROR	0	No error.
NDDE_BUF_TOO_SMALL	2	Buffer is too small to hold information.
NDDE_INVALID_APPNAME	13	Application name is not valid.
NDDE_INVALID_ITEMNAME	9	Item name is not valid.
NDDE_INVALID_LEVEL	7	Invalid level; nLevel parameter must be 2.
NDDE_INVALID_PASSWORD	8	Password is not valid.
NDDE_INVALID_SERVER	4	Computer name is not valid; lpszServer parameter must be NULL.
NDDE_INVALID_SHARE	5	Share name is not valid.
NDDE_INVALID_TOPIC	10	Topic name is not valid.
NDDE_OUT_OF_MEMORY	12	Not enough memory to complete request.
NDDE_SHARE_ALREADY_EXISTS	15	Existing shares cannot be replaced.

There are two steps to establish a network Dynamic Data Exchange (DDE) link between two computers running Microsoft Windows for Workgroups. First, create the DDE source application. Second, create the DDE destination application.

Step One -- Create DDE source application

The following steps show you how to create a Visual Basic DDE source and destination application that communicates through a network DDE link.

1. From the DDE source computer, start Visual Basic or if Visual Basic is already running, from the File menu, choose New Project (ALT, F, N). Form1 is created by default.

2. Change the LinkTopic property of Form1 to VBTopic.

3. If you are running Visual Basic version 2.0 or 3.0 for Windows, change the LinkMode property of Form1 to 1 - Source. In Visual Basic version 1.0, this property is already set to 1 - Server; don't change it.

4. Add a text box (Text1) to Form1.

5. Change the Name property (CTlName in version 1.0) of Text1 to VBItem.

6. Add a timer (Timer1) to Form1.

7. From the File menu, choose New Module (ALT, F, M). Module1 is created.

8. Add the following code to the general declarations section of Module1, and enter all lines as a single line even though they may be shown on multiple lines for readability:

```
' DDE access options
Global Const NDDEACCESS_REQUEST = &H1
Global Const NDDEACCESS_ADVISE = &H2
Global Const NDDEACCESS_POKE = &H4
Global Const NDDEACCESS_EXECUTE = &H8
Global Const NDDEACCESS_START_APP = &H10
Global Const MAX_NDDESHARENAME_PLUSONE = 65
Type NDDESHAREINFO
   szShareName As String * MAX_NDDESHARENAME_PLUSONE
   lpszTargetApp As Long    'LPSTR lpszTargetApp
   lpszTargetTopic As Long  'LPSTR lpszTargetTopic
   lpbPassword1 As Long     'LPBYTE lpbPassword1
   cbPassword1 As Long      'DWORD  cbPassword1;
   dwPermissions1 As Long   'DWORD  dwPermissions1;
   lpbPassword2 As Long     'LPBYTE lpbPassword2;
   cbPassword2 As Long      'DWORD  cbPassword2;
   dwPermissions2  As Long  'DWORD  dwPermissions2;
   lpszItem As Long         'LPSTR  lpszItem;
   cAddItems As Long        'LONG   cAddItems;
   lpNDdeShareItemInfo As Long
End Type
Declare Function NDdeShareAdd Lib "NDDEAPI.DLL" (Server As Any, ByVal
   Level As Integer, ShareInfo As NDDESHAREINFO,
   ByVal Size As Long) As Integer
Declare Function lstrcpy Lib "KERNEL" (szDest As Any, szSource As Any)
   As Long
'If using Visual Basic version 1.0, add the following declarations
'Global Const False = 0
'Global Const True = Not False
```

9. Add the following code to the Form_Load event of Form1:

```
Sub Form_Load ()
   Dim r As Integer
   Dim szShareName As String       ' Net DDE share name
   Dim szTargetName As String      ' Net DDE target name
   Dim szTopicName As String       ' Net DDE source topic name
   Dim szItemName As String
   Dim szReadOnlyPassword As String ' Read-only pw Net DDE share
   Dim szFullAccessPassword As String ' Full access password
   Dim ShareInfo As NDDESHAREINFO

   Dim ShareInfoSize As Long
   Dim Result As Integer
   szShareName = "VBDDESource$" + Chr$(0)
   szTargetName = "VBTARGET" + Chr$(0)
   szTopicName = "VBTopic" + Chr$(0)
   szItemName = Chr$(0)                    'All items are allowed
   szReadOnlyPassword = Chr$(0)            'No password
   szFullAccessPassword = Chr$(0)
   'Provide the share, target, topic, and item names along with
   'passwords that identify the network DDE share
   ShareInfo.szShareName = szShareName
   ShareInfo.lpszTargetApp = lstrcpy(ByVal szTargetName,
      ByVal szTargetName)
   ShareInfo.lpszTargetTopic = lstrcpy(ByVal szTopicName,
      ByVal szTopicName)
   ShareInfo.lpszItem = lstrcpy(ByVal szItemName, ByVal szItemName)

   ShareInfo.cbPassword1 = 0
   ShareInfo.lpbPassword1 = lstrcpy(ByVal szReadOnlyPassword,
      ByVal szReadOnlyPassword)
   ShareInfo.dwPermissions1 = NDDEACCESS_REQUEST Or NDDEACCESS_ADVISE Or
      NDDEACCESS_POKE Or NDDEACCESS_EXECUTE Or NDDEACCESS_START_APP
   ShareInfo.cbPassword2 = 0
   ShareInfo.lpbPassword2 = lstrcpy(ByVal szFullAccessPassword,
      ByVal szFullAccessPassword)
   ShareInfo.dwPermissions2 = NDDEACCESS_REQUEST Or NDDEACCESS_ADVISE Or
      NDDEACCESS_POKE Or NDDEACCESS_EXECUTE Or NDDEACCESS_START_APP
   ShareInfo.lpNDdeShareItemInfo = 15
   Result = NDdeShareAdd(ByVal 0&, 2, ShareInfo, Len(ShareInfo))
   ' Start the timer that will continually update the text box and
   ' the DDE link item with random data.
   timer1.Interval = 1000
   timer1.Enabled = True

End Sub
```

10. Add the following code to the Timer1_Timer event procedure:

```
Sub Timer1_Timer ()
   ' Display random value 0 - 99 in the text box (DDE source data).
   Randomize Timer
   VBItem.Text = Format$(Rnd * 100, "0")
End Sub
```

11. From the File menu, choose Make EXE File...

12. Name the file VBTARGET.EXE and choose OK to create the .EXE file.

13. From the File Manager or Program Manager, run VBTARGET.EXE to display a random value in the text box every second.

Step Two -- Create the DDE destination application
---

14. From the DDE destination computer, start Visual Basic or if Visual Basic is already running, from the File menu, choose New Project (ALT, F, N). Form1 is created by default.

15. Add a text box (Text1) to Form1.

16. Add the following code to the Form_Load event of Form1:

```
Sub Form_Load ()
   Dim r As Long
   Dim szComputer As String     ' Network server name.
   Dim szTopic As String
   ' Identify the network server where the DDE source application
   ' is running. The following statement assumes the source computer
   ' name is COMPUTER1. Change it to your source computer name.
   szComputer = "\\COMPUTER1"
   ' Identify the DDE share established by the source application
   szTopic = "VBDDESource$"
   Text1.LinkMode = 0
   ' The link topic identifies the computer name and link topic
   ' as established by the DDE source application
   Text1.LinkTopic = szComputer + "\" + "NDDE$" + "|" + szTopic
   Text1.LinkItem = "VBItem" ' Name of text box in DDE source app

   Text1.LinkMode = 1        ' Automatic link.
End Sub

'For this program to work, set the szComputer variable (above) to the
'computer name that holds the DDE source application. Find the name
'in the Network section of Windows for Workgroups Control Panel.
```

17. From the Run menu, choose Start to run the program.

You should see the same random values generated on the source computer displayed in the text box of the destination computer. If you receive the error message "DDE method invoked with no channel open" on the Text1.LinkMode = 1 statement in Step 16, make sure the szComputer variable is set correctly.

Additional reference words: 1.00 2.00 3.00 NETDDE
KBCategory:
KBSubcategory: APrgNet IAPDDE

Use COMPRESS-r to Avoid Error: Could not execute: SETUP1.EX 2

Article ID: Q93426

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Files used with the Setup Kit must be decompressed or compressed by using COMPRESS -r <filename>. The following error can occur if you use a method other than COMPRESS -r to create a file with an underscore as the last character:

```
Error - Could not execute: SETUP1.EX 2
```

However, VER.DLL must be named VER.DL_ on the setup disk and must not be compressed.

MORE INFORMATION

This information is included with the Help file provided with the Professional Edition of Microsoft Visual Basic version 3.0 for Windows.

The filename listed in the error message above can be different from SETUP1.EX if you customized the Setup Kit.

The following two commands both create a file named SETUP1.EX_, but they are not equivalent:

```
COMPRESS -r SETUP1.EXE             (correct)
COMPRESS SETUP1.EXE SETUP1.EX_     (incorrect)
```

The COMPRESS.EXE option -r compresses a file, replaces the last character of the filename with an underscore (_), and stores the replaced character in the compressed file. When the Setup Kit uses VER.DLL to decompress a file, VER.DLL reads the character from the file and restores the file to its original name.

If you create a file with an underscore as the last character without using COMPRESS -r, VER.DLL renames the file by removing the underscore. For example, SETUP1.EX_ becomes SETUP1.EX.

NOTE: If you create a custom setup, the default SETUP.LST will not include VER.DLL (or more precisely VER.DL_) as a file to copy. Ensure that you do copy the .DLL file. You will want to make sure it is there in all cases.

Additional reference words: 2.00 3.00 errmsg
KBCategory:
KBSubcategory: TlsSetWiz

DDE Conversation Can Cause Error Message: DDE Channel Locked

Article ID: Q95462

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The error message "DDE channel locked" indicates that an attempt is being made to open a DDE conversation between two objects that are already engaged in a conversation. However, usually the error message occurs in a Visual Basic application as a result of a non-Visual Basic DDE Server application failing to post or send a DDE message Visual Basic is expecting.

The best overall solution is to alter the DDE server application so that it correctly sends the appropriate DDE messages.

Both "DDE channel locked" and "Timeout while waiting for DDE response" are errors that can be trapped in Visual Basic, so you can work around the problem by performing the following steps:

1. Turn on error trapping. For example:

   ```
   On Local Error GoTo DDEerrhand:
   ```

2. In your error handling routine, trap error #284 ("DDE channel locked") and set the LinkTimeout property to 1. This triggers the error message "Timeout while waiting for DDE response" much quicker.

3. Also, in your error handling routine trap error #286 ("Timeout while waiting for DDE response"), reset the LinkTimeout value, re-establish the link, and execute a RESUME statement, as in this example:

   ```
   DDEerrhand:
   Select Case Err
      Case 284:
         OldLinkMode = Text1.LinkMode
         OldTimeout = Text1.LinkTimeout
         Text1.LinkTimeout = 1
         Resume
      Case 286:
         Text1.LinkTimeout = OldTimeout
         Text1.LinkMode = 0
         Text1.LinkMode = OldLinkMode
         Resume
    End Select
   ```

MORE INFORMATION

The DDE conversation guidelines set by the Windows Software Development Kit (SDK) require that Visual Basic sometimes wait for an expected DDE message. If that message is never correctly sent or posted to Visual Basic, the following scenario is likely to occur, leading to the error message "DDE channel locked":

1. At some point between when Visual Basic established the conversation and the conversation terminated, the DDE server application fails to post or send a message that Visual Basic is expecting as a normal part of the DDE termination procedure.

2. At this point, Visual Basic is in a PeekMessage loop waiting for a message from the server indicating that the server application has also terminated the DDE conversation. Because Visual Basic is yielding the CPU inside the loop, the Visual Basic code continues to execute and the DDE conversation appears to have terminated normally from the server side.

3. Because Visual Basic is still waiting for the expected DDE message from the server application, the DDE channel is still open. Any attempt to reopen the channel (such as setting the LinkMode property for the control performing the DDE) results in a "DDE channel locked" error.

If no further DDE actions are attempted, you will receive a "Timeout while waiting for DDE response" error message. The timeout will occur after a number of milliseconds equal to the communicating control's LinkTimeout property.

Additional reference words: 1.00 2.00 3.00 errmsg
KBCategory:
KBSubcategory: IAPDDE

How to Use DDE to Display Microsoft Access Data in VB

Article ID: Q96845

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Professional Toolkit for Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

This article demonstrates how to use DDE to obtain and display data from a Microsoft Access database. While Microsoft Access does not support poking data into a database, it does provide several LinkTopics, so you can get information out from a database.

Using supported LinkTopics, you can receive:

- The contents of a Microsoft Access table
- The result of a stored query in the Microsoft Access database
- The result of a SQL expression that you pass to Microsoft Access
- Specifics about a Microsoft Access database

MORE INFORMATION

Below you'll find example code and a detailed list of the LinkTopics and LinkItems supported by Microsoft Access. For the most updated list of LinkTopics and LinkItems supported by Microsoft Access, query on the following words in the Microsoft Knowledge Base:

```
access and DDE and item and topic and server
```

LinkTopics Supported

Here are the LinkTopics supported by Microsoft Access:

```
System               : List of supported LinkTopics.
<Database>           : <Database> is the filename of an existing database.
<TableName>          : <TableName> is a table within the specified database.
<QueryName>          : <QueryName> is a query within the specified database.
SQL <SQL Statement>  : Result of a SQL Query where <SQL Statement> is a
                       valid SQL expression.
```

LinkItems Supported for Each LinkTopic

Here are the LinkItems supported for each LinkTopic and the results they return

System:

```
   SysItems    - List of LinkItems supported by the System LinkTopic.
   Formats     - List of formats Microsoft Access can post to the clipboard.
   Status      - Busy or Ready.
   Topics      - List of all open databases.
   <Macro>     - Name of a macro to be executed.

Database:

   TableList   - List of tables
   QueryList   - List of queries
   MacroList   - List of scripts
   ReportList  - List of reports
   FormList    - List of forms
   ModuleList  - List of modules
   <Macro>     - The name of a macro to be executed.

Table Name, Query Name, and SQL <expression>:

   All         - All the data in the table including the column names.
   Data        - All rows of data without the column names.
   FieldCount  - Count of columns in the table or query results.
   FieldNames  - List of Columns.
   NextRow     - The next row in the table or query. When the conversation
                 begins, NextRow returns the first row. If the current row
                 is the last record, a NextRow request fails.
   PrevRow     - The previous row in the table or query. If PrevRow is the
                 first request over a new channel, the last row of the
                 table or query is returned. If the current row is the
                 first record, a PrevRow request fails.
   FirstRow    - Data in the first row.
   LastRow     - Data in the last row.
   <Macro>     - The name of a macro to be executed.

Although all three LinkTopics (table name, query name, and SQL expression)
return contents from the database and all three support the same LinkItems,
their syntax structures differ slightly. Each LinkTopic must specify the
database the object is in, a semicolon (;), the keyword (TABLE, QUERY, or
SQL), and the name of an existing table, query, or SQL expression. Here are
the syntax structures:

   [db Name];TABLE <Table name>
   [db Name];QUERY <Query name>
   [db Name];SQL <SQL expression>;

Here are examples:

   Text1.LinkTopic = "C:\ACCESS\NWIND.MDB;TABLE Employees"
   Text1.LinkTopic = "C:\ACCESS\NWIND.MDB;QUERY Sales Totals"
   Text1.LinkTopic = "C:\ACCESS\NWIND.MDB;SQL Select * from Employees;"

Note that all SQL statements must end with a semicolon (;).

Step-by-Step Example
--------------------

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N)
   if Visual Basic is already running. Form1 is created by default.
```

2. From the File menu, choose Add File. In the Files box, select the GRID.VBX custom control file. The grid tool appears in the Toolbox.

3. Add two list boxes (List1 and List2) to Form1. The List1 box holds the list of Tables and the List2 box holds the Queries.

4. Add two command buttons (Command1 and Command2) to Form1, placing the Command1 button beneath the List1 box and the Command2 button beneath the List2 box. Change the following properties:

Default	Name	Caption
Command1	GetTableList	Get &Table List
Command2	GetQueryList	Get &Query List

5. Add a grid control (Grid1) to Form1 giving it the following properties:

Default	Name	FixedCols
Grid1	Grid1	0

   The user chooses to display a table or the results of a query in Grid1.

6. Add two text boxes (Text1 and Text2) to Form1. The Text2 box acts as the destination for the data added to List1 and List2, so the user doesn't need to see this text box. But the Text1 box needs to be visible to the user because it acts as the destination for individual rows returned from a query or table.

7. Add two more command buttons to Form1, placing them beneath the Text1 box. Give the two command buttons the following properties:

Default	Name	Caption
Command1	NextRow	&Next Row
Command2	PrevRow	&Previous Row

8. Add the following code to the General Declarations section of Form1:

```
Const None = 0
Const Automatic = 1
Const Manual = 2
Const dbname = "C:\ACCESS\NWIND.MDB"  ' Change Paths as necessary
Const accesspath = "C:\ACCESS\MSACCESS.EXE "
```

9. Add the following three Sub procedures to the General Declarations section of Form1:

```
Sub ClearGrid ()
   ' Select all grid cells.
   Grid1.SelStartCol = 0
   Grid1.SelStartRow = 1
   Grid1.SelEndCol = Grid1.Cols - 1
   Grid1.SelEndRow = Grid1.Rows - 1
   ' Clear the cells.
   Grid1.Clip = ""
```

```
    ' Clean up the grid.
    Grid1.Col = Grid1.FixedCols
    Grid1.Row = Grid1.FixedRows
    Grid1.SelEndCol = Grid1.SelStartCol
    Grid1.SelEndRow = Grid1.SelStartRow
End Sub

Sub PopulateGrid (IsTable%, QueryOrTable$)
    If IsTable% Then
        Text1.LinkTopic = "MSACCESS|" + dbname + ";TABLE " + QueryOrTable$
    Else
        Text1.LinkTopic = "MSACCESS|" + dbname + ";QUERY " + QueryOrTable$
    End If
    Text1.LinkItem = "FieldCount"
    Text1.LinkMode = Automatic
    Grid1.Cols = Val(Text1.Text)

    Text1.LinkItem = "FieldNames"
    Grid1.FixedRows = 0      ' Cannot additem to a fixed row
    Grid1.AddItem Text1.Text, 0
    Grid1.FixedRows = 1

    On Error GoTo LastRowErr
    Text1.LinkItem = "LastRow"
    Grid1.AddItem Text1.Text, 1
    Text1.LinkItem = "PrevRow"
    Do
        Grid1.AddItem Text1.Text, 1
        Text1.LinkRequest
    Loop
    Exit Sub
LastRowErr:
    Exit Sub  ' Error occurs when last row is reached
End Sub

Sub GetList (L As ListBox, ListType$)
    text2.LinkMode = None
    text2.LinkTopic = "MSAccess|" + dbname
    text2.LinkItem = ListType$
    text2.LinkMode = Automatic
    StartPos% = 1
    Do
        Pos% = InStr(StartPos%, text2.Text, Chr$(9))
        If Pos% = 0 Then Exit Do
        L.AddItem Mid$(text2.Text, StartPos%, Pos% - StartPos%)
        StartPos% = Pos% + 1
    Loop
End Sub
```

10. Add the following code to the Form_Load event of Form1:

```
Sub Form_Load ()
   result% = Shell(accesspath + dbname, 1)
End Sub
```

11. Add the following code to the GetQueryList_Click event procedure:

```
Sub GetQueryList_Click ()
   GetList List2, "QueryList"
End Sub
```

12. Add the following code to the GetQueryList_Click event procedure:

```
Sub GetTableList_click ()
   GetList List1, "TableList"
End Sub
```

13. Add the following code to the List1_Click event procedure:

```
Sub List1_Click ()
   Table$ = List1.Text
   ClearGrid
   PopulateGrid True, Table$
End Sub
```

14. Add the following code to the List2_Click event procedure:

```
Sub List2_Click ()
   Query$ = List1.Text
   ClearGrid
   PopulateGrid False, Query$
End Sub
```

15. Add the following code to the NextRow_Click event procedure:

```
Sub NextRow_click ()
   On Error GoTo NextRowErrHand:
   Text1.LinkItem = "NextRow"  ' Get the next row of results
   Exit Sub
NextRowErrHand:
   MsgBox "Last row reached"
   Exit Sub
End Sub
```

16. Add the following code to the PrevRow_Click event procedure:

```
Sub PrevRow_Click ()
   On Error GoTo PrevRowErrHand
   Text1.LinkItem = "PrevRow"
   Exit Sub
PrevRowErrHand:
   MsgBox "First Row Reached"
   Exit Sub
End Sub
```

17. From the Run menu, choose Start (ALT, R, S) to run the program. Microsoft Access is shelled with the NWIND.MDB sample database open and Form1 showing on the screen.

18. Choose the Get Table List button to see a list of all the tables in the NWIND database displayed in the List1 box.

19. Choose the Get Query List button to see a list of the previously defined queries that exist in the NWIND database displayed in the List2

box.

20. Select one of the items in either the List1 or List2 box to see the results displayed in Grid1.

21. Choose the Next Row button to see the second row displayed in the Text1 box. Continue to choose the Next Row button to display successive rows until you get to the last row. When you get to the last row, a message box appears to tell you that you reached the last row.

22. Choose the Prev Row button. The row previous to the one displayed in the Text1 box is displayed.

Additional reference words: 1.00 2.00
KBCategory:
KBSubcategory: IAPDDE

OLE Embedding & Linking Word for Windows Objects into VB Apps

Article ID: Q97618

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 2.0
- Microsoft Word for Windows, version 2.0

SUMMARY

This article shows by example how to use the object linking and embedding (OLE) client custom control (OLECLIEN.VBX) with Microsoft Word for Windows. The example demonstrates both how to embed and how to link a Word for Windows document into a Visual Basic application.

NOTE: In Word for Windows, version 6.0 or 6.0a, the Bookmark menu item moved from the Insert menu to the View menu.

MORE INFORMATION

Embedding an object encapsulates the data displayed in the Visual Basic OLE client control and makes the data inaccessible to other applications, unlike the data in an linked object. In addition, embedding an object does not require that a file already exist for the object to be usable.

Linking an object, on the other hand, does require that a file already exist, and it requires a LinkItem setting. For a Word for Windows document, the LinkItem can be any bookmark within the document.

The example shown below demonstrates how to use:

- Embedded Word for Windows objects
- Linked Word for Windows objects

The following OLE client control property settings are required to create a Word for Windows OLE object:

Property	Value
Class	"WordDocument"
Protocol	"StdFileEditing"

In addition, linked objects require the following OLE client control property settings:

Property	Value
SourceDoc	The full path of the document to use (such as C:\OLETEST.DOC)
SourceItem	A bookmark (OLE_Link is used in this example)

Here are the steps you need to follow to create the example:

Step One: Create the Word for Windows Document You Want to Link Or Embed

1. Start Word for Windows. Document1 is created by default.

2. Press CTRL+SHIFT+END to select to the end of the document.

3. From the Insert menu, choose Bookmark. Under Bookmark Name, type:

   OLE_Link

   and press ENTER to set a bookmark for the entire document. This bookmark functions as the LinkItem.

4. From the File menu, choose Save As, and save the document with the name C:\OLETEST.DOC. (If the path is different, change the ServerDoc property on OleClient1 to reflect the correct path.)

Step Two: Create the Visual Basic Application That Will Hold the Document

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. From the File menu, choose Add File and add OLECLIEN.VBX to the project.

3. Add the following controls to Form1, and give them the properties shown:

Default Name	Caption	Name
OleClient1	N/A	OleClient1
Option1	&Embed Object	OptionEmbed
Option2	&Link Object	OptionLink
Command1	Embed WinWord Object	Command1

4. Change the Value property on OptionEmbed to True.

5. Add the following code to the general declarations section of Form1:

```
Dim fshowing As Integer

Const OLE_LINKED = 0
Const OLE_EMBEDDED = 1
Const OLE_STATIC = 2

Const OLE_CREATE = 0
Const OLE_CREATE_FROM_FILE = 1
Const OLE_UPDATE = 6
Const OLE_ACTIVATE = 7
Const OLE_DELETE = 10
```

6. Add the following code to the click event of Command1:

```
Sub Command1_Click ()

   ' Unload the current object so a new object can be loaded
   If fshowing Then
      OleClient1.Action = OLE_DELETE
```

```
      End If

      OleClient1.Class = "WordDocument"
      OleClient1.Protocol = "StdFileEditing"
      If OptionEmbed Then
         ' Data is managed by Visual Basic
         OleClient1.ServerType = OLE_EMBEDDED
         OleClient1.Action = OLE_CREATE
      Else
         OleClient1.SourceDoc = "C:\OLETEST.DOC"
         OleClient1.SourceItem = "OLE_Link"
         OleClient1.ServerType = OLE_LINKED
         OleClient1.Action = OLE_CREATE_FROM_FILE
      End If
      OleClient1.Action = OLE_UPDATE
      fshowing = True

   End Sub
```

7. Add the following code to the DblClick event of OleClient1:

```
   Sub OleClient1_DblClick ()
      OleClient1.Action = OLE_ACTIVATE
   End Sub
```

8. Add the following code to the Click event of OptionEmbed:

```
   Sub OptionEmbed_Click ()
      Command1.Caption = "Embed WinWord Object"
   End Sub
```

9. Add the following code to the Click event of OptionLink:

```
   Sub OptionLink_Click ()
      Command1.Caption = "Link WinWord Object"
   End Sub
```

8. From the Run menu, choose Start (ALT+R, S) to run the program.

9. Click the Embed WinWord Object button to activate Word for Windows.

10. Type some text into the active Word document.

11. Close Word and click the Yes button when asked if you want to update the Object in OleClient1. The Word for Windows icon is painted in the OleClient1 control.

12. Double-click the OLE client control to reactivate Word and redisplay the text you entered.

13. Click OptionLink. The caption of button changes to Link WinWord Object.

14. Click the Link WinWord Object button. The Word icon remains in the OLE client control, however it is now linked to the document created in the first part of this example, not the embedded object.

15. Double-click the OLE client control to activate Word for Windows and

redisplay the text you entered in the first document.

Additional reference words: 2.00
KBCategory:
KBSubcategory: IAPOLE

PRB: Error: Setup could not be completed due to system errors

Article ID: Q98554

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SYMPTOMS

During Visual Basic setup, you may encounter this error:

   Setup could not be completed due to system errors

Then setup terminates. This error usually occurs while setup is calculating the amount of free disk space.

CAUSE

This error occurs because Visual Basic setup is attempting to use an older version of LZEXPAND.DLL that it found on your computer. In all reported cases of this problem, the LZEXPAND.DLL file is dated 7-Aug-91 and is usually located in the WINDOWS directory.

One product that may install a copy of LZEXPAND.DLL dated 7-Aug-91 is MicroHelp Muscle version 1.0. However, it is possible that other products not sold by Microsoft may also install this file.

RESOLUTION

To overcome this problem, perform the following steps:

1. Exit from Windows to MS-DOS.

2. Find the LZEXPAND.DLL file that's dated 7-Aug-91. It may be located in the WINDOWS or WINDOWS\SYSTEM directory, but it is usually in the WINDOWS directory.

3. Delete or rename the LZEXPAND.DLL dated 7-Aug-91 to a new name.

4. Locate a copy of LZEXPAND.DLL on your computer that has a date later than 7-Aug-91, and put it in the WINDOWS\SYSTEM directory.

5. If you don't find a later version of LZEXPAND.DLL, run the Windows Setup program from the Windows distribution disks. This will install a later version of LZEXPAND.DLL in the WINDOWS\SYSTEM directory.

6. Start Windows.

7. Run the Visual Basic setup program again.

The error should no longer occur.

Additional reference words: 2.00 3.00
KBCategory:
KBSubcategory: Setins

PRB: GP Fault with Visual Basic DDE Sample & Word for Windows

Article ID: Q99812

The information in this article applies to:

- Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Word for Windows, versions 2.0a, 2.0b, and 2.0c

SYMPTOMS

Running the Visual Basic DDE sample with Microsoft Word for Windows may cause a general protection (GP) fault.

STATUS

Microsoft has confirmed this to be a problem with Microsoft Word for Windows versions 2.0a, 2.0b, and 2.0c. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

Steps to Reproduce Problem

1. Start Word for Windows (WINWORD.EXE).

2. Start Visual Basic version 3.0 for Windows.

3. From the File menu, choose Open Project (ALT, F, O). Then open the DDE.MAK project from the \VB\SAMPLES\DDE directory.

4. From the Run menu, choose start (ALT, R, S), or press F5. The main form of DDE.MAK is titled DDE Experimenter.

5. From the DDE Experimenter form, select WinWord as the Application and Document1 as the Topic. The Item automatically becomes \Doc.

6. Select the Manual option.

7. Click the Connect button. The caption for the command button will change to Disconnect.

7. Type text into the text box in the Destination Data section of the DDE Experimenter form.

8. Click the Poke button.

9. Select the Automatic option.

At this point, a GP fault occurs in USER.EXE. The address of the GP fault

varies depending on the version of Word for Windows. Although the message indicates that Visual Basic caused the GP fault, the problem is actually caused by Word for Windows, not Visual Basic.

Additional reference words: 3.00 WinWord 2.00
KBCategory:
KBSubcategory: IAPDDE

How to Change the Setup Application Name in SETUP1.EXE

Article ID: Q101743

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, version 3.00.

SUMMARY

The Setup Wizard in Visual Basic version 3.0 creates SETUP1.EXE that when executed displays a blue background with white letters that say: "<EXE NAME> Setup," where <EXE NAME> is the name of your application. This article explains how to change that message to something other than the default.

MORE INFORMATION

To change the display of <EXE NAME>, follow these steps:

1. Run the Setup Wizard as you normally would to create installation disks.

2. Start Visual Basic and load the project SETUP1A.MAK (The Setup Wizard created this project in the C:\VB\SETUPKIT\SETUP1 directory).

3. In the General Declarations section of SETUP1A.FRM, change the value of constant APPNAME:

   ```
   Const APPNAME = "<Whatever you want to put here>"
   ```

4. From the File menu, choose Make EXE to create the file SETUP1.EXE.

5. Exit to MS-DOS.

6. Copy and compress the file SETUP1.EXE to your distribution disk.

   ```
   C:\VB\SETUPKIT\KITFILES\COMPRESS -r SETUP1.EXE A:\
   ```

   This will copy over the old SETUP1.EX_ that was created on the distribution disk by the Setup Wizard.

Additional reference words: 3.00
KBCategory:
KBSubcategory: TlsSetWiz

Additions to 'Determining the Files You Need to Distribute'

Article ID: Q103439

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY

After producing an executable program (.EXE file) in Microsoft Visual Basic version 3.0 for Windows, if you want to distribute, sell or test that .EXE file on another computer that does not have Visual Basic version 3.0 for Windows installed, you need to know which files to distribute with your .EXE file. These files are listed on pages 579-582 in the Visual Basic version 3.0 for Windows "Programmer's Guide." This article gives a list of files to be appended to that list.

MORE INFORMATION

Note that VBRUN300.DLL must always be distributed with your executable program.

Below is a list of files that need to be appended to the list provided on pages 579-582 in the "Programmer's Guide."

File Names to Distribute:	Required if your program....
PDIRJET.DLL	Uses Crystal Reports for Visual Basic.
PDBJET.DLL	
MSAJT110.DLL	
MSAES110.DLL	
PDSODBC.DLL	Uses ODBC and Crystal Reports for Visual Basic.
MSAFINX.DLL	Uses the IIF or any of the financial functions.

Additional reference words: 3.00
KBCategory:
KBSubcategory: TlsSetWiz

How to Run a WinHelp Macro from a Help File

Article ID: Q104165

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

This article shows by example how to invoke WinHelp macros in a Help file upon first entering a topic or upon clicking a hot spot. The examples show you how to modify the example help topic source file VB\HC\ICONWRKS.RTF using Microsoft Word version 2.0 for Windows. After making the changes and compiling the .RTF file into a Help file (.HLP file), you will be able to execute a macro upon first entering the topic or upon clicking a hot spot.

MORE INFORMATION

To run a macro when a topic is first entered, enter the macro call into a custom footnote with an exclamation (!) as the footnote mark. To do this in Microsoft Word version 2.0 for Windows, follow these steps:

1. Open the file VB\HC\ICONWRKS.RTF.

2. Locate the beginning of this topic heading:

   ```
   Editor: Commands and Tools
   ```

   Place the text caret at the beginning of the line, in front of the pound (#) character.

3. From the Insert menu, choose Footnote and select Custom Footnote Mark. Enter the exclamation mark (!) and choose the OK button. The Footnotes area appears at the bottom of the window with the caret on a line that begins with the exclamation mark (!).

4. Enter this text:

   ```
   About()
   ```

5. To save this change, from the File menu, choose Close. In each of the subsequent three dialogs, press the Enter key to select the default button.

6. At the command prompt, set the current directory to VB\HC. Then enter the following command to compile the help file:

   ```
   HC31 ICONWRKS.HPJ
   ```

7. Load the resulting ICONWRKS.HLP file into WINHELP.EXE. To do this in the Program Manger, from the File menu, choose Run. Then enter the full path

of VB\HC\ICONWRKS.HLP. The IconWorks help file appears.

8. Click the hot spot Commands and Tools to jump to the topic that contains the macro call. When the topic appears, the About() macro displays a dialog box titled About Help.

To run a macro immediately when a hot spot is clicked, format the hot spot text as double-underlined text followed by an exclamation mark (!) and the macro call -- both formatted as hidden text. To do this in Microsoft Word version 2.0 for Windows, follow these steps:

1. Open the file VB\HC\ICONWRKS.RTF.

2. From the Tools menu, choose Options. Select the View Category. In the section labeled Nonprinting Characters, check Paragraph Marks and Hidden Text.

3. Place the text caret at the beginning of the second line of the file (the line following the heading). Enter the text "Call Macro!About()" without the quotation marks, and press the Enter key.

4. Select the text "Call Macro" up to but not including the exclamation mark. From the Format menu, choose Character. Change the setting in the combo box labeled Underline from None to Double. Choose the OK button.

5. Select the text "!About()" up to but not including the paragraph character at the end of the line. From the Format menu, choose Character. In the Style section, check Hidden.

6. Close the file. Compile it using the Help compiler (HC31.EXE). Then view the compiled .HLP file using WINHELP.EXE. See the previous example for an explanation of how to do this.

7. Click the Call Macro hot spot. The About() macro displays a dialog box titled About Help.

Additional reference words: 3.00
KBCategory: Tls
KBSubCategory: TlsHC

How to Manipulate Groups & Items in Program Manager Using DDE

Article ID: Q104943

The information in this article applies to:

- Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

Program Manager has a DDE command-string interface that allows other applications to create, display, delete, and reload groups; add items to groups; replace items in groups; delete items from groups; and close Program Manager. The following commands perform these actions:

- CreateGroup
- Reload (Windows 3.1 only)
- DeleteGroup
- ShowGroup
- ReplaceItem (Windows 3.1 only)
- DeleteItem (Windows version 3.1 only)
- AddItem

MORE INFORMATION

Perform the following steps to produce an application that manipulates Program Manager using DDE:

1. Start Visual Basic or if Visual Basic is already running, choose New Project from the File menu (ALT, F, N). Form1 is created by default.

2. Add a Textbox control (Text1) to Form1

3. Add a Label control (Label1) to Form1 and change the caption to Group.

4 Add a Textbox control (Text2) to Form1 and change the caption to GGroup.

5. Add a Label control (Label2) to Form1 and change the caption to Item.

6. Add a Textbox control (Text3) to Form1 and change the caption to GItem.

7. Add a Label control (Label3) to Form1 and change the caption to Command Line.

8. Add a Textbox control (Text4) to Form1 and change the caption to ItemExe.

9. Add a Command Button control (Command1) to Form1 and name it CGroup for create group.

10. Add the following code to the CGroup_Click event of Form1:

```
Sub CGroup_Click ()
Dim cmd As String
   On Error GoTo CGError
   text1.LinkMode = 0
   text1.LinkTopic = "Progman|Progman"
   text1.LinkMode = 2
   cmd = "[CreateGroup(" + GGroup.Text + ")]"
   text1.LinkExecute cmd
CGDone:   text1.LinkMode = 0
   Exit Sub
CGError:
   MsgBox "Error Adding Group"
   Resume CGDone
End Sub
```

11. Add a Command Button control (Command2) to Form1 and name it DGroup for Delete Group.

12. Add the following code to the DGroup_Click event of Form1:

```
Sub DGroup_Click ()
Dim cmd As String
   On Error GoTo DGError
   text1.LinkMode = 0
   text1.LinkTopic = "Progman|Progman"
   text1.LinkMode = 2
   cmd = "[DeleteGroup(" + GGroup.Text + ")]"
   text1.LinkExecute cmd
DGDone:   text1.LinkMode = 0
   Exit Sub
DGError:
   MsgBox "Error Deleting Group"
   Resume DGDone
End Sub
```

13. Add a Command Button control (Command3) to Form1 and name it SGroup for ShowGroup.

14. Add the following code to the SGroup_Click event of Form1:

```
Sub SGroup_Click ()
Dim cmd As String
   On Error GoTo SGError
   text1.LinkMode = 0
   text1.LinkTopic = "Progman|Progman"
   text1.LinkMode = 2
   cmd = "[ShowGroup(" + GGroup.Text + ", 1" + ")]"
   text1.LinkExecute cmd
SGDone:   text1.LinkMode = 0
   Exit Sub
SGError:
   MsgBox "Error Showing Group"
   Resume SGDone
End Sub
```

15. Add a Command Button control (Command4) to Form1 and name it

Reload.

16. Add the following code to the Reload_Click event of Form1:

```
Sub Reload_Click ()
Dim cmd As String
   On Error GoTo RLError
   text1.LinkMode = 0
   text1.LinkTopic = "Progman|Progman"
   text1.LinkMode = 2
   cmd = "[Reload(" + GGroup.Text + ")]"
   text1.LinkExecute cmd
RLDone:   text1.LinkMode = 0
   Exit Sub
RLError:
   MsgBox "Error Reloading Group"
   Resume RLDone
End Sub
```

17. Add a Command Button control (Command5) to Form1 and name it AItem for add item.

18. Add the following code to the AItem_Click event of Form1:

```
Sub AItem_Click ()
Dim cmd As String
   On Error GoTo AIError
   text1.LinkMode = 0
   text1.LinkTopic = "Progman|Progman"
   text1.LinkMode = 2
   '*** The ShowGroup is necessary because AddItem changes the group
   '*** with the focus. ShowGroup forces the group you want the
   '*** action taken to get the focus.
   If (Len(GGroup.Text) > 0) Then
      cmd = "[ShowGroup(" + GGroup.Text + ", 1" + ")]"
      text1.LinkExecute cmd
   End If
   cmd = "[Additem(" + ItemExe.Text + "," + GItem.Text + ")]"
   text1.LinkExecute cmd
AIDone:
   text1.LinkMode = 0
   Exit Sub
AIError:
   MsgBox "Error adding Item"
   Resume AIDone
End Sub
```

19. Add a Command Button control (Command6) to Form1 and name it DItem for delete item.

20. Add the following code to the DItem_Click event of Form1:

```
Sub DItem_Click ()
   Dim cmd As String
   On Error GoTo DIError
   text1.LinkMode = 0
   text1.LinkTopic = "Progman|Progman"
```

```
      text1.LinkMode = 2
      '*** ShowGroup is necessary because DeleteItem changes the group
      '*** with the focus. ShowGroup forces the group you want the action
      '*** taken to get the focus.
      If (Len(GGroup.Text) > 0) Then
         cmd = "[ShowGroup(" + GGroup.Text + ", 1" + ")]"
         text1.LinkExecute cmd
      End If
      cmd = "[DeleteItem(" + GItem.Text + ")]"
      text1.LinkExecute cmd
   DIDone:   text1.LinkMode = 0
      Exit Sub
   DIError:
      MsgBox "Error Deleting Item"
      Resume DIDone
   End Sub
```

21. Add a Command Button control (Command7) to Form1 and name it RItem for replace item.

22. Add the following code to the RItem_Click event of Form1:

```
   Sub RItem_Click ()
   Dim cmd As String
      On Error GoTo RIError
      text1.LinkMode = 0
      text1.LinkTopic = "Progman|Progman"
      text1.LinkMode = 2
      '*** ShowGroup forces the group you want the action taken on
      '*** to get the focus.
      If (Len(GGroup.Text) > 0) Then
         cmd = "[ShowGroup(" + GGroup.Text + ", 1" + ")]"
         text1.LinkExecute cmd
      End If
      cmd = "[ReplaceItem(" + GItem.Text + ")]"
      text1.LinkExecute cmd
      cmd = "[Additem(" + ItemExe.Text + "," + GItem.Text + ")]"
      text1.LinkExecute cmd
   RIDone:   text1.LinkMode = 0
      Exit Sub
   RIError:
      MsgBox "Error Replacing Item"
      Resume RIDone
   End Sub
```

23. From the Run menu, choose Start (ALT, R, S) or press the F5 key to run the program. Enter the group you want created in the GGroup textbox and click the Create Group button. You will now see the group you created in Program Manager. To add an item to a group, enter the group in the GGroup textbox. Enter the item you want added in the GItem textbox and enter the command line in the ItemExe textbox. The item will now be in the group you specified.

For more information, refer to the "Programmers Reference, Volume 1: Overview Microsoft Windows SDK," chapter 17, "Shell Dynamic DataExchange Interface." Also, look in the Windows SDK Help file in the Progman topic.

Additional reference words: 1.00 2.00 3.00
KBCategory:
KBSubcategory: IAPDDE

How to Use DDE Between Excel and Visual Basic

Article ID: Q105447

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article outlines the steps necessary to initiate dynamic data exchange (DDE) between an Excel Macro and Visual Basic application at run time.

This article demonstrates how to:

- Prepare an Excel Macro that will perform DDE.
- Initiate a manual DDE link.
- Use Poke to send information from the Excel Worksheet to the Visual Basic application.

MORE INFORMATION

A destination application sends commands through DDE to the source application to establish a link. Through DDE, the source provides data to the destination at the request of the destination or accepts information at the request of the destination.

Example Showing How to Establish a DDE Conversation

The steps below give an example of how to establish a DDE conversation between an Excel Worksheet and a Visual Basic application.

Preparing Visual Basic for DDE

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a text box to form1.

3. Change the following properties :

Control	Name	Caption	Properties
Form1	Form1	Form1	LinkMode : 1-Source
			LinkTopic : Form1
TextBox	Text1	Text1	LinkItem : Text1

4. Save the project as DDE.MAK, and save the form as DDE.FRM.

5. From the File menu, choose Make Exe File, and name the .EXE file
   DDE.EXE to enable the link.

6. Close Visual Basic.

Preparing Excel for DDE
---

1. When you start Excel, Sheet1.XLS is created for you by default.
   In Cell C2, type the text you want to send to the Visual Basic
   application. Save the Sheet1 spreadsheet.

2. Create a new Macro Sheet in Excel by choosing New from the File Menu
   and then selecting Macro Sheet.

3. Type the following macro :

```
Record1 (a)
chan=INITIATE("dde","Form1")
=POKE(chan,"text1",'A:\SHEET1.XLS'!C2)
=TERMINATE(chan)
=RETURN(chan)
```

   The Initiate function starts the Visual Basic application if it is
   not already running and establishes the link with Form1. The Poke
   function puts text from cell C2 of the Sheet1 worksheet into the
   Text1 box on form1. The Terminate function terminates the link, and
   the Return function ends the macro.

   Important Notes

   The following are requirements. You must:

   - Save the Worksheet before attempting the Link.

   - Store the application in the path by making sure the application
     directory is in the path.

   - Not specify the path to the application in the Initiate function.

4. From the Macro menu, choose Run.

5. You will see a message box saying Run Macro. In the reference section,
   type 'Macro1.xlm!Record1' and click the OK button.

6. If DDE.EXE is not running, you will a message box saying "Remote Data
not
   accessible. Start application 'DDE.EXE'." Click Yes.

7. If DDE.EXE was in the path, it will be run minimized. When you maximize
   it, you will see the data from Sheet1 in the Text1 box. If you don't see
   the data, save Sheet1 and try again.

Additional reference words: 3.00
KBCategory:

KBSubCategory: IAPDDE

How to Copy and Paste DDE Links Using CF_LINK in Visual Basic

Article ID: Q106238

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Visual Basic programs can provide an Edit Copy command that puts text onto the Clipboard along with the information needed to make a DDE link to that text. Likewise, you can provide an Edit Paste command that extracts the text and an Edit Paste Link command that extracts the DDE link information, and then initiates a DDE conversation to the source text. You can implement these clipboard operations using the GetText and SetText format CF_LINK (&HBF00) with a string of the format:

   Application|Topic!Item

The Visual Basic manuals and Help menu do not describe how to use the Clipboard format CF_LINK.

NOTE: Visual Basic only supports Clipboard DDE operations for text.

MORE INFORMATION

When a Visual Basic DDE source program uses SetText with format CF_LINK, the format of the string is exe|topic!item.

 - exe is the value of App.EXEName.
 - topic is a form's LinkTopic value.
 - item is the Name of a control.

For example:

   Project1|Form1!Text1

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose Open Project (ALT, F, O) if Visual Basic is already running. Form1 is created.

2. Change the form property LinkMode to 1 - Source.

3. Place a text box named Text1 on the form.

4. From the Window menu, choose Menu Design and create the following menu structure.

   Caption   Name   Indent Level

```
-------------------------------------
Edit        mEdit         0
Copy        mCopy         1
Paste       mPaste        1
Paste Link  mPasteLink    1
```

5. Enter the following code in the general declarations section:

```
Const CF_TEXT = 1
Const CF_LINK = &HBF00
```

6. Enter the following code into the form:

```
Sub mEdit_Click ()
   mCopy.Enabled = Text1.SelLength > 0
   mPaste.Enabled = Clipboard.GetFormat(CF_TEXT)
   mPasteLink.Enabled = Clipboard.GetFormat(CF_LINK)
End Sub

Sub mCopy_Click ()
   Clipboard.Clear
   Clipboard.SetText Text1.SelText, CF_TEXT
   Clipboard.SetText "Project1|Form1!Text1", CF_LINK
End Sub

Sub mPaste_Click ()
   Text1.LinkMode = 0  ' discontinue previous link
   Text1.SelText = Clipboard.GetText(CF_TEXT)
End Sub

Sub mPasteLink_Click ()
   Dim topic As String  ' app|topic!item
   Dim bang As Integer  ' index of ! within topic

   topic = Clipboard.GetText(CF_LINK)
   bang = InStr(topic, "!")
   If bang <> 0 Then
      Text1.LinkMode = 0
      Text1.LinkTopic = Mid$(topic, 1, bang - 1)
      Text1.LinkItem = Mid$(topic, bang + 1)
      On Error Resume Next
      Text1.LinkMode = 1  ' automatic
      If Err <> 0 Then
         MsgBox "Cannot paste link"
      End If
   End If
End Sub
```

7. From the File menu, choose Make EXE and click OK. This creates PROJECT1.EXE.

8. Launch PROJECT1.EXE twice so that two instances are running. Position them so that you can see them both at the same time. In the first instance, select the text in the text box, then from the Edit menu choose copy. Switch to the second instance and from the Edit menu choose Paste Link. Switch back to the first instance and change the contents of the text box. These changes appear immediately in the

second instance.

Additional reference words: 3.00
KBCategory:
KBSubCategory: IAPDDE

Sample .MAK for Compiling VB Custom Control in Borland C++ 3.1

Article ID: Q107776

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

This article gives a sample .MAK file for compiling a Microsoft Visual Basic custom control using Borland C++ version 3.1. Sample C code is not provided with this example.

Borland C++ is manufactured by Borland International, Inc., a vendor independent of Microsoft. We make no warranty, implied or otherwise, regarding this product's performance or reliability.

MORE INFORMATION

Run the MAKE utility without the -N compatibility option when using the sample .MAK file below.

SAMPLE.MAK

```
.nosilent
OBJPATH = ..\obj\               # comment to prevent trailing \
MAINTPATH = ..\maint\           # comment to prevent trailing \
BMPFILES = winqcd.bmp winqcu.bmp winqeu.bmp winqmu.bmp
DLGFILES =
HDRFILES = $(KMSTOOLS)\kms-win.h winq.h icdefs.h vbdefs.h
OBJFILES = $(OBJPATH)winqmain.obj $(OBJPATH)winqsubs.obj
           $(OBJPATH)winqfile.obj
# $(OBJPATH)splay.obj
RESFILES = $(KMSTOOLS)\kms-res.h winq-res.h

.path.obj = $(OBJPATH)
.path.res = $(OBJPATH)
.path.rsp = $(MAINTPATH)
.path.sym = $(OBJPATH)

LIBPATH = $(BORPATH)\LIB;c:\vb3.0\cdk;c:\idk\lib\win
CDEFS = -DSTRICT
MEM = s
DLL = !
MODEL = $(MEM)$(DLL)
CFLAGS = -n$(OBJPATH) -m$(MODEL) -WD /I$(KMSTOOLS) /I$(BORPATH)/include
/Ic:\vb3.0\cdk /Ic:\idk\include

#       Implicit rules
```

```
!if $d(OBJPATH)
.rc.res:
      $(RC) $(RFLAGS) -r $&.rc
      copy  $&.res $(OBJPATH)$&.res
      del   $&.res
!else
.rc.res:
      $(RC) $(RFLAGS) -r $&.rc
!endif

#       Links

winq100.vbx: $(MAINTPATH)bcc.rsp $(OBJFILES) winq.res winq.def
    $(LNK) /Twd/v/x/P-/L$(LIBPATH) @&&|
c0d$(MEM) $(OBJFILES)
$<
$(OBJPATH)$&.map
vbapi.lib cwc.lib icwin.lib import
winq.def
|
    rc $(OBJPATH)winq.res $<

winq.res: $(BMPFILES) $(DLGFILES) $(RESFILES) winq.rc

$(MAINTPATH)bcc.rsp: $(MAINTPATH)makefile.mak
    copy &&|
$(CDEFS) $(CFLAGS)
| $(MAINTPATH)bcc.rsp

#       Compiles

winqinc.sym: $(MAINTPATH)bcc.rsp winqinc.cpp $(RESFILES) $(HDRFILES)
     del $(OBJPATH)*.sym
     $(CC) -H=$(OBJPATH)winqinc.sym @$(MAINTPATH)bcc.rsp {$&.cpp }

$(OBJPATH)winqmain.obj:
     $(MAINTPATH)bcc.rsp winqinc.sym $(RESFILES) $(HDRFILES)
winqmain.cpp
     $(CC) -H=$(OBJPATH)winqmain.sym @$(MAINTPATH)bcc.rsp {$&.cpp }

$(OBJPATH)winqsubs.obj: $(MAINTPATH)bcc.rsp winqinc.sym $(RESFILES)
     $(HDRFILES)
winqsubs.cpp
     $(CC) -H=$(OBJPATH)winqinc.sym @$(MAINTPATH)bcc.rsp {$&.cpp }

$(OBJPATH)winqfile.obj: $(MAINTPATH)bcc.rsp winqinc.sym $(RESFILES)
     $(HDRFILES)
winqfile.cpp
     $(CC) -H=$(OBJPATH)winqinc.sym @$(MAINTPATH)bcc.rsp {$&.cpp }

# $(OBJPATH)splay.obj: $(MAINTPATH)bcc.rsp $(RESFILES) $(HDRFILES)
splay.cpp
#      $(CC) @$(MAINTPATH)bcc.rsp {$&.cpp }

Additional reference words: 3.00
KBCategory: Tls
KBSubcategory: TlsCDK
```

VB Ver 3.0 CDK TN001.TXT: Support for DT_OBJECT Properties

Article ID: Q107872

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

The following article contains the complete contents of the TN001.TXT file installed in the CDK directory of the Professional Edition of Visual Basic version 3.0 for Windows.

MORE INFORMATION

TN001.TXT

```
Microsoft Visual Basic 3.00
Microsoft Corporation Technical Notes

TN001.TXT:  Support for DT_OBJECT Properties

This note describes how to use OLE Automation by creating a custom control
property whose data type is DT_OBJECT.

================================================================================

------------
Introduction
------------

The Visual Basic version 3.0 Control Development Kit allows you to create
custom controls that support OLE Automation. A custom control can define
a DT_OBJECT type property, whose value is a 4-byte pointer to an IDispatch
interface. If your control can contain or refer to an OLE object, you
may want to expose this ability via a property of type DT_OBJECT.

This allows you to use Visual Basic statements such as:

   Dim MyObject As OBJECT
   Set MyObject = Control.Object
   MyObject.Method...

or, more directly:

   Control.Object.Method

----------------
Reference Counts
----------------
```

OLE has strict guidelines for maintaining reference counts on interface pointers. Functions that return interface pointers increment the reference count on the pointer on behalf of the caller. For example, calling VBGetControlProperty for a DT_OBJECT property causes the returned interface pointer to be incremented. However, the caller is responsible for eventually releasing the reference to the interface pointer by using IUnknown::Release().

If the property is PF_fGetMsg, you are responsible for incrementing the reference count on the interface pointer you return in your VBM_GETPROPERTY message code.

Example

You may want to create a control that is an OLE container, meaning that you expose a pointer to an IDispatch interface via a DT_OBJECT property. When the OLE object is initially created within the control, the control could establish a connection to the OLE object by setting a pointer to an IDispatch interface via IUnknown::QueryInterface(). The reference count for the interface pointer would then be incremented from zero to one.

When the control is destroyed, or the control causes the OLE object it contains to be released, the control would also need to release the interface pointer using IUnknown::Release(). If the DT_OBJECT property is PF_fGetData, then all reference count maintenance associated with fetching the property is automatically handled by Visual Basic. If, on the other hand, the DT_OBJECT property is PF_fGetMsg, you would need to call IUnknown::AddRef() on the interface pointer in your VBM_GETPROPERTY message code.

If there is no valid IDispatch interface, VBGetControlProperty returns NULL (DWORD 0).

Property Flags

Because the IDispatch interface pointer can be used only at run time, you cannot save or load an interface pointer while saving or loading a form. For this reason, you cannot use the PF_fSaveData flag on your DT_OBJECT property.

You might want use to PF_fSaveMsg to enable you to do more sophisticated processing at save or load time. For example, you could handle VBM_SAVEPROPERTY by serializing the OLE object corresponding to the interface pointer into the form file, and handle VBM_LOADPROPERTY by deserializing the object and getting a pointer to an IDispatch interface to it to set your property.

A property defined as DT_OBJECT cannot be set. This means that you cannot use the property flags PF_fSetData or PF_fSetMsg. In addition, you should specify the PF_fNoRuntimeW flag for a DT_OBJECT property.

Because you cannot view or modify the value of a DT_OBJECT property at

design time, you should specify the PF_fNoShow flag for your DT_OBJECT property.

Note

You should not designate a DT_OBJECT property as the default property for your control.

Additional reference words: 3.00
KBCategory: IAP
KBSubcategory: IAPOLE

PRB: VB.LIC License File Not Found, Can't Load MSOUTLIN.VBX

Article ID: Q107991

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SYMPTOMS

Under certain conditions, running the Professional Edition of Visual Basic version 2.0 or 3.0 can give the following sequence of two error messages:

- License file for custom control not found. You do not have an appropriate license to use this custom control in the design environment.

- Can't load Custom Control DLL: 'D:\WINDOWS\SYSTEM\MSOUTLIN.VBX'

CAUSE

This can be caused by an old VB.LIC file. This is commonly caused by installing the Visual Control Pack after installing Visual Basic for Windows.

WORKAROUND

To work around this problem, decompress the VB.LIC file from your master diskettes for Visual Basic version 3.0.

The PACKING.LST file on master disk number 1 describes on which disk to obtain VB.LIC. PACKING.LST describes how to decompress VB.LIC. Replace your current VB.LIC file with the decompressed VB.LIC from the master disk.

STATUS

This behavior is by design.

MORE INFORMATION

VB.LIC File

For more information about the VB.LIC file, query on the following words in the Microsoft Knowledge Base:

   License and VB.LIC and Visual and Basic

Microsoft Visual Control Pack (VCP)

If you have the Professional edition of Microsoft Visual Basic version 2.0 or 3.0 for Windows, you have everything that the Microsoft Visual Control Pack (VCP) contains and more.

All controls, tools, and documentation shipped with the Microsoft Visual Control Pack are identical to those same controls, tools, and documentation shipped with the Professional Edition of Microsoft Visual Basic versions 2.0 for Windows, with two exceptions:

- A new copy of the MSCOMM.VBX custom control that works with Visual C++ version 1.0 comes with the Visual Control Pack version 1.0.

- Enhanced Control Development Kit (CDK) documentation including helpful hints on creating custom controls for use with Microsoft Visual C++ version 1.0 comes with the Visual Control Pack version 1.0.

Additional reference words: 2.00 3.00
KBCategory: Tls
KBSubcategory: TlsCDK

How VB Can Use OLE Automation with Word Version 6.0

Article ID: Q108043

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0
- Microsoft Word for Windows, version 6.0

SUMMARY

This article shows by example how to use Microsoft Word version 6.0 Object Linking and Embedding (OLE) Automation from Visual Basic. Microsoft Word version 6.0 offers a single OLE object that supports most WordBasic statements and functions as methods. This allows you to create and run WordBasic code from Visual Basic.

NOTE: The technique described in this article may not work if Microsoft Word version 6.0 is set to do background printing. When background printing is on, setting Word=Nothing may cause the Print Job to be canceled. If you encounter this problem, you can work around it by making the Word object variable's scope local to the form rather than to the Sub procedure. Or you can avoid the problem by turning background printing off (On is the the default for Word for Windows). To turn background printing off, choose Options from the Tools menu. Then click the Print tab, and clear the checkbox for background printing.

MORE INFORMATION

Example of OLE Automation

You can invoke the CreateObject function in Visual Basic using Word.Basic as the class name for the WordBasic object. The following example creates and uses a WordBasic OLE object:

```
Sub WordExample ()
   Dim Word As Object                    'Declare an object variable
   Set Word = CreateObject("Word.Basic") 'Set the object pointer
   Word.FileNew                          'Create a new File in Word
   Word.Bold                             'Make the Font Bold
   Word.FontSize 24                    'Make the Font 24 point in size
   Word.CenterPara                     'Center Text on page
   Word.Insert "Isn't VB Great!!"      'Insert some text
   Word.FilePrintDefault               'Print the current document
   Word.FileClose 2                    'Close file without saving.
   Set Word = Nothing                  'Clear the object pointer.
End Sub
```

The CreateObject function will launch Word version 6.0 if it is not already running, otherwise it will use the currently-active instance of Word.

The Set Word = Nothing statement will exit Word if Word was launched by the CreateObject statement.

OLE Automation cannot invoke the FileExit method of WordBasic. Because OLE Automation cannot start a new instance of Word after the initial instance, OLE Automation assumes that the user started Word and the user is responsible for exiting the application.

Troubleshooting Common Problems When Using OLE Automation

The following are answers to common problems that you may encounter when using the Word.Basic OLE object from Visual Basic:

1. The CreateObject function could cause an error under any of the following circumstances:

   - Word is not registered in the Windows REG.DAT file.

   - Windows is low on system resources.

   - Your user-defined NORMAL.DOT template and/or automatically loading macros in Word could run automatic actions that might conflict with your requested OLE Automation commands.

   - The OLE server application is not found. With Windows version 3.1, object linking and embedding (OLE) clients look for a server application in the following order:

     1) The location specified in the Windows REG.DAT file.
     2) The location specified in the WIN.INI file.
     3) The WINDOWS directory.
     4) The WINDOWS\SYSTEM directory.
     5) The location specified in the MS-DOS PATH environment variable (which is specified in the AUTOEXEC.BAT file).

2. The WordBasic language allows certain shortcuts that are not supported by Visual Basic. For example, the following statement is valid in WordBasic but not in Visual Basic:

   ```
   FormatFont .Bold = 1
   ```

   Visual Basic does not support named parameters, such as .Bold above. Visual Basic requires you to convert this to the following:

   ```
   Dim Word As Object                    'Declare an object variable
   Set Word = CreateObject("Word.Basic") 'Set the object pointer
   Word.FormatFont ,,,,,,,,,,,,,,,True   'Format selection as bold.
   ```

   Fortunately, many WordBasic methods are implemented with more than one method, which can simplify the syntax required by Visual Basic. For example, WordBasic has a direct Bold method which you can invoke as follows from Visual Basic:

   ```
   Word.Bold
   ```

3. Visual Basic requires you to pass all arguments up to the last necessary argument. The following example shows the arguments for the ToolsMacro method of WordBasic.

   To run a Word macro use this syntax:

```
Dim Word As Object                     'Declare an object variable
Set Word = CreateObject("Word.Basic") 'Set the object pointer
Word.ToolsMacro "MyMacro", True       'Run the macro called MyMacro
```

   To rename a Word macro use this syntax:

```
Word.ToolsMacro "MyMacro", False, False, 0, False, True,
   "Description for Your Macro", "NewMacroName"
```

4. The online help for Microsoft Word version 6.0 doesn't always show the arguments in the correct order. For example, Word ToolsMacro parameters should be in this order:

```
ToolsMacro .Name = text [, .Run][, .Edit][, .Show = number][, .Delete]
   [,.Rename] [, .Description = text][, .NewName = text][, .SetDesc]
```

5. WordBasic methods that return strings have a syntax that includes a dollar sign, $, to indicate the return type. Visual Basic requires you to enclose these $ methods in square brackets []. The following example returns the text stored in the bookmark "MyBookMark":

```
Dim Word As Object                     'Declare an object variable
Set Word = CreateObject("Word.Basic") 'Set the object pointer
MyVar = Word.[GetBookMark$]("MyBookMark")  'Return text from bookmark
```

What is OLE Automation?

Object linking and embedding (OLE) Automation is a Windows protocol that allows an application to share data or control another application. OLE Automation is an industry standard that applications use to expose their OLE objects to development tools, macro languages, and other containers that support OLE Automation.

Word for Windows provides other applications with an object called Basic and a class name called Word.Basic. Using this object, other applications can send WordBasic instructions to Microsoft Word version 6.0 for Windows.

Applications such as Visual Basic version 3.0 applications that support OLE Automation can use OLE Automation with Word version 6.0, but Word cannot use OLE Automation to gain access to other applications. Using the terminology of Dynamic Data Exchange (DDE), this means that Word can act as a server for another application but cannot act as the client.

A spreadsheet application may expose a worksheet, chart, cell, or range of cells -- all as different types of OLE objects. A word processor might expose OLE objects such as application, paragraph, sentence, bookmark, or selection. You use Visual Basic to manipulate these objects by invoking methods on the object, or by getting and setting the objects properties, just as you would with the objects in Visual Basic.

REFERENCES
==========

- "Visual Basic version 3.0: Programmer's Guide," Chapter 23, "Programming Other Applications' Objects."

- See the following online Help topics in Visual Basic version 3.0: OLE Automation, CreateObject Function, Object Property, Set Statement

Additional reference words: 3.00 OLE2 OA winword w4wmacro wm_word officeinterop 6.00 6.00a
KBCategory: IAP
KBSubcategory: IAPOLE

PRB: DDE Error When Running Setup on Norton Desktop

Article ID: Q108498

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

The Setup Toolkit provided with Visual Basic version 3.0 is used to create distribution disks for applications created with Visual Basic. If Norton Desktop for Windows version 2.0 or 2.2 is being used as the Windows Shell, Windows will report the error message, "An application using DDE did not respond to the system's exit command," shortly after SETUP1 is complete.

CAUSE

The error message occurs because the Norton Desktop shell and the Program Manager each handle DDE messages differently. SETUP1 calls two routines to create groups and items, CreateProgManGroup and CreateProgManItem. Each routine establishes a DDE link and then terminates it after the group or item is created. After the termination of the link, not enough time is being given to the Norton Desktop shell to acknowledge the terminated link.

RESOLUTION

Modify the SETUP1.MAK program to allow the Norton Desktop shell to acknowledge the termination of DDE links. To do this, call the DoEvents function after each call to CreateProgManGroup or CreateProgManItem.

Here are the changes you need to make to the SETUP1.FRM file in the Form_Load event. The SETUP1.FRM is part of the SETUP1.MAK file, which is typically in the \VB\SETUPKIT\SETUP1 directory.

```
   CreateProgManGroup Setup1, "My Loan Application", "LOAN.GRP"

   For i% = 1 To 10
      z% = DoEvents()
   Next

   CreateProgManItem Setup1, destPath$ + "LOAN.EXE", "My Loan Application"

   For i% = 1 To 10
      z% = DoEvents()
   Next
```

As you can see, the loop of DoEvents is included after each and every call to CreateProgManGroup and CreateProgManItem. If more calls to these procedures are added, additional loops of DoEvents must be included after those calls. If the SETUP1A.MAK program is being used, the same changes

must be made to the SETUP1A.FRM Form_Load event. These changes will not affect installations on computers that use the Program Manager shell.

MORE INFORMATION

The SETUP1.MAK program is part of the Setup Toolkit provided with Visual Basic version 3.0. SETUP1.MAK can be modified by the user or the Setup Wizard to design a custom setup that will install the user's application on other computers. It is typically located in the \VB\SETUPKIT\SETUP1 directory.

When the setup program is run on a target computer, the SETUP1.EXE program is copied to the computer's hard drive and launched. The SETUP1 program then proceeds to copy all of the user's application over to the hard drive. At the conclusion of this process, SETUP1 initiates a series of DDE links to the Program Manager. This is to create the program groups and items for the user's application.

When you use Norton Desktop version 2.0 or 2.2 for Windows as the Windows Shell, the DDE links are successful in creating the necessary program groups and items, and the user's application is installed intact. However, some time after the SETUP1 program ends, the error message (An application using DDE did not respond to the system's exit command) appears. Options to retry, continue, and close are displayed in the dialog box. If the user chooses close, the error will disappear and the user's application will still be installed successfully.

Additional reference words: 3.00 norton desktop progman dde error
KBCategory: Tls
KBSubCategory: TlsSetWiz

PRB: Extra Repaint of VB CDK Graphical Custom Control

Article ID: Q108710

The information in this article applies to:

- Professional Edition of Microsoft Visual Basic for Windows, version 3.0

SYMPTOMS

You can use the Visual Basic Control Development Kit (CDK) to design a graphical custom control to perform visual effects. However, when another control such as a Visual Basic button becomes visible on the same form as your custom control, the graphical control repaints even though its client area has not been invalidated.

CAUSE

The repaint logic in Visual Basic causes this behavior. Graphical controls get a VBM_PAINT message every time their container gets a WM_PAINT message, regardless of whether the graphical control is in the invalidated area.

The VBM_PAINT message is the paint notification for graphical controls. The value of the VBM_PAINT constant is defined in the VBAPI.H file in Visual Basic's CDK directory. WM_PAINT is a message for windowed controls. For examples of VBM_PAINT and WM_PAINT, see the PIX.C file in Visual Basic's CDK directory.

WORKAROUND

You can work around this behavior in the VBM_PAINT handler with the following C code:

```
   if (!RectVisible((HDC)wParam,(LPRECT)lParam)) {
      // Control is not in the paint region
      // Don't bother to repaint
      break;
      }
```

Additional reference words: 3.00 flash flicker
KBCategory: Tls
KBSubcategory: TlsCDK

Category Keywords for All Visual Basic KB Articles

Article ID: Q108753

The information in this article applies to:

- Microsoft Visual Basic for Windows, versions 2.0 and 3.0

SUMMARY

Each article in the Visual Basic for Windows collection contains at least one keyword (called a KBSubcategory keyword) that places the article in an appropriate category. This article lists all the KBSubcategory keywords.

MORE INFORMATION

Category & Subcategory Description	KBSubcategory Keyword
Setup / Installation (Setins)	Setins
Environment-specific Issues (Envt)	
VB Design Environment	EnvtDes
Run-Time Environment	EnvtRun
Programming (Prg)	
Visual Basic Forms and Controls	
Standard Controls / Forms	PrgCtrlsStd
Custom Controls	PrgCtrlsCus
Third-Party Controls	PrgCtrlsThird
Optimization	
Memory Management	PrgOptMemMgt
General Optimization Tips	PrgOptTips
General VB Programming	PrgOther
Advanced programming (APrg)	
Network	APrgNet
Windows Programming (APIs / DLLs)	
Printing	APrgPrint
Graphics	APrgGrap
Windowing	APrgWindow
INI Files	APrgINI
Other API / DLL Programming	APrgOther
Data Access	
ODBC	APrgDataODBC
IISAM	APrgDataIISAM
Access	APrgDataAcc
General Database Programming	APrgDataOther
3rd Party DLL's	APrgThirdDLL

Inter-Application Programmability (IAP)

OLE	IAPOLE
DDE	IAPDDE
3rd Party Interoperability	IAPThird

Tools (Tls)

Setup Toolkit / Wizard	TlsSetWiz
Control Development Kit (CDK)	TlsCDK
Help Compiler (HC)	TlsHC

References (Refs)

Documentation / Help File Fixes	RefsDoc
Product Information	RefsProd
Third-Party Information	RefsThird
PSS-Only Information	RefsPSS

Using Keywords to Query the KB

At Microsoft, we use the subcategory keywords to organize the articles for Help files and for the FastTips Catalog. You can use them to query the Microsoft Knowledge Base for Visual Basic articles that apply to that category or subcategory. For example, you can find all the general database programming articles by querying on the following words in the Microsoft Knowledge Base:

```
visual and basic and APrgDataOther
```

Use the asterisk (*) wildcard to find articles that fall into the general categories or into an intermediate subcategory. The first element in each keyword is the category. For example, to find all the articles that apply to Visual Basic Forms and Controls regardless of whether they are standard, custom, or third-party controls, use the following words to query the Microsoft Knowledge Base:

```
visual and basic and PrgCtrls*
```

To find all advanced programming articles, query on these words:

```
visual and basic and APrg*
```

Add KBSubcategory Keyword to Each Article

When contributing an article to the Visual Basic Knowledge Base, add the appropriate KBSubcategory keyword to the bottom of the article on the KBSubcategory line. Each article in the Visual Basic for Windows collection contains the following section at the bottom of the article:

```
Additional reference words:
KBCategory:
KBSubcategory: <keyword>
```

An article usually has only one subcategory keyword, but it may have more.

If you are interested in contributing, please obtain the guidelines by

querying on the following words in the Microsoft Knowledge Base:

   visual and basic and kbguide and kbartwrite

Additional reference words: 3.00 dskbguide subcatkey
KBCategory:
KBSubcategory: RefsPSS

How to View Microsoft Word Toolbars Using OLE Control

Article ID: Q112044

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0
- Microsoft Word for Windows, version 6.0

SUMMARY

This article shows by example how to make a Visual Basic program display Microsoft Word version 6.0 toolbars. It is not possible to display the toolbars using the OLE control alone, so the following example uses an OLE object in combination with the OLE control.

MORE INFORMATION

When editing a Microsoft Word document that is in an OLE control, you may find it helpful to use the Microsoft Word toolbars. However, if you close all your toolbars, there is no way to use the OLE control alone to get the toolbars back. The following example shows how to access the Word toolbars after closing them.

Step-by-Step Example

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add an MSOLE2 control (OLE1) to Form1. When the Insert Object dialog box asks what type of object to insert, choose the Create From File option. and select a Microsoft Word .DOC file.

3. Set the SizeMode property of the OLE1 control to 1 (Stretch).

4. Add a command button (Command1) to Form1.

5. Add the following code to the click event of the Command1 button:

```
Sub Command1_Click()
   Dim wbObject As Object
   ole1.Action = 7
   Set wbObject = CreateObject("Word.basic")
   wbObject.ViewToolbars "Standard", , , , , , , 1
End Sub
```

6. Run the program.

7. Double-click the OLE1 control. If you see the toolbars, close them and press the ESC key to return control to Visual Basic. Double-click the OLE1 control again. This time the toolbars are gone. Now click the

Command1 button to bring up the Standard toolbar. To bring up any of the other toolbars from this point, move the mouse pointer over the Standard toolbar. Then click the right mouse button, and choose the toolbar you want.

Additional reference words: 3.00
KBCategory:
KBSubcategory: IAPOLE PrgCtrlsCus

How to Navigate Excel Objects from Visual Basic Version 3.0

Article ID: Q112194

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0
- Microsoft Excel for Windows, version 5.0

SUMMARY

This article explains three methods you can use to navigate and access Excel Application objects:

- Using longhand and default properties
- Using aliasing
- Using the Parent and Application methods of Excel version 5.0 objects to access any Excel object

MORE INFORMATION

Each object in Microsoft Excel version 5.0 exists somewhere in the application's hierarchy of objects. You choose among these objects by navigating down that application's hierarchy. At the top of this hierarchy is the Application object. Whatever events or actions you assign to the Application object affect the entire application. For example:

```
' Close the application
[Object].Application.Quit
```

Replace [Object] with any variable that points to any valid Excel Application Object, which can be created from the following example:

```
' MyObject represents [Object] and OLE1 represents an OLE control
' that contains an Excel Worksheet object.
MyObject = OLE1.Object
Set MyObject = CreateObject("Excel.Sheet")
Set MyObject = GetObject("C:\EXCEL\EXAMPLES\SAMPLES.XLS")
```

The Application object contains other large objects. For example, you can use the following code to refer to the collection of Workbooks currently loaded in Excel:

```
[Object].Application.Workbooks
```

If you want to retrieve a single workbook from the collection, use the Item method. For example, to refer to the first workbook:

```
[Object].Application.Workbooks.Item(1)
```

To close the first workbook:

```
   [Object].Application.Workbooks.Item(1).Close
```

Accessing Objects Using Longhand Reference or Default Properties

Each workbook contains a collection of worksheets, each worksheet contains a collection of cells, and so on. (See the Excel documentation and Help menu for specific details about Excel's object hierarchy.) In code, referring to a specific cell could look like this:

```
   ' Following refers to cell A1 on Sheet1 in the first workbook.
   ' Enter the following two lines as one, single line:
   [Object].Application.Workbooks.Item(1).
      WorkSheets.Item("Sheet1").Cells.Item(1,1)
```

This reference can be lengthy and complex; however shortcuts are available. Understanding the navigation operator (.) is fundamental to successful object programming.

Short Cuts:

All objects have a default property and method. For collections the default method is the Item method. For most objects the Name property is the default property. This convention was implemented to simplify programming. For example the previous sample can be simplified to:

```
   [Object].Application.Workbooks(1).WorkSheets("Sheet1").Cells(1,1)
```

Accessing Objects by Aliasing Objects

You can use aliasing to simplify object programming. If you were to write a lot of code that was manipulating Sheet1, for example, the syntax could become lengthy. To prevent this, create an object that points to the lowest common object. This is known as aliasing. Use the Set statement to create an alias.

```
   Dim Sheet1 as Object
   ' Alias Sheet1 to represent [Object]...WorkSheets("Sheet1")
   Set Sheet1 = [Object].Application.Workbooks(1).WorkSheets("Sheet1")
   ' Now just use the variable Sheet1 to refer to Sheet1.
   Sheet1.Cells(1,1).Value = "Title"
   Sheet1.Cells(1,2).Value = "ID"
   Sheet1.Cells(1,3).Value = "Cost"
   Sheet1.Cells(2,1).Value = "Phone"
   Sheet1.Cells(2,2).Value = 123413423
   Sheet1.Cells(2,3).Value = 89.95
```

Accessing Objects by Using Parent and Application Methods

The Parent and Application methods allow you to navigate back up the object hierarchy. The Application method navigates back to the application object, and the Parent method navigates up one level of the object hierarchy. All the examples in this article started with [Object]. As long as [Object] is a valid Excel object, all of those statements are also valid. Regardless of the context of [Object].

This is very helpful when programming the Excel object from Visual Basic version 3.0. Excel exposes only the three objects that can be used as entry points to Excel. These are:

- Excel.Application
- Excel.Sheet
- Excel.Chart

Don't be confused by Excel.Application.5. Excel.Application will always point to the latest version of Excel. Excel.Application.5 will point only to Excel version 5.0.

There is no exposed Workbook object, so there's no way to access the Workbook object directly. However, this is not a problem because the Parent method of a Worksheet or Chart object returns the Workbook object. The following example code illustrates this point.

NOTE: oleExcel is an OLE control that contains an Excel.Sheet object.

```
   ' Declare object references:
   Dim Xlapp As object
   Dim XLWkb As object
   Dim XLWks As object
   Dim XLWksNew As object

   oleExcel.Action = 7   ' Activate OLE Object

   Set XLWks = oleExcel.Object   ' Alias Worksheet object
   Set XLWkb = XLWks.Parent      ' Alias WorkBook object
   Set Xlapp = XLWks.Application ' Alias Application object

   ' Add a new worksheet to the Workbook and name it:
   Set XLWksNew = XLWkb.Worksheets.add  ' Assign alias to new Worksheet
   XLWksNew.Name = "VB3 OLE Automation" & XLWkb.Worksheets.count

   ' Make the 3rd Worksheet of the Workbook active:
   XLWkb.Worksheets(3).Activate

   ' Display the dialog for InsertPicture:
   Xlapp.dialogs(342).[Show]  ' xlDialogInsertPicture = 342
```

REFERENCES
==========

- Office Development Kit, Programming Integrated Solutions

Additional reference words: 5.00 3.00 officeinterop w_VBApp W_Excel WM_OLE OA OLE Automation
KBCategory:
KBSubcategory: IAPOLE

How to Print an Embedded Word Document in Visual Basic

Article ID: Q112196

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0
- Microsoft Word for Windows, version 6.0

SUMMARY

Microsoft Word for Windows version 6.0 disables the ability to use the FilePrint and FilePrintDefault methods while an object is being edited in an OLE container. While the menu options may not be enabled, it is still possible to get around this in code. This article explains how.

MORE INFORMATION

Commands that are part of the workspace are the responsibility of the top container (the Visual Basic application). That is, the application is responsible for the organization of windows, file level operations, and how edits are ultimately saved. The top container must supply a single File menu that contains file level commands such as Open, Close, Save, and Print. If the object is an opened object server application, the commands in its File menu are modified to show containership (Close & Return to <container doc>, Exit & Return to <container doc>).

A well-behaved OLE server will not allow workspace commands to be executed. This is why they are disabled. To work around the problem, edit the object in the server application instead of using in-place editing. In the server workspace, commands are enabled. Therefore, you can edit the object in the server workspace and use OLE Automation to control the server to execute the Workspace commands.

Example Program Using OLE Automation

The following example activates the Word object in the server, and uses OLE Automation to execute the FilePrintDefault method.

NOTE: By default, Word sets background printing On. If Word quits before printing is completed, the print job is aborted. There are two ways to work around this:

- Define the Word Objects globally. The objects will remain in memory until the container application (Visual Basic) quits. This is the easiest way to do it.

  -or-

- Disable background printing in Word. You can do this by using OLE automation. The command is not available during in-place editing. The

following example shows how to do this in code.

1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.

2. Add a command button (Command1) to Form1.

3. Add an MSOLE2.VBX control (OLE1) to Form1. When the Insert Object dialog comes up, choose the Create From File option button, and select a Word for Windows document.

4. Add the following code to the Command1_Click event:

```
Sub Command_Click()
    ' Open application in separate application Window:
    ole1.Verb = -2
    ' Activate Object:
    ole1.Action = 7
    Dim WB As object
    ' Alias WordBasic Object:
    Set WB = ole1.Object.application.wordbasic
    ' Disable background printing:
    WB.ToolsOptionsPrint , , , , , , , , , , , 0
    WB.FilePrintDefault  'Print the Word Object.
    ' Hint: it may be necessary to check page layout parameters before
    ' printing. If parameters are outside of the printable region, Word
    ' will display an error message.
End Sub
```

5. Run the program, and click the Command1 button.

Additional reference words: 3.00
KBCategory:
KBSubcategory: IAPOLE PrgCtrlsCus

Retrieving Groups & Items from Program Manager Using DDE in VB

Article ID: Q112384

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic programming system for Windows, versions 1.0, 2.0, and 3.0

SUMMARY

Program Manager (Progman) has a DDE command-string interface that allows other applications to view the groups and items that currently exist within Progman.

MORE INFORMATION

Perform the following steps to produce an application that will retrieve the group and items from Progman by using DDE:

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add two Text box controls (tGroups and tItems) to Form1. Set the visible property on both to false. The DDE link needs these controls.

3. Add two Combo box controls (ComGroups and ComItems) to Form1.

4. Add a Command button (cGroups) to Form1 and change the caption property to Groups.

5. Add the following code to the cGroups_Click () event:

```
Sub cGroups_Click ()
   Dim sGroups As String
   Dim pos As Integer
   On Error GoTo GError
   tGroups.LinkMode = 0
   tGroups.LinkTopic = "Progman|Progman"
   tGroups.LinkMode = 2
   tGroups.LinkItem = "groups"
   tGroups.LinkRequest

   ' Parse groups that come back:
   sGroups = tGroups.Text
   pos = InStr(1, sGroups, Chr(13))
   While pos
      ComGroups.AddItem RTrim$(Mid$(sGroups, 1, pos - 1))
      sGroups = LTrim$(Mid$(sGroups, pos + 2))
      ' The + 2 on the previous line gets past the line feed chr(10)
      pos = InStr(1, sGroups, Chr(13))
   Wend

   ' Select first member in combo box:
```

```
      ComGroups.ListIndex = 1
      GDone:
         tGroups.LinkMode = 0
         Exit Sub
      GError:
         MsgBox "Error in getting groups"
         Resume GDone
   End Sub
```

6. Add another Command button (cItems) to Form1, and change the caption to Items.

7. Add the following code to the cItems_Click () event:

```
   Sub cItems_Click ()
      Dim sItems As String
      On Error GoTo IError

      ' Clear the combo box:
      ComItems.Clear
      If (Len(ComGroups.Text)) Then
         tItems.LinkMode = 0
         tItems.LinkTopic = "Progman|Progman"
         tItems.LinkMode = 2
         tItems.LinkItem = ComGroups.Text
         tItems.LinkRequest

         ' Parse items that come back:
         sItems = tItems.Text
         pos = InStr(1, sItems, Chr(13))
         While pos
            ComItems.AddItem RTrim$(Mid$(sItems, 1, pos - 1))
            sItems = LTrim$(Mid$(sItems, pos + 2))
            ' The + 2 on the previous line gets past the line feed chr(10)
            pos = InStr(1, sItems, Chr(13))
         Wend
      End If

      ' Select first member in combo box:
      ComItems.ListIndex = 1
      IDone:
         tItems.LinkMode = 0
         Exit Sub
      IError:
         MsgBox "Error in getting items"
         Resume IDone
   End Sub
```

8. From the Run menu, choose Start (ALT, R, S) or press the F5 key to run the program. Press the Groups button and all the groups on Progman will be loaded into the ComGroups Combo box. Then press the Items button and all the items in the group selected in the ComGroups Combo box will be put into the ComItems Combo box.

REFERENCES
==========

More information can be found in:

- "Programmers Reference, Volume 1: Overview Microsoft Windows SDK," Chapter 17, "Shell Dynamic Data Exchange Interface DDEML."
- Progman topic in the Windows SDK Help menu.

Additional reference words: 1.00 2.00 3.00 interop icon
KBCategory:
KBSubcategory: IAPDDE

How to Find Articles On Visual Basic For Applications

Article ID: Q112391

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Word for Windows, version 6.0
- Microsoft Excel, version 5.0
- Microsoft Office for Windows, version 4.0

For information about Visual Basic for Applications and OLE Automation, use a combination of the following reference words to query the Microsoft Knowledge Base:

Reference Word	Meaning
6.00	Applies to Word 6.0
6.00a	Applies to Word 6.0a
IAPOLE	Inter-Application Programming OLE issues
IAPOLEAuto	Inter-Application OLE Automation issues
officeinterop	Issues between office applications
w4wmacro	WordBasic issues
XL5	In the title of Excel version 5.0 *only* articles

If you want to limit your query to one or two products, add the product mnemonic keywords to your query:

Product name	Product Mnemonic
Excel (Cross-platform issues)	WM_Excel
Excel for Windows	W_Excel
Microsoft Office for Windows	W_Office
Visual Basic for Applications	W_VBApp
Visual Basic for Windows	B_VBasic
Microsoft Word (Windows & Macintosh)	WM_Word

Additional reference words: 5.00 3.00 officeinterop w_VBApp W_Excel WM_OLE OA OLE Automation
KBCategory:
KBSubcategory: IAPOLE

How to Create Excel Chart w/ OLE Automation from Visual Basic

Article ID: Q112417

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows version 3.0
- Microsoft Excel, version 5.0

SUMMARY

This article shows by example how to insert data into a Microsoft Excel version 5.0 worksheet and create a chart by using OLE automation in a Visual Basic application.

MORE INFORMATION

There are five key points you need to keep in mind when creating an Excel chart in a Visual Basic program.

- A chart can be either on a Worksheet or a Chart sheet.
- A chart on a Worksheet is a ChartObject.
- A ChartObject has a Chart property, which is a Chart Object.
- The data associated with the Chart is part of the SeriesCollection.
- You can add a new data series with a Named Range.

NOTE: Complete definitions for these objects can be found in the Excel documentation.

A Workbook has a Charts collection, which is the collection of all Chart sheets in the workbook. All Charts on WorkSheets are part of that WorkSheet's ChartObjects collection. Therefore to add a new Chart to a Worksheet, you can use the Add method on the worksheets ChartObject collection.

The SeriesCollection property of the Chart object contains the reference to the data linked to the table. In the example below, you'll add two data series -- each of which contains 10 data points -- by using the Add method on the SeriesCollection object.

Although the example passes the Range of cells containing the data as a named range, you could specify a Range in R1C1 notation.

Steps to Create Example Program

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a command button (Command1) to Form1.

3. Add the following code to the Command1_Click event procedure:

```
Sub Command1_Click()
   Dim objXLsheet As Object ' Object reference to Excel Worksheet
   Dim objRange1 As Object  ' First series in the chart
   Dim objRange2 As Object  ' Second series in the chart
   Dim objChart1 As Object  ' Object reference to the chart we create

   Dim iRow As Integer ' Index variable for the current Row
   Dim iCol As Integer ' Index variable for the current Row

   Dim strTmpRange As String ' Temporarily hold Range in R1C1 notation

   Const cNumCols = 10 ' Number of points in each Series
   Const cNumRows = 2  ' Number of Series

   ' Create a Worksheet Object:
   Set objXLsheet = CreateObject("Excel.Sheet")

   Randomize Timer

   ' Insert Random data into Cells for the two Series:
   For iRow = 1 To cNumRows
      For iCol = 1 To cNumCols
         objXLsheet.Cells(iRow, iCol).Value = Int(Rnd * 50) + 1
      Next iCol
   Next iRow

   ' Insert Named Ranges:
   For iRow = 1 To cNumRows
      ' Enter the following two lines as one, single line:
      strTmpRange = "R" & iRow & "C" & Format$(1) & ":R" & iRow & "C"
         & Format$(cNumCols)
      ' Enter the following two lines as one, single line:
      objXLsheet.Parent.Names.Add "Range" & Format$(iRow), "=Sheet1!"
         & strTmpRange
   Next iRow

   ' Add a ChartObject to the worksheet:
   Set objChart1 = objXLsheet.ChartObjects.Add(100, 100, 200, 200)

   ' Assign the Ranges created above as the individual series
   ' for the chart:
   For iRow = 1 To cNumRows
      objChart1.Chart.SeriesCollection.Add "Range" & Format$(iRow)
   Next iRow

   ' Make Excel Visible:
   objXLsheet.application.Visible = True
   DoEvents

   ' Save the Worksheet to disk. The parent of a WorkSheet is WorkBook.
   objXLsheet.Parent.SaveAs "C:\VB\XLCHART.XLS"

   ' Close this instance of Excel:
   objXLsheet.application.Quit
End Sub
```

4. Press the F5 key to run the program, and click the command button.

At this point, Excel starts, and it loads and displays the worksheet with the newly created chart. If you dont already have a file name as specified on the jXLSheet.Parent.SaveAs line of code, Excel saves the file and closes itself down. If you already have a file with the same name, Excel brings up a dialog asking you if you would like to overwrite the existing file.

Additional reference words: 3.00
KBCategory:
KBSubcategory: IAPOLE

How to Save an Embedded Word Document in Visual Basic

Article ID: Q112440

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Word for Windows, version 6.0

SUMMARY

Word version 6.0 for Windows disables the ability to do a FileSaveAs while an object is being edited in an OLE container. While these methods may not be enabled, it is possible to work around this limitation in code. This article explains how.

MORE INFORMATION

Commands that are part of the workspace are the responsibility of the top container (the Visual Basic application). That is, the application is responsible for the organization of windows, file level operations, and how edits are ultimately saved. The top container must supply a single File menu that contains file level commands such as Open, Close, Save, and Print.

If the object is an opened object server application, the commands in its File menu are modified to show containership (Close & Return to <container doc>, Exit & Return to <container doc>).

A well-behaved OLE server will not allow workspace commands to be executed. This is why they are disabled. To work around the problem, edit the object in the server application -- without using in-place editing. In the server, you'll find that the workspace commands are enabled. Therefore edit the object in the server and use OLE Automation to control the server to execute

the Workspace commands.

Step-by-Step Example

The following example uses an OLE2 control called OLE1, which contains an embedded Word version 6.0 document and a CommonDialog control called CMDialog1. To make the code generic, the OLE1 control is passed to the WordFileSave subroutine.

1. Start a new project in Visual Basic, Form1 is created by default.

2. Add a command button (Command1), MSOLE2.VBX (OLE1) control, and a CMDIALOG.VBX (CMDialog1) control to Form1.

3. Add the following code to the Command1_Click event:

```
Sub Command1_Click ()
   ' Pass the name of the Control to WordFileSave subroutine.
   WordFileSave OLE1
End Sub
```

4. Add the following code to the general declarations section of Form1:

```
Sub WordFileSave (OLECtrl As Control)
   'Purpose: Example of how to save an embedded Word object from
   'Visual Basic as a Word Document.

   'Overview of technique:      '
   'Activate Object. Select its contents. Copy contents to clipboard.
   'Launch a hidden instance of Word. Create a new file.
   'Paste clipboard into document. Save document.

   Dim Word As Object  'Alias to Hidden instance of Word.
                       'Only if Word is not already running.
   Dim WB As Object    'alias to WordBasic object.

   OLECtrl.Action = 7  'Activate OLE control. This must be done in order
                       'to have the Word Basic alias act on the correct
                       'instance of Word.
   Set WB = CreateObject("Word.Basic") 'Set the object variable.

   WB.editselectall    'Select the contents of the embedded object.
   WB.EditCopy         'Copy the selection to the clipboard.
   OLECtrl.Action = 9  'Deactivate the OLE control. This must be
                       'done before the following set statements to
                       'reference the correct instances of Word.

   'Use the Common dialog control to display a SaveAs dialog.
   CMDialog1.Filter = "Word Document (*.Doc)|*.doc"  'Set the filter
   CMDialog1.DefaultExt = "*.doc"            'Set the default extension
   CMDialog1.FileName = OLECtrl.SourceDoc 'Set default filename
   CMDialog1.Action = 2                      'Display the dialog.


   Set WB = Nothing    'Free the WB object reference.
   Set Word = GetObject("", "Word.Document.6")  'Create a hidden inst.
   Set WB = Word.application.Wordbasic  'Set WB to the WordBasic object
                                        'of the new instance of Word.

   WB.filenew        'Create a New file in hidden instance of Word.
   WB.editpaste      'Paste contents of clipboard into new document.
   WB.filesaveas CMDialog1.Filename  'Save file as selected by user.
   WB.fileclose       'Close document.

   Set WB = Nothing   'Free WordBasic object
   Set Word = Nothing 'Free Word Document object, if Word wasn't
                      'running previously, Word will shut itself down
                      'from memory; otherwise, it is up to the user to
                      'shut Word down.
End Sub
```

5. Run the program. The program will ask you to input a name and then save

the document to the name that you input.

Additional reference words: 3.00
KBCategory:
KBSubcategory: IAPOLE PrgCtrlsCus

How VB Can Use OLE Automation with Excel Version 5.0

Article ID: Q112443

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0
- Microsoft Excel, version 5.0

SUMMARY

This article shows by example how to embed a Microsoft Excel version 5.0 Worksheet object in a Visual Basic application, and then manipulate it by using OLE Automation and an MSOLE2.VBX control.

Microsoft Excel version 5.0 offers OLE objects that support Worksheet and Chart functionality using Visual Basic for Applications.

MORE INFORMATION

A worksheet in Excel is sometimes called a spreadsheet. It is the primary document used in Excel to store and manipulate data. A worksheet consists of cells organized into columns and rows and is always a part of a workbook.

Step-by-Step Example of OLE 2.0 Automation

The following example shows how to use OLE automation to accomplish some common tasks on a worksheet, such as adding data, computing a sum and selecting cells.

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add the following constant declarations, taken from the CONSTANT.TXT file, into the general declarations section of Form1:

   ```
   Const OLE_CREATE_EMBED = 0
   Const OLE_ACTIVATE = 7
   ```

3. Add the MSOLE2.VBX file to the project, using the Add File option in the File Menu. MSOLE2.VBX is found in the WINDOWS\SYSTEM directory. The OLE control will appear as an option on the Visual Basic toolbar. Add an OLE control (OLE1) to Form1. Cancel the Insert Object Dialog box that pops up. You will be left with an empty OLE1 object on Form1. Set the OLE1 control's SizeMode property to Stretch.

4. Add a command button (Command1) to Form1. Set the Caption to: Embed Excel 5.0 Object. Add the following code to the Command1 Click event to embed an Excel version 5.0 worksheet into the OLE1 control. Because the SizeMode property is set to Stretch, the Worksheet automatically sizes itself in the OLE1 control when the code is executed.

```
Sub Command1_Click()
   ole1.Class = "Excel.Sheet.5"
   ole1.Action = OLE_CREATE_EMBED
End Sub
```

5. Place another Command button (Command2) on Form1. Change the Command button's Caption to: Add Data. Add the following code to the Command2 click event:

   ```
   Sub Command2_Click ()
      ole1.Action = OLE_ACTIVATE
      ole1.Object.cells(1, 1).value = "Jan"
      ole1.Object.cells(2, 1).value = 3
      ole1.Object.cells(3, 1).value = 4
      ole1.Object.cells(4, 1).value = 6
   End Sub
   ```

   The "ole1.Object" part is Visual Basic code. The rest of the line (cells(2,1).value = 1) is Excel's Visual Basic for Applications code.

6. Choose Start from the Run menu or press the F5 key to run the program. Click Command1 to see the worksheet. Click Command2 to see the information added to the worksheet. Choose End from the Run menu to return to development.

7. Add another Command button (Command3) to experiment with functions. Add the following code to the Command3 Click event code. The SUM function is one of many Excel functions that you can use in an experiment. Run the application, and press the command buttons to see the effect.

   ```
   Sub Command3_Click()
      ole1.Action = OLE_ACTIVATE
      ole1.Object.Range("A2:A4").Select

      ' Try any one of the following lines, or add some pauses between them
      ' to see the selections taking place and the active cell changing.
      ' To try a line, remove the single quotation mark to uncomment the
      ' line:

      ' ole1.Object.Range("C6").Activate
      ' ole1.Object.cells(6, 1).value = "=SUM(R2C:R4C)"
      ' ole1.Object.Range("A6").Select

   End Sub
   ```

How to Find Out More

To find out more about Microsoft Excel's Visual Basic for Applications, open a new module sheet in Excel, and choose Object Browser from the View menu, or press the F2 key. The Object Browser lists all the objects in Excel and their related objects and methods. The Object Browser demonstrates the heirachical nature of the object model.

If you want to try something new, but are unsure of the syntax, it is a good idea to start the Macro recorder in Microsoft Excel, step through the

process manually, switch off the Macro recorder, and view the code in the current module. Then cut and paste the code into the Visual Basic event procedure. Usually all that is required is a prefix of ole1.object.

Additional reference words: 3.00 W_VBApp
KBCategory:
KBSubcategory: IAPOLE

POSITION.HLP File for VB OLE Automation w/ Word for Windows

Article ID: Q112733

The information in the article applies to:

- Standard and Professional Editions of Microsoft Visual Basic, for Windows, version 3.0
- Microsoft Word for Windows, version 6.0

SUMMARY

The contents of POSITION.TXT, a file currently being shipped with the Word Development Kit (WDK), is available as a Help file (POSITION.HLP) for use by Visual Basic version 3.0 programmers who want to use OLE Automation to access WordBasic in Microsoft Word version 6.0.

POSITION.HLP provides a listing of WordBasic functions and all their required parameters in the proper order, which is not the case in Word version 6.0 Help file.

To get obtain POSITION.HLP, download POSITION.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for POSITION.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download POSITION.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get POSITION.EXE

MORE INFORMATION

WordBasic statements or functions in Microsoft Word version 6.0 can take named arguments. But Visual Basic version 3.0 does not support named arguments. This means that when a Visual Basic version 3.0 application sends WordBasic commands through OLE Automation, Visual Basic must specify all the parameters to a WordBasic function in the proper order. The same call in WordBasic may only require some of the parameters using named arguments, and those parameters could be provided in any order.

For most WordBasic statements the positioning of the arguments is documented in the WordBasic Help topics or printed reference entries for those statements. However, some statements' arguments are not listed in proper order, or the arguments are irrelevant or have no effect. These

arguments are not documented in WordBasic Help or in the printed reference.

For example, the InsertIndex statement corresponds to the Index tab in the Index and Tables dialog box. The InsertIndex statement takes a number of arguments that have to do with other tabs in the dialog box, such as the Table of Contents tab. Because these arguments are irrelevant to inserting an index, they are ignored and therefore not documented in WordBasic Help or in the printed reference. But the Visual Basic version 3.0 OLE Automation programmer needs to be aware of these arguments so that he or she can correctly specify arguments by position.

Additional reference words: 3.00 6.00 2.00 OLE2 readme softlib S14663
KBCategory: IAP
KBSubCategory: IAPOLE

How to Perform Microsoft Access Macro Action Via DDE from VB

Article ID: Q112767

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY

From Visual Basic, you cannot directly use a form or report that was created by the Microsoft Access engine. This article shows by example how to use DDE to do it indirectly. The example prints one of the built-in reports from the NWIND.MDB sample database by using DDE and the OpenReport macro action in Microsoft Access.

MORE INFORMATION

For more information about Microsoft Access macro actions, please see the Microsoft Access documentation or Help menu. A Visual Basic application can call most of these actions by using DDE.

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.

2. Add a text box (Text1) and Command button (Command1) to Form1.

3. Place the following code in the Command1 button's click event:

```
' Note the time-out has to be long enough to allow for the print
' to complete or an error will occur.
Sub Command1_Click ()
   Text1.LinkTimeout = 600 'Set DDE Time-out for 60 Seconds
   Text1.LinkTopic = "MSACCESS|SYSTEM"
   Text1.LinkMode = 2  ' Establish manual DDE link to Microsoft Access.
   Text1.LinkExecute "[OPENREPORT Catalog]" 'Open and Print     Report
   Text1.LinkMode = 0  ' Terminate the DDE link to Microsoft Access
End Sub
```

4. Start Microsoft Access and open the NWIND.MDB sample database.

5. Run the Visual Basic program, and click the Command1 button.

Additional reference words: 3.00
KBCategory:
KBSubcategory: IAPDDE APrgDataAcc