Visual Basic for Applications Tips

Microsoft Visual Basic for Applications Tips

Visual Basic for Applications Tips

- XL5: No Visual Basic Method for SET.UPDATE.STATUS
- XL5: Can't Access File Opened as Read-Only in Visual Basic
- XL5: GP Fault When Macro Closes Its Own Workbook
- XL5: Selection.Rows.Count May Return Incorrect Result
- XL5: Error Getting the Value of a Name in Visual Basic

XL5: Range.EntireRow May Return Incorrect Result

- XL5: Running Subroutines and Macros from Visual Basic
- XL5: Macro to Add a Number to a Selected Cell
- XL5: OLE Automation Err Msg: Method Not Applicable
- Menus.Count Returns Different Number When Workbook Maximized
- XL5: Manually Recalculating a Single Cell or Range
- XL5: Cells.Find Returns Error When No Match Found
- XL5: Visual Basic Macro to Hide and Restore All Toolbars
- XL5: Macro to Open the Most Recently Used File
- XL5: All PageSetup Settings Are Recorded into Macro
- XL5: Function Subroutines Can Show Message Boxes
- XL5 Err Msg: "Not Enough Memory" With Indirect Defined Names

XL5: Verifying the Value of a Check Box

- XL5: Cannot Use Array As Source Argument with SeriesCollection
- XL5: Branching to Other Sections of Code with GoTo and Call
- XL5: Visual Basic and the Microsoft Excel Version 4.0
- XL5: Using Visual Basic to Exit Windows From Within Excel
- XL5: Update Remote References Option Selected by Default
- XL5: Can't Use Line Continuation Character in Some Locations
- XL5: List of the Files You Need to Run Microsoft Excel
- XL5: GP Fault If REFTEXT() Refers to Closed Workbook
- XL5: Macro to Find Directory in WIN.INI for an Application
- XL5: Visual Basic Example to Delete Blank Rows
- Computing Periodic Annual Interest Rate in Microsoft Excel

XL5 Err: "Call to Undefined Dynalink" in Visual Basic Macro

- XL5 OLE Automation: Workbook Hidden Using GetObject Function
- XL5: Calculating Elapsed Time for a Visual Basic Procedure
- XL5: Can't Define Name with Same Name as a Subroutine
- XL5: Borders Method Applies Inconsistent Format
- XL5: CreateObject Function Starts Invisible Instance of Excel
- Excel: Creating Macros for Different Language Versions
- XL5: Hiding Button Objects with Visual Basic for Applications
- XL5: Determining which Items Are Selected in a List Box
- XL5: Application.OperatingSystem Returns 3.10 in WFW 3.11
- XL5: FOR Behaves Differently in Visual Basic Than in 4.0 Macro
- XL5 Err Msg: "Not Enough Stack Space to Run Macro"

XL5: Controls in Dialog Box May "Snap" to Preset Values

- XL5: Can't Use Replace Command to Search for Blank Cells
- Selection.Cells with Nonadjacent Selection Returns Single Cell
- XL5: Cannot Print Multiple Copies to DeskJet 500C/550C Driver

- XL5: "Cannot Find Macro..." Using Run Method with Add-In Macro
- XL5: 'For Each Item in List' Doesn't Work
- XL5: Getting Windows Status Information from Windows API
- XL5: Using the Windows OpenFile Dialog Box
- XL5: Incorrect Use of "Is" Function Causes GP Fault
- XL5: OLE Automation Dialog Box Redrawn Incorrectly
- XL5: Setting Status Bar Text and ToolTips for Toolbar Buttons
- Excel AppNote: Built-in Constants in VB, Applications Edition

XL5: Can't Use Group Edit Mode on Multiple Workbooks

- XL5: Can't Use Dialog Method to Bring Up Spelling Dialog Box
- XL5: Dialogs(xlDialogFont) Changes Normal Style
- XL5: OLE Automation, Can't Use Named Arguments in Visual Basic
- XL5: Can't Print Collated Copies
- XL5: "Invalid Procedure Call" with SendKeys Statement
- XL5: Docerr: VB for Apps Sub Example Incorrect
- XL5: Visual Basic Macro to Concatenate Columns of Data
- XL5: User-Defined Function to Put Sheet Name in a Cell
- XL5: Using Visual Basic to Multiply Cell Contents
- XL5: Caption Property of Menu Contains Ampersands
- XL5: Visual Basic Code to Use Instead of DIRECTORIES()

XL5: Shell Function Doesn't Accept Built-in Constants

- XL5: Visual Basic Nested Case Statement Example
- XL5: Can't Select Label Attached to Filled Radar Series
- XL5: Recorded AutoFit Selection Changes Entire Row or Column
- XL5: Error Message Using Name of Constant with Show Method
- XL5: OLE Automation Error Using Quit Method with GetObject
- XL5: Command to Create Add-in File from Visual Basic Module
- XL5: Can't Set Value Property in For-Each with Variant Type
- How to Manipulate Object's Properties w/ Property Set/Let/Get
- Line Numbers Greater Than 65529 Not Supported
- Null Character Truncates String in Visual Basic
- PRB: Error When Excel VBA Proc & Implicit Var Have Same Name

PRB: Sub Name Can't Be Valid Cell Reference in Excel VBA

- BUG: VBA FileCopy Updates Destination File's Date & Time Stamp
- BUG: VBA Shell Returns Invalid Proc Call Even If App Starts
- "Invalid Data Format" Referencing File that Contains Procedure
- How to Dimension a Variable as Name Type
- Using Square Brackets in a Visual Basic Procedure
- BUG: SendKeys Fails to Send Right Brace Character in Excel VBA
- BUG: Is Operator in VBA Incorrectly Evaluates Excel Objects

Recursive Routines Possible in Excel Function Macros

Article ID: Q47949

The information in this article applies to:

- Microsoft Excel for Windows, versions 2.x, 3.0, 4.0, 5.0

SUMMARY
=======

A Microsoft Excel function macro can call itself in a recursive fashion to compute an answer. For example, the following macro computes the factorial of a number:

```
A1: factorial
A2: =ARGUMENT("parameter")
A3: =IF(parameter=1,RETURN(1))
A4: =RETURN(parameter*factorial(parameter-1))
```

Note: This function macro is provided only as an example. Microsoft Excel has a built-in FACT() function that computes the factorial of a number more efficiently.

The macro must be defined as a function macro.

If you are using Microsoft Excel version 5.0, you can use the equivalent macro in Visual Basic for Applications:

```
Function factorial(parameter As Integer) As Integer
   If parameter = 1 Then
      factorial = 1
   Else
      factorial = parameter * factorial(parameter - 1)
   End If
End Function
```

Additional reference words: 2.0 2.00 2.01 2.1 2.10 3.0 3.00 4.0 4.00 5.0 5.00 VBAppcode

Spreadsheet Won't Visibly Update Behind Triggered Dialog Box

Article ID: Q68508

The information in this article applies to:

- Microsoft Excel for Windows, versions 3.0, 4.0, 5.0

SUMMARY

If a macro is allowed to continue execution by the use of a triggered dialog box (which means the dialog box will be allowed to remain on the screen as the macro continues), any changes made by the macro to the document behind the custom dialog box will not be displayed. A triggered dialog box has the same effect as ECHO(FALSE) before using a macro to make changes to a spreadsheet. That is, the changes are actually made, but the screen is not updated (to save time).

If you want to display updating as it takes place, you must use an ECHO(TRUE) statement before statements in the continuing macro code that make changes to the active document. However, this will cause the dialog box to be erased from the screen until it is redisplayed by another DIALOG.BOX statement.

The information above also applies to version 5.0 if you are working with version 4.0 macros. Visual Basic for Applications code allows a spreadsheet to update while a Dialog box (created from a Dialog sheet) is displayed.

REFERENCES

"User's Guide 2," version 4.0, pages 280-281
"User's Guide," version 3.0, page 639

Additional reference words: 3.0 3.00 4.0 4.00 5.0 5.00

Activating An Embedded Chart Using A Macro

Article ID: Q71539

The information in this article applies to:

- Microsoft Excel for Windows, versions 3.0, 4.0, 5.0

SUMMARY
=======

You can create an embedded chart object using the chart tool on the Standard Toolbar. You must create an embedded chart in the instance of Microsoft Excel that contains the supporting data. The embedded chart is then dynamically linked to this data.

To add a legend, title, etc. to your embedded chart, you must activate the chart object. You can do this by double-clicking anywhere on the chart object. In Microsoft Excel version 5.0, when you activate an embedded chart, a hatched border appears around the chart and you can format the chart elements.

In Microsoft Excel versions 3.0 and 4.0, when you activate an embedded chart, a copy of the chart object opens in a new window. This new window displays the chart menu bar, allowing you to format the chart elements. The embedded chart on the worksheet immediately reflects all changes made to the copy in the window.

You can also activate an embedded chart object using a macro as in the following examples:

Microsoft Excel version 5.0 Visual Basic Example

```
' Dimension variable
Dim Data As String
' Assign variable Data to selected cells
Data = Selection.Address
' Create embedded chart from selected data
ActiveSheet.ChartObjects.Add(82.8, 76.8, 321.6, 100.2).Select
ActiveChart.ChartWizard Source:=Range(Data)
' Activate embedded chart for editing
Selection.Activate
```


Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line

to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Microsoft Excel versions 3.0 and 4.0 Macro

```
A1: =COPY()
A2: =CREATE.OBJECT(5,!$B$5,5,5,!$D$15,5,5)
A3: =DEFINE.NAME("Temp")
A4: =GET.NAME("!Temp")
A5: worksheet=GET.DOCUMENT(1)
A6: =UNHIDE(worksheet&" "&A2)
A7: =RETURN()
```

This macro does the following:

1. Copies the selected data for the chart from the worksheet

2. Creates the embedded chart

3. Assigns the name "Temp" to the chart object

4. Gets the definition of the name "Temp"

5. Assigns the name "worksheet" to the name of the worksheet

6. Activates the chart object

Microsoft provides macro examples for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This macro is provided 'as is' and Microsoft does not guarantee that the following code can be used in all situations. Microsoft does not support modifications of that code to suit specific customer requirements.

REFERENCES
==========

"User's Guide," version 4.0, page 394-399, 435

"User's Guide," version 3.0, page 325, 391-394

"Function Reference," version 4.0, pages 65-66, 77-79, 196, 201, 441

"Function Reference," version 3.0, pages 36, 40, 107, 111, 241

Additional reference words: 3.0 3.00 4.0 4.00 5.0 5.00

Using a Mouse to Enter References into a Custom Dialog Box

Article ID: Q74267

The information in this article applies to

- Microsoft Excel for Windows, versions 3.0, 4.0, 5.0
- Microsoft Excel for OS/2 version 3.0

SUMMARY
=======

Entering cell references in a custom dialog box with a mouse requires a movable dialog box. In version 3.0 and 4.0 this is done by typing a title in the first row of the text column in the dialog box definition table. In version 5.0, using Visual Basic for Applications, dialog boxes are moveable regardless of whether they have a title.

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

A Reference Edit box allows references to be entered into the dialog box. Cell references are manually entered by typing directly into the edit box. After clicking the Reference Edit box, nothing can be selected outside of the dialog box if the dialog is not movable.

By making the dialog box movable, cells on the worksheet can be selected. After clicking the Reference Edit box, you can then manually select the desired range of cells with your mouse. Menus and cells in nonactive documents can also be chosen if you want to enter information into the dialog box.

To create a movable custom dialog box, type a title in the first row of the text column in the dialog box definition table. This is the sixth column. The title will then appear in the title bar of your dialog box, which can then be used to move the dialog box.

REFERENCES
==========

"User's Guide 2," version 4.0, pages 270-282
"User's Guide," version 3.0, pages 631-641

Additional reference words: 3.0 3.00 4.0 4.00 4.0a 4.00a 5.0 5.00 moveable

Using GET.CELL with Type_num 15 in Excel

Article ID: Q75622

The information in this article applies to:

- Microsoft Excel for Windows, versions 2.x, 3.0, 4.0, 5.0
- Microsoft Excel for OS/2, versions 2.2, 3.0

SUMMARY

The GET.CELL macro function with type number 15 returns TRUE or FALSE, depending on whether the selected cell has been hidden by selecting Cell Protection from the Format menu and checking the Hidden check box. This function does not test to see whether the cell has been hidden using the Row Height or Column Width commands in the Format menu.

MORE INFORMATION

Formatting cell protection to hidden prevents a selected cell's formula from being displayed or changed in the formula bar. GET.CELL(15) will return TRUE if the cell protection is formatted as hidden in this manner. GET.CELL(15) does NOT test for columns or rows that have been hidden by choosing the Row Height or Column Width command from the Format menu and choosing the Hide button. You can use GET.CELL(16) and/or GET.CELL(17) to test for hidden rows or columns.

For help with GET.CELL in Microsoft Excel 5.0, from the Help menu, choose Contents, then select Reference Information. Select Microsoft Excel Macro Functions Contents, then choose Alphabetical List of Macro Functions and scroll down the list until you reach GET.CELL.

In Microsoft Excel version 5.0, you can also use Visual Basic for Applications code to confirm whether an active cell is in a hidden row or hidden column.

```
Sub testhiddenrow()
   hiddenrow = Rows(ActiveCell.Row).Hidden
   MsgBox hiddenrow
End Sub

Sub testhiddencolumn()
   hiddencol = Columns(ActiveCell.Column).Hidden
   MsgBox hiddencol
End Sub
```

REFERENCES

Online Help, version 5.0
"Function Reference," version 4.0, page 191-194
"User's Guide," version 3.0, page 564

"Function Reference," version 3.0, page 103

Additional reference words: 2.0 2.00 2.01 2.1 2.10 2.2 2.20 2.21 3.0 3.00 4.0 4.00 5.0 5.00 VBAppCode

Excel: =RESULT(8+64) Causes #VALUE! with 1-by-1 Array

Article ID: Q84076

The information in this article applies to:

- Microsoft Excel for Windows, versions 3.0, 4.0, 5.0
- Microsoft Excel for the Macintosh, version 3.0
- Microsoft Excel for OS/2, version 3.0

SYMPTOMS

When you are using RESULT() within a function macro in Microsoft Excel, specifying a return value of 72 (8+64=reference and array) will cause the function macro to return #VALUE! when specifying a 1-by-1 return array (one cell).

Note: this also applies to version 5.0 when you use version 4.0 macrosheets. It does not apply to Visual Basic for Applications modules.

WORKAROUNDS

- In most cases, the DEREF() is not necessary. The problem will be corrected if you stop using it.

- Use =RESULT(75). This will specify result types 1, 2, 8 and 64 (number, text, reference and array).

Steps to Reproduce Problem

Type the following into a macro sheet, and define cell A1 as a function macro:

```
A1: =RESULT(72)
A2: {=RETURN(DEREF(A1))}
```

Note: the curly brackets are not typed into cell A2, the contents of A2 are entered as an array formula. In Microsoft Excel for Windows, use CTRL+SHIFT+ENTER. In Microsoft Excel for the Macintosh, use COMMAND+RETURN.

In a worksheet, paste the function into a cell. Since the array only has one element, it is not necessary to enter the formula as an array formula. The result of the function will be #VALUE!. This does not occur on arrays larger than 1-by-1.

REFERENCES

"Function Reference" for Windows, version 4.0, page 361
"Function Reference" for the Macintosh, version 3.0, pages 198-199
"Function Reference" for Windows, version 3.0, pages 198-199

Additional reference words: 3.00 3.0 4.00 4.0 5.00 5.0 mpf

Excel: ASSIGN.TO.TOOL and ENABLE.TOOL Cause Macro Error

Article ID: Q87540

The information in this article applies to:

- Microsoft Excel for Windows, version 4.0, 5.0
- Microsoft Excel for the Macintosh, version 4.0

SUMMARY

Microsoft Excel assigns a position to each tool and gap on a toolbar. When the ENABLE.TOOL function or the ASSIGN.TO.TOOL function tries to utilize a position number corresponding to a gap, a macro error results. In version 5.0, this information applies to macrosheets only, not Visual Basic for Applications modules.

MORE INFORMATION

The ENABLE.TOOL function allows you to selectively enable and disable tools on toolbars within Microsoft Excel. Disabled tools have the same appearance as enabled tools, but they do not respond to your actions until re-enabled.

ASSIGN.TO.TOOL function is equivalent to choosing the Assign To Tool command from the Macro menu or from the Tools shortcut menu. It assigns a macro to be run when a tool is clicked with the mouse. For detailed information about the Assign To Tool command, see online Help.

Attempting to use these functions on a position in a toolbar that is occupied by a gap, results in the following error message, where Macro Sheet Name refers to the macrosheet where the function is located and Cell Reference specifies the cell in the macro where the command is located:

```
Macro error at cell: <Macro Sheet Name>!<Cell Reference>
```

To avoid this error, use the ENABLE.TOOL function and the ASSIGN.TO.TOOL function in conjunction with the GET.TOOL function. The following line checks to see if the position number (5) to be enabled is a tool or a gap. If it is a tool, it enables the tool (or assigns the macro). Otherwise, it does nothing:

```
=IF(GET.TOOL(1,"SampleToolbar",5)=0,,ENABLE.TOOL("SampleToolbar",5,TRUE))
```

REFERENCES

"Microsoft Excel Function Reference," version 4.0, pages 28, 131, 206

Additional reference words: 4.0 4.00 4.0a 4.00a 5.0 5.00 buttons mpf

Third-Party Vendors of Excel-Related Products

Article ID: Q90508

The information in this article applies to:

- Microsoft Excel for Windows, versions 3.0, 4.0, 4.0a, 5.0
- Microsoft Excel for the Macintosh, version 4.0

SUMMARY

The following is a list of third-party vendors who provide additional tools, macros, templates, books, and training products for use with Microsoft Excel. Some of these vendors may offer add-in macros and templates for earlier versions of Microsoft Excel and for both Microsoft Excel for Windows and Microsoft Excel for the Macintosh.

For more information, please call the vendor listed.

The products mentioned here are manufactured by vendors independent of Microsoft; we make no warranty, implied or otherwise, regarding these products' performance or reliability.

MORE INFORMATION

These third-party vendors provide additional tools, macros, and templates for Microsoft Excel:

Tools, Templates, and Macros

Real estate templates and macros
A la Mode, Inc.
(801) 268-6911
4547 S. 700 E.
Suite 200
Salt Lake City, UT 84107.

Financial and investment planning
Apropos Software, Inc.
64 Hillview Avenue
Los Altos, CA 94022

DIF to SYLK conversions
Template Systems, Inc.
(617) 533-2203
7 Industrial Park Rd.
Medway, MA 02053.

Personal income tax templates and schedules
James Associates
(303) 484-5296

1525 East County Road 58
Fort Collins, CO 80524.

Real estate and personal finance templates
Real Data, Inc.
(203) 255-2732
P.O. Box 691
Southport, CT 06490.

Individual Software Inc.
5870 Stoneridge Drive, #1
Pleasanton, CA 94588
(415) 734-6767
Customer Support: (800) 822-3522

The following venders also provide custom macros for Microsoft Excel:

Add-ins, macros, templates, and a few engineering products.
Heizer Software
1941 Oak Park Blvd., Suite 30
P.O. Box 232019
Pleasant Hill, CA 94523
(510) 943-7667 Main Office
(800) 888-7667 Order Line
(510) 943-6882 Fax

The Baarnes Utilities for MS Excel, Custom add-ins
and Expert Corporate Development Using Microsoft Excel
Baarns Consulting Group, Inc.
(800)377-XCEL (9235)
12807 Borden Avenue
Sylmar, CA 91342.

@Risk for Excel: A risk analysis tool.
Palisade Corporation
31 Decker Road
Newfield, NY 14867 USA
(607) 277-8000

Evolver: An all-purpose problem solver.
Axcellis
4668 Eastern Ave. N.
Seattle, WA 98103-6932 USA
(206) 632-0885 Customer line
(206) 632-3681 Fax

Statistics: A set of complex statistical analysis tools.
Spreadware
82521 Market Ave. #110
Indio, CA 92201 USA
(619) 347-2365

Crystal Ball: What-if analysis tools.
Decisioneering Inc.
1380 Lawrence Street, Suite 610
Denver, CO 80204 USA
(303) 292-2291

Braincel: Tools for forecasting using neural network
technology.
Promised Land Tech. Inc.
900 Chapel St. Suite 300
New Haven, CT 06510
(203) 562-7335

Training Products and Books

These vendors provide additional training products and books for Microsoft Excel:

Personal Training Systems
Training products covering Excel fundamentals, business graphing, databases, linking spreadsheets and macros
(408) 559-8635
P.O. Box 54240
San Jose, CA 95154.

The Cobb Group
Monthly journal and books for Microsoft Excel users
(800) 223-8720 or (502) 425-7756
P.O. Box 24480
Louisville, KY 40224.

"Microsoft Excel Visual Basic for Applications Reference"
Microsoft Corporation
$24.95 ($33.95 Canada)
ISBN 1-55615-624-3

"Microsoft Excel Visual Basic for Applications Step by Step"
Reed Jacobson,
$29.95 ($39.95 Canada)
ISBN is 1-55615-589-1

Microsoft Press books can be found in bookstores everywhere, through CompuServe electronic mail (type GO MSP), or you can order direct by calling (800) MSPRESS. In Canada, call (416) 293-8464, extension 340.

If you are outside the United States, contact the Microsoft subsidiary for your area. To locate your subsidiary, call Microsoft International Customer Service at (206) 936-8661.

Excel 5 Courseware Developer's Kit (CDK)
Microsoft Corporation
$895 US ($1195 Canada)
(Win) ISBN 1-55615-719-3

Includes license agreement (grants permission to customize the student book, the student practice files, and the instructor outline files for reproduction), eight disks (student practice files and student book on disk), Instructor's guide, certificate of completion master copy, hardcopy overhead and handout masters, and a template for creating your own lessons

Microsoft Excel 5 Software Development Kit
$49.95 ($69.95 Canada)
ISBN 1-55615-632-4
(Available in April)

Additional reference words: 2.00 2.01 2.10 3.00 4.00 4.00a 5.00 macro developer phone writer books 3rdparty 3pty telephone isv phoneref third party macros programs

Excel: Finding Open Applications that are DDE Capable

Article ID: Q102362

The information in this article applies to:

- Microsoft Excel for Windows, versions 2.x, 3.0, 4.0, 4.0a, 5.0
- Microsoft Excel for the Macintosh, versions 3.0, 4.0

SUMMARY

In Microsoft Excel, you can find which of your currently open applications are capable of dynamic data exchange (DDE) by using the following macro function:

```
=INITIATE("","")
```

This function will display a DDE dialog box that will list all of the open applications and topics to which Microsoft Excel can open a DDE channel. You can then use other DDE commands such as EXECUTE() and SEND.KEYS() to control the other application.

In Microsoft Excel version 5.0, you can use the following Visual Basic for Applications procedure to determine which of your open applications are capable of DDE:

```
SUB Initiate_DDE ()
    ChanNum = Application.DDEInitiate("", "")
End Sub
```

NOTE: Microsoft Excel for the Macintosh requires system software version 7.0 or later for this and other DDE functions to work.

REFERENCES

"Function Reference," version 4.0, page 240

Additional reference words: 2.0 2.00 2.01 2.1 2.10 3.0 3.00 4.0 4.00 4.0a 4.00a 5.0 5.00 mpf VBAppcode

XL5: Can't Link Edit Box to Worksheet Cell from Control Tab

Article ID: Q105110

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, although you can establish a link between some types of dialog box controls and a worksheet cell, edit boxes do not have this capability. To allow the value of an edit box to appear in a cell, you must create a Visual Basic procedure.

MORE INFORMATION

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following Visual Basic procedure creates the equivalent of a cell link between an edit box and cell A1 on a worksheet. The subroutine should be assigned to the edit box by choosing Assign Macro from the Tools menu.

```
'-----------------------------------------------------------------------
Sub EditBox1_Change()
   'Enter the following two lines of code as one single line:
   Worksheets("Sheet1").Cells(1, 1).Value =
      ActiveDialog.EditBoxes("Edit Box 1").Text
End Sub
'-----------------------------------------------------------------------
```

In this example the name of the edit box is "Edit Box 1," the name of the worksheet "Sheet1". This example assumes that all dialog sheets, Visual Basic modules, and worksheets are located in the same workbook.

This sample code runs whenever the content of Edit Box 1 is changed. When this happens, the code changes the value of cell A1 to be equal

to the text property of Edit Box 1.

REFERENCES
==========

"Visual Basic User's Guide," version 5.0, Chapter 11, "Linking Controls to Worksheets," and "Assigning Code to Controls and Dialog Boxes"

Additional reference words: 5.00 spin scroll button list

XL5: Cannot Use Name Box to Define 3-D Name

Article ID: Q105770

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

Page 144 of the "User's Guide," version 5.0, states that you can define a name with a 3-D reference. While this information is correct, note that you cannot use the name box (on the Formula bar) to define a 3-D name.

WORKAROUND

To define a 3-D name, specify the 3-D reference in the Refers To box in the Define Name dialog box (from the Insert menu, choose Name, and choose Define).

MORE INFORMATION

To manually create a 3-D name

The following example assumes you have a workbook that contains five worksheets: Sheet1, Sheet2, Sheet3, Sheet4, and Sheet5, appearing in that order.

To create a 3-D defined name "TestRange" that refers to the range $A$1:$D$10 on sheets Sheet1 through Sheet4:

1. With a single worksheet tab selected, choose Define from the Insert menu, and choose Name.

2. In the Names In Workbook box, type "TestRange" (without the quotation marks).

3. In the Refers To box, type "=Sheet1:Sheet4!$A$1:$D$10" (without the quotation marks).

4. Choose OK.

To create a 3-D name with a Visual Basic procedure

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied,

including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided "as is" and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line-continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

To create the 3-D name "TestRange" in Visual Basic, you could use a procedure similar to the following:

```
' Enter the following two lines of code as one single line and remove the
' underline character:
ActiveWorkbook.Names.Add Name:="TestRange", _
   RefersToR1C1:="=Sheet1:Sheet4!R1C1:R10C4"
```

REFERENCES
==========

"User's Guide," version 5.0, pages 132, 136, 144

Additional reference words: 5.00 define.name

XL5: Can't Use PrintOut When Custom Dialog Box Is Visible

Article ID: Q105876

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

Because of the way in which Visual Basic code and custom dialog boxes interact in Microsoft Excel version 5.0, it is not possible to use the PrintOut method, which allows you to print a document, or the PrintPreview method, which allows you to print preview a document, while a custom dialog box is visible on the screen. Instead, you must use a method called "tunneling" to remove the dialog box, perform your print action, and redisplay the dialog box. The following information discusses methods that you can use to accomplish this task.

MORE INFORMATION

In Microsoft Excel version 5.0, you can use the PrintOut method to send a document to your printer. For example, to print a worksheet called Sheet1, you would use the command:

```
Worksheets("Sheet1").PrintOut
```

When a custom dialog box is visible on the screen, if your macro uses the PrintOut method, you may receive the error message:

```
PrintOut method of Worksheet class failed
```

-or-

```
Runtime error 1004
Printout Method of sheets class failed
```

To use the PrintOut method, you must first hide or dismiss all custom dialog boxes.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from

one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

This Visual Basic code example uses tunneling to display a dialog box, hide the dialog box before it prints a worksheet, and then redisplay the dialog box when the print operation is complete.

This example assumes that you have a dialog sheet (Dialog1) and a worksheet (Sheet1) that are located in the same workbook. The dialog sheet contains two buttons: DoneButton and PrintButton.

Before executing the macro you need to assign the appropriate macros to the DoneButton and the PrintButton. To do this, follow these steps:

1. Activate the dialog sheet.
2. Select the DoneButton.
3. From the Tools menu choose Assign Macro.
4. Select the DoneButton_Click macro and choose OK.

To assign the PrintButton_Click macro to the PrintButton, repeat steps 1 through 4 and substitute PrintButton for DoneButton and PrintButton_Click for DoneButton_Click.

To run the example, position the cursor in the line that reads "Sub MainMacro()" and either press the F5 key or choose Start from the Run menu.

```
'--------------------------------------------------------------------------
Option Explicit
Public DoneFlag As Integer, PrintFlag As Integer

Sub MainMacro()
   PrintFlag = 0                          'initialize PrintFlag
   DoneFlag = 0                           'initialize DoneFlag

   'While the DoneFlag does not equal 1 (which will only occur if the
   'DoneButton is clicked), continue to loop through the Subroutine.

   Do
      If PrintFlag = 1 Then               'if the PrintFlag is set, then
         Worksheets("Sheet1").PrintOut    'print Sheet1 and
         PrintFlag = 0                    'reset the PrintFlag
      End If
      DialogSheets("Dialog1").Show        'display the dialog box
   Loop Until DoneFlag = 1                'loop until DoneButton clicked
End Sub

Sub DoneButton_Click()
   DoneFlag = 1                           'set the DoneFlag
   DialogSheets("Dialog1").Hide           'hide the dialog box
End Sub

Sub PrintButton_Click()
   DoneFlag = 0                           'ensure DoneFlag set to 0
```

```
   PrintFlag = 1                          'set the PrintFlag
   DialogSheets("Dialog1").Hide           'hide the dialog box
End Sub
'--------------------------------------------------------------------------
```

When either of the DoneButton or PrintButton buttons are activated, the appropriate subroutines (DoneButton_Click or PrintButton_Click) are run: within each subroutine, the Dialog1 dialog box is hidden and a flag (DoneFlag or PrintFlag) is set to 1. The MainMacro subroutine then resumes and loops back: if PrintFlag equals 1, the macro prints the worksheet and redisplays the dialog box; if DoneFlag equals 1, the macro exits the loop and ends the macro.

In this way, the PrintOut method is only executed if the Dialog1 dialog box is not visible on the screen, and the dialog box will be redisplayed until you exit the loop by activating the DoneButton.

Additional reference words: 5.00

XL5: RemoveItem and RemoveAllItems Methods May Not Work

Article ID: Q105877

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, the RemoveItem and RemoveAllItems methods cannot be used to remove items from a list box if the list box is linked to a range on a worksheet or a Microsoft Excel 4.0 macro sheet.

This is by design in Microsoft Excel version 5.0.

MORE INFORMATION

In Microsoft Excel 5.0, when you use a Visual Basic module to create a list box, there are two ways you can add items to the list:

- Use the AddItem method to add items to the list box.

  -or-

- Do the following to link a worksheet or a Microsoft Excel 4.0 macro sheet range to the list box by setting an input range for the list box.

  1. In the dialog sheet, select the list box.

  2. From the Format menu, choose Object.

  3. Select the Control tab.

  4. In the Input Range edit box, enter the range where your list items are stored (for example, Sheet1!$A$1:$A$10).

  5. Choose OK to accept the change.

If your list box is linked to a worksheet or a Microsoft Excel 4.0 macro sheet range, you cannot use the RemoveItem or RemoveAllItems methods to remove items from the list. Attempting to do so will result in the following error message:

    RemoveItem method of ListBox class failed

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for

illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

This Visual Basic code example displays a dialog box. When you select one button, it hides the dialog box, removes the first item from the list in the dialog box, and redisplays the dialog box. Another button ends the procedure.

You need to use this example ONLY if your list box is linked to cells on a worksheet or a Microsoft Excel 4.0 macro sheet: lists created using AddItem can use the RemoveItem and RemoveAllItems methods to remove items from a list.

This example assumes that you have a dialog sheet (Dialog1) and a worksheet (Sheet1) both contained in the same workbook. The dialog sheet contains two buttons, DoneButton and RemoveButton, and a list box, List Box 1. The worksheet contains a list of items in cells $A$1:$A$10. The list box has its Input Range set to Sheet1!$A$1:$A$10.

To run the example, position the cursor in line the line which reads "Sub MainMacro()" and either press the F5 key or choose Start from the Run menu.

```
'-------------------------------------------------------------
Option Explicit
Public DoneFlag As Integer

Sub MainMacro()
   DoneFlag = 0                          'initialize DoneFlag

'  While the DoneFlag does not equal 1 (which will only occur if the
'  DoneButton is clicked), continue to loop through the Subroutine.

   Do
      DialogSheets("Dialog1").Show       'display the dialog box
   Loop Until DoneFlag = 1               'loop until DoneButton clicked
End Sub

Sub RemoveButton_Click
   DoneFlag = 0                          'ensure DoneFlag set to 0
   DialogSheets("Dialog1").Hide          'hide the dialog box

'  The following line deletes cell A1 ["Cells(1, 1)"] from worksheet
```

```
'  Sheet1.

   Worksheets("Sheet1").Cells(1, 1).Delete
End Sub

Sub DoneButton_Click
   DoneFlag = 1                          'set the DoneFlag
End Sub
'-----------------------------------------------------------
```

When either of the DoneButton or RemoveButton buttons are activated, the appropriate subroutines (DoneButton_Click or RemoveButton_Click) are run: in the RemoveButton subroutine, an item is deleted from the list, which is automatically updated, and in the DoneButton subroutine, a flag which indicates that you want to end the macro is set.

The MainMacro subroutine then resumes and proceeds to either loop back upon itself if DoneFlag equals 0 (if the RemoveButton was activated) or exit the loop and end the macro (if the DoneButton was activated).

Note that this example uses the Delete method to eliminate a cell from the list. If you want to redefine the list without deleting the cell, you will need to use the ListFillRange property to determine the proper range to use for your list.

Note also that when you use this method to remove items from a list, you must hide and reshow the dialog box in order for the list to appear correctly on the screen. This is accomplished by using the DialogSheets("Dialog1").Hide command in the RemoveButton_Click subroutine.

Additional reference words: 5.00

XL5: Using SENDKEYS to Change Paper Source in Printer Setup

Article ID: Q105878

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

The following Microsoft Excel, version 5.0, Visual Basic macro will change the paper source on the Hewlett-Packard (HP) LaserJet 4/4M to lower tray.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Visual Basic Macro

```
Sub HP4_Paper_Source()

     Application.ActivePrinter = "HP LaserJet 4/4M on LPT1:"
     SendKeys "%(f)(p)(r)%(s)%(s){pgup}{down}{down}
     {down}{down}~~~"

End Sub
```

This macro sends key commands to the Printer Setup dialog box. The following is an explanation of the string used in the SendKeys function:

This command	Performs this action
%(f)(p)(r)	Chooses Printer Setup in the Print dialog
%(s)	Chooses Setup in the Printer Setup dialog
%(s)	Chooses Paper Source in the HP 4/4M Setup dialog
{pgup}	Moves Paper Source selection up
{down}	Moves Paper Source selection down

MORE INFORMATION

==========

In the Printer Setup dialog box, there is no accelerator key combination that will directly select a paper source such as lower tray. Instead, the arrow key should be used to select from the list of available sources. For example, on the HP LaserJet 4/4M there are six possible choices for the paper source:

- Auto Select
- MP Tray
- Paper Cassette
- Manual Feed
- Lower Cassette
- Envelope Feeder

Because the current paper source selection is not known before execution of the macro, use the {pgup} command to select the first item in the Paper Source list. This will guarantee that Auto Select is chosen. Four DOWN ARROW keystrokes will select the lower cassette.

NOTE: This macro may work for other printers or may require a different sequence of keystrokes depending on the printer driver.

REFERENCES
==========

"Visual Basic User's Guide," version 5.0, Chapter 6

Additional reference words: 5.00 print

XL5: Visual Basic Interprets mm:ss as Month Not Minutes

Article ID: Q105954

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

When a date format is evaluated in a Visual Basic module, it may be changed from months to minutes.

The following macro command will return months followed by seconds:

```
MsgBox Format(Now, "mm:ss")
```

While Microsoft Excel uses the format mm:ss for minutes and seconds, Visual Basic uses the format nn:ss for minutes and seconds.

This issue will not be seen by using the format "h:mm:ss". Visual Basic will interpret this correctly as hours:minutes:seconds, as this is a less ambiguous format.

MORE INFORMATION

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Example

The following Visual Basic macro creates a message box with three numbers that display the system date and time. The first number is formatted as month and seconds, while the second two numbers display minutes and seconds.

```
Sub My_string()
```

```
   Dim Msg, NL
   NL = Chr(10)
   Msg = Format(Now, "mm:ss") & NL            'Returns "Month & Seconds"
   Msg = Msg & Format(Now, "nn:ss") & NL      'Returns "Minutes & Seconds"
   Msg = Msg & Application.Text(Now, "mm:ss") 'Returns "Minutes & Seconds"
   MsgBox Msg
End Sub
```

REFERENCES
==========

"Visual Basic User's Guide," version 5.0, pages 111-113

For more information about User Defined Date/Time Formats, choose the Search button in Visual Basic Reference Help and type:

    Formatting Times

Additional reference words: 5.00

XL5: Macro to Restore Tab Split Box to Default Position

Article ID: Q106009

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, it is possible to adjust the space allocated for displaying sheet tabs and the horizontal scroll bar by moving the tab split box. The tab split box is located between the sheet tabs and the horizontal scroll bar.

This article contains some sample Visual Basic macros and a sample Microsoft Excel version 4.0 macro to restore the tab split box to its default.

MORE INFORMATION

Some video drivers, particularly very high resolution (1024x768, 1280x1024, and so on) drivers, may exhibit "jumpiness" when you switch from one insertion point mode to another (for example when you move the insertion point from the worksheet, where it usually appears as an arrow or a plus sign (+), to the tab split box or the split box, where the insertion point appears as a two-way split symbol).

If your screen is "jumpy", it may be difficult to determine whether your insertion point is positioned over the split box or the tab split box when these boxes are right next to each other. You may need to use a macro to restore the tab split box to its default position.

The following macros each restore the tab split box to its default position.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Visual Basic Procedure

```
Sub CenterTabSplitBox
    ActiveWindow.TabRatio = 0.6
End Sub
```

Microsoft provides macro examples for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This macro is provided 'as is' and Microsoft does not guarantee that the following code can be used in all situations. Microsoft does not support modifications of the code to suit customer requirements for a particular purpose.

Microsoft Excel Version 4.0 Macro

```
A1: CenterTabSplitBox
A2: =WORKBOOK.TAB.SPLIT(0.6)
A3: =RETURN()
```

To use this macro:

1. On the macro sheet, select cell A1.

2. From the Insert menu, choose Name, and choose Define.

3. In the Define Name dialog box, select the Command option.

4. Choose OK to accept the change.

When you want to run either of the macros, choose Macro from the Tools menu. Select the name of the macro from the list of macros and choose Run to run the macro. Your tab split box will be restored to the default position.

REFERENCES
==========

"User's Guide 1," Chapter 7

Additional reference words: 5.00

XL5: Excel 4.0 Menus Option Setting Not Saved

Article ID: Q106357

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

When you switch to Microsoft Excel Version 4.0 menus, the setting is not retained when you exit Microsoft Excel.

CAUSE

In Microsoft Excel, you can switch to Microsoft Excel version 4.0 menus by choosing Options from the Tools menu and selecting the Microsoft Excel Version 4.0 Menus option on the General tab. However, when you exit Microsoft Excel, the Microsoft Excel Version 4.0 Menus setting is not retained.

This behavior is by design.

The Microsoft Excel Version 4.0 Menus option is provided to help you adjust to the 5.0 menus. However, when this option is selected, you do not have full access to all the functionality of Microsoft Excel version 5.0.

WORKAROUND

The following sampe macro automatically switches to Microsoft Excel Version 4.0 menus each time you start Microsoft Excel.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Example of Procedure

To create the procedure, do the following:

1. In a new workbook, choose Macro from the Insert menu and choose Module.

2. In the new Module, enter the following code:

```
' Macro name must be auto_open() to run automatically
Sub auto_open()
   ' Switch to Microsoft Excel version 4.0 menus
   Application.DisplayExcel4Menus = True
   ' Switch back to Sheet1 of default workbook
   Sheets("Sheet1").Select
End Sub
```

4. From the File menu, choose Save.

5. In the File Name box, enter a name for the workbook.

6. In the Directories box, select the Excel Startup directory (XLSTART) and choose OK.

7. From the Window menu, choose Hide.

8. From the File menu, choose Exit. When asked to save the workbook, choose Yes.

The next time you start Microsoft Excel, this workbook will be opened and the Microsoft Excel version 4.0 menus will be displayed.

REFERENCES
==========

"User's Guide," version 5.0, page 642

"Visual Basic User's Guide," version 5.0, page 266

For more information about Switching To The Version 4.0 Menus, choose the Search button in Help and type:

    Microsoft Excel 4.0

Additional reference words: 5.00

XL5: Unlocked Cells Not Underlined With Protection Enabled

Article ID: Q106390

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel versions earlier than 5.0, while worksheet protection is enabled and gridlines are turned off, unlocked cells appear underlined. This underline identifies the cells that you can edit.

Although this feature is not available in Microsoft Excel version 5.0, you can create a macro that will allow you to simulate this behavior.

MORE INFORMATION

The following Visual Basic macro places a bottom border on all unlocked cells in a worksheet. The macro also removes any bottom border from locked cells.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

To create the macro:

1. From the Insert menu, choose Macro, and then choose Module.

2. In the module, type the following macro code:

```
Sub Format_Unlocked_Cells()
   'Declare procedure-level variables
   Dim x As Range, lastcell As Range
   'Prevent screen redraw to speed up the macro
   Application.ScreenUpdating = False
```

```
         'Unprotect the worksheet to allow editing
         ActiveSheet.Unprotect ("my_password")
         'Set lastcell to point to the last cell on the sheet
         Set lastcell = Selection.SpecialCells(xlLastCell)

    'Place bottom borders on unlocked cells and remove any bottom
    'borders from locked cells
         For Each x In Range("A1", lastcell)
             With x.Borders(xlBottom)
                 If x.Locked = False Then
                     .Weight = xlHairline
                     .ColorIndex = xlAutomatic
                  Else
                     .LineStyle = xlNone
                 End If
            End With
         Next x

          ActiveSheet.Protect ("my_password") 'Re-apply worksheet
protection.
      End Sub
```

NOTE: If you want to format a specific cell range, replace Range("A1", lastcell) with the range you want to format, for example, Range("A1:G100").

To use the macro:

1. Activate the worksheet that you want to format.

2. From the Tools menu, choose Macro.

3. From the list of macros, select the Format_Unlocked_Cells macro and choose Run.

REFERENCES
==========

"Visual Basic User's Guide," version 5.0, Chapter 5

Additional reference words: 5.00

XL5: Worksheet May Be Activated When Method Is Applied

Article ID: Q106463

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

When you apply methods or change properties of a worksheet, the target worksheet may need to be activated before the macro can perform the requested action. If the worksheet is not already active, a brief flash will be noticeable on the screen when the sheet is activated.

MORE INFORMATION

Some methods and properties that are applied to worksheets will cause the worksheet to be briefly activated.

Steps to Reproduce Behavior

1. From the File menu, choose New, and then choose Workbook.

2. From the Insert menu, choose Macro, and then choose Module.

3. In the Visual Basic module, type the following text:

```
Sub Protect_Sheet1()
   ActiveWorkbook.Sheets("Sheet1").Protect
End Sub
```

4. With the Visual Basic module active, choose Macro from the Tools menu. Select Protect_Sheet1 from the list and choose Run.

The following procedures will also briefly activate the worksheet:

```
ActiveWorkbook.Sheets("Sheet1").Unprotect
ActiveWorkbook.Sheets("Sheet1").Visible = True
```

Some methods or properties, such as the one in the following example, may activate the sheet and cause it to remain the active sheet:

```
ActiveWorkbook.Sheets("Sheet1").Move
```

Additional reference words: 5.00

XL5: Using Visual Basic to Return Screen Elements

Article ID: Q106719

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, when you develop a custom application that will run on computers with different screen resolutions, it may be useful to determine the amount of space that different workspace elements such as the menu bar, formula bar, status bar and toolbars will use. After you determine the amount of space these element use, you can then determine how and where to position documents and which elements you want to be displayed.

MORE INFORMATION

With the Microsoft Visual Basic Programming System, Applications Edition, you can write code that will retrieve workspace properties. For example the APPLICATION.USABLEWIDTH and APPLICATION.USABLEHEIGHT command allow you to return the width and height (in points) of the workspace.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Sample Macro Code

The following example determines the height of the status bar:

```
Sub statusbarheight()
   'Declares the Variables as Integers
   Dim Nostatusbar as  Integer
   Dim Withstatusbar as  Integer
   Dim StatusHeight as  Integer
```

```
      Application.DisplayStatusBar = False          'Turns status off
      nostatusbar = Application.UsableHeight     'finds height without
                                                 'status bar
      Application.DisplayStatusBar = True          'Turns the status bar on
      withstatusbar = Application.UsableHeight     'Finds height with
                                                   'status bar on
      statusheight = nostatusbar - withstatusbar   'Determines the
                                                   'difference
      MsgBox ("The height of the status bar in points is " & statusheight)
End Sub
```

REFERENCES
==========

"Visual Basic User's Guide," version 5.0, Chapter 5

Additional reference words: 5.00 Get.workspace(13) Get.Workspace(14) howto

XL5: Alert Function Does Not Beep in Excel Version 5.0

Article ID: Q107137

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel versions earlier than 5.0, the ALERT() macro function automatically causes the computer to beep. In Microsoft Excel version 5.0, the computer does not automatically beep when you use the ALERT() macro function or the Visual Basic MsgBox function.

MORE INFORMATION

Microsoft Excel version 5.0 does not automatically beep when an ALERT() box or MsgBox is displayed. This allows you to determine what sound (if any) you want to play when a message box is displayed.

If you do want to hear a beep sound, you must use the BEEP() function or the Visual Basic Beep statement before you issue the ALERT() function or the MsgBox command. The following examples illustrate how to do this.

Visual Basic Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

In Microsoft Excel 5.0, open a new Visual Basic module and type the following code:

```
Sub Test()
   Beep                    'Causes the computer to beep
```

```
   MsgBox "Hello", 64      'Displays an information message box
End Sub
```

Microsoft Excel 4.0 Macro Example

In Microsoft Excel 5.0, open a new Microsoft Excel 4.0 macro sheet and type the following macro:

```
   A1: =BEEP()
   A2: =ALERT("Hello")
   A3: =RETURN()
```

Explanation of Macro Code

A1: Causes the computer to beep.
A2: Displays an information alert box with the text: "Hello"
A3: Ends the macro.

Instead of using the =BEEP() function or the Beep procedure, you can use the SOUND.PLAY() function to play any sound before displaying the message box. For example, for a Visual Basic macro, use the following code:

```
   Application.ExecuteExcel4Macro "SOUND.PLAY(,""C:\WINDOWS\CHIMES.WAV"")"
```

For a Microsoft Excel version 4.0 macro, use the following code:

```
   =SOUND.PLAY(,"C:\WINDOWS\CHIMES.WAV")
```

Additional reference words: 5.00 dialog

XL5: Formatting Name of Months in All Capital Letters

Article ID: Q107139

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY
=======

In the built-in date format in Microsoft Excel, only the first letter of the month is uppercase. If you want to make the entire month appear in uppercase letters, you can automate this process with a Visual Basic macro, or, you can use the UPPER() function in a Microsoft Excel 4.0 macro.

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

Visual Basic Procedure

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

This procedure takes a date in any number format, changes it to a text string in 'mmm' format, and then makes all the letters uppercase. For example, if a cell contained a date of January 1, 1993, in the format Jan-93, it would be converted to JAN.

CAUTION: This procedure will delete the value representing the date.

To create and use this procedure, do the following:

1. Enter the following code in a module sheet

```
Sub Upper()
   Dim Cell As Object     'Declare the Cell variable.
   For Each Cell In Selection
      'If the cell is blank or a text string, then
      'skip to the next cell in the selection.
```

```
     If Cell.Value <> "" And Val(Cell.Value) > 0 Then
        'Format the cell as text in a 'mmm' number format,
        'and change it to uppercase.
        Cell.Value = UCase(Format(Cell.Value, "mmm"))
        End If
    Next
End Sub
```

2. To run the above code, select any range of cells, choose Macro from the Tools menu, select the Upper macro, and choose Run.

Microsoft Excel Version 4.0 Macro

You can use the UPPER() function to create a macro that will change the letters in a three-letter month abbreviation to uppercase, as in the following example

```
   =UPPER(TEXT(cell_ref,"mmm"))
```

where cell_ref is the cell that contains the date.

Additional reference words: 5.00 mpf

XL5: WORKBOOK.ACTIVATE() Returns Error If Sheet_Name Omitted

Article ID: Q107385

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In Microsoft Excel version 5.0, if you do not specify the sheet_name argument when you use the WORKBOOK.ACTIVATE() command in a Microsoft Excel version 4.0 macro, the command will return a macro error.

CAUSE

In earlier versions of Microsoft Excel, you can omit the first argument of the WORKBOOK.ACTIVATE() command to activate the workbook contents screen. Because Microsoft Excel version 5.0 does not have a workbook contents screen, sheet_name is a required argument.

The workbook contents screen has been replaced with tabs located along the bottom of the Microsoft Excel application window. There is a tab for each worksheet, module, Microsoft Excel 4.0 macro, and chart in the workbook.

WORKAROUND

Avoid using a macro command to activate a workbook contents window. You can perform the desired action on a workbook in Microsoft Excel version 5.0 without this command.

For example, the following macro in Microsoft Excel version 4.0 activates the workbook contents window in the current workbook and protects it.

```
A1: Protect_Contents
A2: =WORKBOOK.ACTIVATE(,FALSE)
A3: =PROTECT.DOCUMENT(TRUE,FALSE,,FALSE)
A4: =RETURN()
```

If you run this same macro in Microsoft Excel version 5.0, a macro error occurs at cell A2 because the second argument in this function is not supported. To accomplish the same action in Microsoft Excel version 5.0 using a macro, use the following examples:

Visual Basic Macro

```
Sub Protect_Contents()
   ActiveWorkbook.Protect Structure:=True, Windows:=False
End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Microsoft Excel Version 4.0 Macro

```
A1: Protect_Contents
A2: =WORKBOOK.PROTECT(TRUE,FALSE)
A3: =RETURN()
```

Microsoft provides macro examples for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This macro is provided as is and Microsoft in no way guaranties that the following code can be used in all situations and will not support modifications of the code to suit specific customer requirements.

REFERENCES
==========

"Function Reference," version 4.0, page 460

Additional reference words: 5.00 4.00 4.00a

XL5: Finding Default Printer and Port Settings

Article ID: Q107621

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY
=======

Microsoft Excel version 5.0, does not have the built-in functionality to directly recognize default printer and port settings. To return this printer information, use the Microsoft Windows Dynamic Link Libraries (DLLs) to read the DEVICE line of the [WINDOWS] section of the WIN.INI file.

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

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following macro will return the default printer type and port.

```
'Enter the following Declare Function as a 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

Sub Default_Printer_Port()
Dim Result As String * 255
Dim Printer As String
Dim Port As String
   'Gets info from win.ini
    Call GetProfileString("Windows","Device","",result,254)
   'Gets first part of device line
    Printer = Left(result,InStr(result, ",")-1)
   'Gets last part of device line
    Port = Mid(result, InStr(InStr(result, ",") +1, result, ",") +1)
```

```
   'Displays Printer and port in successive message boxes
    MsgBox "The current default printer is " & Printer
    MsgBox "The current default port is " & Port
End Sub
```

Additional reference words: 5.00 call register

XL5: Cannot Add Sheet After Last Sheet in a Single Action

Article ID: Q107622

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, it is not possible to add a new sheet after the last sheet in the workbook in a single action. For example, if you have a workbook that contains Sheet1 and Sheet2 (in left-to-right sheet tab order), you cannot add a new sheet to the right of Sheet2.

This situation occurs regardless of whether you add the new sheet manually, use Visual Basic code, or use Microsoft Excel 4.0 macro code.

You can add a new sheet to a workbook and make the sheet become the last sheet in the workbook, but at least two separate actions are required to do this.

MORE INFORMATION

If your application requires that a new sheet be added or inserted after the last sheet in the workbook, use either of the following procedures:

- If you manually add the new sheet to the workbook (by choosing the appropriate item in the Insert menu, for example), you can drag the new sheet's tab to the right of all of the other sheet tabs. This makes the new sheet also become the last sheet in the workbook.

  -or-

- If you are using a Visual Basic subroutine to add the new sheet to the workbook, you can use the Move method to move the new sheet to the proper location. An example of how to do this is shown below.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a

comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following Visual Basic code example inserts a new sheet into a workbook and makes the new sheet become the last sheet in the workbook by moving it to the right of the rightmost sheet.

This example assumes you have a workbook that contains at least one Visual Basic module and any number of other sheets.

To run the example, position the cursor in the line that reads "Sub AddNewSheet()" and either press the F5 key or choose Start from the Run menu.

```
'--------------------------------------------------------------------
Sub AddNewSheet()
   Application.ScreenUpdating = False    'prevents screen refreshing
   'This line adds a new worksheet to the workbook. Alternatively, use:
   '
   'DialogSheets.Add                    to add a dialog sheet
   'Charts.Add                          to add a chart sheet
   'Modules.Add                         to add a Visual Basic module
   'Sheets.Add Type:=xlExcel4MacroSheet to add a MS Excel 4.0 Macro sheet
   Worksheets.Add
   'This line makes the new sheet (which is also the active sheet) the
   'last sheet in the workbook.
   ActiveSheet.Move After:=Sheets(ActiveWorkbook.Sheets.Count)
   Application.ScreenUpdating = True     'reenables screen refreshing
End Sub
'--------------------------------------------------------------------
```

When you run the AddNewSheet subroutine, it will add a new sheet to the active workbook and make the new sheet become the last sheet in the workbook.

If you try to add a new sheet after the last sheet in the workbook, all in a single action, you will receive the error message:

   Add method of Sheet class failed

For example, the following command will return an error message

   Worksheets.Add After:=Sheets(ActiveWorkbook.Sheets.Count)

because it attempts to add a new sheet after the last sheet in the workbook.

Additional reference words: 5.00

XL5: Error Unhiding Multiple Sheets in Visual Basic

Article ID: Q107623

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In the Microsoft Visual Basic Programming System, Applications Edition, it is not possible to unhide multiple sheets with a single command. If you try to change the Visible property for more than one sheet to True in a single command such as following

```
Sheets(Array("Sheet1", "Sheet2", "Sheet3")).Visible = True
```

you will receive the error:

```
Unable to set the Visible property of the Sheets class
```


MORE INFORMATION

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following code examples assume you have a workbook that contains three worksheets (Sheet1, Sheet2, and Sheet3) and a single Visual Basic module (Module1).

If you run the HideSheets subroutine, the sheet tabs for Sheet1, Sheet2, and Sheet3 will disappear from the screen as the sheets are hidden.

To unhide the sheets, you can use the UnhideSheets subroutine. This subroutine loops through an array to unhide the sheets.

```
'-----------------------------------------------------------------
```

```
Option Explicit

Sub HideSheets()
   'This line hides the sheets listed in the array
   'by setting the Visible property to False.
    Sheets(Array("Sheet1", "Sheet2", "Sheet3")).Visible = False
End Sub

Sub UnhideSheets()
   'Dimension some variables.
   Dim Collection As Variant, Item As Variant
   'This line creates an array of sheets, called "Collection".
   Collection = Array("Sheet1", "Sheet2", "Sheet3")
   'Iterate through the loop once for each sheet in the array.
   For Each Item In Collection           'for each sheet in the array,
      Sheets(Item).Visible = True        'unhide the current sheet
   Next                                  'repeat the loop until all done
End Sub
'--------------------------------------------------------------------
```

By establishing an array (called "Collection" in this example) and using a For Each loop, it is possible to unhide the sheets one at a time. If you are unhiding many sheets, you may want to use the following line of code to disable your screen while the sheets are being unhidden:

```
   Application.ScreenUpdating = False
```

To re-enable the screen when you are done, use the following command:

```
   Application.ScreenUpdating = True
```

Additional reference words: 5.00

XL5: Disabling Microsoft Excel Control Menu Commands

Article ID: Q107689

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

You can use the Microsoft Windows Dynamic Link Libraries (DLLs) to disable the commands in the Microsoft Excel Control menu. For example, you can use these tools to delete the Minimize and Maximize buttons.

MORE INFORMATION

Microsoft Excel does not have the built-in functionality to modify the Control menu commands. However, you can use the Microsoft Windows Declare functions to disable Control menu items.

The following sample Visual Basic procedure disables the entire Control menu in Microsoft Excel version 5.0. The macro disables the Control Menu for the current session of Microsoft Excel (when you restart Microsoft Excel the Control menu will be reset).

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Visual Basic Procedure to Disable Control Menu

```
Declare Function GetActiveWindow Lib "User" () As Integer

'Enter this Declare statement as a single line
Declare Function GetSystemMenu Lib "User" (ByVal hWnd As Integer,_
 ByVal_ bRevert As Integer) As Integer

'Enter this Declare statement as a single line
```

```
Declare Function DeleteMenu Lib "User" (ByVal hMenu As Integer,_
 ByVal_ nPosition As Integer, ByVal wFlags As Integer) As Integer

Sub Disable_Control()
   Dim X as Integer
   For X = 1 to 9
   'Deletes the first Menu command and loops until all commands are deleted
   Call DeleteMenu(GetSystemMenu(GetActiveWindow, False),0,1024)
   Next X
End Sub
```

The commands on the control menu are numbered starting a zero. On the default, Control Menu Restore is item 0, Move is item 1, Size is item 2 and so on. Even if items are deleted, the first item always starts at zero.

To delete individual items from the control menu without deleting the entire menu, you can specify the menu command to delete. For example the following two lines of code when used in place of the above For Next loop, will delete the Maximize (item 4) and Minimize (item 3) commands and disable the Maximize and Minimize buttons.

```
Call DeleteMenu(GetSystemMenu(GetActiveWindow, False),4,1024)
Call DeleteMenu(GetSystemMenu(GetActiveWindow, False),3,1024)
```

Additional reference words: 5.00 call register remove

XL5: Continue Macro Execution While Playing a .WAV File

Article ID: Q107690

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

The APPLICATION.SOUND.PLAY() or SOUNDNOTE.PLAY functions allow a macro to execute and play wave(.WAV) files. When you use these functions in a macro, Microsoft Excel must wait for the entire sound file to play before it can run the rest of the macro.

You can use the following

- The Declare Functions,

  -and-

- Microsoft Windows version 3.1(which includes MultiMedia Extensions),

  -and-

- A sound card (such as the Windows Sound System or MediaVision's Pro Audio Spectrum),

to play the sound file with a macro. When you play the sound file in this manner, the macro will not have to wait for the wave file to finish playing. As soon as MultiMedia Extensions takes over playing the sound file, Microsoft Excel continues to execute the rest of the macro.

MORE INFORMATION

When you play the .WAV file this way, the sound file continues to play while the macro finishes executing, and it appears as if Microsoft Excel is performing two tasks at once. To demonstrate this behavior, play a .WAV file describing the contents of a dialog box and then display the dialog box. The following sample macro can be used in Microsoft Excel, version 5.0, for this purpose.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is

doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

WARNING: The Declare Function statements listed in the following macro are very sensitive. If used incorrectly, these functions may cause a general protection (GP) fault in Windows or cause serious software problems.

```
'The following two lines should be entered as a single line on a 'module
sheet
Declare Function sndPlaySound Lib "MMSYSTEM.DLL" (ByVal lpszSoundName
As String,_ ByVal wFlags As Integer) As Integer

Sub main()
   'Calls the sndPlaySound function and passes it the name of
   'the sound file to play
   Call sndPlaySound("c:\bat\wave\close.wav", 0)
End Sub
```

You can replace c:\bat\wave\close.wav with the path to any valid .wav file. This macro assumes the that you are using Microsoft Windows 3.1 and have a compatible sound card installed. Software sound drivers such as SPEAK.EXE will not give the desired results (these drivers must play the entire sound file before allowing Microsoft Excel to continue execution of the macro).

For more complete information about the sndPlaySound function of the MMSYSTEM.DLL, refer to the "Multimedia Programmer's Reference." This manual is part of the Microsoft Windows operating system version 3.1, Software Development Kit (SDK).

Additional reference words: 5.00 5.0 blaster soundblaster wss creative labs

XL5: Client Not Updated with Multiple Instances of Excel

Article ID: Q107745

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

When multiple instances of Microsoft Excel are simultaneously serving as dynamic data exchange (DDE) clients to a single DDE server, only one instance of Microsoft Excel will be updated.

MORE INFORMATION

This is a problem with the current implementation of the DDE management library (DDEML). The problem will only occur when all of the following conditions are met:

- There are two or more instances of the same application serving as DDE clients.
- The applications all have objects with the same name (such as "Sheet1")
- The applications are all trying to use FETCH.ADVISE

Steps to Reproduce Problem

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

1. Start Microsoft Query.
2. Open two instances of Microsoft Excel and run the following Visual Basic macro from each instance:

   ```
   Sub dde()
   ```

```
    chanNum = DDEInitiate("msquery", "system")
    ' Use the path for your query file for the following:
    DDEExecute chanNum, "[open('c:\excel\query1.qry')]"
    nRows = DDERequest(chanNum, "NumRows")
    nCols = DDERequest(chanNum, "NumCols")
    DDEExecute chanNum, "[fetch.advise('Excel','sheet1','r1c1:r" & _
    nRows(LBound(nRows)) & "c" & nCols(LBound(nCols)) & "','all')]"

End Sub
```

NOTE: You will need to create a query file to run this macro if you don't already have one. Use the path for this file in line 2 of the macro.)

3. Check Sheet1 on both instances of Excel; they should display a copy of the query.

4. Switch to Query Tool and from the Records menu, choose Allow Editing.

5. Change some of the records in the query.

Check Sheet1 on both instances of Microsoft Excel; the records should have been updated on one but not the other.

Additional reference words: 5.00

XL5: Macro to Print a Group of Files Located in Same Directory

Article ID: Q107880

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

In Microsoft Excel, you can use the following Visual Basic macro to print a batch of files that are all located in the same directory. The macro requests a directory name and then prints all the files in that directory.

Sample Macro Code

```
Sub Batch_Print()
Dim Input_Dir, Print_File As String
    Input_Dir = InputBox("Input directory path containing the files
       to print")
    ' Defines Print_File equal to the first Microsoft Excel file found in
    ' the directory specified in the InputBox above.
    Print_File = Dir(Input_Dir & "\*.xl*")
      ' Loops through the directory specified in the above InputBox
      ' and opens each workbook in the directory, prints all sheets
      ' in the workbook and closes the workbook. Continues until are
      ' all files are printed.
      Do While Len(Print_File) > 0
          Workbooks.Open Filename:=Input_Dir & "\" & Print_File
          ActiveWorkbook.PrintOut Copies:=1
          ActiveWorkbook.Close
          Print_File = Dir()
      Loop
End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

REFERENCES
==========

"Visual Basic Reference," version 5.0, pages 143-145

Additional reference words: 5.00

XL5: Determining Which DLLs Are Registered

Article ID: Q108002

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY
=======

In Microsoft Excel version 5.0, to determine which DLLs are registered, you can do either of the following:

- Choose the About Microsoft Excel command from the Help menu to display a list of all DLLs currently registered on your system.

  NOTE: Microsoft System Info is a Setup option. If it is not available on your system, rerun Setup, choose the Complete/Custom button, select the Tools option, and select the System Information Checking option.

  -or-

- Use a Visual Basic statement to return a list of all the DLLs and XLLs that provide functions registered in Microsoft Excel.

To use Microsoft System Info to display a list of registered DLLs
---

1. From the Help menu, choose About Microsoft Excel.

2. Choose the System Info button.

3. From the Choose a Category list, select System DLLs.

The Microsoft System Info dialog box displays a list of all registered DLLs, indicating the DLL filename, version, date, size in bytes, and displays the word "Yes" next to each DLL that is currently in memory.

To use a Visual Basic procedure

You can use the RegisteredFunctions property to return an array of every Microsoft Excel function that is provided by DLLs and other code resources, along with the name of the associated file.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note

also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide.

The following example code creates a three-column array of all registered functions in Microsoft Excel, where column 1 displays the name of the DLL or code resource; column 2 displays the name of the procedure in the DLL or code resource; and column 3 displays strings specifying the data types of the return values, and the number and data types of the arguments:

```
Sub GenerateDLLList()
   theArray = Application.RegisteredFunctions
      If IsNull(theArray) Then
         MsgBox "No registered functions"
      Else
      For i = 1 To UBound(theArray)
         For j = 1 To 3
            Cells(i, j).Formula = theArray(i, j)
         Next j
      Next i
   End If
End Sub
```

If you need to generate a list of all DLLs on your system, not just the ones used by Microsoft Excel, you can use the following code to export the System Info information (see above) to a text file named MSINFO.TXT, saved to your WINDOWS directory:

```
   Application.SendKeys "%h"
   Application.SendKeys "a"
   Application.SendKeys "%s"
   Application.SendKeys "%s"
   Application.SendKeys "{ESC}"
   Application.SendKeys "{ESC}"
   Application.SendKeys "{ESC}"
```

After saving MSINFO.TXT, your macro can then open the file and access the DLL registration information contained in Column A, under the System DLLs heading.

Additional reference words: 5.00

XL5: Visual Basic Procedure to Display Screen Metrics

Article ID: Q108012

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

The following macro allows you to determine the metrics of your video display (that is, horizontal resolution, vertical resolution, number of colors) by using a macro in Microsoft Excel, version 5.0. This information may be helpful in creating macros that create variable-sized custom dialog boxes based on display size or take into account the effect of display resolution on printed output.

MORE INFORMATION

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

```
'Each Declare statement should be entered as a single line
Declare Function GetActiveWindow Lib "USER" () As Integer

Declare Function GetDeviceCaps Lib "GDI" (ByVal hDC As Integer, ByVal _
nIndex As Integer) As Integer

Declare Function GetDC Lib "USER" (ByVal hWnd As Integer) As Integer

Const Horz_res = 8
Const Vert_res = 10
Const Planes = 14
Const BitsPixel = 12

Sub Display_info()
   'Returns the handle of the active window
   Act_Win = GetActiveWindow()
```

```
   'Returns the handle to the Device Context
   Dev_Con = GetDC(Act_Win)
   'Displays The Horizontal Video resolution
   MsgBox "Horz Video Resolution = " & GetDeviceCaps(Dev_Con, Horz_res)
   'Displays The Vertical Video resolution
   MsgBox "Vert Video Resolution =" & GetDeviceCaps(Dev_Con, Vert_res)
   'Displays the number of Planes
   MsgBox "Planes = " & GetDeviceCaps(Dev_Con, Planes)
   'Displays the number of Bits per Pixels
   MsgBox "Bits per Pixel = " & GetDeviceCaps(Dev_Con, BitsPixel)
End Sub
```

The above macro displays 4 message boxes showing the following:

- Horizontal video resolution
- Vertical video resolution
- Number of Planes
- Bits per Pixel

Additional Reference Words: 5.00 5.0 high res mapping screen driver

OLE Automation: GetObject Function with Filename Opens File

Article ID: Q108074

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In a Microsoft Visual Basic application, if you use the GetObject function with a filename to activate a file in Microsoft Excel, you may receive the following error message in Microsoft Excel:

> Book1.xls is already open. Reopening will cause your changes to be discarded. Do you want to reopen?

CAUSE

When you use the GetObject function with a filename to activate a workbook in Microsoft Excel, the file is opened, whether or not the file is already open. If the file is already open, and contains changes you have not saved, the above error message is returned.

WORKAROUND

To activate the workbook BOOK1.XLS as an object linking and embedding (OLE) Automation object and avoid opening BOOK1.XLS if it is already open, do the following:

1. Activate the Microsoft Excel Application object.

2. Check for an open workbook called BOOK1.XLS.

3. Activate BOOK1.XLS if it is open.

   -or-

   Open BOOK1.XLS if it is not open.

The following sample code demonstrates the workaround described above. The code scans the open workbooks in Microsoft Excel and activates BOOK1.XLS if it is open.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an

apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

```
   'Dimension variable xl as object type
   Dim xl As Object
   'Dimension variable Workbook as object type
   Dim Workbook As Object
   'Activate Microsoft Excel and assign to variable xl
   Set xl = GetObject(, "Excel.application.5")
   'Set n to number of open workbooks
   n = xl.Workbooks.Count
      'Loop through each open workbook
      For c = 1 To n
         'Test to see if workbook is BOOK1.XLS
         'If true, activate BOOK1.XLS and exit loop
         'BOOK1.XLS in this line of code must be capitalized
         If xl.Workbooks(n).Name = "BOOK1.XLS" Then
            xl.Workbooks("BOOK1.XLS").Activate
            Exit For
         End If
      Next c
```

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

You can use the Visual Basic GetObject function to access an OLE automation object and assign the object to an object variable. You can then use the object variable to reference the OLE automation object in your Visual Basic code.

Steps to Reproduce Problem
---

1. In a new worksheet, from the File menu, choose Save.

2. In the File Name box, type "BOOK1.XLS" (without the quotation marks) and choose OK.

3. In BOOK1.XLS, type "1" (without the quotation marks) in cell A1.

4. In Microsoft Visual Basic for Windows, enter and run the following code.

```
   'Dimension variable xlsheet as object type
   Dim xlsheet As object
   'Activate BOOK1.XLS and assign to variable xlsheet. Enter these two
   'lines as a single line. Note: use the appropriate path for your file
   '(this example assumes BOOK1.XLS has been saved in C:\EXCEL5\FILES).
   Set xlsheet = _
      GetObject("c:\excel5\files\[book1.xls]sheet1","Excel.Sheet.5")
   'Insert 10 in cell A2
```

```
xlsheet.Cells(2, 1).value = 10
```

The following error message appears in Microsoft Excel:

```
Book1.xls is already open. Reopening will cause your changes to be
discarded. Do  you want to reopen?
```

If you choose Yes, the file will be closed and opened again without saving the change you made in step 3. If you choose No, the workbook will remain open with the change you made in step 3, but you will receive an error message and the next line of code will not run.

For more information about the GetObject Function or OLE Automation, choose the Search button in Visual Basic Help and type:

```
OLE Automation
```

Additional reference words: 5.00

XL5: Calculating Depreciation with the Production Method

Article ID: Q108271

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY
=======

Microsoft Excel includes functions to calculate straight line, sum of the years digits, and double-declining balance depreciation (SLN, DDB, SYD). A fourth depreciation method, the production method, is widely used in business to calculate depreciation on items that can produce discrete units. Items such as vehicles and machinery are most often depreciated in this manner. The custom function module shown below calculates depreciation using this method.

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

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

1. Enter the following into a new module:

```
'Defines the variables
Function  Prod_depr(usage as Double, cost as Double, residual_value
as Double, useful_life As Double) As Double
'Performs the mathematical computation.
Prod_depr = usage * ((cost - residual_value) / useful_life)
'Terminates function
End Function
```

2. Enter the arguments for the function based on the following descriptions:

   - "usage" is the amount of usage, in units, the item has received during the depreciation period. This can be measured in miles or

hours used.
- "Cost" is the original price paid for the item.
- "Residual_value" is the estimated salvage or trade-in value.
- "useful_ life" is the planned usage, in units, of the item during its lifetime.

Example

Suppose a truck is purchased for $25,000, and has a residual value of $1500. The useful life of the truck is 250,000 miles. During a specific period of operation, the truck is driven for 15,000 miles. Using the above function, the depreciation for that period is $1410.

```
usage            =15,000 miles
cost             =$25,000
residual_value   =$1,500
use_life         =250,000 miles
```

NOTE: It may be easier to enter the formula

```
   = usage * ((cost - residual_value) / useful_life
```

directly into your worksheet if you do not intend to use the function frequently.

Additional reference words: 5.0 Production Depreciation

XL5: Visual Basic Procedure to Display System Resources

Article ID: Q108277

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

In Microsoft Excel 5.0, you can create a Visual Basic procedure that uses the Microsoft Windows API (application programming interface) Library to return the percentage of available User, GDI, and System resources.

Sample Visual Basic Procedure

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

```
'The following Declare Function should be entered on a single line
Declare Function GetFreeSystemResources Lib "USER" (ByVal fuSysResource _
As Integer) As Integer

'Declaration of Constants
Const GFSR_SYSTEMRESOURCES = 0
Const GFSR_GDIRESOURCES = 1
Const GFSR_USERRESOURCES = 2

'The Following macro code creates a Message box that displays
' the percentage of GDI, USER and SYSTEM RESOURCES
'available to Microsoft Windows on individual lines

Sub Get_Resource()
   Static LN As String * 1
   Dim Msg As String
   'LN forces the Message Box to wrap to a new line
      LN = Chr(13)
   'Percentage of Free GDI RESOURCES
   Msg = "GDI Resources = " & _
       GetFreeSystemResources(GFSR_GDIRESOURCES) & " %" & LN
   'percentage of Free USER RESOURCES
   Msg = Msg & "User Resources = " & _
```

```
        GetFreeSystemResources(GFSR_USERRESOURCES) & " %" & LN
    'percentage of Free SYSTEM RESOURCES
    Msg = Msg & "System Resources = " & _
        GetFreeSystemResources(GFSR_SYSTEMRESOURCES) & " %"
    'Display results in a message box
    MsgBox Msg
End Sub
```

REFERENCES
==========

Microsoft Windows SDK
Microsoft Visual Basic 3.0 Professional - WINAPI31.HLP help file

Additional reference words: 5.00 USER.EXE find call register howto VB

XL5: Macros to Return Windows and System Directories Paths

Article ID: Q108278

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel, version 5.0, you can create a macro to return the path to the Windows or Windows System directory by using the Declare Functions to access built-in functions in Microsoft Windows, version 3.1. The following macros use the GetWindowsDirectory and GetSystemDirectory function calls to retrieve the desired directory information.

MORE INFORMATION

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

```
'The following two declare statements need to be entered as single
'individual lines on the module sheet.
Declare Function GetSystemDirectory Lib "KERNEL" (ByVal lpBuffer As _
String, ByVal nSize As Integer) As Integer

Declare Function GetWindowsDirectory Lib "KERNEL" (ByVal lpBuffer As _
String, ByVal nSize As Integer) As Integer

Sub GetDir()
'sets the buffer length for both variables to 144
Dim Win_Dir As String * 144
Dim Sys_Dir As String * 144
   'returns the windows directory
   y = GetWindowsDirectory(Win_Dir, Len(Win_Dir))
   'Displays the windows directory in a Message box
   MsgBox Win_Dir
```

```
   'Returns the Windows\System directory
   x = GetSystemDirectory(Sys_Dir, Len(Win_Dir))
   'Displays the windows\system directory in a Message box
   MsgBox Sys_Dir
End Sub
```

REFERENCES
==========

"Function Reference," version 4.0, pages 42, 350
On-line Help in the Microsoft Windows SDK

Additional reference words: 5.00

XL5: Function to Compute Average Without High and Low Values

Article ID: Q108280

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

For statistical purposes, you may want to calculate the average of a sample excluding its highest and lowest values. In Microsoft Excel version 5.0, you can create the following formula to calculate an average after excluding the highest and lowest values:

```
=(SUM(region)-MAX(region)-MIN(region))/(COUNT(region)-2)
```

NOTE: You must include the parentheses the way they appear in the above formula so that the function is computed in the right order.

MORE INFORMATION

The following is the same function as above, incorporated into a Microsoft Excel version 4.0 Visual Basic macro:

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Sample Visual Basic Macro

```
Function HiloAverage(region) As Variant
 With Application
  HiloAverage=(.Sum(region)-.Max(region)-.Min(region))/(.Count(region) - 2)
 End With
End Function
```

REFERENCES

==========

"Visual Basic Language Reference," version 3.0, Page 231

Additional reference words: 5.00

XL5: Macro to Toggle spooler= Line of WIN.INI

Article ID: Q108281

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel, version 5.0, you can use the following Visual Basic code to toggle the spooler= setting in the WIN.INI file (from Yes to No or from No to Yes).

MORE INFORMATION

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

WARNING: The Declare Function statements listed in the following macro are sensitive. If you use these statements incorrectly, these functions may cause a general protection (GP) fault in Windows or cause serious software problems.

```
'Enter each of the following Declare Functions as a single line.

Declare Function GetProfileString Lib "Kernel" (ByVal Section As _
   String, ByVal Entry As Any, ByVal Default As String, _
   ByVal Setting As String, ByVal BytesOutput As Integer) As Integer

Declare Function WriteProfileString Lib "Kernel" (ByVal _
   Section As String, ByVal Entry As Any, ByVal EntryValue As Any) _
   As Integer

Sub ToggleSpooler()
   'Create a buffer to return the string into.
   Dim Setting As String * 4
   'Find the current value of Spooler in the WIN.INI file.
```

```
   'Note: enter the following two lines as a single line.
   Length = GetProfileString("windows", "spooler", "", Setting, _
      Len(Setting))
   'If Spooler = yes then set Spooler to no.
   'Note: enter the following two lines as a single line.

   If Left(Setting, Length) = "yes" Then
      Length = WriteProfileString("windows", "spooler", "no")
      MsgBox "Spooler has been turned off. Please restart Windows."
   Else
      'If Spooler = anything other than yes, set spooler to yes.
      Length = WriteProfileString("Windows", "spooler", "yes")
      MsgBox "Spooler has been turned on. Please restart Windows."
   End If
End Sub
```

Additional reference words: 5.00 change settings

XL5: Distinguishing Sheet Types In Visual Basic

Article ID: Q108350

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY
=======

In Microsoft Excel, version 5.0, the Visual Basic TypeName function will return the "Worksheet" value when you use it to identify worksheets, MS Excel 4.0 Macro sheets, or MS Excel 4.0 International Macro sheets.

Because these sheets all return the same value, you must use the Type property to distinguish between these three types of sheets. The following Visual Basic code demonstrates how to use this property.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following Visual Basic code examples assume that you have a workbook that contains one worksheet, one MS Excel 4.0 Macro sheet, one chart sheet, one dialog sheet, and one Visual Basic module, in that order. The code example is located in the Visual Basic module.

To run the macro position the insertion point in the line that reads "Sub Sheet_Type()" and choose Start from the Run menu.

```
'-----------------------------------------------------------------------
Option Explicit

Sub Sheet_Type()
   'Dimension variables.
   Dim X As Variant
   'Iterate through the loop once for each sheet in the workbook.
```

```
   For Each X In ActiveWorkbook.Sheets
      'If the sheet's TypeName is "Worksheet", then
      If TypeName(X) = "Worksheet" Then
      'Check for each Type (xlWorksheet, xlExcel4MacroSheet,
      'xlExcel4IntlMacroSheet) and display the appropriate message
      'box.
         If X.Type = xlWorksheet Then
            MsgBox "Worksheet"
         ElseIf X.Type = xlExcel4MacroSheet Then
            MsgBox "MS Excel 4.0 Macro Sheet"
         ElseIf X.Type = xlExcel4IntlMacroSheet Then
            MsgBox "MS Excel 4.0 International Macro Sheet"
         End If
      'Otherwise, display a message box with the appropriate TypeName.
       Else
          MsgBox TypeName(X)          'show sheet type in message box
      End If
   Next                               'repeat the loop until all done
End Sub
'---------------------------------------------------------------------
```

When you run the Sheet_Type subroutine, the messages you receive are:

```
   Worksheet, MS Excel 4.0 Macro Sheet, Chart, DialogSheet, Module
```

The Sheet_Type subroutine determines if the TypeName of the current sheet is "Worksheet" and then gets the current sheet's Type property to determine the final result. This allows different, more correct results for the worksheet and the MS Excel 4.0 Macro sheet.

Additional reference words: 5.00

XL5: ChDir May Fail When Changing to a Root Directory

Article ID: Q108351

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel, version 5.0, the Visual Basic ChDir statement will fail if the path argument does not contain a backslash character (\). This requirement may cause problems if you use ChDir to change to a root directory, because the path of a root directory does not necessarily contain a backslash.

For example, the following Visual Basic code will fail

```
ChDir "C:"
```

and you will receive the following error message:

```
Run-time error '76':
Path not found
```

The following lines of code will succeed, because they each contain a backslash:

```
ChDir "C:\"  <or>  ChDir "C:\EXCEL\LIBRARY"
```

It is possible to incorporate error-handling into a ChDir function to prevent an error from occurring. The Visual Basic code example below demonstrates one way to do this.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

This Visual Basic code example displays an input box which asks you which drive or directory you would like to switch to. When you enter a drive or directory, the example checks whether the path you enter contains a backslash and adds one, if needed, before switching to that drive or directory.

To run the example, position the insertion point in the line that reads "Sub MainMacro()" and choose Start from the Run menu.

```
'--------------------------------------------------------------------------
Option Explicit

Sub MainMacro()
   'Dimension some variables.
   Dim NewDir As Variant
   'Prompts you to enter a drive/directory. For example, D: or C:\EXCEL.
   'If you type in an invalid directory, the subroutine will fail. For
   'example, typing just "C" (without the quotes) is not going to work.
   NewDir = InputBox("Switch to which drive/directory?")
   'If the NewDir ends in a colon, indicating it is a root directory,
   'concatenate a backslash onto the end of it. For example, C: would
   'become C:\. If you actually enter "C:\", the subroutine doesn't add
   'another backslash.
   If Right(NewDir, 1) = ":" Then
      NewDir = NewDir & "\"
   End If
   'Switches to the proper drive (the first letter in NewDir: 'Left'
   'gives us this) and directory.
   ChDrive Left(NewDir, 1)             'change to the drive (C:, etc.)
   ChDir NewDir                        'change to the directory
   'Display the name of the current directory so you know it worked.
   MsgBox "Current directory is " & CurDir()
End Sub
'--------------------------------------------------------------------------
```

The If-End If routine is the error-checking procedure. Whenever you use a ChDir statement, preceding it with the If-End If routine (or some variation of it) can help prevent the "no backslash" error.

Additional reference words: 5.00 5.0

XL5: Can't Run Macros That Use File Functions Add-in Functions

Article ID: Q108356

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

Although Microsoft Excel version 5.0 does not include the File Functions (FILEFNS.XLA) add-in macro (or the associated FILEFNS.DLL file), if you install Microsoft Excel version 5.0 without deleting the earlier version of Microsoft Excel (3.0 or 4.0), the FILEFNS.XLA add-in macro will still be available and will function normally (because the file is not deleted when you install version 5.0). However, if you do a "clean" installation of Microsoft Excel 5.0 (that is, if you delete the earlier version of Microsoft Excel before you install version 5.0) and you then run a macro that uses any of the FILEFNS.XLA add-in functions you will receive a macro error.

Most of the file maintenance tasks that you can do with the File Functions add-in functions can be done with Visual Basic procedures.

MORE INFORMATION

The following table lists the functions that are available in the File Functions add-in macro and their Visual Basic equivalents.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Function from File Functions	Equivalent Visual Basic Statement
=CREATE.DIRECTORY()	MdDir "MYDIR"

```
=DELETE.DIRECTORY() RmDir "MYDIR"

=DIRECTORIES()      MyDir = Dir(pathname,16)
                    'call repeatedly until Dir return a zero-
                    'length string

=OPEN.DIALOG()      MyDialog = Application.GetOpenFilename(fileFilter, _
                    FilterIndex,Title,buttonText)
                    ' This second form will bring up Microsoft
                    ' Excel's internal Open Dialog Box
                    MyDialog2=Application.Dialogs(xlDialogOpen).Show
                    ' will bring up default open dialog

=SAVE.DIALOG()      MyDialog=Application.GetSaveAsFilename _
                    (initialFilename,fileFilter,filter, _
                    Index,title,buttonText)
                    ' This second form will bring up Microsoft
                    ' Excel's 'internal SaveAs Dialog Box
                    MyDialog2=Application.Dialogs(xlDialogSaveAs).Show
                    ' will bring up default Save As dialog

=FILE.EXISTS()      MyFile = Dir(pathname)
```

REFERENCES
==========

For more information about GetOpenFilename, choose the Search button in Help and type:

   getop

For more information about GetSaveasFilename, choose the Search button in Help and type:

   gets

Additional reference words: 5.0 5.00

XL5: How to Manipulate Macro Operation With Key Strokes

Article ID: Q108357

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel, you can create a macro that performs different operations depending on which keys you press when you run the macro (for example, if you want your macro to perform one operation when you press the CTRL key and another if you don't press the CTRL key).

To control the operation of a macro, you can use one key (for example the CTRL key) or a combination of keys (for example CTRL+ALT). Your macro will test for the state of these keys (pressed or not pressed) to determine which action to take.

MORE INFORMATION

To create a macro that responds to different key states, use an IF function. The IF function allows you to control your macro based on the results of calls it makes to the Windows environment. These calls to the Windows environment allow the IF function to identify the state of the keys you want to use. To return the key state of one or more keys, use the Windows function, GetKeyState().

You can use the above procedure to change the operation of macros that are already assigned to objects, buttons, menu commands, or hot keys.

NOTE: Making this kind of change will not affect the normal behavior of the object to which the macro is assigned.

You can use any keys as long as they are not either of the following:

- The same keys that are used to start the macro.

  -or-

- Keys that change how Microsoft Excel interacts with the object to which your macro is assigned.

The GetKeyState() Function

The GetKeyState() function returns a number indicating the current state of a specific key at the time the DLL call is made. A negative number indicates that the key is pressed, a positive number indicates that the key is not pressed.

The following GetKeyState() Visual Basic code is an example that returns

the current state of a key and takes action based on the key state returned.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

WARNING: The Declare Function statements listed in the following macro are sensitive. If used incorrectly, these functions may cause a general protection (GP) fault in Windows or cause serious software problems.

```
'Type the following two lines as a single line in the module
Declare Function GetKeyState Lib "USER" (ByVal nVirtKey As Integer) _
As Integer

Const SHIFT_KEY = 16
Const CTRL_KEY = 17
Const ALT_KEY = 18

Sub find_key()

   'Checks key states to see if both CTRL and ALT Keys are pressed
   If GetKeyState(CTRL_KEY) < 0 And GetKeyState(ALT_KEY) < 0 Then
     MsgBox "CTRL + ALT KEYS PRESSED"
   'Checks key states to see if only CTRL key is pressed
   ElseIf GetKeyState(CTRL_KEY) < 0 Then
     MsgBox "CTRL KEY PRESSED"
   'Checks key states to see if only ALT key is pressed
   ElseIf GetKeyState(ALT_KEY) < 0 Then
     MsgBox "ALT KEY PRESSED"
   'Checks key states to see if only SHIFT key is pressed
   ElseIf GetKeyState(SHIFT_KEY) < 0 Then
     MsgBox "SHIFT KEY PRESSED"
   End If

End Sub
```

The above example macro checks the state of the following keys or key combinations: CTRL+ALT, CTRL, and ALT. The macro displays an alert box describing the key or key combination that is pressed when you run the macro.

The argument used to call the DLL function GetKeyState() is a number indicating which key to test. The number used is the virtual key code that the GetKeyState function uses to identify which key to test.

When you write your macro, you should be aware of the following:

- You should check for key state at the beginning of the macro. The GetKeyState() function returns the state of the key at the time the DLL function is called not at the time the macro is run. So, it is possible to miss the key state if you don't check for it right away.

- The test you use for checking which key is pressed must follow correct programming logic.

- Your macro should not check for too many key possibilities. The more possible key combinations you check for, the longer it takes your macro to find the right condition and, therefore, the greater the possibility that the key will no longer be pressed.

REFERENCES
==========

"Microsoft Windows Software Developers Kit Reference," Volume 1, pages 1-30, 4-183

"Microsoft Windows Software Developers Kit Reference," Volume 2, Appendix A

"Microsoft Quick C for Windows," pages 341, 935-938

Additional reference words: 5.00 call register keystate GPF

XL5: No Visual Basic Method for SET.UPDATE.STATUS

Article ID: Q108384

The information in this article applies to:

- Microsoft Excel for Windows version 5.0

In Microsoft Excel 5.0, to set the links update status, you must use the ExcecuteExcel4Macro method to run the SET.UPDATE.STATUS() macro command.

Macro Example

This macro sets the update status to manual in the Links dialog box which is available when you choose Links from the Edit menu.

```
Sub LinkStatus()

   ExecuteExcel4Macro "SET.UPDATE.STATUS__
(""Word.Document.6|Doc!!DDE_LINK2"",2,2)"

End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Additional reference words: 5.00

XL5: Can't Access File Opened as Read-Only in Visual Basic

Article ID: Q108432

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

A file opened as read-only through a Visual Basic macro will remain read-only to other applications until it is closed.

This is normal operation of Visual Basic. Visual Basic opens the file for input in compatibility mode; this prevents other users from opening the file in anything other than compatibility-input mode.

This functionality is different than the FOPEN function in Microsoft Excel version 4.0. The statement =FOPEN(filename,2) will allow read-only access while allowing full access to other applications.

Macro Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following code opens a file (C:\TEST.TXT) as read only and forces the file as read only to other applications until you close the file or quit Microsoft Excel:

```
Sub Test()
   Dim FileNumber as Integer
   FileNumber = FreeFile()
   Open "c:\test.txt" For Input Access Read As #FileNumber
End Sub
```

1. Open the file C:\TEST.TXT in a text editor (such as Write, which is located in the accessories group of Program Manager).

2. From the File menu, choose Save.

3. You will receive a message stating that the file is read-only and should be saved as a different name.

Additional reference words: 5.00

XL5: GP Fault When Macro Closes Its Own Workbook

Article ID: Q108507

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In Microsoft Excel version 5.0 for Windows, you may receive a General Protection (GP) fault if a Visual Basic macro is started by an OnSheetDeactivate command and the macro then closes the workbook in which it is contained.

CAUSE

This error occurs only if a sheet is deleted by choosing Delete Sheet from the Edit menu or from the Sheet Tabs menu; it does not occur if a sheet is deactivated by switching to another sheet.

STATUS

Microsoft has confirmed this to be a problem in Microsoft Excel version 5.0 for Windows. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 5.00 crash hang freeze lock-up

XL5: Selection.Rows.Count May Return Incorrect Result

Article ID: Q108508

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, the Visual Basic statement Selection.Rows.Count, which returns the number of rows in the current selection, will return an incorrect result if the current selection is nonadjacent.

This incorrect result occurs because the Selection.Rows.Count statement only counts the number of rows in the first Area of the selection. An Area is defined as a single piece of a nonadjacent selection. For example, the selection $1:$2, $4:$6, $8:$11 consists of three Areas: Selection.Rows.Count will only count rows in the first Area (the range $1:$2).

The above information also applies to Selection.Columns.Count, which counts the number of columns in the current selection (or first Area of a nonadjacent selection).

The Visual Basic code example shown below demonstrates one way to return the correct number of rows (or columns) in a nonadjacent selection.

MORE INFORMATION

You can create a nonadjacent selection by selecting one range and then selecting another range while holding down the CTRL key or using Microsoft Excel version 4.0 or Visual Basic commands that select two or more ranges at the same time.

To count the number of rows or columns in a selection when the selection is nonadjacent, you must separate the selection into Areas. This process is illustrated in the Visual Basic code example below.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is

doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

NOTE: To convert this example to work with columns, substitute "Column" for "Row."

To use the CountRows subroutine, select any combination of rows on a worksheet. Then, choose Macro from the Tools menu, select CountRows from the list of macro names, and choose Run.

```
'-----------------------------------------------------------------------
Option Explicit

Sub CountRows()
   'Dimension some variables.
   Dim Counter As Integer, NumRows As Integer
   Dim X As Variant, Y As Variant
   NumRows = 0                              'initialize NumRows
   'Sets range X equal to the current selection.
   Set X = Selection
   'Initializes range Y equal to the first Area in X.
   Set Y = X.Areas(1)
   'Iterate through the loop once for each Area (nonadjacent piece)of
   'the range X. This loop consolidates the different Areas in order to
   'prevent row miscounts due to overlapping Areas.
   For Counter = 1 To X.Areas.Count
      'Set Y equal to the union of its previous range and the range of
      'the rows which encompass the current Area.
      Set Y = Application.Union(Y, X.Areas(Counter).EntireRow)
   Next                                     'loop until all done
   'Iterate through the loop once for each Area (nonadjacent piece)of
   'the range Y. This loop actually counts the rows.
   For Counter = 1 To Y.Areas.Count
      'Set NumRows equal to its previous value plus the number of rows
      'in the current Area.
      NumRows = NumRows + Y.Areas(Counter).Rows.Count
   Next                                     'loop until all done
   'Show the number of rows.
   MsgBox Str(NumRows) & " rows selected."
End Sub
'-----------------------------------------------------------------------
```

For example, if you select the range $1:$2, $4:$6, $8:$11 on a worksheet, and run the CountRows subroutine, the subroutine determines how many Areas there are in the selection (there are three: $1:$2 , $4:$6 , and $8:$11), counts the number of rows in each Area (2, 3, and 4). The subroutine then adds those numbers together for a total of nine rows. when the subroutine is completed, you receive the following message

```
   9 rows selected.
```

Additional reference words: 5.00

XL5: Error Getting the Value of a Name in Visual Basic

Article ID: Q108517

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0 you may receive an error message when you use the Visual Basic Range(Name).Value method to get the value of a name. This error message will occur if the name in question refers to a constant value or a formula rather than a range.

MORE INFORMATION

If a particular name will refer to a constant value or a formula, you can use the .Evaluate(Name) method to get the value of the name. The .Evaluate(Name) method can evaluate formulas and constant values even if they don't refer to an actual range.

However, note that the .Evaluate(Name) method may return an error message if the name refers to an error value, such as #REF! or #N/A. In these cases, you may need to employ error checking (such as the On Error Resume Next function) or use an alternative method, such as the Names(Name).RefersTo method, to detect or handle an error value.

For example, if you have a sheet named Sheet1 in a workbook that contains the following global names

This Name	Refers To	Cell Information
Alpha	=Sheet1!$B$1	Cell B1 contains the number 1
Bravo	=2	
Charlie	=SUM(Sheet1!$B$3:$C$3)	Cell B3 contains the number 3, cell C3 contains the number 4
Delta	=Alpha	
Echo	=Charlie	
Foxtrot	=Alpha+Charlie	
Golf	=#N/A	

and you use the Range(Name).Value and .Evaluate(Name) methods to get the values of the names you will receive the following results:

Name	Range(Name).Value	Sheets("Sheet1").Evaluate(Name)
Alpha	1	1
Bravo	[Error message 1]	2
Charlie	[Error message 1]	7
Delta	1	1
Echo	[Error message 1]	7

```
Foxtrot  [Error message 1]  8
Golf     [Error message 1]  [Error message 2]
```

The error messages 1 and 2 are as follows.

Error message 1

```
   Run-time error '1004':
   Range method of Application class failed
```

Error message 2

```
   Run-time error '13':
   Type mismatch
```

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

To check the value of a name, you could use the following subroutine:

```
'-----------------------------------------------------------------------
Option Explicit

Sub CheckNameValue()
   Dim Test As Variant
   Test = Range("Alpha").Value

   'To check the value by using .Evaluate(Name), use
   '
   '   Test = Sheets("Sheet1").Evaluate("Alpha")
   '
   'in place of the previous test line.

   MsgBox Test
End Sub
'-----------------------------------------------------------------------
```

In order to prevent an error when the name refers to an error value (in this case, if the name is Golf), use the Names(Name).RefersTo method to check the name before getting its value. For example, you could use:

```
Sub CheckForError()

   'If the name Golf refers to an error value, such as #REF! or #N/A,
   If IsError(Evaluate(Names("Golf").RefersTo)) Then
      'then show an error message to that effect,
      MsgBox "Golf is an error name!"
   Else
      'otherwise state that the name refers to a good reference.
      MsgBox "Golf is OK!"
   End If
End Sub
```

If the name refers to an error value, the IsError test will be true and the error message box will be displayed. Otherwise, the OK message box will be displayed. For example, if the name is Golf, which refers to =#N/A, the error message box will be displayed. Using any of the other example names will result in the OK message box.

Additional reference words: 5.00

XL5: Range.EntireRow May Return Incorrect Result

Article ID: Q108518

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, the Range.EntireRow method may return an incorrect result if the Range is a nonadjacent selection. For example, if the current Range is the nonadjacent selection $A$1:$A$3, $C$11:$C$13, the Range.EntireRow method will return $1:$13, not $1:$3,$11:$13 as would be expected.

This also occurs when you use the Range.EntireColumn method to return entire columns when the Range is nonadjacent.

MORE INFORMATION

You can create a nonadjacent selection by selecting a range and then selecting another range while holding down the CTRL key on your keyboard or by using Microsoft Excel version 4.0 or Visual Basic commands that select two or more ranges at the same time.

To return the correct rows or columns in a selection when the selection is nonadjacent, you must separate the selection into Areas, where an Area is defined as a single piece of a nonadjacent selection. This process is illustrated in the Visual Basic code example below.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

NOTE: To convert this example to work with columns, use "Column" in place

of "Row."

To use the SelectRows subroutine, select any combination of cells on sheet1. (This example assumes the current selection is $A$1:$A$3,$C$11:$C$13 on Sheet1.) Then, choose Macro from the Tools menu, select the name SelectRows from the list of macro names, and choose Run.

```
'-----------------------------------------------------------------------
Option Explicit

Sub SelectRows()
   'Dimension some variables.
   Dim Counter As Integer, X As Variant, Y As Variant
   Sheets("Sheet1").Activate            'ensure the worksheet is active
   'Sets range X equal to the current selection.
   Set X = Selection
   'Initializes range Y equal to the first Area in X.
   Set Y = X.Areas(1)
   'Iterate through the loop once for each Area (nonadjacent piece)
   'of the range X.
   For Counter = 1 To X.Areas.Count
      'Set Y equal to the union of its previous range and the range of
      'the rows which encompass the current Area.
      Set Y = Application.Union(Y, X.Areas(Counter).EntireRow)
   Next                                 'loop until all done
   Y.Select                             'select the range Y
End Sub
'-----------------------------------------------------------------------
```

For example, if you select the range $A$1:$A$3,$C$11:$C$13 on worksheet Sheet1 and then run the SelectRows subroutine, the subroutine determines how many Areas there are in the selection (there are two: $A$1:$A$3 and $C$11:$C$13), determines which combination of rows encompasses each Area ($1:$3, $11:$13), and selects the range $1:$3,$11:$13.

Additional reference words: 5.00 5.0

XL5: Running Subroutines and Macros from Visual Basic

Article ID: Q108519

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, you can run Visual Basic subroutines and Microsoft Excel version 4.0 macros from a Visual Basic subroutine by using the Application.Run and Application.ExecuteExcel4Macro methods. You can also run Visual Basic subroutines with the Call method or by entering the name of a subroutine on a line by itself.

This article illustrates several methods that you can use to run subroutines and Microsoft Excel version 4.0 macros from Visual Basic in Microsoft Excel.

MORE INFORMATION

To Use the Application.Run Method

You can use the Application.Run method to run Visual Basic subroutines or Microsoft Excel version 4.0 macros from other Visual Basic subroutines. The Application.Run method requires one named argument: the name of the macro or subroutine to be run. (However, other optional arguments may also be included.) This name can be a text string (for example, "TestXLM") or it can be a variable that is equal to the name of the macro.

For example, to run an Microsoft Excel version 4.0 macro called TestXLM, you could use this method:

```
Application.Run ("TestXLM")
```

If you have the variable "MacroToRun", whose value is equal to "TestXLM", you could use this method:

```
Application.Run (MacroToRun)
```

To Use the Application.ExecuteExcel4Macro Method

You can also use the Application.ExecuteExcel4Macro method to run Microsoft Excel version 4.0 macros or other Visual Basic subroutines, but the syntax is somewhat different. To use Application.ExecuteExcel4Macro to run a macro or subroutine, you must also include the Microsoft Excel version 4.0 RUN() function, as in the following examples:

```
Application.ExecuteExcel4Macro "RUN(""TestXLM"")"
```

-or-

```
Application.ExecuteExcel4Macro "RUN(""" & MacroToRun & """)"
```

Note that when you use Application.ExecuteExcel4Macro you must use quotation marks. For example, to use the RUN() function, you must enclose the name of the argument in quotation marks:

```
RUN("TestXLM")
```

Because the entire string must also be enclosed in quotation marks, when you add quotation marks to the outside of the string, you must also add an additional quotation mark adjacent to each quotation mark within the string. The resulting string is as follows:

```
"RUN(""TestXLM"")"
```

The Application.ExecuteExcel4Macro command that uses a variable inside the RUN() function is more complex than the equivalent Application.Run method. For the command to be properly evaluated, the macro string must be entered as:

```
"RUN(""" & MacroToRun & """)"
```

This command is evaluated as

```
RUN("" & MacroToRun & "")
```

which is a valid Microsoft Excel version 4.0 macro command.

To Use the Call Method

The Call method may be used to run Visual Basic subroutines, but not Microsoft Excel version 4.0 macros. For example, to run the subroutine TestVBSub, you would use this method:

```
Call TestVBSub
```

Note that you cannot pass a variable name to the Call method. For example, if you have the variable "SubToRun", whose value is equal to "TestVBSub", you cannot run the TestVBSub subroutine with the following:

```
Call SubToRun
```

To Run a Subroutine Using Only Its Name

You can also run a Visual Basic subroutine by entering its name on a line by itself. For example, if you want your subroutine to run the TestVBSub subroutine, you would enter

```
TestVBSub
```

on a line by itself. When that line in the subroutine is executed, it will run the TestVBSub subroutine.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

To create six subroutines that illustrate the most common methods you can use to run a Visual Basic subroutine or Microsoft Excel version 4.0 macro from another Visual Basic subroutine, do the following:

1. In a new workbook, insert a Microsoft Excel 4.0 macro sheet called Macro1 and a Visual Basic module called Module1.

2. On the macro sheet, enter the following macro:

   ```
   A1: TestXLM
   A2: =ALERT("TestXLM works!")
   A3: =RETURN()
   ```

   This macro displays an alert box.

3. On the macro sheet, select cell A1.

4. From the Insert menu, choose Name, and then choose Define.

5. Verify that the following information appears in the Define Name dialog box:

   - The Names In Workbook box contains the name TestXLM.
   - The Refers To box contains the reference =Macro1!$A$1.
   - The Command option is selected under Macro.

   When the settings are as specified above, choose OK to define the name of the macro.

6. In Module1, enter the following subroutines:

```
'--------------------------------------------------------------------
Option Explicit

'The TestVBSub subroutine displays a message box: it is the Visual
'Basic equivalent of the TestXLM macro shown above.
```

```
Sub TestVBSub()
   MsgBox "TestVBSub works!"             'displays a message box
End Sub

'The Test1 subroutine makes use of the Application.Run method with
'hard-coded macro/subroutine names.

Sub Test1()
   Application.Run ("TestVBSub")
   Application.Run ("TestXLM")
End Sub

'The Test2 subroutine makes use of the Application.Run method with
'variable macro/subroutine names.

Sub Test2()
   Dim SubToRun As String, MacroToRun As String
   SubToRun = "TestVBSub"
   MacroToRun = "TestXLM"
   Application.Run (SubToRun)
   Application.Run (MacroToRun)
End Sub

'The Test3 subroutine makes use of the Application.ExecuteExcel4Macro
'method with hard-coded macro/subroutine names.

Sub Test3()
   'Note the extra quotation marks which are contained within the RUN
   'statements. These are required in order for the command to evaluate
   'properly.
   Application.ExecuteExcel4Macro "RUN(""TestVBSub"")"
   Application.ExecuteExcel4Macro "RUN(""TestXLM"")"
End Sub

'The Test4 subroutine makes use of the Application.ExecuteExcel4Macro
'method with variable macro/subroutine names.

Sub Test4()
   Dim SubToRun As String, MacroToRun As String
   SubToRun = "TestVBSub"
   MacroToRun = "TestXLM"
   'Note the extra quotation marks which are contained within the RUN
   'statements. These are required in order for the command to evaluate
   'properly.
   Application.ExecuteExcel4Macro "RUN(""" & SubToRun & """)"
   Application.ExecuteExcel4Macro "RUN(""" & MacroToRun & """)"
End Sub

'The Test5 subroutine uses the Call method with hard-coded
'subroutine names.

Sub Test5()
    Call TestVBSub
End Sub

'The Test6 subroutine runs the TestVBSub subroutine because its name is
```

```
'entered on a line by itself.

Sub Test6()
   TestVBSub
End Sub
'-----------------------------------------------------------------------
```

When you run Test1, Test2, Test3, or Test4, two alert boxes will appear with the messages "TestVBSub works!" and "TestXLM works!" When you run Test5 or Test6, one alert box will appear with the message "TestVBSub works!"

Additional reference words: 5.00

XL5: Macro to Add a Number to a Selected Cell

Article ID: Q108642

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, to add a number to an existing number and put the result into the same cell, you can use a Visual Basic procedure or a Microsoft Excel 4.0 macro.

MORE INFORMATION

The following Visual Basic procedure uses the active cell, asks for the number to be added to the existing cells value, and places the result back into the active cell:

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

```
'-----------------------------------------------------
Sub AddToCell()

   ' ActiveCell.Value places the value or formula into the active cell
   ' on the active worksheet.
   '
   ' ActiveCell.Value returns the value in the active cell of the active
   ' window.
   '
   ' InputBox calls up the Excel input box for you to type in the number
   ' to be added to the original value. Val takes the text string
   ' returned from the input box and turns it into a number.

   ActiveCell.Value = ActiveCell.Value + Val(InputBox("Enter a Number",_
   default:=1))
```

```
End Sub
'-----------------------------------------------------

REFERENCES
==========

For more information about ActiveCell.Value, choose Programming With
Visual Basic from the Help Contents and then choose the Search button and
type:

    ActiveCell

Additional reference words: 5.00
```

XL5: OLE Automation Err Msg: Method Not Applicable

Article ID: Q108661

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0
- Microsoft Visual Basic version 3.0

SYMPTOMS

When you run code in a Microsoft Visual Basic version 3.0 application and the code uses a method or property of an object linking and embedding (OLE) automation object, you may receive the following error message

```
Method not applicable for this object
```

CAUSE

Some of the methods and properties of OLE Automation objects are keywords in Visual Basic version 3.0 and will cause the error message above when you use them in a Visual Basic application.

WORKAROUND

To use a method or property of a Microsoft Excel object in a Visual Basic application when the method or property is a Visual Basic keyword, use square brackets ([]) around the keyword in the Visual Basic code. For example, to use the Close method of the Microsoft Excel Workbooks object in a Visual Basic application, use the following syntax:

```
XL.Workbooks(1).[close]
```

MORE INFORMATION

OLE Automation is an industry standard that applications use to expose OLE objects to development tools, macro languages, and other containers that support OLE Automation. A few of the objects exposed by Microsoft Excel are workbooks, worksheets, and charts. These objects can be accessed with Visual Basic commands.

You use Visual Basic commands 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).

Steps to Reproduce Behavior

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied,

including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide.

1. Run Microsoft Excel.

2. Run Microsoft Visual Basic version 3.0.

3. In Microsoft Visual Basic, enter and run the following code:

```
' Dimension variable xl as object type
Dim xl as Object
' Activate Microsoft Excel and assign to variable xl
Set xl = GetObject(,"Excel.Application.5")
' Create a new workbook so we can close it
xl.Workbooks.Add
' Close the workbook we just created
xl.ActiveWorkbook.Close
```

The following error message appears

   Method not applicable to this object

even though Close is a method of the Microsoft Excel Workbooks object.

To make this piece of code work correctly, change the last line to:

```
xl.ActiveWorkbook.[Close]
```

REFERENCES
==========

For more information about the GetObject Function, choose the Search button in Microsoft Visual Basic Help and type:

    OLE Automation

Additional reference words: 5.00

Menus.Count Returns Different Number When Workbook Maximized

Article ID: Q108662

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In the Microsoft Visual Basic Programming System, Applications Edition, the following code

```
ActiveMenuBar.Menus.Count
```

returns a different number of menus on the active menu bar depending on whether or not the active workbook is maximized.

CAUSE

When the active workbook is maximized, the control menu appears next to the File menu. Because this menu is counted by ActiveMenuBar.Menus.Count, the number returned by the code will be one greater for a maximized workbook than for a workbook that is restored or minimized.

WORKAROUND

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide.

To return the actual number of menus on the active menu bar, excluding the control menu if the worksheet is maximized, use the following code.

Sample Visual Basic Code

```
Sub Menu_Count()
   ' Declare variable x as an Integer and MyMenu as an Object
   Dim x As Integer
   Dim mymenu As Object
   ' Set initial value of x to zero
   x = 0
      ' Use For Next loop to count number of menus
      For Each mymenu In ActiveMenuBar.Menus
          x = x + 1
     Next
     ' Display number of menus in a message box
     MsgBox x
End Sub
```

REFERENCES
==========

For more information about the Menu Bar Object, choose the Search button in Visual Basic Online Help and type:

    menu bar object

Additional reference words: 5.00

XL5: Manually Recalculating a Single Cell or Range

Article ID: Q108799

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, the Calculate Now command on the Options menu causes every cell in the open worksheet to be recalculated (even when calculations are set to manual).

The following information discusses three ways to recalculate a single cell or range without causing the entire document to be recalculated.

MORE INFORMATION

Method 1

1. Position the insertion point in the cell that you want to recalculate.
2. Press the F2 key.
3. Press ENTER.

To recalculate only the cells in a single array, do the following:

1. Select one cell in the array.
2. From the Edit menu, choose Go To and choose Special.
3. Select Current Array.
4. Press the F2 key.
5. Press CTRL+SHIFT+ENTER. These steps reenter the formula in the array, updating the values they return.

Method 2

1. Be sure calculation is set to manual. To do this, choose Options from the Tools menu, select the Calculation Tab and select the Manual option. Then choose OK to accept the change.
2. From the Edit menu, choose Replace.
3. In the Find What box, type "=" (without quotation marks) and in the Replace With box type "=".

4. Choose Replace (not Replace All).

NOTE: To recalculate a selected range rather than a single cell, choose Replace All.

Method 3 - Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

To recalculate the current cell, use the following procedure:

```
Sub Calc()
   ActiveCell.Calculate
End Sub
```

To recalculate a range on the active worksheet, use the following procedure:

```
Sub Calc_Range()
   Range("A1:D1").Calculate
End Sub
```

Additional reference words: 5.00

XL5: Cells.Find Returns Error When No Match Found

Article ID: Q108892

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

When you run a recorded Visual Basic macro, the Cells.Find, columns.find, Selection.Find, and Range.Find functions will produce the following macro error if no data is found:

```
Runtime error '91':
Object variable not Set
```

This macro error occurs because the Visual Basic Find method returns a NULL value which makes activating a cell impossible.

WORKAROUND

The following Visual Basic example will attempt to find the text string "mystring" and produces an alert message which will not cause the macro to halt. The example can be entered into a Microsoft Excel module sheet "as is" or on one line without the underscore characters.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

```
Sub Find_MyString

If Not Application.ExecuteExcel4Macro _
(String:="FORMULA.FIND(""mystring"",1,2,1,1,FALSE)") _
Then MsgBox "Cannot Find Matching Data."

End Sub
```

Additional reference words: 5.00 VB

XL5: Visual Basic Macro to Hide and Restore All Toolbars

Article ID: Q109064

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY
=======

In Microsoft Excel version 5.0, the following Visual Basic Code will show all, hide All, or toggle the currently displayed toolbars Visible property.

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

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

```
'This is a sub-routine to Hide all of the toolbars.

Sub HideAllToolbars()
   ' loop variable
   Dim i As Integer
   ' Loop through the Total number of toolbars
   For i = 1 To Application.Toolbars.Count
       'Hide each toolbar
       Application.Toolbars(i).Visible = False
   ' end of loop
   Next i
End Sub

'This is a sub-routine to show all of the toolbars.

Sub ShowAllToolbars()
   'loop variable
   Dim i As Integer
   'Loop through the Total number of toolbars
```

```
   For i = 1 To Application.Toolbars.Count
       'Show each toolbar
       Application.Toolbars(i).Visible = True
   'end of loop
   Next i
End Sub

' The following routine when run the first time will store all of the
' toolbars visible property then hide all of the visible toolbars. When
' the routine is run a second time, it will restore the toolbars to their
' original state the first time the code was executed.

' This subroutine will toggle all currently visible toolbars to hidden
' and when rerun will restore the toolbars

Sub ToggleToolbars()
' Creates a 20 element array to keep track of current toolbar settings on
' the first iteration through the routine.
' This limits this routine to 20 toolbars total (increasing this number
' will allow for more custom toolbars).

Static CurrentToolSet(20) As Boolean
Static Flag As Boolean
Dim i As Integer
'If this is the first time through the routine do this
If Flag = False Then
    ' Loop through all of the toolbars
    For i = 1 To Application.Toolbars.Count
        'Store the visible property of each toolbar in the array
        'CurrentToolSet
        CurrentToolSet(i) = Application.Toolbars(i).Visible
        'Hide all of the toolbars
        Application.Toolbars(i).Visible = False
    ' End of loop
    Next i
    ' Set flag to true to skip this section next time through
    Flag = True
Else
    'loop through all of the toolbars
    For i = 1 To Application.Toolbars.Count
        'restore toolbar setting to the original value as saved in the
        'Array
        Application.Toolbars(i).Visible = CurrentToolSet(i)
    'End of loop
    Next i
    'Set flag back to false to hide visible toolbars on the next time
    'this is run
    Flag = False
'End of block if statement
End If
End Sub

Additional reference words: 5.00 tool bar
```

XL5: Macro to Open the Most Recently Used File

Article ID: Q109113

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, you can access a list of the four most recently used files by selecting the File menu. The following macro opens the first file in that list:

MORE INFORMATION

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide." Further description goes under this heading. You should provide details of how the feature works. If it's by design but not what you would normally expect to happen, describe what does occur and why (if you have that information).

The following macro will open the most recently used file. Before you run this macro you must make sure that the most recently used file list is turned on. To verify that this option is selected, choose Options from the Tools menu and then select the General tab. Make sure that the Recently Used File List box is checked.

```
Sub Open_File_Macro()
   'sets a variable for "item" to the last most recently
   'open file
   'the following two lines should be entered as one single line
   Item = MenuBars("Worksheet").Menus("file")._
   MenuItems("file list").Caption
   'sets item to "file" and deletes the first 3 letters
   'from the file list
   file = Mid(Item, 4, 24)
```

```
   Workbooks.Open filename:=file  'opens the file
End Sub
```

Additional reference words: 5.00

XL5: All PageSetup Settings Are Recorded into Macro

Article ID: Q109205

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In Microsoft Excel version 5.0, if you record a Visual Basic macro which includes a Page Setup command, all of the Page Setup settings are recorded.

Because of this, you may find that running a recorded Visual Basic PageSetup function may take an unusually long amount of time, up to two minutes or more (depending on the speed of your computer). Also, the screen may flicker or blink repeatedly while the function is being executed.

CAUSE

If you record a Page Setup in a Visual Basic macro, all the settings are recorded because of the way in which page setup information is returned to the macro recording system.

The flickering occurs because of the way in which the PageSetup function updates the sheet's different Page Setup settings. The amount of flickering is related to the number of Page Setup settings you change with the PageSetup function: changing more settings results in more flickering.

WORKAROUND

After you record a Visual Basic PageSetup function, you will probably want to eliminate unneeded settings from the PageSetup function.

To prevent the flickering, set the Application.ScreenUpdating property to False before executing your PageSetup function. Then, when the PageSetup function has completed, you can set the Application.ScreenUpdating property back to True to reenable screen redraws.

MORE INFORMATION

Visual Basic Code Examples

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided "as is" and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by

an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the Visual Basic "User's Guide."

The first Visual Basic code example shows the results of recording a Visual Basic PageSetup function.

The second Visual Basic code example shows one way in which you can prevent the screen from flickering or blinking while a PageSetup function is being executed.

Example One - Recording a Page Setup:

1. Create a new workbook.

2. Activate a worksheet in the workbook.

3. From the Tools menu, choose the Record Macro submenu, and then choose Record New Macro.

4. In the Record New Macro dialog box, select the Options button.

5. In the language section, make sure that the "Visual Basic" option button is selected. In the "Store in" section, make sure the "This Workbook" option button is selected.

6. Select OK to begin recording.

7. From the File menu, choose Page Setup.

8. In the Page Setup dialog box, choose OK.

9. From the Tools menu, choose the Record Macro submenu, and then choose Stop Recording.

10. Activate the new Visual Basic module. Your recorded subroutine should appear similar to the following (comments have been added for explanation--they are not actually recorded).

```
'--------------------------------------------------------------------
Sub Macro1()
   With ActiveSheet.PageSetup            'this is the first part
      .PrintTitleRows = ""
      .PrintTitleColumns = ""
   End With
   ActiveSheet.PageSetup.PrintArea = ""  'this is the second part
   With ActiveSheet.PageSetup            'this is the third part
      .LeftHeader = ""
      .CenterHeader = "&A"
      .RightHeader = ""
      .LeftFooter = ""
```

```
      .CenterFooter = "Page &P"
      .RightFooter = ""
      .LeftMargin = Application.InchesToPoints(0.75)
      .RightMargin = Application.InchesToPoints(0.75)
      .TopMargin = Application.InchesToPoints(1)
      .BottomMargin = Application.InchesToPoints(1)
      .HeaderMargin = Application.InchesToPoints(0.5)
      .FooterMargin = Application.InchesToPoints(0.5)
      .PrintHeadings = False
      .PrintGridlines = True
      .PrintNotes = False
      .CenterHorizontally = False
      .CenterVertically = False
      .Orientation = xlPortrait
      .Draft = False
      .PaperSize = xlPaperLetter
      .FirstPageNumber = xlAutomatic
      .Order = xlDownThenOver
      .BlackAndWhite = False
      .Zoom = 100
   End With
End Sub
'-----------------------------------------------------------------------
```

When the PageSetup function is recorded, the settings are recorded in three parts:

1. The first part is a With-End With section which sets the PrintTitleRows and the PrintTitleColumns.

2. The second part sets the PrintArea.

3. The third part is a With-End With section, which sets all of the other settings.

If you do not actually want to change certain settings, such as .Draft, you can remove those lines from the subroutine. For example, if you only want to change the PrintTitleRows, the PrintArea, and the Orientation, you could use the following:

```
   Sub Macro1()
      With ActiveSheet.PageSetup
         .PrintTitleRows = "$1:$3"
         .PrintArea = "$A$4:$C$100"
         .Orientation = xlLandscape
      End With
   End Sub
```

Since the other settings do not need to be changed, it is not necessary to include them in the subroutine. (However, you must remove them yourself.)

Also, note that the entire PageSetup procedure can be incorporated into a single With-End With section. It is not necessary for the PrintArea, PrintTitleRows, or PrintTitleColumns settings to be changed separately from the other settings--it is only recorded that way.

Example Two - Preventing Screen Flicker While PageSetup Executes:

The following subroutine demonstrates one way in which you may prevent the screen from flickering while a PageSetup function is being executed.

```
'--------------------------------------------------------------------------
Sub PreventScreenFlicker()

   'This line turns off screen updating.
   Application.ScreenUpdating = False
   'Apply each of the following properties to the active sheet's Page
   'Setup.
   With ActiveSheet.PageSetup
      .PrintTitleRows = "$1:$3"          'set print title rows
      .PrintTitleColumns = "$A:$C"       'set print title columns
      .LeftHeader = ""                   'set the left header
     'More commands could appear before the End With. They are not
     'shown here in order to keep the example short.
   End With                              'end of With section
   'Re-enable screen updating. This line is optional: you may not need
   'or want to re-enable screen updating.
   Application.ScreenUpdating = True
End Sub
'--------------------------------------------------------------------------
```

The subroutine turns off screen updating just before executing the PageSetup function and then turns screen updating back on when the PageSetup function is complete.

If screen updating is not turned off, as each line in the With section (.PrintTitleRows, .PrintTitleColumns, and so forth) is executed, the screen may flicker slightly.

Additional reference words: 5.00

XL5: Function Subroutines Can Show Message Boxes

Article ID: Q109208

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, you can use the MsgBox function to display a message box in a Visual Basic function macro. The following Visual Basic procedure demonstrates one way to do this.

NOTE: In earlier versions of Microsoft Excel it is not possible to display a message box from a macro sheet.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following Visual Basic function subroutine illustrates one way to display a message box in a function.

1. In a new Visual Basic module in a new workbook, enter the following subroutine:

```
'---------------------------------------------------------------------
'This function accepts two arguments: X, an integer, and Y, an integer.
'It returns an integer.

Function Test(X As Integer, Y As Integer) As Integer
   'Multiply X by Y to get the value of Test.
   Test = X * Y
   'Shows the value of Test in a message box.
   MsgBox "The result is " & Test & "."
End Function
```

'---

2. In a new worksheet in the same workbook, enter the following formulas:

   A1: =Test(4,5)
   A2: =Test(3,6)
   A3: =Test(20,20)

   After you enter each formula, a message box will be displayed. The messages shown are the following:

   The result is 20.
   The result is 18.
   The result is 200.

NOTE: If your Visual Basic function subroutine, includes the line

   Application.Volatile

a message box will be shown for each cell that uses the subroutine EVERY TIME YOU RECALCULATE THE WORKSHEET. For example, if you insert the Application.Volatile line into your function subroutine and then enter 500 formulas that use that subroutine, each time you recalculate the worksheet, you will receive 500 message boxes. In this case, there is no way to prevent the display of the message boxes as the worksheet is recalculated.

For this reason, it is recommended that you not use the MsgBox function and Application.Volatile in the same function subroutine.

Additional reference words: 5.00

XL5 Err Msg: "Not Enough Memory" With Indirect Defined Names

Article ID: Q109209

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, defined names are normally limited to 20 levels of indirection. If you use a defined name that exceeds this limit, you will receive the "Not enough memory" error message.

MORE INFORMATION

A level of indirection in defined names is created when one name refers to another name or to a cell reference.

If you have the following names defined in a workbook (Book1)

Name	Refers to	Level of indirection
Test1	=Sheet1!$A$1	1
Test2	=Book1!Test1	2
Test3	=Book1!Test2	3
Test4	=Book1!Test3	4
Test5	=Book1!Test4	5
Test6	=Book1!Test5	6
Test7	=Book1!Test6	7
Test8	=Book1!Test7	8
Test9	=Book1!Test8	9
Test10	=Book1!Test9	10
Test11	=Book1!Test10	11
Test12	=Book1!Test11	12
Test13	=Book1!Test12	13
Test14	=Book1!Test13	14
Test15	=Book1!Test14	15
Test16	=Book1!Test15	16
Test17	=Book1!Test16	17
Test18	=Book1!Test17	18
Test19	=Book1!Test18	19
Test20	=Book1!Test19	20 (the limit)
Test21	=Book1!Test20	21

where each line represents one level of indirection, and if you attempt to use a name that exceeds the 20th level of indirection, you will receive the "Not enough memory" error message.

You may also receive this error message if you try to delete a name that has too many levels of indirection below it. For example, if you

try to delete the name "Test1", you may receive the error message because the name "Test21" is 20 or more levels of indirection away.

Steps to Reproduce Problem

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

To create an example that demonstrates what can happen if you exceed the limit of 20 levels of indirection, do the following:

1. Open a new workbook (Book1) that contains a Visual Basic module (Module1) and a worksheet (Sheet1) and does not contain any defined names. In the Visual Basic module, type the following:

```
'-----------------------------------------------------------------------
Option Explicit

Sub CreateNames()
   'Dimension some variables.
   Dim X As Integer
   'Create the name "Test1", which refers to Sheet1!$A$1.
   ActiveWorkbook.Names.Add Name:="Test1", RefersTo:="=Sheet1!$A$1"
   'Iterate through the loop 20 times, creating the names "Test2"
   'through "Test21".
   For X = 2 To 21
   'Create the name "Test(X)", which refers to the name "Test(X-1)".
   'For example, Test2 refers to Test1, Test3 refers to Test2, etc.
   'IMPORTANT: These two lines should be entered as one line.
      ActiveWorkbook.Names.Add Name:="Test" & X, _
         RefersTo:="=Book1!Test" & X - 1
   Next                                  'loop until all done
End Sub
'-----------------------------------------------------------------------
```

2. To run the CreateNames() subroutine, position the insertion point in the line that reads Sub CreateNames() and press the F5 key.

   When you run the CreateNames subroutine, you will have 21 names defined on Sheet1. Each name Test[X], where [X] is a number from 2 to 21, refers to Test[X-1]. Test1 refers to Sheet1!$A$1.

3. On Sheet1, enter the following values:

   B1: =Test20
   B2: =Test21

   When you enter the formula =Test21, you will receive the "Not enough memory" error message message. This is because the name Test21 is at the 21st level of indirection relative to Test1, to which it ultimately refers. The "=Test20" formula works because Test20 is only at the 20th level of indirection relative to Test1.

4. From the Insert menu, choose Name, and then choose Define.

5. From the Names In Workbook list, select Test1 and choose the Delete button.

   Again, you will receive the "Not enough memory" error message. The name Test1 will not be deleted.

NOTE: If you delete the name Test21 first, you can delete Test1 because you are then within the 20 levels of indirection limit.

Additional reference words: 5.00

XL5: Verifying the Value of a Check Box

Article ID: Q109418

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, check boxes can have one of three values: xlOn, xlOff, or xlMixed.

To find the value of a check box, you can use Visual Basic code. An example of how to do this is shown below. Or, if the check box has been linked to a cell, you can use Visual Basic commands to determine the value of that cell.

MORE INFORMATION

If a check box is linked to a cell on a worksheet, the cell will display one of three different values:

Value	Check Box Status
TRUE	Selected (On)
FALSE	Cleared (Off)
#N/A	Mixed

To examine the value of the check box, examine the value of the cell to which it has been linked.

To link a check box to a cell on a worksheet:

1. In the dialog sheet, select the check box.

2. From the Format menu, choose Object.

3. Select the Control tab.

4. In the Cell Link box, enter the appropriate cell reference. For example, "Sheet1!$A$1" (without the quotation marks).

5. Choose OK to accept the change.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure

is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following example assumes that you have a dialog sheet (Dialog1) that contains one or more check boxes and is located in the same workbook as the Visual Basic module. It checks the value of the check boxes, regardless of whether they have been linked to cells.

To run the example, position the insertion point in the line that reads "Sub ExamineCheckBoxes()" and either press the F5 key or choose Start from the Run menu.

```
'--------------------------------------------------------------------
Option Explicit

Sub ExamineCheckBoxes()

   'Declare the variable used as the For Each loop counter
   Dim Box as CheckBox

   'Iterate once for each check box in the dialog sheet.
   For Each Box in DialogSheets("Dialog1").CheckBoxes
      'Get the value of the current check box.
      Select Case Box.Value
      Case xlOn                           'if the check box is ON
         MsgBox "Check box is on!"        'show the "On" message
      Case xlOff                          'if the check box is OFF
         MsgBox "Check box is off!"       'show the "Off" message
      Case xlMixed                        'if the check box is MIXED
         MsgBox "Check box is mixed!"     'show the "Mixed" message
      End Select
   'Repeat the loop until all done.
   Next Box
End Sub
'--------------------------------------------------------------------
```

This subroutine will examine each check box in the dialog sheet and display its value.

Additional reference words: 5.00

XL5: Cannot Use Array As Source Argument with SeriesCollection

Article ID: Q109575

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In the Microsoft Visual Basic Programming System, Applications Edition, you cannot add a new series to a chart in Microsoft Excel using an array as the Source argument with the Add method of the SeriesCollection object.

WORKAROUND

To add a new series to a chart in Microsoft Excel using the Add method of the SeriesCollection object, use a Range as the Source argument. For example, to add the range C1:C4 on Sheet1 as a new series to the active chart, use the following syntax:

```
ActiveChart.SeriesCollection.Add source:=Sheets("Sheet1").Range("C1:C4")
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide.

MORE INFORMATION

The information in the Visual Basic Reference in Help incorrectly states that the source argument of the Add method (SeriesCollection) specifies the new data and can be either a Range or an array of data points. However, if you specify the new data as array of data points, you receive the following error message and your data points are not added to the chart:

```
Run-time error '1004':
```

Reference is not valid.

REFERENCES
==========

For more information about the Add Method (SeriesCollection), choose the Search button in Help and type:

Add Method

For more information about the Array Function, choose the Search button in Help and type:

array

Additional reference words: 5.00

XL5: Branching to Other Sections of Code with GoTo and Call

Article ID: Q109780

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, you can use the GoTo and Call statements to branch to other sections of a Visual Basic procedure.

NOTE: In earlier versions of Microsoft Excel, you can do this with the GOTO() and RUN() functions.

MORE INFORMATION

GoTo

The GoTo statement jumps to and executes the instructions at the specific line label and continues executing until the end of the program is reached.

Call

The Call statement runs a procedure and then returns to the line immediately after the Call statement in the originating procedure. The Call statement can also pass arguments of any declared data type to the called procedure and can transfer control to a sub procedure, function procedure, dynamic link library (DLL) procedure, or Macintosh code resource procedure.

Visual Basic Examples

The following examples demonstrate how to use GoTo and Call.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for

Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Example 1--The Difference Between Call and GoTo

The following Visual Basic procedure demonstrates the difference between using the GoTo statement and the Call statement.

```
Sub One()
   MsgBox ("One()")       'Displays message box
   ' Sends control to Line10 below
   ' The GoTo Line10 statement directs the program to branch to the
   ' line labeled "Line10:", the message box "unexecuted code" is skipped.
   GoTo Line10
   ' This message box is not displayed
   MsgBox ("unexecuted code")
Line10:
   ' Displays message box
   MsgBox ("Line10 of One()")
   ' The "Call Two()" line executes the subroutine defined as "Two()",
   ' sends a message box indicating that the subprocedure is executing,
   ' then returns to Sub One executing the line immediately after the Call
   ' statement.
   ' Call made to subprocedure named two()
   Call Two
   ' Message box displayed
   MsgBox ("back to One(); returning from Two()")
End Sub

Sub Two()
   'Message box displayed
   MsgBox ("Two()")
End Sub
```

NOTE: The sample code above does not pass any arguments.

Example 2--An Alternative to the Call Statement

The following Visual Basic code makes a call to a another subprocedure without using the Call statement

```
Sub One()
   MsgBox ("One()")              'Message box displayed
   two                           'Call made to subprocedure named two()
   MsgBox ("return from Two()")  'Message box displayed
End Sub

Sub Two()
   MsgBox ("Two()")          'Message box displayed
End Sub
```

Example 3--Using GoTo for Conditional Branching

The following code shows how you can combine the IF...Then...Else statement with the GoTo statement to provide more branching options or to create the ability to return to a specific location after the GoTo is executed.

```
Sub GetInput()

   DIM number as Integer ' used for input variable
   'User input requested. Val() to turn input text into a number.
   number = Val(InputBox("Enter a 1 or a 2."))
   'Condition that is evaluated based on user input
   If number = 1 Or number = 2 Then
      GoTo Line1
   Else
      GoTo Line2
   End If
Line1:
   MsgBox ("Great! You entered a " & number & ".")
   GoTo LastLine
Line2:
   MsgBox ("Sorry, you must enter a 1 or a 2.")
LastLine:
   MsgBox ("End of program.")
End Sub
```

Additional reference words: 5.00

XL5: Visual Basic and the Microsoft Excel Version 4.0

Article ID: Q109976

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel, there are two programming languages: the Microsoft Visual Basic Programming System, Applications Edition, and the Microsoft Excel version 4.0 macro language. Microsoft included both languages to ease the transition to Visual Basic. Although both languages give you complete control of Microsoft Excel, you should use the Visual Basic language because it is more flexible and powerful than the Microsoft Excel version 4.0 macro language. In addition, Visual Basic will be the only programming language offered in future versions of Microsoft Excel.

In Microsoft Excel version 5.0, you can still run your existing Microsoft Excel 4.0 macros. You can also combine Visual Basic and Microsoft Excel version 4.0 macros: you can run your existing version 4.0 macros from Visual Basic procedures and you can run Visual Basic procedures from your version 4.0 macros.

Future versions of Microsoft Excel will still support running your Microsoft Excel version 4.0 macros, but will not include enhanced Microsoft Excel version 4.0 macro language commands for new features.

The following is a list of resources you can use to learn Microsoft Visual Basic for Microsoft Excel.

Online Help

Online help is provided for all Visual Basic and Microsoft Excel version 4.0 macro commands to help make the transition easy.

Also provided is an online reference of Visual Basic Equivalents For Macro Functions and Commands. To access this list, do the following:

1. In Microsoft Excel, bring up the Help Contents screen by choosing Contents from the Help menu.

2. Choose the Reference Information icon.

3. Under General Reference, choose Microsoft Excel Macro Functions Contents.

4. Choose Visual Basic Equivalents for Macro Functions and Commands.

Reference Materials

"Microsoft Excel Visual Basic for Applications Reference"
Microsoft Corporation
$24.95 ($33.95 Canada)
ISBN 1-55615-624-3
(Available February 1994)

"Microsoft Excel Visual Basic for Applications Step by Step"
Reed Jacobson,
$29.95 ($39.95 Canada)
ISBN is 1-55615-589-1
(Available January 1994)

Microsoft Press books can be found in bookstores everywhere, through CompuServe electronic mail (type GO MSP), or you can order direct by calling (800) MSPRESS. In Canada, call (416) 293-8464, extension 340.

If you are outside the United States, contact the Microsoft subsidiary for your area. To locate your subsidiary, call Microsoft International Customer Service at (206) 936-8661.

Excel 5 Courseware Developer's Kit (CDK)
Microsoft Corporation
$895 US ($1195 Canada)
(Win) ISBN 1-55615-719-3
(Available in February)

Includes license agreement (grants permission to customize the student book, the student practice files, and the instructor outline files for reproduction), eight disks (student practice files and student book on disk), Instructor's guide, certificate of completion master copy, hardcopy overhead and handout masters, and a template for creating your own lessons

Microsoft Excel 5 Software Development Kit
$49.95 ($69.95 Canada)
ISBN 1-55615-632-4
(Available in April)

Additional reference words: 5.00

XL5: Using Visual Basic to Exit Windows From Within Excel

Article ID: Q110005

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

You can exit Windows from within Microsoft Excel version 5.0 by using a Visual Basic macro. The procedure below makes a call to a Windows DLL (dynamic-link library) that is similar to choosing the Exit command from the File menu in Program Manager.

Other Windows applications and instances of Microsoft Excel will prompt you to confirm that you want to quit (just as if you had exited Windows manually). If you choose Cancel when you are prompted to save a file, the exit request is canceled as well.

MORE INFORMATION

CAUTION: The Declare, Call and Register Functions listed in the following macro are very sensitive. If used incorrectly, these functions may cause a general protection fault (GPF) in Windows or cause other software problems.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty, either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following procedure will allow you to exit Windows from Microsoft Excel:

```
'The following Declare statement should be entered on a single line
Declare Function ExitWindows Lib "User" (ByVal dwReturnCode As Long,
ByVal wReserved  As Integer) As Integer
```

```
Sub ExitWin
   Call ExitWindows(1,0)
End Sub
```

This will cause Windows to close all open applications immediately and return to MS-DOS.

CAUTION: Any sheets that you are editing in the current instance of Microsoft Excel (the instance from which the macro is run) will be closed without confirmation. You will lose any changes made since the last time these files were saved. Other Windows applications and instances of Microsoft Excel prompt you for confirmation (just as if you had exited Windows manually).

REFERENCES

Microsoft Windows SDK
Microsoft Visual Basic 3.0 Professional - WINAPI31.HLP help file

Additional reference words: 5.00 Sdk dynamic link

XL5: Update Remote References Option Selected by Default

Article ID: Q110006

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

When you open a workbook in Microsoft Excel version 5.0, the Update Remote References calculation setting for the workbook is usually selected. The exception to this rule occurs when a workbook has been linked to an external data source and you do not allow Microsoft Excel to update links to the external data source.

MORE INFORMATION

The Update Remote References calculation setting determines whether or not Microsoft Excel updates formulas that include references to external data sources (such as Microsoft Word for Windows documents or Microsoft Access databases). Note that Microsoft Excel workbooks and documents are NOT external data sources.

When you open a workbook, the Update Remote References setting will be set according to the following table:

Condition	Update Remote References Check Box
Workbook not linked to any other documents	Selected
Workbook linked to other workbooks or Microsoft Excel documents	Selected
Workbook linked to external data source and you select No when you receive the prompt "This document contains links. Re-establish links?"	Cleared (not selected)
Workbook linked to external data source and you select Yes when you receive the prompt "This document contains links. Re-establish links?"	Selected

To clear the Update Remote References setting in the active workbook:

1. From the Tools menu, choose Options.

2. Select the Calculation tab.

3. Clear the Update Remote References check box.

4. Choose OK to accept the change.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following Visual Basic code example assumes that you have a file called TEST.XLS, which is located in the C:\EXCEL directory. The example opens the TEST.XLS file and then clears (turns off) the Update Remote References check box. As the subroutine opens the file, it also prevents Microsoft Excel from updating any of the workbook's external or remote references.

```
'-----------------------------------------------------------------------
Sub Example()
   'The zero after updateLinks indicates that neither external nor
   'remote references should be updated when the file is opened.
   Workbooks.Open fileName:="C:\EXCEL\TEST.XLS", updateLinks:=0
   'Turn off the Update Remote Referencessetting for the workbook.
   ActiveWorkbook.UpdateRemoteReferences = False
End Sub
'-----------------------------------------------------------------------
```

Note that the Update Remote References check box is not a universal setting: if two workbooks are open, one workbook can have its Update Remote References setting selected, while in the other workbook the check box is cleared. However, when you open a workbook, its Update Remote References check box will be selected, no matter how it was saved, unless it contains links to an external data source.

Additional reference words: 5.00 calc

XL5: Can't Use Line Continuation Character in Some Locations

Article ID: Q110237

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In the Microsoft Visual Basic Programming System, Applications Edition, when you use the line-continuation character(_) in your procedure, you may receive one of the following error messages:

   Syntax error

   -or-

   Invalid character

   -or-

   General protection (GP) fault (only if you use the line-continuation character within square brackets)

CAUSE

The line-continuation character indicates that a logical line of code is continued from one physical line in the procedure to the next. Line continuation is detected in a Visual Basic procedure by searching for a space followed by an underscore (_) at the end of a line. These characters are then removed before the line of code is parsed. However, you cannot use the line continuation character in the following locations in your code.

Between Quotation Marks

```
   A = "This is an example of assigning a very long _
   string variable"
```

Between a Colon and an Equal Sign

```
   Cells.CheckSpelling CustomDictionary:="CUSTOM.DIC", IgnoreUppercase: _
   =False, AlwaysSuggest:=True
```

Within Square Brackets

```
   [Test _
   ]
```

WORKAROUND
==========

You should only use the line continuation character in your procedure when you have a very long line of code that is wider than your screen. In general, you can use the line continuation character in the following locations:

- Before or after a comma

- Before or after an equal sign (except between a colon and equal sign)

- Before or after a colon and an equal sign (:=)

- Before or after an operator (such as &, +, -, LIKE, NOT, AND)

The following are examples of using the line continuation character that do not cause an error message when they are run:

```
A = "This is an example of assigning a very long " & _
"string variable"

A = _
"This is an example of assigning a very long string variable"

A _
= "This is an example of assigning a very long string variable"

Cells.CheckSpelling CustomDictionary:="CUSTOM.DIC", _
IgnoreUppercase:=False, AlwaysSuggest:=True

Cells.CheckSpelling CustomDictionary:="CUSTOM.DIC", IgnoreUppercase _
:=False, AlwaysSuggest:=True

Cells.CheckSpelling CustomDictionary:="CUSTOM.DIC", IgnoreUppercase:= _
False, AlwaysSuggest:=True

ActiveSheet.PageSetup.LeftHeader = _
"this is an example of a very long left header for your worksheet"

Answer = NOT _
(A<B)
```

NOTE: Do not use the line-continuation character within square brackets. There is no workaround for this behavior.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that

code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide.

NOTE: Though you can type the lines in the above examples without the line-continuation character as one physical line, type them exactly as shown above to illustrate using the line-continuation character.

Additional reference words: 5.00 gpf

XL5: List of the Files You Need to Run Microsoft Excel

Article ID: Q110594

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY
=======

The following tables contain the filenames and descriptions of the files that you need to be able to run the various components of Microsoft Excel version 5.0.

Files needed to run Microsoft Excel:

Filename	Description
COMMTB.DLL	Microsoft Button Editor DLL
COMPOBJ.DLL	OLE 2.0 Library
EXCEL.EXE	Microsoft Excel Executable
MSFFILE.DLL	Microsoft Find File Library
OLE2.DLL	OLE 2.0 Library
OLE2DISP.DLL	OLE Automation Library
OLE2NLS.DLL	OLE NLS Library (Intl character set translation searching and sorting)
OLE2PROX.DLL	OLE2PROXY 2.0 Library
SCP.DLL	Code Page Translation Library
SDM.DLL	Standard Dialog Manager
STDOLE.TLB	OLE Type Library
STORAGE.DLL	OLE 2.0 Library (docfile DLL)
TYPELIB.DLL	OLE Automation Type Information Interfaces
XLINTL.DLL	Microsoft Excel Localized Resources
XLEN50.OLB	Microsoft Excel Object Library
VBAEN.DLL	Visual Basic for Applications International Resources
VBAEN.OLB	Visual Basic for Applications Object Library
VBA.DLL	Visual Basic for Applications Development Environment

Files needed to run Help:

Filename	Description
MAINXL.HLP	- Microsoft Excel Main Help
SHARERES.DLL	- Shared Resources
XLHELP.DLL	- Microsoft Excel Help Extended Macros

Files needed to run Microsoft Query

Filename	Description

COMMDLG.DLL   - Common Dialogs Libraries
MSQUERY.EXE   - Microsoft Query Executable
QRYINTL.DLL   - Microsoft Query International DLL

For a list of all the files installed with Microsoft Excel 5.0, including file size and location on the hard drive, open FILELIST.TXT located in the Microsoft Excel directory.

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

Microsoft Excel requires that you have the Share program loaded on your computer before you run Microsoft Excel. For more information about the Share program, query on the following in the Microsoft Knowledge Base:

   share.exe and load and must

Additional reference words: 5.00 required setup

XL5: GP Fault If REFTEXT() Refers to Closed Workbook

Article ID: Q110595

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In Microsoft Excel version 5.0, when you use the REFTEXT() function in a Microsoft Excel 4.0 macro sheet, and the reference includes the name of a closed workbook, you receive a general protection (GP) fault.

WORKAROUND

To avoid receiving the GP fault when you use the REFTEXT() function and the reference contains a workbook name, do the following:

1. Before you type the function in the macro, open the workbook you want to reference in the REFTEXT() function.

2. Before you run your macro, do either of the following :

   - Insert a command in your macro that opens the workbook before the REFTEXT() function runs.

     -or-

   - Manually open the workbook before running the macro.

To convert a reference to a reference in the form of text, you can use the following Microsoft Visual Basic Programming System, Applications Edition functions:

```
' Converts name on a workbook to reference in text form,
' where file$ is name of workbook containing name
' and ref$ is name to return reference for
Function GetNameVal(file$, ref$)
   GetNameVal = GetObject(file$, "Excel.Sheet").Parent.Names(ref$)
End Function

' Converts range reference to text
' file$ is name of workbook containing range
' sheet$ is name of worksheet containing range
' ref$ is range reference to convert to text
Function GetRangeVal(file$,sheet$,ref$)
   GetRangeVal = _
GetObject(file$,"Excel.Sheet").Parent.Sheets(sheet$).Range(ref$).Address
End Function
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied,

including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide.

MORE INFORMATION

The REFTEXT() function converts a reference to an absolute reference in the form of text and is useful when you need to manipulate references with text functions. This function uses the following syntax

```
   REFTEXT(reference, a1)
```

where reference is the reference you want to convert. When you type this command in your macro and press ENTER, or when you run a macro that contains this command, you receive a GP fault if the reference argument contains the name of a closed workbook.

REFERENCES

For more information about the Address Method, choose the Search button in Help and type:

```
    Address Method
```

For more information about the Name Object, choose the Search button in Help and type:

```
    Name Object
```

Additional reference words: 5.00

XL5: Macro to Find Directory in WIN.INI for an Application

Article ID: Q110692

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel 5.0 for Windows, you can create a macro to find the default directory of any application that registers itself in the [EXTENSIONS] section of the WIN.INI file. Given the three-character extension for the application's file, the macro below uses the Microsoft Windows dynamic-link libraries (DLLs) to return the default directory information.

MORE INFORMATION

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Sample Macro Code

```
Option Explicit

'Enter the following declare statement needs as a 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

Sub Application_Directory()
   Dim Path As String * 255, Ext As String, Directory As String
   Dim Back_Slash As Integer, Length As Integer
   Ext = InputBox("Enter the 3 letter extension of the application.")
   'If Cancel was chosen then end procedure
```

```
   If Len(Ext) = 0 Then Exit Sub
   'Gets information from the WIN.INI file and stores in the "path"
   'variable
   Length = GetProfileString("Extensions", Ext, "", Path, Len(Path))
   'Tests for no application found
   If InStr(Path, "\") = 0 Then
       MsgBox "Application not found."
   Else
       Back_Slash = 255
       While Mid(Path, Back_Slash, 1) <> "\"
           Back_Slash = Back_Slash - 1
       Wend
       Directory = Left(Path, Back_Slash - 1)
       MsgBox "The directory of the application which uses the " & _
          "extension '" & Ext & "' is " & Directory & "."
   End If
End Sub
```

To run this macro, place the insertion point anywhere in the Sub Application_Directory() line and either press F5 or choose Start from the Run menu.

When you are prompted to enter a three-letter extension, enter an extension and choose OK. The macro will use the Microsoft Windows API to determine which directory contains the application that uses that three-letter extension. For example, if you enter "xls" (without the quotation marks), the message box will tell you where the Microsoft Excel application is installed.

MORE INFORMATION

For information about creating a similar Visual Basic macro to find the default printer and port assignments, query on the following words in the Microsoft Knowledge Base:

   device and port and visual and basic and printer

REFERENCES

Microsoft Windows SDK
Microsoft Visual Basic 3.0 Professional - WINAPI31.HLP help file

Additional reference words: 5.00 5.0 call register

XL5: Visual Basic Example to Delete Blank Rows

Article ID: Q110759

The information in this article applies to:

- Microsoft Excel version 5.0

SUMMARY
=======

The following Visual Basic code locates blank cells within a range and deletes the entire row that contains that cell. The code brings up a dialog box that prompts the user for the total number of rows to process. The macro begins at the active cell and moves down the specified number of rows.

Visual Basic Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

```
'The following code deletes blank rows from the active worksheet.

Option Explicit

Dim Count
Dim i As Integer

Sub DelRow()

'Input box to determine the total number of rows in the worksheet.
Count = InputBox("Enter the total number of rows to process")

'Loops through the desired number of rows.
For i = 1 To Count
    'Checks to see if the active cell is blank.
    If ActiveCell = "" Then
        Selection.EntireRow.Delete
```

```
        'Decrements count each time a row is deleted. This ensures
                'that the macro will not run past the last row.
            Count = Count - 1
        Else
            'Selects the next cell.
            ActiveCell.Offset(1, 0).Select
        End If

Next i

End Sub
```

Note: you can also delete lines that contain spaces or other non-display characters. To do this, change the line that reads

```
   If ActiveCell = "" Then
```

-to-

```
   If Len(Trim(ActiveCell)) = 0 Then
```

and those lines will also be deleted.

Additional reference words: 5.00 VBAppCode

Computing Periodic Annual Interest Rate in Microsoft Excel

Article ID: Q110854

The information in this article applies to:

- Microsoft Excel for Windows, versions 2.x, 3.0, 4.0, 4.0a, 5.0
- Microsoft Excel for the Macintosh, versions 2.x, 3.0, 4.0
- Microsoft Excel for the OS/2, versions 2.x, 3.0

SUMMARY

The RATE() function in Microsoft Excel, and the @RATE function in Lotus 1-2-3 both return the periodic interest rate necessary for an investment to grow to a specific value over a specified number of compounding periods. However, in Microsoft Excel, RATE() assumes a known stream of payments. If the payment amount per period is not known, you can still the RATE() function, but you must modify it slightly.

A manually entered formula must be used to determine the annualized interest rate for an investment with more than one compounding period per year. You can also use a custom function to make this calculation easier.

MORE INFORMATION

To use the RATE() function in Microsoft Excel when the payment per period is not known, you must enter the number of compounding periods, the present value as a negative number, and the future value.

For Example, an initial investment of $1,000, that has grown to $2,000 when compounded quarterly over six years would return 2.93% per quarter when entered as follows:

```
=RATE(24,,-1000,2000)
```

To determine the annualized interest rate for an investment, use the following formula

```
=((FV/PV)^(1/N)^M)-1
```

where FV is the future value of an investment, PV is the initial investment, N is the total number of compounding periods, and M is the number of compounding periods per year.

The following formula, when used with the numbers from the previous example, will return 12.2462% per year:

```
=((2000/1000)^(1/24)^4)-1
```

To create a function macro in version 5.0

To create this function using Visual Basic, Applications Edition, enter the following in a new Visual Basic module:

```
Function Yearly_Rate(FV, PV, N, M) As Double     'Defines variables
    Yearly_Rate = ((FV / PV) ^ (1 / N)^M) - 1    'Performs computation
End Function                                     'Ends function
```

To use the custom function:

1. Enter the following values on a worksheet:

```
A1:   20000
A2:   10000
A3:      72
A4:      12
```

2. In any blank cell, enter the following formula:

```
=Yearly_Rate(A1,A2,A3,A4)
```

The annual interest rate returned is 12.2462%.

To create a function macro for versions 2.x, 3.0 and 4.0

You can use a custom function in Microsoft Excel to compute the annual interest rate. To create this function enter the following in a Microsoft Excel macro sheet:

```
B1:  Yearly_Rate
B2:  =RESULT(1)
B3:  =ARGUMENT("FV",1)
B4:  =ARGUMENT("PV",1)
B5:  =ARGUMENT("N",1)
B6:  =ARGUMENT("M",1)
B7:  =RETURN((((FV/PV)^(1/N)^M))-1)
```

After entering the above formulas on a Microsoft Excel macro sheet, define it as a function macro. For information on defining function macros see the Microsoft Excel User's Guide 2, pages 203-213.

Example

To compute the periodic interest rate for a $10,000 investment that matures in six years, compounded monthly, with a maturity value of $20,000, do the following:

1. Enter the following on a Microsoft Excel worksheet:

```
A1:   20000
A2:   10000
A3:      72
A4:      12
```

2. Select cell B1, choose Paste Function from the formula menu.

3. Select the User Defined category, and then scroll to the bottom of the Paste Function list.

4. Choose your custom function and choose the OK button.

   If the custom function macro is placed on a sheet named "RATE.XLM", the following formula will appear in cell B1.

   =RATE.XLM!Yearly_Rate(FV,PV,N,M)

5. Enter the values "A1", "A2", "A3", and "A4" (without the quotation marks)in place of the arguments in the formula (FV, PV, N, and M) and press ENTER.

The annual interest rate 12.2462% is returned.

REFERENCES
==========

"Visual Basic User's Guide," version 5.0, pages 41-48
"User's Guide 2," version 4.0, pages 203-213
"Function Reference," version 4.0 pages 347-348
"Lotus 1-2-3 for DOS 2.3, @Functions and Macro Guide," Pages 69-70

Additional reference words: 2.0 3.0 4.0 4.0a 5.0 growth

XL5 Err: "Call to Undefined Dynalink" in Visual Basic Macro

Article ID: Q111089

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

When a Microsoft Excel version 5.0 Visual Basic for Applications macro is executing procedures to manipulate dialog boxes, you may receive the following error message:

```
Application Error, Call to Undefined Dynalink.
```

CAUSE

The files OLE2CONV.DLL, OLE2DISP.DLL, and OLE2NLS.DLL are used by many applications for OLE version 2.0 support. Applications other than Microsoft Excel version 5.0 may install older versions of these files. These outdated versions can cause the above error message.

Third Party Applications

The following applications are known to cause the above error message.

Corel Draw:

Corel Draw version 4.00.B3 installs OLE2DISP.DLL into the WINDOWS\SYSTEM directory. The file is an older version than the OLE2DISP.DLL that ships with Microsoft Excel version 5.0.

The older .DLL that Ships with Corel Draw version 4.0 does not recognize the new procedures that Microsoft Excel version 5.0 can pass to OLE 2.0 objects (such as manipulating dialog boxes in Microsoft Excel 5.0). Listed below are the OLE 2.0 DLL files that Corel Draw will overwrite.

Corel Draw	Date	Size
OLE2CONV.DLL	7/7/93	57328
OLE2DISP.DLL	7/7/93	80384
OLE2NLS.DLL	7/7/93	26624

RESOLUTION

To resolve this problem, verify that the files OLE2CONV.DLL, OLE2DISP.DLL, and OLE2NLS.DLL are the correct size for Microsoft Excel version 5.0. Below is a list of these files and the sizes necessary for Microsoft Excel version 5.0.

Filename	Date	Size
OLE2CONV.DLL	12-14-93	57328
OLE2DISP.DLL	12-14-93	90144
OLE2NLS.DLL	12-14-93	99200

If any of the above files are not the correct size, use the decompression utility (DECOMP.EXE) that is shipped with Microsoft Excel version 5.0 to decompress the files to the correct size. These files are located on Disk 7 of the 3.5-inch disks and Disk 9 of the 5.25-inch disks.

To use the decompression utility:

1. Copy the DECOMP.EXE file to your hard drive.

2. Exit Windows completely.

3. Change to the drive\directory that you copied DECOMP.EXE to.

4. Insert the Microsoft Excel disk containing the OLE*.* files into the appropriate floppy drive.

   For 3.5" HD disks, use Disk 7.

5. Decompress the file to your WINDOWS\SYSTEM directory.

   Below is an example of the syntax that would be used to decompress OLE2CONV.DLL:

```
decomp a:\ole2conv.dl_ c:\windows\system\ole2conv.dll
```

Additional reference words: 5.00

XL5 OLE Automation: Workbook Hidden Using GetObject Function

Article ID: Q111247

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

If you use the GetObject function in a Microsoft Visual Basic version 3.0 procedure to open and save a Microsoft Excel workbook, the workbook is saved as hidden and does not appear when you open the workbook again.

WORKAROUND

To avoid saving a workbook as hidden when you use the GetObject function in a Visual Basic procedure to open and save the workbook, unhide the workbook before saving it, as in the following Visual Basic procedure:

```
' Dimension variable xl as Object type
Dim xl As Object
' Set xl equal to Excel workbook as OLE Automation object
' and open file BOOK1.XLS located in c:\excel5
Set xl = GetObject("C:\excel5\book1.xls")
' Make the hidden workbook visible
xl.Parent.Windows("BOOK1.XLS").Visible = True
' Save the file BOOK1.XLS
xl.Parent.Save
' Close the file BOOK1.XLS
xl.Parent.[Close]
' Free the memory used for storing xl variable
Set xl = Nothing
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide.

MORE INFORMATION

Note that the above behavior also occurs when you use the CreateObject function in a Visual Basic, Applications Edition, procedure to open and save a Microsoft Excel workbook. However, because Microsoft Excel has an object library, use the functions defined in that library rather than the GetObject or CreateObject function when you access Microsoft Excel objects in a Visual Basic, Applications Edition, procedure.

Additional reference words: 5.00 3.00 B_VBasic

XL5: Calculating Elapsed Time for a Visual Basic Procedure

Article ID: Q111268

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Visual Basic, Applications Edition, you can time procedures, statements, and functions by entering the Timer function before and after the statements that you want to time.

MORE INFORMATION

The Visual Basic Timer function can be used to record the starting and ending times for a series of commands. The following example sets the variable StartTime to the current system time, runs the code to be timed, then sets the variable EndTime to the current system time. Finally, the elapsed time between StartTime and EndTime is printed to the debug window using the Debug.Print command.

Example of Visual Basic Code

```
Sub ElapsedTime()
   Dim StartTime!, EndTime!
   StartTime! = Timer 'Stores start time in variable "StartTime"

   'Place code to be timed here

   EndTime! = Timer 'Stores end time in variable "EndTime"
   'Prints execution time in the debug window
   Debug.Print "Execution time in seconds: ", EndTime! - StartTime!
End Sub
```

The following Visual Basic macro creates a text file containing a list of error messages that can be generated in Visual Basic, Applications Edition. The amount of time it takes to accomplish this procedure will be displayed in a message box.

Example of Visual Basic Code

```
Sub ErrorCodes()
   Dim StartTime!, EndTime!, x
   'creates a file of error messages
   startTime! = Timer 'Stores start time in variable "startTime"
   Open "vbaerror.txt" For Output As #1
   For x = 1 To 3300
      Print #1, x, Error$(x)
   Next x
   Close #1
```

```
   endTime! = Timer 'Stores end time in variable "endTime"
   'Shows Message Box with elapsed time
   MsgBox "Execution time in seconds: " + Format$(EndTime! - StartTime!)
End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

REFERENCES

"Visual Basic User's Guide," version 5.0, Chapter 8, page 158

Additional reference words: 5.00 timing calculating benchmark bench mark

XL5: Can't Define Name with Same Name as a Subroutine

Article ID: Q111281

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, you cannot define a new name that is the same as the name of a currently running Visual Basic subroutine. This is by design.

MORE INFORMATION

When you run a Visual Basic subroutine, Microsoft Excel creates a temporary name for the subroutine in memory. As long as this temporary name exists, it is not possible to define a new name that is the same as the temporary name. You can, however, redefine an existing name.

If you attempt to define a new name while an identically named subroutine is running, you will receive the following error message:

```
Run-time error '1004':
Add method of Names class failed
```

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following subroutines illustrate the various types of situations that may cause this error to occur. Enter the following subroutines into a new module sheet in a new workbook.

To run the subroutines, position the cursor in the Sub line for the subroutine you want to run and either press the F5 key or choose Start from the Run menu.

```
'---------------------------------------------------------------------
'The Test1 subroutine will fail if the name "Test1" does not already
'exist in the workbook.

Sub Test1()
   'Enter the following two lines as a single line.
   ActiveWorkbook.Names.Add Name:="Test1", _
      RefersToR1C1:="=Sheet1!R1C1:R10C10"
End Sub

'The Test2 subroutine runs the MakeName subroutine, which attempts to
'define the name "Test2". As with the above subroutine, it will fail if
'the name "Test2" does not already exist in the workbook.

Sub Test2()
   MakeName                              'runs the Test2 subroutine
End Sub

Sub MakeName()
'Enter the following two lines as a single line.
   ActiveWorkbook.Names.Add Name:="Test2", _
      RefersToR1C1:="=Sheet1!R1C1:R10C10"
End Sub
'---------------------------------------------------------------------
```

Additional reference words: 5.00 routine

XL5: Borders Method Applies Inconsistent Format

Article ID: Q111309

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In Microsoft Excel, if you use the Borders method in a Visual Basic procedure to set the Weight or Color property of the borders of a selection on a worksheet, the formatting of the selection may not be consistent.

CAUSE

When you use the Borders method on a selection, and there is already a border around the selection, if you set only the Color or the Weight property, two of the borders of the entire selection are formatted differently that the rest of your selection.

Take for example the range B2:C4 on a worksheet that has been formatted with a thick green outline border. If you run the following procedure

```
Sub Color()
   Dim x As Object
   Set x = Range("B2:C4")
   ' Set color of all cell borders in range to red
   x.Borders.Color = RGB(255, 0, 0)
End Sub
```

the result is a thick red border around every cell in the selection, but a thin red border on right and bottom of the selection.

If you run the following procedure

```
Sub Weight()
   Dim x As Object
   Set x = Range("B2:C4")
   ' Set thickness of all cell borders to thin
   x.Borders.Weight = xlThin
End Sub
```

the result is a thin automatic color border around every cell, but a green border on the top and left of the selection.

WORKAROUND

To avoid having unexpected results when you use the Borders method on a selection of cells on a worksheet, apply both the Weight and Color

property as in the following example:

```
Set x = Range("B2:C4")
With x.Borders
   .Color = RGB(255, 0, 0)
   .Weight = xlThick
End With
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Additional reference words: 5.00

XL5: CreateObject Function Starts Invisible Instance of Excel

Article ID: Q111311

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

When you use the CreateObject function in a Visual Basic version 3.0 procedure to create a Microsoft Excel OLE Automation object, a new instance of Microsoft Excel starts, but you cannot see it.

For example, the following Visual Basic command starts a new instance of Microsoft Excel, but you cannot see it, and it does not appear in the Task List:

```
Set x = CreateObject("Excel.Application")
```

WORKAROUND

If you want to see Microsoft Excel when you use the CreateObject function to access the application object, use the Visible property as in the following example:

```
Sub Run_Excel
   Dim x As Object
   Set x = CreateObject("Excel.Application")
   x.Visible = true
End Sub
```

MORE INFORMATION

When you use the CreateObject function to run Microsoft Excel, you start a new hidden instance of Microsoft Excel. Because this new instance uses memory and resources on your system, you should exit the program within the same procedure that runs the instance. If you don't exit the program in the procedure, the invisible instance runs until you exit Microsoft Windows.

To free the memory used by the object variable assigned to the OLE Automation object, set the variable equal to Nothing. The following Visual Basic procedure uses the CreateObject function to start an invisible instance of Microsoft Excel, quits the instance, and sets the OLE Automation object variable equal to Nothing.

```
Sub Run_Excel
   ' Dimension variable x as Object type
   Dim x As Object
   ' Set x equal to Excel object
   Set x = CreateObject("Excel.Application")
   ' Make running instance of Excel visible
```

```
      x.Visible = true
      ' Insert desired Excel commands here
      ' Quit Microsoft Excel
      x.Quit
      ' Set x equal to nothing to free memory object was using
      Set x = Nothing
   End Sub
```

CreateObject Versus GetObject Functions

CreateObject and GetObject are two Visual Basic functions that you can use to return an OLE Automation object. The CreateObject function creates an OLE Automation object; the GetObject function retrieves an OLE Automation object from a file. The way an application behaves when you start it with one of these functions depends on the application. For example, when you use the CreateObject function to access a Microsoft Excel Application object, an invisible instance of Microsoft Excel runs. When you use the CreateObject function to access a Microsoft Word WordBasic object, a visible instance of Microsoft Word runs.

Use the following list to determine the behavior of Microsoft Excel when you use the CreateObject or GetObject function:

Function	Behavior
CreateObject("Excel.Application")	Always loads a new invisible instance
GetObject("", "Excel.Application")	Always loads a new visible instance
GetObject(, "Excel.Application")	Either returns an already running instance, or fails with error message "OLE Automation server cannot create object"

Microsoft Word for Windows

When you use the CreateObject function to access a Microsoft Word for Windows WordBasic object, a new visible instance runs if Word for Windows is not currently running. If you set the variable returned by the CreateObject function equal to nothing in this case, Word for Windows is closed. Otherwise, if Word for Windows is already running, the CreateObject function uses the running instance. If you set the variable returned by the CreateObject function equal to nothing in this case, Word for Windows is not closed, because the instance was running before you ran the procedure.

The following Visual Basic procedure uses the CreateObject function to access the Word for Windows WordBasic object, performs some Word commands, and then sets the OLE Automation object variable equal to Nothing.

```
   Sub Run_Word ()
      ' Dimension variable word as Object type
```

```
    Dim word As Object
    ' Set word equal to Word for Windows object
    ' Start Word for Windows if not already running
    Set word = CreateObject("Word.Basic")
    ' Create new file
    word.FileNew
    ' Insert text in new file
    word.Insert "Some Text"
    ' Save file as TEXT.DOC
    word.FileSaveAs "text"
    ' Quit Word if it was not already running before this procedure ran
    ' Set word equal to nothing to free memory used for object variable
    Set word = Nothing
End Sub
```

NOTE: The above information applies both to Visual Basic version 3.0, and Visual Basic, Applications Edition. However, because Microsoft Excel has an object library, use the functions defined in that library when you access Microsoft Excel objects in a Visual Basic, Applications Edition procedure, rather than the GetObject or CreateObject function. Because Microsoft Word for Windows does not have an object library, you must use the CreateObject or GetObject function to access a Microsoft Word object in any version of Visual Basic.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

For more information on the GetObject function, query on the following words in the Microsoft Knowledge Base:

   ole and automation and getobject

REFERENCES
==========

For more information about the GetObject Function and the CreateObject Function, choose the Search button in the Visual Basic Reference and type:

    OLE Automation

Additional reference words: 5.00 officeinterop WM_Word B_VBasic

Excel: Creating Macros for Different Language Versions

Article ID: Q111388

The information in this article applies to:

- Microsoft Excel for Windows, versions 4.0, 4.0a, 5.0
- Microsoft Excel for the Macintosh, version 4.0

SUMMARY
=======

In Microsoft Excel versions 4.0 and 5.0, you can determine the country code that corresponds to the version of Microsoft Excel you are running. These country codes can be helpful in creating custom applications.

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

Microsoft Excel is available in sixteen international languages. These languages and their corresponding country codes are listed below:

Chinese	86
Danish	45
Dutch	31
English	1
Finnish	358
French	33
German	49
Greek	30
Hebrew	972
Italian	39
Japanese	81
Korean	82
Norwegian	47
Portuguese	351
Spanish	34
Swedish	46

NOTE: The following language versions are not available in Microsoft Excel version 4.0 for the Macintosh: Chinese, Greek, Hebrew, Korean, and Portuguese.

In a custom application it may be necessary to determine which language version of Microsoft Excel is running. For example, if you are writing a custom application for your company, which has offices in two different countries, the country code would make it possible to write a single macro for both offices. You could display different dialog boxes based on which language version of Microsoft Excel is being used. Below are examples of returning and using the country code in a Visual Basic macro and an XLM macro.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

In Microsoft Excel version 5.0, the APPLICATION.INTERNATIONAL function can be used to return information about the current country and international settings of Microsoft Excel. The built-in constant "xlCountryCode" will return the country code of the version of Microsoft Excel you are running.

Below is a macro example that will return the country code and will then, based on that code, display "Hello" in the appropriate language:

```
Sub Code()
   Country_Code = Application.International(xlCountryCode)
   If Country_Code = 1 Then
     MsgBox ("Hello")
   ElseIf Country_Code = 34 Then
     MsgBox ("Ola")
   End If
End Sub
```

XLM Macro Code Example

Microsoft provides macro examples for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This macro is provided 'as is' and Microsoft does not guarantee that the following code can be used in all situations. Microsoft does not support modifications of that code to suit specific customer requirements.

In Microsoft Excel version 4.0, the Get.Workspace function can be used to return information about the workspace. GET.WORKSPACE has one argument, type_num. Type_Num is a number specifying the type of workspace information you want.

To return the country code of the version of Microsoft Excel you are running, use the type number of 37. Type 37 returns a 45-item horizontal array of the items related to country versions and settings. You must index this array to return a specific item. The country code is the first item in this array.

Below is a macro example that will index the array of items returned by

GET.WORKSPACE(37) to get the country code and then, based on that code, display "Hello" in the appropriate language:

```
A1: Code_Macro
A2: =INDEX(GET.WORKSPACE(37),1)
A3: =IF(A2=1)
A4: =ALERT("Hello")
A5: =ELSE.IF(A2=34)
A6: =ALERT("Ola")
A7: =END.IF()
A4: =RETURN()
```

REFERENCES
==========

"Function Reference," version 4.0, pages 209-212

Additional reference words: 4.00 4.00a 5.00 mpf foreign

XL5: Hiding Button Objects with Visual Basic for Applications

Article ID: Q111392

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

To print a worksheet without showing button objects, follow the appropriate procedure below.

If the Worksheet Does Not Contain Other Graphic Objects

1. From the Edit menu, choose Go To.

2. Choose the Special button to open the Go To Special dialog box.

3. In the Go To Special dialog box, select the Objects option to select all objects on the worksheet. Select the OK button.

4. From the Format menu, choose Object, and select the Properties tab.

5. On the Properties tab, clear the Print Object check box. Select the OK button to accept the change.

NOTE: Using the Objects option in the Go To Special dialog box activates the macros that the buttons are assigned to instead of selecting the button object.

If the Worksheet Contains Other Graphic Objects In Addition To Buttons

1. From the Insert menu, choose Macro, and then choose Module.

2. In the new module, type the following Visual Basic code:

```
'The following macro code hides only the macro button objects on the
'worksheet. To re-show the buttons, use
'
'   ActiveSheet.Buttons.Visible = True
'
'instead.

Sub HideButtons()
   ActiveSheet.Buttons.Visible = False
End Sub
```

3. Return to the sheet that has the buttons and choose Macro from the Tools menu.

4. In the Macro dialog box, select the HideButtons macro, and choose the Run button.

The buttons on your worksheet will be hidden but other graphic objects will

be visible.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

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

When you choose the Objects option in the Go To Special dialog box, all objects are selected. This means that if the Print Object check box is cleared, none of the graphic objects will be printed.

Additional reference words: 5.00 VBApp VBAEq

XL5: Determining which Items Are Selected in a List Box

Article ID: Q111564

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY
=======

In Microsoft Excel version 5.0, you can determine which items in a multi selection list box are selected by using the Selected property of the list box. This article contains an example of how this can be done with a Visual Basic subroutine.

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

In custom dialog boxes, multi selection list boxes allow you to choose any number of items from a list. For example, if a list contains Alpha, Bravo, and Charlie, you can select any, none, or all of those items.

To determine which items are selected, you can use the Selected property of the list box. The Selected property of a multi selection list box is an array of values: each value is either True, if the item is selected, or False, if the item is not selected. For example, if the list contains 1, 2, 3, and 4, and 2 and 3 are selected, the Selected property would be the following array:

```
   False, True, True, False
```

Because the first item (1) is not selected, the second and third items (2 and 3) are selected, and the fourth item (4) is not selected.

An example of how to read the Selected array in a subroutine is shown below.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line

continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

1. In a new workbook, insert a new worksheet (Sheet1), a dialog sheet (Dialog1), and a Visual Basic module (Module1).

2. In the worksheet, enter the following values:

   A1: Alpha
   A2: Bravo
   A3: Charlie
   A4: Delta
   A5: Echo
   A6: Foxtrot
   A7: Golf
   A8: Hotel

3. In the dialog sheet, add an OK button that is set to dismiss the dialog box.

4. In the dialog box, do the following to add a multi selection list box ("List1") that is linked to Sheet1!$A$1:$A$8.

   a. To create a list box in the dialog sheet, use the List Box button.

   b. Select the list box.

   c. In the name box, type "List1" (without the quotation marks) and press the ENTER key.

   d. From the Format menu, choose Object.

   e. Select the Control tab.

   f. In the Input Range box, type "Sheet1!$A$1:$A$8" (without the quotation marks).

   g. Under Selection Type, select the Multi option.

   h. Choose OK to accept the change.

   The list box is now linked to Sheet1!$A$1:$A$8 and is a multi selection list box.

5. In Module1, enter the following subroutine:

```
'--------------------------------------------------------------------
Sub ShowSelectedItems()

   'Dimension some variables.
   Dim LTemp As Variant
   Dim LItem As Variant
   Dim Counter As Integer
   Dim CurList as ListBox
```

```
    'Show the dialog box.
    DialogSheets("Dialog1").Show

    'Set an object name for easy referencing of the list box.
    Set CurList = DialogSheets("Dialog1").ListBoxes("List1")

    'Put the Selected array into the variable LTemp.
    LTemp = CurList.Selected

    'Initialize the Counter variable.
    Counter = 1

    'Iterate through the loop once for each item in the array (which is
    'the same as iterating once for each item in the list box).
    For Each LItem In LTemp

       'If the value of the current item is True...
       If LItem = True Then

          '...show a message box indicating the item is selected.
          'CurList.List(Counter) gets us the value of the selected item
          '("Alpha", "Bravo", etc.).
          MsgBox CurList.List(Counter) & " is selected."

       'Otherwise...
       Else

          '...indicate that it isn't selected.
          MsgBox CurList.List(Counter) & " is NOT selected."
       End If

       'Increment the Counter so we can get the value of the next
       'selected item.
       Counter = Counter + 1
    Next                                  'repeat until all done
End Sub
'---------------------------------------------------------------------------
```

6. To run the subroutine, position the insertion point in the line that reads "Sub ShowSelectedItems()" and either press F5 or choose Start from the Run menu.

7. When the dialog box is displayed, select one or more items in the list box, and choose the OK button. A message box will be displayed for each item in the list, indicating whether it is selected or not.

Note that you can remove the MsgBox functions and substitute any functions that perform a desired task using the list items. For example, you might want to insert the current list item into a cell on a worksheet. To do this, you could use

```
Sheets("Sheet1").Cells(10, 10).Value = CurList.List(Counter)
```

to put the current list item into cell J10 on Sheet1.

Additional reference words: 5.00 howto multiple selection

XL5: Application.OperatingSystem Returns 3.10 in WFW 3.11

Article ID: Q111710

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel 5.0 for Windows, the OperatingSystem property of the Application object returns the name and version number of the current operating system. When used under Microsoft Windows for Workgroups version 3.11, OperatingSystem will return version 3.10.

MORE INFORMATION

Microsoft Excel 5.0 for Windows uses a "GetVersion" Windows application programming interface (API) call that returns version 3.10 from the kernel. The version that the kernel reports was not changed for this version of Microsoft Windows for Workgroups.

When you run the following Visual Basic procedure, which displays the current operating system in a dialog box, Microsoft Windows version 3.11 will be incorrectly identified as version 3.10.

```
Sub ShowOS()
    Msgbox Application.OperatingSystem
End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line-continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

REFERENCES

"Visual Basic User's Guide", page 460

Additional reference words: 5.00

XL5: FOR Behaves Differently in Visual Basic Than in 4.0 Macro

Article ID: Q111725

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In the Microsoft Visual Basic Programming System, Applications Edition, that ships with Microsoft Excel, FOR loops behave differently than they do in the Microsoft Excel version 4.0 macro language.

The respective macro commands are as follows:

Visual Basic, Applications Edition

```
For counter = 1 To endValue.
Next counter
```

Microsoft Excel Version 4.0 Macro Commands

```
=FOR("counter",1,endValue)
=NEXT()
```

MORE INFORMATION

In Visual Basic, Applications Edition, the number of iterations for the loop cannot be changed by changing the value of the variable used to set the ending value for the loop while the loop is in progress (this behavior is standard for most programming languages that use FOR loops). However, you can change the value of the variable in a Microsoft Excel version 4.0 style macro using the equivalent functions, FOR() and NEXT().

Microsoft Excel Version 4.0 Macro Example

Microsoft provides macro examples for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This macro is provided 'as is' and Microsoft does not guarantee that the following code can be used in all situations. Microsoft does not support modifications of the code to suit customer requirements.

In the following Excel 4.0 style macro, the loop is originally set to run ten times. However, it will only run 5 times because the ending value is modified during the loop. To test this behavior, enter the following on a Microsoft Excel 4.0 macro sheet:

```
A1: y=10
A2: =FOR("x",1,y)
A3: y=5
A4: =NEXT()
A5: =ALERT(x)
A6: =RETURN()
```

To run the macro, select cell A1 and choose Macro from the Tools menu, then choose the Run button.

The ALERT() statement in A5 will display the value 6, which means that the loop only ran 5 times, as opposed to the 10 times that the original value of "y" was set for.

Visual Basic Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line-continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

In the following Visual Basic, Applications Edition, macro, the loop is originally set to run 10 times and it will run 10 times, even though the macro changes the value of the variable that is used to set the ending value of the loop.

To test this macro:

1. Enter the following in a new Visual Basic module:

```
Sub MyLoop()
Dim x,y As Integer

y=10
   For x = 1 To y
     y=5
   Next x
MsgBox x
End Sub
```

2. Position the insertion point in the line that reads "Sub MyLoop()" and either press F5 or choose Start from the Run menu.

The MsgBox statement in the above macro displays the value 11, which means the loop ran through 10 times, even though we changed the value of the variable used to set the ending value of the loop.

REFERENCES
==========

"Visual Basic User's Guide," version 5.0, pages 143-147

For more information about FOR, choose Contents from the Help menu, select Programming With Visual Basic, and then choose the Search button in Help and type the following:

   For

Additional reference words: 5.00

XL5 Err Msg: "Not Enough Stack Space to Run Macro"

Article ID: Q111867

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In Microsoft Excel, when you choose a control in a dialog box that is assigned to an event macro when there are a total of three dialog boxes on the screen that have not been dismissed, you may receive the following error message:

   Not Enough Stack Space to Run Macro

WORKAROUND

To avoid receiving this error message when you call nested dialog boxes, do not assign a dialog control (such as a button or a check box) to a macro event that calls another dialog box. Instead, assign the control to first dismiss the active dialog box, then call the desired dialog box from the same the macro that called the first dialog box. To dismiss the active dialog box, do any of the following:

 - Format the control with the Dismiss property:

   1. Select the control and choose Object from Format menu.

   2. In the Format Object dialog box, select the Dismiss check box on the Control tab .

   -or-

 - Assign the control to a macro that contains the following command:

      ActivateDialog.Hide

   -or-

 - Format the control with the Cancel property:

   1. Select the control and choose Object from Format menu.

   2. In the Format Object dialog box, Select the Cancel check box on the Control tab.

MORE INFORMATION

A dialog box is not updated until after the event macro has finished. The event macro is the code that is run when an action, such as choosing a

button, is taken in the dialog box. Because the dialog box is not updated, if you call a dialog from another dialog, the first dialog is still loaded (stacked) and is not released until the code that it ran (calling the second dialog) has completed. This condition exists even if the property of the button that called the second dialog is set to dismiss.

In Microsoft Excel 5.0, you can stack dialog boxes (that is, display more than one dialog box on the screen at one time) with the top dialog box active. You can stack two dialog boxes, and still run code assigned to controls on the second dialog box. For example, you can call Dialog Two from Dialog One, and then run a macro assigned to a control on Dialog Two by choosing the control. In addition, you can call Dialog Three from Dialog Two, but you cannot run any event macros from this top level (Dialog Three) dialog box.

Note that you cannot hide a dialog box until the macro that hides the dialog box ends.

Visual Basic Example

In the following example, there are three dialog boxes. Each dialog box can call either of the other two dialog boxes. Each button is assigned to a macro that sets the value of a variable. The value of the variable determines which dialog box to display next.

1. Create three dialog sheets. Modify each dialog sheet to contain the following:

Dialog Sheet Buttons	Dialog1	Dialog2	Dialog3
OK	yes	yes	yes
Cancel	yes	yes	yes
Go_To_Dialog1	no	yes	yes
Go_To_Dialog2	yes	no	yes
Go_To_Dialog3	yes	yes	no

2. Set the dismiss property for each button by doing the following:

   a. Select the button on the dialog sheet.

   b. From the Format menu, choose Object. Select the Control tab, and select the Dismiss check box. Choose OK.

3. In a new module in the same workbook, type the following:

```
' Define the variable as Integer type
Dim dialog_number As Integer

Sub Main()
   dialog_number = 1                  'initialize the variable
   DialogSheets("Dialog1").Show       'display the first dialog sheet
   While dialog_number > 0            'while variable is greater than 0
      Select Case dialog_number       'display a dialog based on the
                                      'value of dialog_number
```

```
            Case 1
                                              'dialog_number is 1
               DialogSheets("Dialog1").Show   'display dialog1
            Case 2
                                              'dialog_number is 2
               DialogSheets("Dialog2").Show   'display dialog2
            Case 3
                                              'dialog_number is 3
               DialogSheets("Dialog3").Show   'display dialog3

         End Select
      Wend
   End Sub

'The following code sets the value of the dialog_number variable

Sub Go_To_Dialog1()
    dialog_number = 1
End Sub

Sub Go_To_Dialog2()
    dialog_number = 2
End Sub

Sub Go_To_Dialog3()
    dialog_number = 3
End Sub

Sub OK_Or_Cancel()
    dialog_number = 0
End Sub
```

4. Select the Dialog1 sheet tab. Select the Go_To_Dialog2 button, and choose Assign Macro from the Tools menu. From the Macro Name/Reference list, select Go_To_Dialog2() and choose OK.

5. Repeat step 4 for each button, assigning the corresponding macro to each button on each dialog sheet. Assign the OK_Or_Cancel macro to each OK and Cancel button.

 Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

REFERENCES
==========

"Visual Basic User's Guide," version 5.0, page 219-239

Additional reference words: 5.00 5.0

XL5: Controls in Dialog Box May "Snap" to Preset Values

Article ID: Q111899

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOM

In Microsoft Excel version 5.0, when you use Visual Basic commands to adjust the dimensions of controls in custom dialog boxes by manipulating the Top, Left, Height, and Width properties of the controls, the controls will always "snap" to certain numbers. For example, if you change the Left property of a control, it will always snap to a multiple of 0.75.

CAUSE

This behavior occurs because of the design of the custom dialog box system.

MORE INFORMATION

"Snapping" occurs when a control is placed in a certain position that is not precisely supported in a custom dialog box. For example, the Left property of a control always snaps to the next lowest multiple of 0.75. If you change the Left property of a control to any of the following values, the Left property will then snap to the indicated value:

When <Control>.Left is set to this value	It snaps to this value
0	0
0.25	0
0.50	0
0.75	0.75
1.00	0.75
1.25	0.75
1.50	1.50

If you try to set the Left property of a control to 1.12 or 1.25, for example, Microsoft Excel will snap the control's Left property to 0.75, because that is the greatest multiple of 0.75 less than or equal to 1.12 or 1.25.

In addition, if one property is changed by snapping, it may affect other related properties. For example, in the Visual Basic code example below, an edit box is initially created with a Top value of 50 and a Height value of 15. The Top value initially snaps to 49.5; however, when the Height value snaps, the Top value is changed to 47.25, even though 49.5 is a valid Top value. The Left and Width values are similarly related. The final Top, Left, Height, and Width values will

always be valid, but you may observe controls moving slightly because of snapping.

Listed here are the guidelines that Microsoft Excel uses when determining how a property of a control will be snapped.

Control Type	Units Snapped to
ALL CONTROLS	The Left property of all controls snaps to the nearest multiple of 0.75 less than or equal to the indicated value. The Top property of all controls snaps to the nearest multiple of 0.75 less than or equal to the indicated value.
Edit Boxes and Labels	The Height property of edit boxes and labels use the following table to determine the true Height value: Height Set to / Height Snaps to 0.00 - 18.50 / 13.50 18.75 - 28.25 / 23.25 28.50 - 38.00 / 33.00 38.25 - 47.75 / 42.75 For subsequent ranges, add 9.75 for each range. The Width property of edit boxes and labels snaps to the nearest multiple of 0.75 less than or equal to the indicated value.
Buttons	The Height property of buttons snaps to the nearest multiple of 0.75 less than or equal to the indicated value, but no less than 15.75. The Width property of buttons snaps to the nearest multiple of 0.75 less than or equal to the indicated value, but no less than 3.00.
Dialog Box Frame	All properties of a dialog frame snap to the nearest multiple of 0.75 less than or equal to the indicated value.
Group Boxes	The Height property of group boxes snaps to the nearest multiple of .75 less than or equal to the indicated value, but no less than 18.75. The Width property of group boxes snaps to the nearest multiple of 0.75 less than or equal to the indicated value, but no less than 23.25.
Check Boxes	The Height property of check boxes is always 16.50. The Width property of check boxes snaps to the nearest multiple of 0.75 less than or equal to the

indicated value, but no less than 23.25.

Option Buttons	The Height property of option buttons is always 16.50. The Width property of option buttons snaps to the nearest multiple of 0.75 less than or equal to the indicated value, but no less than 23.25.
List Boxes	The Height property of list boxes uses the following table to determine the true Height value: Height Set to Height Snaps to 0.00 - 26.75 21.75 27.00 - 36.50 31.50 36.75 - 46.25 41.25 46.50 - 56.00 51.00 For subsequent ranges, add 9.75 for each range. The Width property of list boxes snaps to the nearest multiple of 0.75 less than or equal to the indicated value, but no less than 12.00.
Drop-Down Lists	The Height property of drop-down lists is always 15.00. The Width property of drop-down lists snaps to the nearest multiple of 0.75 less than or equal to the indicated value, but no less than 12.00.
Drop-Down Edit Boxes	The Height property of drop-down edit boxes is always 13.50. The Width property of drop-down edit boxes snaps to the nearest multiple of 0.75 less than or equal to the indicated value, but no less than 12.00.
Scroll Bars	The Height property of scroll bars snaps to the nearest multiple of 0.75 less than or equal to the indicated value, but no less than 12.00. The Width property of scroll bars is always 12.00.
Spinners	The Height property of spinners snaps to the nearest multiple of 0.75 less than or equal to the indicated value, but no less than 13.50. The Width property of spinners is always 9.00.

Visual Basic Code Example

particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line-continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following subroutine creates a new dialog sheet, adds an edit box to it, changes the Top, Left, Height, and Width properties of the edit box, and then displays what the various properties have snapped to.

```
'-------------------------------------------------------------------------
Sub ControlSnapDemo()
   'Add a new dialog sheet to the active workbook.
   Set DemoDlg = ThisWorkbook.DialogSheets.Add
   'Add an edit box to the current dialog sheet. The dimensions
   'supplied are arbitrary.
   Set EdBox = DemoDlg.EditBoxes.Add(50, 50, 50, 50)
   'Set the Top, Left, Height, and Width properties of the edit box.
   EdBox.Top = 50
   EdBox.Left = 70
   EdBox.Height = 15
   EdBox.Width = 80
   'Construct a message string which will be shown in a message box.
   'Chr$(9) is a tab character: Chr$(10) is a line feed.
   MsgString = Chr$(9) & "Original Setting" & Chr$(9) & "Snaps To"
   MsgString = MsgString & Chr$(10) & "Top" & Chr$(9) & "50" & Chr$(9)
   MsgString = MsgString & Chr$(9) & EdBox.Top & Chr$(10) & "Left"
   MsgString = MsgString & Chr$(9) & "70" & Chr$(9) & Chr$(9)
   MsgString = MsgString & EdBox.Left & Chr$(10)& "Height" & Chr$(9)
   MsgString = MsgString & "15" & Chr$(9) & Chr$(9) & EdBox.Height
   MsgString = MsgString & Chr$(10) & "Width" & Chr$(9) & "80" & Chr$(9)
   MsgString = MsgString & Chr$(9) & EdBox.Width
   'Show the message.
   MsgBox MsgString
End Sub
'-------------------------------------------------------------------------
```

When you run this subroutine, a new dialog sheet containing a new edit box will be created. A message box will appear with this message:

```
            Original Setting   Snaps To
   Top      50                 47.25
   Left     70                 69.75
   Height   15                 13.5
   Width    80                 79.5
```

The properties of the edit box have snapped to values appropriate for

an edit box. Note that the Top property has been "over-snapped" because of the substantial change in the Height property.

Additional reference words: 5.00

XL5: Can't Use Replace Command to Search for Blank Cells

Article ID: Q111942

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In Microsoft Excel, if you choose Replace from the Edit menu, and you leave the Find What box blank and choose Find Next, Replace, or Replace All to fill blank cells on a worksheet, you receive the following error message:

   Search string must be specified

WORKAROUND 1

The following workaround is presented on page 33 in "Microsoft Excel Visual Basic for Applications Step by Step," a Microsoft Press book.

1. Highlight the range
2. From the Edit menu, choose Goto
3. Choose Special
4. Select the Blanks check box
5. Choose OK

All the blanks in the range are now highlighted. Enter the value that you want to replace the blanks with. Now press CTRL+ENTER. This places the data in all of the selected cells.

WORKAROUND 2

The following Visual Basic procedure prompts you for the text you want to enter in blank cells, and replaces the contents of the blank cells in the selection with this text.

```
Sub Replace_Blanks()
    ' Dimension variables
    Dim Blanks As Object
    Dim ReplaceWith As String
    Dim Cell As Object
    ' Prompt user for string to enter in blank cells
    ReplaceWith = _
    Application.InputBox("Replace blank cells with what?", _
    "Replace String")
    ' Select blank cells in selection
    Selection.SpecialCells(xlBlanks).Select
    ' Defined Blanks as selection
    Set Blanks = Application.Selection
    ' Fill each cell in selection with user input
    For Each cell In blanks
```

```
         Cell.Activate
         ActiveCell.Formula = replacewith
      Next cell
   End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

MORE INFORMATION

In Microsoft Excel versions earlier than version 5.0, you can replace blank cells with a value by using the Replace dialog box and leaving the Find What box blank. However, in Microsoft Excel version 5.0, you cannot use this method to replace blank cells because you cannot leave the Find What box in the Replace dialog box blank.

REFERENCES

"Microsoft Excel Visual Basic for Applications Step by Step,"
Reed Jacobson, $29.95 ($39.95 Canada), ISBN is 1-55615-589-1

Additional reference words: 5.00 err msg

Selection.Cells with Nonadjacent Selection Returns Single Cell

Article ID: Q112028

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In Microsoft Excel, when you use the Range method or the Selection property with the cells method in a Visual Basic module to return a nonadjacent selection of cells on a worksheet, only the first cell in the selection is returned.

STATUS

Microsoft has confirmed this to be a problem in the versions of Microsoft Excel listed above. We are researching this problem, and will post new information here in the Microsoft Knowledge Base when it becomes available.

WORKAROUND

You can use the following sample Visual Basic procedure to return all the cells in a selection when not all of the cells are adjacent on a worksheet:

```
Sub Apply_Italic()
' Declare variables
Dim x As Object
Dim y As Object
   ' Define y as an area in the selection
   For Each y In Selection.Areas
      ' Define x as a cell in that area
        For Each x In y
        ' Replace the following statement with the code you want to use
        ' with the cells in the selection
        x.Font.Italic = True
      Next x
    Next y
End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one

logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

MORE INFORMATION

Normally, you can use the Range method or the Selection property with the Cells method to return selected cells. However, if the cells in the selection are nonadjacent, the Range method and the Selection property return only the first cell in the selection.

Visual Basic Example

When you run the following Visual Basic procedure, the data in each adjacent cell in the selection appears in italic formatting:

```
Sub Apply_Italic()
' Define variable x
Dim x as Object
   ' Define x as a cell in the selection
   For Each x in Selection.Cells
      ' Format the contents of each cell with Italic formatting
      x.Font.Italic=True
   Next x
End Sub
```

If the selection of cells is nonadjacent, only the data in the first selected cell appears in italic formatting.

NOTE: To select nonadjacent cells on a worksheet, select one cell, hold down the CTRL key, and select another cell. Repeat this procedure until you select every cell in the selection that you want.

REFERENCES

For more information about the Range Method, choose the Search button in the Visual Basic Reference and type:

```
Range Method
```

For more information about the Selection Property, choose the Search button in the Visual Basic Reference and type:

```
Selection Property
```

Additional reference words: 5.00

XL5: Cannot Print Multiple Copies to DeskJet 500C/550C Driver

Article ID: Q112037

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

If you are using a Hewlett-Packard (HP) DeskJet 550C or 500C printer driver version 3.0 or 3.1, you will not be able to print multiple copies from Microsoft Excel 5.0. Changing the number of copies in the Print dialog box will not affect the number of copies printed.

STATUS

Microsoft has confirmed this to be a problem in the versions of Microsoft Excel listed above. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

WORKAROUND 1

The HP DeskJet 500 (black and white) driver version 2.14 that ships with Microsoft Windows 3.1 and 3.11 will print multiple copies correctly. However, this driver does not allow you to print in color.

WORKAROUND 2

The following Visual Basic for Applications code allows you to print multiple copies.

```
'A procedure to print multiple copies
Sub Example()
   Ncopies = Application.InputBox("Please enter number off copies _
       to print: ", "Print Multiple Copies", 1, , , , , 1)
   For counter = 1 To Ncopies
      ActiveWindow.SelectedSheets.PrintOut Copies:=1
   Next
End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from

one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Additional reference words: 5.00 VBAppCode

XL5: "Cannot Find Macro..." Using Run Method with Add-In Macro

Article ID: Q112219

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In a Visual Basic module in Microsoft Excel, if you use the Run method of the Application object to run a macro located in an add-in, you may receive the following error message

```
Run-time error '1004':

Cannot find macro '<ADDIN.XLA>!<Addin_Macro>'
```

where <ADDIN.XLA> is the name of the add-in that contains the macro, and <Addin_Macro> is the name of the macro that you want to run.

CAUSE

This error occurs when the add-in that contains the procedure you used with the Run method is not currently open. Even if the add-in is fast loaded by the following line in the EXCEL5.INI file

```
OPEN=/f C:\EXCEL\ADDIN.XLA
```

where ADDIN.XLA is the add-in file.

STATUS

Microsoft has confirmed this to be a problem in the Microsoft products 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.

WORKAROUND

To use the Run method to run a macro that is contained in an add-in without receiving the error message above, do any of the following:

- In the procedure that contains the Run method, use the Open method of the Workbooks object to open the add-in before the Run method as in the following example:

```
Workbooks.Open "c:\EXCEL\ADDIN.XLA"
Run("ADDIN.XLA!Macro_Name")
```

-or-

- Do the following to manually establish a reference to the add-in from the procedure that is calling the add-in macro:

  1. Activate the module that contains the procedure that is running the add-in macro.

  2. From the Tools menu, choose References.

  3. If the add-in that contains the macro you want to run is not listed under Available References, choose Browse.

  4. From the File Name list, select the add-in you want to reference and choose OK. Choose OK again.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line-continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

MORE INFORMATION

When you fast load and add-in, for example by using the /f switch in the EXCEL5.INI file as explained above, any menus and toolbars contained in the add-in are available, but the add-in file itself is not actually loaded, and any macros contained in the add-in are not loaded. When you demand load an add-in, either by opening the add-in file, or with the following line in the EXCEL5.INI file

```
OPEN C:\EXCEL5\ADDIN.XLA
```

everything in the add-in is available, including any macros in the add-in. Therefore, when you demand load and add-in, you do not receive the above error message when you use the Run method to run a macro located in the add-in.

For more information about using the /F switch to load an Add-in, query on the following words in the Microsoft Knowledge Base:

```
fast and load and add-in
```

Additional reference words: 5.00 addins addin

XL5: 'For Each Item in List' Doesn't Work

Article ID: Q112330

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0 for Windows, the Visual Basic 'For Each...Next' control structure may not function properly if the group argument is a list in a custom dialog box.

MORE INFORMATION

In Microsoft Excel version 5.0, you can use the 'For Each...Next' control structure to repeat a group of statements for each element in an array or collection.

For example, the following code displays dialog boxes containing the contents of each cell in a selected range:

```
For Each mCell In Selection  '"Selection" is the group argument
   MsgBox mCell.Value
Next
```

If the group argument is a list in a custom dialog box, the For Each command may fail, and you will receive the following error message:

```
Run-time error '10':
Duplicate definition
```

WORKAROUND

To avoid this error, set an object equal to the list

```
<Object> = <Listname>.List
```

and then use the object in the For Each statement. For example, instead of

```
xList = DialogSheets("Dialog1").ListBoxes("List1")
For Each mItem in xList.List
   <statements>
Next
```

use

```
xList = DialogSheets("Dialog1").ListBoxes("List1")
mTemp = xList.List
For Each mItem in mTemp
   <statements>
```

```
   Next
```

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

This example shows one way that you can avoid the error associated with calling a list directly from a For Each statement.

The following example assumes you have a workbook that contains a Visual Basic module (Module1) and a dialog sheet (Dialog1). The dialog sheet contains a single list box (List1).

In Module1, enter the following subroutine:

```
'-----------------------------------------------------------------------
Option Explicit

Sub ForEachListItem()

   'Dimension some variables.
   Dim Alpha As Variant, Foxtrot As Variant, Golf As Variant

   'Set an object name for easy referencing of the list box.
   Set Alpha = DialogSheets("Dialog1").ListBoxes("List1")

   'Add three items to the list.
   Alpha.AddItem "Bravo"
   Alpha.AddItem "Charlie"
   Alpha.AddItem "Delta"

   'Set an object name so that the For Each structure can function
   'properly.
   Golf = Alpha.List

   'Iterate through the loop once for each item in Golf (which is
   'the same as iterating once for each item in the list box).
   For Each Foxtrot In Golf

      'Show the current list item in a message box.
```

```
      MsgBox Foxtrot
   Next                                  'repeat until all done
End Sub
'---------------------------------------------------------------------
```

To run the subroutine, position the cursor in the line that reads "Sub ForEachListItem()," and either press F5 or choose Start from the Run menu.

If the For Each line is entered as shown, the subroutine will run properly and message boxes will display the names of the items in the list.

If the For Each line is altered to refer to Alpha.List directly, without using an intermediary function (in this case, Golf), you will receive the error message shown above and the message boxes will not be displayed.

Additional reference words: 5.00

XL5: Getting Windows Status Information from Windows API

Article ID: Q112393

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

The Visual Basic, Applications Edition, example below demonstrates how to obtain system status information similar to the information displayed in the Microsoft Windows Program Manager About box. The example procedure displays the following information using the Windows API functions indicated:

This Statement/ Property	Displays This Information
GetWinFlags	The kind of CPU (80286, 80386, or 80486) and whether a math coprocessor is present
GetWinFlags	Whether Microsoft Windows is running in enhanced mode or standard mode
GetFreeSpace and GlobalCompact	The amount of free memory
SystemHeapInfo	The percentage of free system resources
OperatingSystem	The version of Windows

MORE INFORMATION

To Create a sample macro

1. Start Microsoft Excel for Windows 5.0.

2. From the Insert menu, choose Macro, and then choose Module. Module 1 is created by default.

3. Enter the following code into the newly created 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
```

```
' Type for SystemHeapInfo.
Type SYSHEAPINFO
    dwSize As Long
    wUserFreePercent As Integer
    wGDIFreePercent As Integer
    hUserSegment As Integer
    hGDISegment As Integer
End Type

Declare Function GetWinFlags Lib "KERNEL" () As Long
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

Sub GetWindowsInfo()
    Dim Status As Long
    Dim Memory As Long
    Dim msg As String         ' Status information.
    Dim nl As String          ' New-line.
    Dim shi As SYSHEAPINFO
    nl = Chr$(13) + Chr$(10)  ' New-line.

    Status = GetWinFlags()

    ' Get operating system version.
    ' (Uses Excel's built-in OperatingSystem function rather
    ' than Windows API calls.)
    msg = "OS: " + Application.OperatingSystem

    ' Get CPU kind and operating mode.
    msg = msg + nl + "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$(Memory \ 1024, "###,###,###") + "K" + nl
    Memory = GlobalCompact(&hffff)
    msg = msg + "Largest free block: "
    msg = msg + Format$(Memory \ 1024, "###,###,###") + "K" + nl

    ' Get free system resources.
    msg = msg + "System resources: "
    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

      MsgBox msg, vbOKOnly, "About This PC"
   End Sub
```

4. From the Tools menu, choose Macro.

5. From the Macro Name/Reference list, select GetWindowsInfo, and choose OK to run the macro.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Additional reference words: 5.00 vbappcode

XL5: Using the Windows OpenFile Dialog Box

Article ID: Q112394

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

This article describes how to access the OpenFile dialog box using a Visual Basic, Applications Edition, procedure. The OpenFile dialog box can be used to prompt the user for the name of a file.

This article assumes that you are familiar with Visual Basic, Applications Edition, and with the programming tools provided with Microsoft Excel version 5.0.

MORE INFORMATION

In Microsoft Excel version 5.0, you can access the OpenFile common dialog box by using Windows GetOpenFileName() application programming interface (API) function. This function creates a system-defined dialog box, familiar throughout Windows, that makes it possible for the user to select a file to open. This function will return a valid file name to the programmer that is fully qualified with the path name. Using this function will simplify programming issues for the developer.

A developer can customize the way the system will handle specific situations, such as specifying that the file must exist when the user wants to save a file, through the use of flags. Additionally, multiple files can be selected and returned using this function (which is not true of the built-in function Application.GetOpenFilename, which will only return a single file name).

The OpenFile common dialog routines are stored in a file called COMMDLG.DLL, which is supplied with Microsoft Windows versions 3.1 and later.

Visual Basic Procedure

To use the code below, paste the function and declarations into a Visual Basic module.

You may have some Windows API functions defined in an existing Microsoft Excel module; therefore, your declarations may be duplicates. If you receive a duplicate procedure name error, remove the Declare statement from your code or comment it out.

```
'-------------------------------------------------------
' Global Declaration Section
```

```
'-------------------------------------------------------
Option Explicit

Type tagOPENFILENAME
     lStructSize As Long
     hwndOwner As Integer
     hInstance As Integer
     lpstrFilter As Long
     lpstrCustomFilter As Long
     nMaxCustFilter As Long
     nFilterIndex As Long
     lpstrFile As Long
     nMaxFile As Long
     lpstrFileTitle As Long
     nMaxFileTitle As Long
     lpstrInitialDir As Long
     lpstrTitle As Long
     Flags As Long
     nFileOffset As Integer
     nFileExtension As Integer
     lpstrDefExt As Long
     lCustData As Long
     lpfnHook As Long
     lpTemplateName As Long
End Type

Declare Function GetOpenFileName% Lib "COMMDLG.DLL" (OPENFILENAME As
tagOPENFILENAME)
Declare Function lstrcpy& Lib "KERNEL" (ByVal lpDestString As Any,
ByVal lpSourceString As Any)
Declare Function GetModuleHandle Lib "KERNEL" (ByVal App As String) As
Integer

Dim OPENFILENAME As tagOPENFILENAME

Global Const OFN_READONLY = &h1
Global Const OFN_OVERWRITEPROMPT = &h2
Global Const OFN_HIDEREADONLY = &h4
Global Const OFN_NOCHANGEDIR = &h8
Global Const OFN_SHOWHELP = &h10
Global Const OFN_ENABLEHOOK = &h20
Global Const OFN_ENABLETEMPLATE = &h40
Global Const OFN_ENABLETEMPLATEHANDLE = &h80
Global Const OFN_NOVALIDATE = &h100
Global Const OFN_ALLOWMULTISELECT = &h200
Global Const OFN_EXTENSIONDIFFERENT = &h400
Global Const OFN_PATHMUSTEXIST = &h800
Global Const OFN_FILEMUSTEXIST = &h1000
Global Const OFN_CREATEPROMPT = &h2000
Global Const OFN_SHAREAWARE = &h4000
Global Const OFN_NOREADONLYRETURN = &h8000
Global Const OFN_NOTESTFILECREATE = &h10000

Global Const OFN_SHAREFALLTHROUGH = 2
Global Const OFN_SHARENOWARN = 1
Global Const OFN_SHAREWARN = 0
```

```
'-------------------------------------------------------
' Open Common Dialog Function
'-------------------------------------------------------
Function OpenCommDlg()
  Dim Message$, Filter$, FileName$, FileTitle$, DefExt$
  Dim Title$, szCurDir$, APIResults%

    ' Define the filter string and allocate space in the "c" string
    Filter$ = "Excel Files(*.XL*)" & Chr$(0) & "*.XL*" & Chr$(0)
    Filter$ = Filter$ & "Text(*.txt)" & Chr$(0) & "*.TXT" & Chr$(0)
    Filter$ = Filter$ & Chr$(0)

    ' Allocate string space for the returned strings.
    FileName$ = Chr$(0) & Space$(255) & Chr$(0)
    FileTitle$ = Space$(255) & Chr$(0)

    ' Give the dialog a caption title.
    Title$ = "My File Open Dialog" & Chr$(0)

    ' If the user does not specify an extension, append TXT.
    DefExt$ = "TXT" & Chr$(0)

    ' Set up the defualt directory
    szCurDir$ = CurDir$() & Chr$(0)

    ' Set up the data structure before you call the GetOpenFileName

    OPENFILENAME.lStructSize = Len(OPENFILENAME)
    OPENFILENAME.hwndOwner = GetModuleHandle(Application)
    OPENFILENAME.lpstrFilter = lstrcpy(Filter$, Filter$)
    OPENFILENAME.nFilterIndex = 1
    OPENFILENAME.lpstrFile = lstrcpy(FileName$, FileName$)
    OPENFILENAME.nMaxFile = Len(FileName$)
    OPENFILENAME.lpstrFileTitle = lstrcpy(FileTitle$, FileTitle$)
    OPENFILENAME.nMaxFileTitle = Len(FileTitle$)
    OPENFILENAME.lpstrTitle = lstrcpy(Title$, Title$)
    OPENFILENAME.Flags = OFN_FILEMUSTEXIST Or OFN_READONLY
    OPENFILENAME.lpstrDefExt = lstrcpy(DefExt$, DefExt$)
    OPENFILENAME.hInstance = 0
    OPENFILENAME.lpstrCustomFilter = 0
    OPENFILENAME.nMaxCustFilter = 0
    OPENFILENAME.lpstrInitialDir = lstrcpy(szCurDir$, szCurDir$)
    OPENFILENAME.nFileOffset = 0
    OPENFILENAME.nFileExtension = 0
    OPENFILENAME.lCustData = 0
    OPENFILENAME.lpfnHook = 0
    OPENFILENAME.lpTemplateName = 0

    ' This will pass the desired data structure to the Windows API,
    ' which will in turn use it to display the Open Dialog form.

    APIResults% = GetOpenFileName(OPENFILENAME)

    If APIResults% <> 0 Then

        ' Note that FileName$ will have an embedded Chr$(0) at the
        ' end. You may want to strip this character from the string.
```

```
        Message$ = "The file you chose was " + RTrim$(FileName$)
    Else
        Message$ = "No file was selected"
    End If

    MsgBox Message$

    ' Return the file selected
    OpenCommDlg = RTrim$(FileName$)

End Function
```

This function can be called as shown in the example below.

```
Sub OpenCommDlgTest()
    Dim FileSelected
    Calls the OpenCommDlg function (listed above) and places
    return value in FileSelected variable
    FileSelected = OpenCommDlg
    MsgBox FileSelected
End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

REFERENCES

"Microsoft Windows Software Development Kit Programmer's Reference," Volume 2: Functions, version 3.1

Additional reference words: 5.00 COMMDLG.DLL linked listbox vbappcode list box

XL5: Incorrect Use of "Is" Function Causes GP Fault

Article ID: Q112629

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0 for Windows, if you use the Is function incorrectly, you may receive a general protection (GP) fault.

MORE INFORMATION

The Is function is used to compare two object reference variables. The result of an Is operation is either TRUE (if the variables both refer to the same object) or FALSE (if they do not).

For example, if you have the following code:

```
Set Alpha = ActiveWorkbook.Sheets(1)
Set Bravo = ActiveWorkbook.Sheets(1)

Set Charlie = Alpha
Set Delta = Alpha
```

All of these commands will return the value TRUE:

```
Alpha Is Charlie
Alpha Is Delta
Charlie Is Delta
```

Note that Bravo Is <object>, where <object> is Alpha, Charlie, or Delta, will return FALSE, because the value of Bravo is not the same as the value of Alpha, Charlie, or Delta, even though they both refer to the same object.

If you use the Is function to compare a valid object reference (for example, Alpha) to an invalid object reference (for example, a numeric value, a string, or any other item that is not an object), you may receive a GP fault.

To prevent this error from occurring, make sure both items being compared are valid objects.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and

Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line-continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following code example illustrates the proper use of the Is function.

To run this example, position the cursor in the line that reads "Sub GoodIsSubroutine()" and either press the F5 key or choose Start from the Run menu.

```
'--------------------------------------------------------------------
Option Explicit

Sub GoodIsSubroutine()

   'Dimension variables.
   Dim Alpha As Variant, Bravo As Variant, Charlie As Variant
   Dim Delta As Variant

   'Create object names.
   Set Alpha = ActiveWorkbook.Sheets(1)
   Set Bravo = ActiveWorkbook.Sheets(1)
   Set Charlie = Alpha
   Set Delta = Alpha

   'These are tests to see if one object Is another object. A message
   'box is displayed if the objects are the same. In this case, only the
   'first three messages will be shown.
   If Alpha Is Charlie Then
      MsgBox "Alpha is Charlie!"
   End If

   If Alpha Is Delta Then
      MsgBox "Alpha is Delta!"
   End If

   If Charlie Is Delta Then
      MsgBox "Charlie is Delta!"
   End If

   If Bravo Is Charlie Then
      MsgBox "Bravo is Charlie!"
   End If

   If Bravo Is Delta Then
      MsgBox "Bravo is Delta!"
   End If
```

```
End Sub
'-----------------------------------------------------------------------

When you run GoodIsSubroutine, you will be shown three message boxes,
one for each of the Is functions that succeeds.
'-----------------------------------------------------------------------

Additional reference words: 5.00 hang stop responding gpf
```

XL5: OLE Automation Dialog Box Redrawn Incorrectly

Article ID: Q112630

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In Microsoft Excel version 5.0 for Windows, if you use OLE Automation to control another application, and if you display a dialog box from that application, the dialog box may not be correctly redrawn when you move it on the screen.

Also, if you switch to Microsoft Excel (for example, by pressing ALT+TAB) without dismissing the dialog box, Microsoft Excel may appear to hang. This is by design.

CAUSES

Dialog Box Is Not Redrawn Correctly

This error occurs because of the way in which dialog boxes are handled in OLE Automation. If you display a dialog box from another application, Microsoft Excel is temporarily "frozen" until the dialog box is dismissed. When Microsoft Excel is frozen this way, it does not have the ability to redraw parts of the screen that are occupied by a dialog box, and you may see "ghost" images of the dialog box that only disappear when control is returned to Microsoft Excel. The only solution to this problem is to not move dialog boxes that are shown by OLE Automation.

The redraw behavior used by Microsoft Excel is different than that used by Microsoft Visual Basic for Windows, version 3.0. Visual Basic version 3.0 has the ability to redraw parts of the screen, eliminating the "ghost" images, because of the way that Visual Basic handles dialog boxes that are shown by OLE Automation.

Microsoft Excel Hangs When a Dialog Box Is Displayed

Microsoft Excel cannot be accessed when a dialog box called through OLE Automation is still on the screen.

If you switch back to Microsoft Excel without dismissing the dialog box, an hourglass will appear and Microsoft Excel will appear to hang. If this occurs, you should switch to the dialog box by pressing ALT+TAB and dismissing the dialog box before you returning to Microsoft Excel.

Additional reference words: 5.00 lock up freeze doesn't respond

XL5: Setting Status Bar Text and ToolTips for Toolbar Buttons

Article ID: Q112632

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY
=======

In Microsoft Excel version 5.0, the behavior of status bar text attached to toolbar buttons differs from earlier versions of Microsoft Excel. In Microsoft Excel version 5.0, status bar text becomes attached to a toolbar button by one of two methods. This can be achieved by using a Visual Basic macro or from the Status Bar Text box in the Macro Options dialog box. (To get to this dialog box, choose Macro from the Tools menu, select the name of the macro assigned to the toolbar button, and click the Options button.)

ToolTips is a new feature in Microsoft Excel version 5.0. ToolTips can only be assigned to a toolbar button by using a macro.

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

In Microsoft Excel version 5.0, the status bar text is a property of the macro, not of the toolbar button. When a macro is assigned to a toolbar button, that macro's status bar text is assigned to that toolbar button and will be displayed when the mouse cursor is positioned over that toolbar button.

NOTE: This mouse cursor behavior differs from earlier versions of Microsoft Excel. In earlier versions you have to hold down the left mouse button while the cursor is on top of the toolbar button in order to read the status bar text.

In Microsoft Excel version 5.0, the status bar text will change whenever a macro is assigned to a toolbar button. If no status bar text is assigned to the macro, the toolbar button's status bar text will revert to the default for that toolbar button. For example, the default text for a toolbar button taken from the Custom category is Creates A Button To Which You Can Assign A Macro.

ToolTips are a property of the individual toolbar button. They only change when a Visual Basic macro explicitly sets the Name property for a particular toolbar button.

The following Visual Basic macros add a toolbar button, set its name property, and assign status bar text to a macro named "myMacro". Note that it is not possible to assign status bar text directly to a toolbar button from a Visual Basic macro.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability

and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

```
Sub AddButton()
  'creates a new toolbar titled "My Toolbar"
  With Toolbars.Add("My Toolbar")
     'show the newly created toolbar
     .Visible = True
     'Add the "hand" button to end of the new toolbar
     .ToolbarButtons.Add Button:=229
     'Set ToolTip to "my tool tip"
     .ToolbarButtons(.ToolbarButtons.Count).Name = "my tool tip"
  End With
End Sub
```

After you run the above macro, clicking the hand button will cause the Assign Macro dialog box to appear, where you can assign a macro to this new button. Alternatively, you can assign a macro to the toolbar button above by changing Line 4 to read:

```
     .ToolbarButtons.Add Button:=229, OnAction:="myMacro"
```

The following is the Visual Basic macro code to assign Status Bar text to a macro:

```
Sub AssignStatusText()
     Application.MacroOptions Macro:="myMacro", StatusBar:="My Text"
End Sub
```

NOTE: For the above Subroutine to work you must create a macro named "myMacro".

REFERENCES
==========

"Visual Basic User's Guide," version 5.0, Chapter 12,
"Managing Toolbars and Toolbar Buttons with Visual Basic"

For more information about toolbar buttons, choose the Search button in Help and type:

    toolbar buttons

Additional reference words: 5.00 tool tip tips

Excel AppNote: Built-in Constants in VB, Applications Edition

Article ID: Q112671

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

The Application Note "Built-in Constants in Microsoft Visual Basic Programming System, Applications Edition," (WE0993) provides a list of the Microsoft Excel constants and their values for use in Microsoft Visual Basic Programming System, Applications Edition. The enclosed WE0993 disk contains the Microsoft Excel 5.0 constants file, XLCONST.BAS, and the Visual Basic, Applications Edition, constants file, VBACONST.BAS. You can use these files in Visual Basic version 3.0 projects. These files provide the definitions of these constants so you can use the name of the constants in your Visual Basic version 3.0 modules. The WE0993 disk also includes a Microsoft Excel workbook file, XLCONST.XLS, which includes alphabetical and numerical lists of the Microsoft Excel constants.

You can obtain this Application Note from the following sources:

- CompuServe, GEnie, and Microsoft OnLine
- Microsoft Download Service (MSDL)
- The Internet (Microsoft anonymous ftp server)
- Microsoft Product Support Services

For complete information, see the "To Obtain This Application Note" section at the end of this article.

THE TEXT OF WE0993

==========

```
=========================================================================
  Microsoft(R) Product Support Services Application Note (Text File)
         WE0993: BUILT-IN CONSTANTS IN MICROSOFT VISUAL BASIC
               PROGRAMMING SYSTEM, APPLICATIONS EDITION
=========================================================================
                                                      Revision Date: 3/94
                                                          1 Disk Included
```

The following information applies to Microsoft Excel for Windows(TM), version 5.0.

OVERVIEW

This Application Note provides a list of the Microsoft Excel constants and their values for use in Microsoft Visual Basic Programming System, Applications Edition. The enclosed WE0993 disk contains the Microsoft Excel 5.0 constants file, XLCONST.BAS, and the Visual Basic, Applications Edition, constants file, VBACONST.BAS. You can use these files in Visual Basic version 3.0 projects. These files provide the definitions of these constants so you can use the name of the constants in your Visual Basic version 3.0 modules. The WE0993 disk also includes a Microsoft Excel workbook file, XLCONST.XLS, which includes alphabetical and numerical lists of the Microsoft Excel constants.

WHAT ARE BUILT-IN CONSTANTS?

When you create procedures in Microsoft Excel using the Visual Basic Programming System, Applications Edition, you can use "constants" to represent values that you use frequently in a procedure. Using constants makes your code easier to read and maintain. For example, if you use the value 5.67 frequently, you can define a constant called <MYVALUE> as 5.67, and then you can use <MYVALUE> in your procedure everywhere you would use the value 5.67.

Microsoft Excel has built-in constants that are used in Visual Basic functions, methods, objects, and properties. The Microsoft Excel built-in constants all begin with the letters "xl," and the Visual Basic, Applications Edition, built-in constants begin with the letters "vb."

Built-in constants make it easier for you to create procedures in Microsoft Excel using Visual Basic. For example, to change the orientation of the active worksheet to Landscape or to check whether a cell is center-aligned, you could use the following Visual Basic code:

```
   ActiveWorksheet.PageSetup.Orientation = xlLandscape
   If Range("A1").HorizontalAlignment = xlCenter Then
      MsgBox "Cell A1 is centered!"
   End If
```

The built-in constants (xlLandscape and xlCenter) are easier to remember than the numeric values they represent. In addition, when you use built-in constants, it is easier to read the code and understand the function the code performs. Without Microsoft Excel's built-in constants, the lines above would appear as follows:

```
   ActiveWorksheet.PageSetup.Orientation = 2
   If Range("A1").HorizontalAlignment = -4108 Then
```

Note how difficult it is to understand these lines without knowing what the values 2 and -4108 represent.

USING MICROSOFT EXCEL BUILT-IN CONSTANTS IN VISUAL BASIC VERSION 3.0

In Visual Basic in Microsoft Excel, the Microsoft Excel constants and the Visual Basic, Applications Edition, constants are available automatically. For the Standard or Professional Editions of Visual Basic, you must either add the constants file XLCONST.BAS or VBACONST.BAS to your project so that you can use the constant by name, or you must use the numeric value of the constant. The Microsoft Excel constants file (XLCONST.BAS), and the Microsoft Visual Basic, Applications Edition, constants file (VBACONST.BAS) include the name of each constant along with the numeric value of that constant. These files are available on the WE0993 disk that comes with this Application Note.

LOADING CONSTANTS FROM XLCONST.BAS OR VBACONST.BAS

The XLCONST.BAS and VBACONST.BAS files contain the Microsoft Excel and Visual Basic, Applications Edition, constants that you can use in your code. For example, to set the orientation of a worksheet in Microsoft Excel to landscape, you can either set the worksheet's PageSetup Orientation property to 2, or you can use the constant xlLandscape supplied in XLCONST.BAS. Using constants in your code makes the code easier to read. In addition, when you are entering code, you may find the symbolic constant easier to remember than its numerical specific. The constants are declared globally so they must be loaded into a code module, not into a form. To limit the scope of the constants to a form or procedure, change Global to Const in the declaration and paste them into the Declaration section of a form or procedure.

To load XLCONST.BAS or VBACONST.BAS into a code module

1. In the development environment, choose the New Module command from the File menu. This adds an empty code module to your project.

2. Make sure the module is the active window, and then choose Load Text from the File menu.

3. From the List Files Of Type box, select Basic Files (*.BAS).

4. Select one of the files containing constants, and choose the Replace button. If you want to add a file to a module in which you already have other declarations, choose the Merge button.

WHERE ARE BUILT-IN CONSTANTS LISTED IN MICROSOFT EXCEL?

All of the built-in constants available in Microsoft Excel are listed

in Visual Basic Reference Help. If you want to know which built-in constants are available for a particular function, you can look them up in Microsoft Excel in Visual Basic Reference Help or you can use the Object Browser.

To look up a built-in constant in Microsoft Excel Help

1. In Microsoft Excel, choose Contents from the Help menu.

2. In the Microsoft Excel Help Contents window, choose Programming With Visual Basic.

3. In the Visual Basic Reference window, choose the Search button.

4. In the Search dialog box, type the name of the function you want to view.

5. Choose the Show Topics button.

6. From the Topics list, select the function you want to view, and choose the Go To button.

Any built-in constants that are available to the selected function appear in bold in the help topic text for that function. For example, the ConvertFormula method uses the following six built-in constants, all listed in bold: xlA1, xlR1C1, xlAbsolute, xlAbsRowRelColumn, xlRelRowAbsColumn, and xlRelative.

To look up constants with the Object Browser

1. In a Visual Basic module, choose Object Browser from the View menu.

2. From the Libraries/Workbooks list, select Excel.

3. From the Objects/Modules list, select Constants.

A list of the built-in constants appears in the Methods/Properties list box.

For a complete list of the Microsoft Excel built-in constants, use the XLCONST.XLS file located on the WE0993 disk that is included with this Application Note.

USING THE LISTS OF BUILT-IN CONSTANTS ON THE WE0993 DISK

Two lists of these built-in constants are available in the XLCONST.XLS file that is located on the enclosed WE0993 disk. For your convenience, these lists are organized alphabetically and numerically.

To print the lists from the XLCONST.XLS file

1. Insert the WE0993 disk in a floppy disk drive on your computer.

2. In File Manager, copy the XLCONST.XLS file from the disk to a directory on your hard disk drive.

3. Start Microsoft Excel.

4. Open the XLCONST.XLS file that is located on the directory on your hard disk drive, and select the tab of the worksheet that you want to print.

5. From the File menu, choose Print.

The information in these lists is organized in the following format

```
   Constant      |   Object::Statement   |  Value
-------------------------------------------------
   <xlConstant>  |                       | <Value>
                 | <Object>::<Statement> |
                 | <Object>::<Statement> |
```

where <xlConstant> represents the name of the built-in constant, <Value> is the numerical value of the built-in constant, and <Object> is the object to which <Statement> is applied. The following is an example of an entry for the xlFilterInPlace built-in constant:

```
   Constant         | Object::Statement     | Value
-------------------------------------------------
                    |                       |
   xlFilterInPlace  |                       |   1
                    | Range::AdvancedFilter |
```

In this example, the built-in constant is xlFilterInPlace, its value is 1, and xlFilterInPlace is used by the AdvancedFilter method, which is applied to Range.

Note that a built-in constant (xlNone for example) may be available to multiple functions, so more than one object and statement may be listed below each built-in constant.

TO OBTAIN THIS APPLICATION NOTE
==========

- On CompuServe, GEnie, and Microsoft OnLine, Application Notes are located in the Microsoft Software Library. You can find WE0993 in the Software Library by searching on the word WE0993, the Q number of this article, or S14650. WE0993 is a compressed, self-extracting file. After you download WE0993, run it to extract the file(s) it contains.

- Application Notes are available by modem from the Microsoft Download Service (MSDL), which you can reach by calling (206) 936-6735. This service is available 24 hours a day, 7 days a week. The highest download speed available is 14,400 bits per second (bps). For more information about using the MSDL, call (800) 936-4100 and follow the prompts. To obtain WE0993, download WE0993.EXE. WE0993.EXE is a compressed, self-extracting file. After you

download WE0993, run it to extract the file(s) it contains.

- On the Internet, Application Notes are located on the Microsoft anonymous ftp server, which you can reach by typing "ftp ftp.microsoft.com" (without the quotation marks) at the ">" command prompt.

- If you are unable to access the source(s) listed above, you can have this Application Note mailed or faxed to you by calling Microsoft Product Support Services Monday through Friday, 6:00 A.M. to 6:00 P.M. Pacific time at (206) 635-7070 If you are outside the United States, contact the Microsoft subsidiary for your area. To locate your subsidiary, call Microsoft International Customer Service at (206) 936-8661.

Additional reference words: 5.00

XL5: Can't Use Group Edit Mode on Multiple Workbooks

Article ID: Q112681

The information in this article applies to:

- Microsoft Excel for Windows version 5.0

SUMMARY

Microsoft Excel version 5.0 does not allow group editing across multiple workbooks. Sheets can be grouped within a workbook by selecting multiple sheet tabs in the workbook.

NOTE: In Microsoft Excel version 4.0, you have the option of doing a group edit across multiple workbooks by using the Group Edit command on the Options Menu.

MORE INFORMATION

By grouping sheets, you can perform tasks on all of the sheets in the group simultaneously. For example, you could enter a value in cell A1 on all sheets in one operation.

Group adjacent sheets in the same workbook

Click the first sheet tab, hold down the Shift key and then click the last sheet tab in the group you want to select.

Group nonadjacent sheets in the same workbook

Click the first sheet tab, hold down the CTRL key, and then click the other sheet tabs that you want in the group.

To group sheets in the different workbooks

To group sheets in different workbooks, you must use a macro to display the Microsoft Excel version 4.0 Group Edit box.

Note that it is recommended to use the Excel 5.0 method of creating a group edit, but if there is a situation that requires a group edit across more than one workbook, the procedure below will provide that functionality.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of

this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

1. To use the Microsoft Excel version 4.0 Group Edit dialog box, enter the following in a Visual Basic module sheet:

   ```
   Sub Group_Edit()
      'Brings up the built in Excel Group Edit dialog
      Application.Dialogs(xlDialogWorkgroup).Show
   End Sub
   ```

2. To run the procedure, position the cursor in the line that reads "Group_Edit()" and either press the F5 key or choose Start from the Run menu.

REFERENCES
==========

For more information on using built in dialog boxes in a procedure do the following:

1. In a module sheet, choose Object Browser from the View menu.

2. Select the Excel Library, and then choose the Constants object.

3. Scroll the list in the Methods/Properties box until you find the constants that begin with xlDialog.

Additional reference words: 5.00

XL5: Can't Use Dialog Method to Bring Up Spelling Dialog Box

Article ID: Q112771

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOM

In Microsoft Excel 5.0, the Dialogs method in Visual Basic, Applications Edition, can be used to display over 200 built-in dialog boxes. However, the Spelling tool dialog box cannot be shown using the Dialogs method.

CAUSE

The Spelling tool dialog box cannot be shown using this method because it does not have a defined constant in Microsoft Excel 5.0.

WORKAROUND

This problem only affects the ability to display the dialog box for the Spelling tool. You can run the Spelling tool by using the CheckSpelling method as in the following example:

```
ActiveSheet.CheckSpelling
```

MORE INFORMATION

The Dialogs method can be used to display various built-in Microsoft Excel 5.0 dialog boxes. For example, the following Visual Basic, Applications Edition, procedure uses the Dialogs method and the constant xlDialogOpen to show the Open dialog box (the dialog box you get when you choose Open from the File menu):

```
Sub ShowFileOpen()
    Application.Dialogs(xlDialogOpen).Show
End Sub
```

To look up constants with the Object Browser

1. In a Visual Basic module, choose Object Browser from the View menu.

2. From the Libraries/Workbooks list, select Excel.

3. From the Objects/Modules list, select Constants.

A list of the built-in constants appears in the Methods/Properties list box. To find the dialog box constants, scroll the list until you find the constants that begin with xlDialog. These constants correspond to dialog

box names; for example, the constant for the Find File dialog box is xlDialogFindFile.

For more information about using constants in Microsoft Excel 5.0 and Visual Basic, query on the following words in the Microsoft Knowledge Base:

   appnote and constants and using and visual and basic

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

REFERENCES
==========

"Visual Basic User's Guide," version 5.0, page 236

Additional reference words: 5.00 msapps vbappcode

XL5: Dialogs(xlDialogFont) Changes Normal Style

Article ID: Q112785

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, using the Application.Dialogs(xlDialogFont) built-in dialog box modifies the font of the Normal style of the worksheet, not the font of the current selection.

MORE INFORMATION

In the Microsoft Excel version 4.0 macro language, you can use the following macro code to invoke a built-in dialog box to change the font attribute for the current selection:

```
=FORMAT.FONT?()
```

In the above macro instruction, the question mark (?) tells Microsoft Excel to display the Font dialog box so that you can select the font changes to be applied to the current selection.

In Microsoft Excel version 5.0, the Visual Basic equivalent command is as follows:

```
Sub ChangeFontDialog()
   Application.Dialogs(xlDialogFormatFont).Show
End Sub
```

This code displays a dialog box that changes the font of the current selection. If you use xlDialogFont instead, you will modify the font of the Normal style in the worksheet.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual

Basic User's Guide."

REFERENCES
==========

"Visual Basic User's Guide," version 5.0, Chapter 11,
"Displaying a Custom Dialog Box," "Displaying a Built-in Dialog Box"

For more information about Dialogs, choose the Search button in
Help and type:

    Dialogs

Additional reference words: 5.00 constant

XL5: OLE Automation, Can't Use Named Arguments in Visual Basic

Article ID: Q112813

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY
=======

Using a Microsoft Visual Basic application, you cannot call a Microsoft Excel OLE automation object using named arguments. Instead, you must use arguments in their correct order, using commas as place holders even for omitted arguments.

In Visual Basic in Microsoft Excel (called the Microsoft Visual Basic Programming System, Applications Edition), names are defined for the arguments to their properties and methods. These names allow you to list arguments in any order or omit preceding arguments.

The following is an instruction in Visual Basic in Microsoft Excel that pastes the entire contents of the clipboard on a worksheet in the current selection, skipping any blank cells that were copied:

```
' Using Named arguments
Selection.PasteSpecial Paste:=xlAll, SkipBlanks:=True
```

To use the same instruction in the Visual Basic Standard or Professional Edition, use the following syntax

```
' Using arguments in correct order, commas as place holders
xl.Selection.PasteSpecial xlall, , True
```

where xl is an object variable that refers to Microsoft Excel (the complete Visual Basic Standard or Professional Edition code is included below). The extra comma after the xlall value is a place holder for the Operation argument.

NOTE: In Microsoft Excel, there are many symbolic constants defined that are used for application-specific settings. These constants all begin with the letters "xl". For example, xlall in the above example is used to paste all the contents of the clipboard with the Paste.Special command.

These Microsoft Excel constants are available automatically in Microsoft Visual Basic for Microsoft Excel. For the Standard or Professional Editions of Visual Basic, you must either add the Microsoft Excel constants file XLCONST.BAS to your project to use the constant by name as in the above example, or you must use the numeric value of the constant. The Microsoft Excel constants file includes the name of each constant along with the value of that constant.

This constants file is included with the "Built-in Constants in Microsoft Visual Basic Programming System, Applications Edition"

(WE0993) application note.

For more information about obtaining this application note, query on the following words in the Microsoft Knowledge Base:

   xlconst.bas and application and note

To see a list of the Microsoft Excel constants, do the following:

1. Insert or select a module sheet.

2. From the View menu, choose Object Browser.

3. From the Libraries/Workbooks list, select Excel.

4. From the Objects/Modules list, select Constants.

The Methods/Properties list contains the Microsoft Excel constants.

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

The syntax for the PasteSpecial method in Visual Basic in Microsoft Excel is the following

   object.PasteSpecial(Paste, Operation, SkipBlanks, Transpose)

where the named arguments are included in the parentheses.

In Visual Basic Standard or Professional Edition, if you want to leave out an argument such as operation in the example above, you must indicate the missing argument with a comma. Trailing commas at the end of an instruction can be omitted. In the example above, it is not necessary to add additional commas after the final argument (SkipBlanks).

The following procedure in Visual Basic Standard or Professional Edition opens the workbook TEST.XLS, selects the range A1:B6, copies the cells to the clipboard, selects a new range, and pastes the entire copied range (except for any blank cells) to this location.

```
   Sub Form_Load ()
      ' Dimension variable xl as object type
      Dim xl As object
      ' Activate Microsoft Excel and assign to variable xl
      Set xl = GetObject(, "Excel.Application.5")
      ' Open workbook TEST.XLS
      xl.Workbooks.Open "c:\excel\test.xls"
      ' Select range A1:B6 on active worksheet
      xl.Activesheet.Range("A1:B6").Select
      ' Copy selection to clipboard
      xl.Selection.Copy
      ' Select cell A11
      xl.Activesheet.Range("A11").Select
      ' Paste contents of clipboard to active cell
      ' Skip blank cells in copy range
      xl.Selection.PasteSpecial xlall, , True
```

```
      Set xl = Nothing
   End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Additional reference words: 5.00

XL5: Can't Print Collated Copies

Article ID: Q112821

The information in this article applies to:

- Microsoft Excel for Windows, 5.0

SYMPTOMS

In Microsoft Excel 5.0, when you print multiple copies of a document, you cannot automatically collate the pages unless the printer driver supports collation.

NOTE: In earlier versions of Microsoft Excel, a document can be collated when you print it (regardless of whether the printer driver supports collated printing).

STATUS

This method of printing is by design to allow for faster printing.

WORKAROUNDS

In Microsoft Excel 5.0, to print collated copies of a file, do any of the following:

- Print the document to a file on your hard drive and copy the file to the printer multiple times from the MS-DOS prompt.

  -or-

- Write a macro to print the selected file multiple times. Note that this option will be very time consuming because Microsoft Excel has to print single copies of the file multiple times. It is much faster to print a file multiple times. (Sample code is provided below.

  -or-

- If your printer driver provides the option to print collated copies, use this option. You can set this option by choosing the Print command on the File menu and then choosing Printer Setup in Microsoft Excel to access the printer driver settings.

MORE INFORMATION

An example of collated printing is when you print two copies of a file such that page 1 and page 2 of the first copy are printed, and then page 1 and page 2 of the second copy are printed. In contrast, uncollated printing would print page 1, and then another page 1, and then page 2, and then another page 2.

Printer drivers that have a Copies box in the Print dialog box support collation. Most UNIDRV-based drivers support collation; some printer drivers, such as the Hewlett-Packard (HP) DeskJet, do not. In most programs, you will not be able to clear the Collated Copies option in the Print dialog box when you are using one of these drivers.

SAMPLE VISUAL BASIC FOR APPLICATIONS CODE

The following Visual Basic for Applications code allows you to print multiple copies.

```
'A procedure to print multiple copies
Sub Example()
   Ncopies = Application.InputBox("Please enter number off copies _
       to print: ", "Print Multiple Copies", 1, , , , , 1)
   For counter = 1 To Ncopies
      ActiveWindow.SelectedSheets.PrintOut Copies:=1
   Next
End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Additional reference words: 5.00 multiple copies dialogue VBAppCode

XL5: "Invalid Procedure Call" with SendKeys Statement

Article ID: Q112864

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

The error message "Run Time Error '5': Invalid Procedure Call" may be generated when you use the SendKeys statement in a Visual Basic procedure to send multiple spaces.

WORKAROUND

To work around this problem, use the Application.SendKeys statement in place of the SendKey statement, as demonstrated below:

```
Sub SendKeysTest()
    Application.SendKeys "{  20}" 'There are 20 spaces before the
    number 20
End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

MORE INFORMATION

Steps to Reproduce Behavior

The SendKeys statement in Visual Basic, Applications Edition, sends one or more key commands to the active window as if the they were typed at the keyboard. To specify repeating keys in the SendKeys statement, the syntax used is "{key number}" (without the quotation marks). A space must be placed between key and number. For example, {LEFT 42} means press the LEFT ARROW key 42 times and {h 10} means press h 10 times.

The following example should send 20 spaces to the active window:

```
Sub SendKeysTest()
    SendKeys "{  20}" 'There are two spaces before the number 20
End Sub
```

However, if you run this procedure, your receive the error message "Run Time Error '5': Invalid Procedure Call."

Additional reference words: 5.00 vbappcode

XL5: Docerr: VB for Apps Sub Example Incorrect

Article ID: Q113046

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

The following Visual Basic Help example for the Sub statement contains an incorrect variable:

```
   Sub SubComputeArea(Length, Width)          ' Sub with two arguments.
        Dim Area As Double                      ' Declare local variable.
        If Length = 0 Or Width = 0 Then         ' If either argument = 0.

           Exit Sub                             ' Exit Sub immediately.

         End If

        Area = Length * Width                  ' Calculate area of
   rectangle.
        Debug.Print Area                          ' Print Area to Debug
   window.

   End Sub
```

In the example, a variable called Width is used. =Width is a reserved keyword in Visual Basic and cannot be used as a variable.

WORKAROUND

Changing each reference to Width in the example to gWidth (or another variable name that is not reserved) will allow the example macro to function correctly. The following example is a corrected version of the sample procedure:

```
   Sub SubComputeArea(Length, gWidth)       ' Sub with two arguments.
       Dim Area As Double                   ' Declare local variable.
       If Length = 0 Or gWidth = 0 Then     ' either argument = 0.

           Exit Sub                         ' Exit Sub immediately.

         End If

        Area = Length * gWidth              ' Calculate area of rectangle.
        Debug.Print Area                    ' Print Area to Debug window.

   End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied,

including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Additional reference words: 5.00 vbappcode

XL5: Visual Basic Macro to Concatenate Columns of Data

Article ID: Q113237

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

In Microsoft Excel 5.0, you can use the following Visual Basic, Applications Edition, macro to concatenate the data from two adjacent cells and display the result in the cell to the right of the concatenated cells.

Visual Basic Macro to Concatenate Two Cells

```
Sub ConcatColumns()

   Do While ActiveCell <> ""  'loops until the active cell is blank

      'The "&" must have a space on both sides or it will be
      'treated as a variable type of long integer. Enter the following two
      'lines as a single line.

      ActiveCell.Offset(0, 1).FormulaR1C1 = ActiveCell.Offset(0, -1) & _
         " " & ActiveCell.Offset(0, 0)

      ActiveCell.Offset(1, 0).Select
   Loop

End Sub
```

NOTE: The statement ActiveCell.Offset(0, 1).FormulaR1C1 can be replaced with the statement ActiveCell.Offset(0, 1).Formula. They can be used with equal success if you are using text and numbers only (not formulas). The R1C1 used at the end of the first statement refers to row one, column one and is the form used in examples in Help.

To run the macro:

1. In a Visual Basic module, enter the above macro code.

2. Switch to the worksheet that contains the data that you want to concatenate.

3. Make the top cell in the second column of data the active cell. For example, if your data is contained in the ranges A1:A100 and B1:B100, make B1 the active cell.

4. From the Tools menu, choose Macro, select the ConcatColumns macro, and choose Run.

Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Additional reference words: 5.00 vbappcode operator ampersand

XL5: User-Defined Function to Put Sheet Name in a Cell

Article ID: Q113392

The information in this article applies to:

- Microsoft Excel for Windows version 5.0

SUMMARY

The following Visual Basic, Applications Editions, procedure used in Microsoft Excel version 5.0, results in a user-defined function, Sheetname(), that will place a cell's sheet name into the cell of a Microsoft Excel worksheet:

```
Function Sheetname()
   Application.Volatile
   Sheetname = Application.Caller.Parent.Name
End Function
```

MORE INFORMATION

For more information about creating user-defined functions in Microsoft Excel 5.0, choose the Search button in Help and type:

   user-defined function

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

REFERENCES

Additional Reference Words 5.00

XL5: Using Visual Basic to Multiply Cell Contents

Article ID: Q113394

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel, the following Visual Basic procedure multiplies each cell in the currently-selected range by a user-supplied input and displays the results in the original cells on the active worksheet. The procedure then determines the number of rows and columns in the selection.

For example, if the currently-selected range is A1:C3, and you enter 5 into the input box, the subroutine will multiply each cell in the range by 5.

Sample Visual Basic Code

```
Sub MultCellsInRange()
   Mult = InputBox("Enter the number to be used as the multiplier:")
   For Each xCell in Selection
      xCell.Value = xCell.Value * Mult
   Next xCell
End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Additional reference words: 5.00 vbappcode

XL5: Caption Property of Menu Contains Ampersands

Article ID: Q113490

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOM

In Microsoft Excel version 5.0 for Windows, the Caption property of a Menu or a MenuItem will contain an ampersand character (&) if the Menu or MenuItem has a quick key assigned to it. If an ampersand actually appears in the name of the Menu or MenuItem, the Caption property for that item will contain two ampersands right next to each other.

When you read the Caption property in a Visual Basic macro (for example, if you display this property in a message box), this ampersand will appear as part of the text for the caption.

CAUSE

This behavior occurs because, although ampersands are used to assign quick keys to a Menu or MenuItem, they are interpreted as part of the caption text.

MORE INFORMATION

When you create the Menu or MenuItem, either in a macro or by using the Menu Editor, an ampersand is used to indicate that the next character is the designated quick key.

For example, if you want to add the command "Begin Process" to a menu, and you want the P in "Process" to be a quick key, you would type "Begin &Process" in the Caption field in the Menu Editor. The "Begin Process" menu item would then appear with an underline under the letter P.

If you want a Menu or MenuItem to actually contain an ampersand, you must enter two ampersands directly adjacent to each other.

For example, to create a Menu called "Shipping & Receiving", you would enter "Shipping && Receiving" in the Caption field. If you also wanted to assign a quick key to the S in "Shipping", the Caption field would contain "&Shipping && Receiving".

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure

is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Example 1--Results of reading the Caption property
of a Menu or MenuItem that has a quick key assigned to it

To run this first example, position the cursor in the line which reads "Sub GetCaptions1()" and either press the F5 key or choose Start from the Run menu.

```
'-------------------------------------------------------------------------
Option Explicit

Sub GetCaptions1()

   'Dimension variables.
   Dim Alpha As String

   'Put the Caption of the third menu item of the first menu of the
   'first menu bar into the variable Alpha.
   Alpha = MenuBars(1).Menus(1).MenuItems(3).Caption

   'Show the Caption. It should be "&Close".
   MsgBox Alpha

   'Put the Caption of the first menu of the first menu bar into the
   'variable Alpha.
   Alpha = MenuBars(1).Menus(1).Caption

   'Show the Caption. It should be "&File".
   MsgBox Alpha
End Sub
'-------------------------------------------------------------------------
```

Example 2--The results of reading the Caption property
of a Menu or MenuItem that actually contains an ampersand

To run the second example, position the cursor in the line which reads "Sub GetCaptions2()" and either press the F5 key or choose Start from the Run menu.

```
'-------------------------------------------------------------------------
Option Explicit
```

```
Sub GetCaptions2()

   'Dimension variables.
   Dim Bravo As String

   'Add a menu item "Shipping & Receiving" to the first menu bar.
   'Enter the following two lines as a single line.
   MenuBars(1).Menus.Add Caption:="&Shipping && Receiving", _
      Before:="File"

   'Put the Caption of the first menu of the first menu bar into the
   'variable Bravo.
   Bravo = MenuBars(1).Menus(1).Caption

   'Show the Caption. It should be "&Shipping && Receiving".
   MsgBox Bravo
End Sub
'-----------------------------------------------------------------------
```

Additional reference words: 5.00

XL5: Visual Basic Code to Use Instead of DIRECTORIES()

Article ID: Q113491

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

This article contains sample Visual Basic code that you can use to duplicate the behavior of the DIRECTORIES() function that was included in the Microsoft Excel 4.0 FILEFNS.XLA add-in.

NOTE: The FILEFNS.XLA add-in is not included in Microsoft Excel 5.0.

MORE INFORMATION

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Sample Visual Basic Code

```
' The main procedure calls dir_test and passes it to the specified
' directory.

Sub main()
   'Always make sure the path ends with a backslash.
   'Calls dir_test and passes it the directory to check
   dir_test "c:\"
End Sub

' The dir_test procedure returns an array of all subdirectories contained
' in the specified directory. To do this, it creates the array dir_array;
' dir_array is declared as Static to hold its values after the macro has
' finished.

Sub dir_test(directory_text)
```

```
   'Dimensions a variable to hold the temp Directory name
   Dim temp_var As String

   'Dimensions a dynamic array to hold the Directory Array
   Static dir_array() As String

   'Turns off Error Checking
   On Error Resume Next

   'Call the DIR function and returns the first item the Directory
   temp_var = Dir(directory_text, vbDirectory)

   'Initializes the variable for building the array
   counter = 0

'Set a loop until the DIR function returns
Do Until temp_var = ""
    'Temp_var stores the individual Directory name
    temp_var = Dir()
    'Checks to see if temp_var is an empty string
    If temp_var <> "" Then
       'The following code will create an array of the directories
        If GetAttr(directory_text + temp_var) = 16 Then
            'enlarge the array to hold a new item
            ReDim Preserve dir_array(counter)
            'add directory to the array
            dir_array(counter) = temp_var
            'This line just shows the variable was added to the array
            'You can comment this line out when using this function
            MsgBox dir_array(counter)
            'Increase counter to make room for the next directory
            counter = counter + 1
       End If
    End If
Loop
End Sub
```

REFERENCES
==========

For more information about FILEFNS.XLA and the Visual Basic equivalents for those functions, query on the following words in the Microsoft Knowledge Base:

   FILEFNS.XLA and add-in and maintenance

Additional reference words: 5.00 Array Dir list directory

XL5: Shell Function Doesn't Accept Built-in Constants

Article ID: Q113631

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0 for Windows, it is not possible to use built-in constants, such as xlNormal, xlMinimized, or xlMaximized, for the "windowstyle" argument of the Visual Basic Shell function. This behavior is by design.

MORE INFORMATION

The Visual Basic Shell function is used to run an executable program from a Visual Basic subroutine. Here is an example of the Shell function that will run the Microsoft Windows Notepad program in a normal window with focus:

```
RetVal = Shell("c:\windows\notepad.exe", 1)
```

The second argument (in this case, 1) is the windowstyle argument of the function. Valid windowstyle values are as follows:

Value	Window Style
1, 5, 9	Normal with focus
2	Minimized with focus
3	Maximized with focus
4, 8	Normal without focus
6, 7	Minimized without focus

The windowstyle you specify determines how the application's window will appear when it is started.

It is not possible to use built-in constants such as xlNormal, xlMinimized, or xlMaximized in place of the standard numerical values shown here. This is because the values of the built-in constants are not valid when applied to the Shell command.

The values of the various built-in constants that might be expected to work as windowstyle arguments are as follows:

Constant	Value
xlNormal	-4143
xlMinimized	-4140
xlMaximized	-4137

If you try to use a built-in constant for the windowstyle argument, you will receive the following error message:

```
   Run-time error '5':
   Invalid procedure call
```

Steps to Reproduce Behavior

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

The following examples illustrate proper, and improper, use of the windowstyle argument in the Shell function.

```
'-----------------------------------------------------------------------
Option Explicit

Sub ProperShellFunction()

   'Dimension variables.
   Dim RetVal As Integer

   'This Shell function will work because the windowstyle argument is
   'valid (1).
   RetVal = Shell("c:\windows\notepad.exe", 1)
End Sub

Sub ImproperShellFunction()

   'Dimension variables.
   Dim SecondRetVal As Integer

   'This Shell function will NOT work because the windowstyle argument
   'is invalid (cannot use this built-in constant in this case).
   SecondRetVal = Shell("c:\windows\notepad.exe", xlNormal)
End Sub
'-----------------------------------------------------------------------
```

REFERENCES
==========

For more information on the Shell function, choose the Search button in Visual Basic Help and type:

Shell

Additional reference words: 5.00

XL5: Visual Basic Nested Case Statement Example

Article ID: Q113659

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY
=======

In Microsoft Excel version 5.0, you can use nested case statements in Visual Basic to execute one of several expressions based on a particular value or expression evaluation similar to an IF statement.

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

The following example returns a text expression in the next column, depending upon whether a particular date falls within a specific beginning and ending date range. The active cell on the worksheet should be the upper-left cell containing the dates.

Macro Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

```
Sub TestDate()

'The following nested case statement determines if a date within
'a column of dates falls within a certain range. If the beginning date
'or the ending date is within or out of the range, it displays a message
'in the next column.

  begin_date = DateValue(InputBox(prompt:="Enter the beginning date"))
  end_date = DateValue(InputBox(prompt:="Enter the ending date"))

  Do While ActiveCell <> ""
```

```
    'Checks the date of the current cell.
    Select Case ActiveCell
        Case Is >= begin_date

        'Nested Case to check for ending date of current cell.
        Select Case ActiveCell

            Case Is <= end_date
            ActiveCell.Offset(0, 2) = "Meets criteria"

            'Does this if the ending date is out of range.
            Case Is >= end_date
            ActiveCell.Offset(0, 2) = "Ending date out of range"

            End Select

    'Does this if the beginning date is out of range.
    Case Else
        ActiveCell.Offset(0, 1) = "Beginning date is out of range"
    End Select

    'Selects the next cell on the active worksheet.
    ActiveCell.Offset(1, 0).Select

    Loop

End Sub
```

Additional reference words: 5.00 VBA HOWTO nest case if

XL5: Can't Select Label Attached to Filled Radar Series

Article ID: Q113795

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, it may not be possible for a Visual Basic subroutine to select an individual data label attached to a filled radar series.

WORKAROUND

If your subroutine must select an individual data label attached to a filled radar series, you can use the Application.ExecuteExcel4Macro method to select the label.

Normally, if you want to select the fourth label in the second series, you would use the following Visual Basic code:

```
ActiveChart.SeriesCollection(2).Points(4).DataLabel.Select
```

However, if the label is attached to a filled radar series, you must use the following instead:

```
Application.ExecuteExcel4Macro "SELECT(""Text S2P4"")"
```

Note that the double quotation marks inside the parentheses are necessary because the method will remove a complete set when the command is executed.

MORE INFORMATION

In Microsoft Excel, there are two types of radar charts:

Type of Radar Chart	Description
Wireframe	Lines connect the points but are not filled.
Filled	All lines connect the points and a fill pattern is applied such that a polygon is formed, with the series forming the perimeter of the polygon.

In the Microsoft Excel ChartWizard, only the type 6 radar chart is a filled radar chart (all other types of radar chart are wireframes). You cannot mix

a wireframe and a filled radar series in the same chart.

If you are recording a Visual Basic macro and you select an individual data label attached to a series (in this example, the fourth point of the second series), the recorded code looks like this:

```
   ActiveChart.SeriesCollection(2).Points(4).DataLabel.Select
```

If you run this line of code when the second series is a filled radar series, you will receive the error message

```
   Run-time error '1006':
   Unable to get the DataLabel property of the Point class
```

The code will run correctly if the series is any other type of series, including a wireframe radar series.

To select an individual data label attached to a filled radar series, you must use the Application.ExecuteExcel4Macro method, shown above.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

This Visual Basic code example selects the label attached to the fifth point of the second series in a chart and moves the label to the coordinates 20,20. The series is a filled radar series, so the Application.ExecuteExcel4Macro is used to avoid the error described above.

```
'-----------------------------------------------------------------

Sub MoveDataLabel()

   'Select the fifth point (P5) of the second series (S2).
   'Normally, you would use
   '
   '   ActiveChart.SeriesCollection(2).Points(5).DataLabel.Select
   '
   'to do this, but this method does not work with a filled radar
   'chart.
   Application.ExecuteExcel4Macro "SELECT(""Text S2P5"")"
```

```
   'Change the selection's (the label's) Left property.
   Selection.Left = 20

   'Change the selection's (the label's) Top property.
   Selection.Top = 20
End Sub
'---------------------------------------------------------------------
```

Additional reference words: 5.00

XL5: Recorded AutoFit Selection Changes Entire Row or Column

Article ID: Q113991

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

In Microsoft Excel 5.0, when you record a macro that includes the AutoFit Selection command, the resulting macro will apply the AutoFit command to the entire row or column rather than to the desired selection.

CAUSE

The recorded Visual Basic code uses the EntireColumn property to qualify the AutoFit method.

The only objects that you can use with the AutoFit method are EntireColumn or EntireRow. Using the Selection or Range objects will result in an error when you run the macro. The AutoFit method does not allow the current selection as an object qualifier, and, therefore, is not able to imitate the functionality of AutoFit Selection menu command.

WORKAROUND

To imitate the behavior of the AutoFit Selection command in a Visual Basic macro, you can use the ExecuteExcel4Macro method to call the Microsoft Excel 4.0 Column.Width function.

The following subroutine illustrates a method to perform an AutoFit on the current selection rather than the entire column. To perform the AutoFit on a row rather than a column, substitute the Row.Height macro function for the Column.Width function.

```
Sub AutoFitColumnSelection()

    'Call the Microsoft Excel 4.0 Macro Function
    'from a Visual Basic subroutine.
    Application.ExecuteExcel4Macro "Column.Width(,,,3)"

End Sub
```

Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Additional reference words: 5.00 auto-fit

XL5: Error Message Using Name of Constant with Show Method

Article ID: Q114183

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

When you use the Show method of the Dialog Object in a Visual Basic macro in Microsoft Excel, and you use the name of a built-in constant as the value of an argument, the following error message may appear:

```
Run-time error '1004':

Show method of Dialog class failed
```

CAUSE

This behavior is by design in Microsoft Excel. You cannot use the name of a built-in constant as an argument for the Show method of the Dialog Object. Because some constants are used as arguments for multiple dialog boxes, there is no way to uniquely identify a constant by name.

WORKAROUNDS

To avoid receiving this error message when you use arguments with the Show method of the Dialog object, use the value of the argument as determined by the Microsoft Excel version 4.0 macro command instead of the name of the constant. For example, if you are using the Show method to display the Save As dialog box, and you want to save the file to the Microsoft Excel Workbook file format, use the following code:

```
Sub SaveAs_Dlg()
   Application.Dialogs(xlDialogSaveAs).Show arg1:="TEST.XLS", arg2:=1
End Sub
```

When you run this macro, the Save As dialog box appears and Microsoft Excel Workbook is selected in the Save File As Type list because the value of the Microsoft Excel workbook file format argument is 1.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line

to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

MORE INFORMATION

To find the value of an individual argument to use with the Show method of the Dialog object in Microsoft Excel, do the following:

1. In Microsoft Excel, choose Contents from the Help menu.

2. From the Microsoft Excel Help Contents window, select Reference Information.

3. From the Reference Information window, select Microsoft Excel Macro Functions Contents.

4. Select the Alphabetical List Of Macro Functions option.

5. From the Alphabetical List Of Macro Functions, select the command that corresponds to the dialog box you want to display using the Show method. For example, if you want to display the Save As dialog box using the Dialog object, select the Save.As command from the Alphabetical List Of Macro Functions.

Additional reference words: 5.00

XL5: OLE Automation Error Using Quit Method with GetObject

Article ID: Q114225

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SYMPTOMS

If you use the GetObject function with a filename in a Microsoft Visual Basic version 3.0 procedure to open a Microsoft Excel workbook and you then use the Quit method to quit Microsoft Excel, the following error message appears:

   OLE Automation error. Worksheet does not have Quit method.

CAUSE

When you use the GetObject function with a filename to open a file, the object linking and embedding (OLE) Automation object that is created is a Worksheet object, not the application object. This error occurs when you use the Quit method with the Worksheet object, because the Quit method does not apply to a Worksheet object.

WORKAROUND

To use the Quit method to quit Microsoft Excel after creating an OLE Automation object in an application that uses Visual Basic, do either of the following:

- Use the Parent property to return the Application object and use the Quit method as in the following Visual Basic procedure:

```
' Dimension variable x as Object type
Dim x As Object
' Set x equal to Microsoft Excel Worksheet object
Set x = GetObject("C:\EXCEL\BOOK1.XLS")
' Quit Microsoft Excel
x.Application.Quit
```

   -or-

- Use the class argument of the GetObject function to return the Application object as the OLE Automation object, then open the desired workbook file as in the following Visual Basic procedure:

```
' Dimension variable x as object type
Dim x As Object
' Set x equal to Microsoft Excel Application object
Set x = GetObject("", "Excel.Application.5")
' Open workbook BOOK1.XLS
```

```
      x.Workbooks.Open "C:\EXCEL\BOOK1.XLS"
      ' Quit Microsoft Excel
      x.Quit
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

MORE INFORMATION

For more information about using the GetObject function or the CreateObject function to activate a Microsoft Excel object as an OLE Automation object, query on the following words in the Microsoft Knowledge Base:

   ole and automation and createobject and getobject

REFERENCES

For more information about the Quit Method, choose the Search button in the Visual Basic Reference and type:

   Quit Method

Additional reference words: 5.00 3.00

XL5: Command to Create Add-in File from Visual Basic Module

Article ID: Q114436

The information in this article applies to:

- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, there is no built-in method in Visual Basic, Applications Edition, that will perform a Make Add-In operation. To create an add-in from a Visual Basic subroutine, you must use the Application.ExecuteExcel4Macro method to call the VBA.MAKE.ADDIN macro command.

MORE INFORMATION

In Microsoft Excel version 5.0, you can create add-in files by choosing Make Add-In from the Tools menu in a Visual Basic module or Microsoft Excel 4.0 macro sheet. When you do this, you are prompted to enter a filename for the add-in that you want to create.

To create an add-in file from a Visual Basic subroutine, use the following line of code

```
   Application.ExecuteExcel4Macro "VBA.MAKE.ADDIN(""TEST.XLA"")"
```

where TEST.XLA is the name of the add-in file to be created.

Note that the following line of code will NOT create an add-in file (even though it seems like it should perform the same function):

```
   ActiveWorkbook.SaveAs "TEST.XLA", xlAddIn
```

This line of code will save the active workbook as a Normal workbook with the filename TEST.XLA.

Visual Basic Code Example

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one

logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

This code shows an example of the correct way create an add-in file using a Visual Basic subroutine:

```
'-----------------------------------------------------------------------
Sub MakeAddIn()

   'This command makes an add-in called TEST.XLA, based on the
   'active workbook.
   Application.ExecuteExcel4Macro "VBA.MAKE.ADDIN(""TEST.XLA"")"
End Sub
'-----------------------------------------------------------------------
```

Additional reference words: 5.00

XL5: Can't Set Value Property in For-Each with Variant Type

Article ID: Q107903

The information in this article applies to:

- Microsoft Visual Basic Programming System, Applications Edition
- Microsoft Excel for Windows, version 5.0

SUMMARY

In Microsoft Excel version 5.0, the default value property of an object or an element in an array cannot be set with a For Each loop if the loop control variable is a Variant data type.

MORE INFORMATION

The following macro example uses a For Each loop to concatenate the letter "A" to the values in a range and then displays the results in a message box. The loop counter in this macro is defined with a Variant data type. When the loop attempts to change the value of the cells in the range by concatenating the variant loop counter, the routine appears to run properly and the new value appears in a message box as expected. However, when a second For Each loop redisplays the values in the range by calling the variant loop counter, the range values have reverted back to their original state (the range values no longer have the "A" concatenated to them).

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code-- comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

Macro Example

```
' In this macro, cellRange is an Object variable
' and cellItem is a Variant variable.

Sub loopTest()

   Set cellRange = ActiveSheet.Range(Selection.Address)
```

```
' The following Concatenates an "A" to the default value of each cell.

   For Each cellItem In cellRange
      cellItem = cellItem & "A"
      MsgBox cellItem
   Next

'The cellRange will return the default value (not value&"A"), even though
'the loop above changed the value of the object.

   For Each cellItem In cellRange
      MsgBox "n = " & cellItem
   Next

End Sub
```

The connection between the object and the variant is broken when the loop attempts to write to the value property of the object. The result of this break is that the new value is assigned to the Variant variable, but the object is not updated. When the first loop references the variant, the Variant variable passes the value that it is holding (instead of getting the object's value property) and the loop appears to work. However, when the second loop references the Variant variable, it is passed the value that the variable is reading from the object--which was never changed.

This behavior is by design.

WORKAROUND
==========

If you are working with a collection of objects, use an explicit value property (cellItem.value=cellItem & "A") or use the Dim statement to declare the control variable as an Object rather than a Variant data type.

If you are working with an array, there is no workaround: when you use a For Each loop with an array, you have read-only access. Although you do not receive an error message if you attempt to write to an array, the array element will not be written to.

Additional reference words: 5.00

How to Manipulate Object's Properties w/ Property Set/Let/Get

Article ID: Q108731

The information in this article applies to:

- Microsoft Visual Basic Programming System, Applications Edition, version 1.0
- Microsoft Excel for Windows, version 5.0

SUMMARY

In Visual Basic for Applications in Microsoft Excel version 5.0 for Windows, you can create your own objects by using modules and then manipulate the properties of those objects by using the Property Set, Property Get, and Property Let statements. This article explains how to use the Property Set/Let/Get syntax.

MORE INFORMATION

Usually, you will define either Property Let and Property Get or Property Set and Property Get. If a property value stores an object reference, use Property Set and Property Get. If a property value stores a variant reference, use Property Let and Property Get.

In the following example, you'll create the object module SGLCount in step 2 and then create another module that uses SGLCount in step 3.

1. Start Microsoft Excel version 5.0. A new Workbook (sheet1) is created by default.

2. Add a new module by using the Insert menu (ALT, I, M, M) or by clicking the Module button on the Visual Basic for Applications toolbar. Name the module SGLCount. Add the following code to the SGLCount module:

```
Option Explicit
Private iMyCount As Integer
Private rMyRange As Variant

' Use Property Let for Variants
Property Let MyCount(iCount As Variant)
   iMyCount = iCount
End Property

Property Get MyCount()
   MyCount = iMyCount
End Property

' Use Property Set for Objects
Property Set MyRange(rRange As Range)
   ' Use Set because rRange is a Range Object
   Set rMyRange = rRange
End Property
```

```
Property Get MyRange()
   Set MyRange = rMyRange
End Property
```

3. Add another new module by using the Insert menu (ALT, I, M, M) or by clicking the Module button on the Visual Basic for Applications toolbar. Use the default name for the module (module1). Add the following code to the module1 module:

```
Option Explicit
Sub TestCount()

   Dim rRange As Range

   Set rRange = ActiveSheet.Range("B1")
   rRange.Value = 7777

   ' Execute module SGLCount Property Set MyRange:
   Set SGLCount.MyRange = ActiveSheet.Range("B1")
   ActiveSheet.Range("A1").Select

   ' Execute module SGLCount Property Set MyCount:
   SGLCount.MyCount = 5

   ' Execute module SGLCount Property Get MyRange:
   rRange = SGLCount.MyRange

   ' Execute module SGLCount Property Get MyCount:
   rRange.Value = SGLCount.MyCount
   rRange.Select

End Sub
```

4. Add a command button (Command1) to Sheet1 and assign the TestCount macro to the button. To assign a macro, place the mouse insertion point on the Command1 button and click the right mouse button. Then select Assign Macro.

Additional reference words: 1.00 5.00
KBCategory:
KBSubcategory:

Line Numbers Greater Than 65529 Not Supported

Article ID: Q111869

The information in this article applies to:

- Microsoft Visual Basic Programming System, Applications Edition, version 1.0

SYMPTOMS

In Microsoft Visual Basic Programming System, Applications Edition, when you type a line number greater then 65529, type code, and then press ENTER to start a new line, you may receive the following error message:

   Expected: line number or label or statement or end of statement

If you do not have the Display Syntax Errors option selected, you receive a syntax error when you run the procedure.

CAUSE

Visual Basic, Applications Edition does not support line numbers greater than 65529.

WORKAROUND

To work around this problem, do one of the following:

- Use line numbers less than 65,529

   -or-

- Use line labels to identify a single line of code. A line label 0 can contain numbers, but must start with a letter and end with a colon.

MORE INFORMATION

A line number is used to identify a single line of code. This number can be any combination of digits that is unique within the module. Line numbers can begin in any column as long as they are the first non-blank characters.

Line numbers greater than 65529 are treated as line labels and cannot be returned by the Erl function.

NOTE: Line numbers are used for backward compatibility; it is not recommended that you use them in your procedures.

Additional reference words: 1.00 5.00 linenum lables

Null Character Truncates String in Visual Basic

Article ID: Q112772

The information in this article applies to:

- Microsoft Visual Basic Programming System, Applications Edition, version 1.0
- Microsoft Excel for Windows, version 5.0

SUMMARY

In a Visual Basic procedure, if you use a string that contains the null value in either the SendKeys statement, the MsgBox function, or the Format function, the string is truncated at the null character.

NOTE: This situation is true for both the Microsoft Excel 5.0 for Windows SendKeys statement (using Application.SendKeys) and the Visual Basic, Applications Edition, SendKeys (using just SendKeys) statement.

MORE INFORMATION

The SendKeys statement sends keystrokes to the active windows as if they were typed on the keyboard. Attempting to send a null character causes the string being sent to be truncated at the location of the null character.

You can use the MsgBox function to display a string expression as a message. However, if you use a string expression that contains the null character in the MsgBox function, the characters in the string after the null character do not display in the message box.

The Format function allows you to return an expression such as a string in a named or user-defined format. If you use the Format function in the Debug window to format a string expression that contains the null character, and if the null character is the first character in the string, the value 0 is returned.

Steps to Reproduce Behavior

The following Visual Basic procedure uses the SendKeys statement to send the word "Testing" to the active window:

```
Sub Testing()
   SendKeys "Testing"
End Sub
```

If you add a null character before the word Testing, the word "Testing" is not sent to the active window. In the following procedure, the keystrokes are not sent to the active window.

```
Sub Testing()
   SendKeys Chr(0) + "Testing"
End Sub
```

If a null character is placed between two (or more) concatenated strings, the string is truncated at the null character. In the following example, the word "Testing" is sent to the active window, but the string "One, Two, Three" is not.

```
Sub Testing()
   SendKeys "Testing" + Chr(0)+ "One, Two, Three"
End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided 'as is' and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line-continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the "Visual Basic User's Guide."

REFERENCES
==========

For more information about the Format Function, choose the Search button in the Visual Basic Reference and type:

```
Format
```

For more information about the Chr Function, choose the Search button in the Visual Basic Reference and type:

```
 Chr
```

Additional reference words: 5.00 Nul Chr$ vbappcode empty

PRB: Error When Excel VBA Proc & Implicit Var Have Same Name

Article ID: Q113947

The information in this article applies to:

- Microsoft Visual Basic, Applications Editions, version 1.0
- Microsoft Excel, version 5.0
- Microsoft Office for Windows, version 4.0

SYMPTOMS

In an Excel Visual Basic for Applications module, if you have a procedure and an implicitly declared variable that share the same name, you will get one of two possible errors:

   Type-declaration character does not match declared data type.

      -or-

   Expected function or variable.

RESOLUTION

Use the Dim statement to explicitly dimension the local variable (ThingOne$ or ThingOne):

```
   Sub ThingOne

   End Sub

   Sub ThingTwo
      Dim ThingOne$   ' Or: Dim ThingOne As Variant
      ThingOne$ = "hi"
   End Sub
```

Or add the Option Explicit statement at the beginning of your code module to force you to explicitly dimension all variables.

STATUS

This behavior is by design. The local variable ThingOne (or ThingOne$) must be explicitly declared or you will get an error. Sub procedures within modules are visible to each other in the Visual Basic, Applications Edition.

Because ThingOne is visible inside ThingTwo (see the code in the More Information section below) and Sub and Function procedures may be called without parameters the reference to ThingOne as a variable is ambiguous.

In the first case, the type char is checked first. The type is determined to be a String. However, the Sub declaration is equivalent to a function

which has a void return. The $ contradicts this void return, so you get an error.

In the second case, without the type character, Visual Basic, Applications Edition checks the return type of the procedure. The return for a Basic Sub is void so it results in the second error.

This behavior can be avoided altogether by using the Option Explicit statement.

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

Steps to Reproduce Behavior
---

1. Start Microsoft Excel version 5.0.

2. From the File menu, choose New to create a new Excel book.

3. From the Insert menu, choose Macro and then choose Module to create a new module in the book.

4. Add the following code to the Excel module:

```
Sub ThingOne

End Sub

Sub ThingTwo
   ThingOne$ = "hi"
End Sub
```

5. Run the macro by choosing Start from the Run menu or by pressing the F5 key. Excel will pop-up an error dialog:

   Type-declaration character does not match declared data type.

6. Replace the above code with the following code.

```
Sub ThingOne

End Sub

Sub ThingTwo
   ThingOne = 4
End Sub
```

7. Run the macro. Excel will pop-up an error dialog:

   Expected function or variable.

Additional reference words: 1.00 4.00 5.00
KBCategory: IAP
KBSubcategory: IAPVBA

PRB: Sub Name Can't Be Valid Cell Reference in Excel VBA

Article ID: Q113952

The information in this article applies to:

- Microsoft Visual Basic, Application Edition, version 1.0
- Microsoft Excel for Windows, version 5.0

SYMPTOMS

Certain Sub or Function procedure names may generate an "Invalid Procedure Name" error in Visual Basic, Applications Edition in Excel even though the syntax is correct.

CAUSE

When creating new modules or renaming existing modules in Excel, Visual Basic, Applications Edition rejects any Sub or Function that is a valid cell reference regardless of whether you are using A1 or R1C1 referencing.

STATUS

This behavior is by design.

MORE INFORMATION

Steps to Reproduce Behavior

1. Create a new book in Excel by choosing New from the File menu.

2. Insert a new module by choosing Macro and then Module from the Insert menu.

3. Add the following code to the new module:

   Sub C3000

4. When you press the ENTER key, you'll see the error "Invalid procedure name." Then the line of code turns red.

Additional reference words: 5.00 1.00
KBCategory:
KBSubcategory: IAPVBA

BUG: VBA FileCopy Updates Destination File's Date & Time Stamp

Article ID: Q113958

The information in this article applies to:

- Microsoft Visual Basic, Applications Edition, version 1.0
- Microsoft Excel, version 5.0
- Microsoft Project, version 4.0

SYMPTOMS

The FileCopy statement in Visual Basic, Applications Edition does not maintain the Date and Time stamp of the source file when the destination file is copied. Unlike the MS-DOS Copy command and the FileCopy statement in Visual Basic version 3.0, the time stamp of the destination file shows the actual time the copy occurs.

WORKAROUND

Because the time stamp placed on the file is based on the current time, setting the system time to the time stamp of the source file prior to copying the file establishes the same time stamp on the destination file. Below is a sample piece of code to copy a file that maintains the same time stamp:

```
Sub TestFileCopy()
   Dim datOriginalDateTime As Date
   Dim datSourceTimeStamp As Date

   Const cSourceFile = "C:\AUTOEXEC.BAT"
   Const cDestFile = "C:\AUTOEXEC.OLD"

   ' Obtain Date/Time Stamp of the Source file:
   datSourceTimeStamp = FileDateTime(cSourceFile)

   ' Store Current time in a temporary variable:
   datOriginalDateTime = Now()

   ' Set System time to that of the Source File:
   Date = datSourceTimeStamp
   Time = datSourceTimeStamp

   FileCopy cSourceFile, cDestFile

   ' Restore System time to correct time:
   Date = datOriginalDateTime
   Time = datOriginalDateTime
   MsgBox "Source Date = " & FileDateTime(cSourceFile) & Chr(13) & _
      Chr(10) & "Destination Date = " & FileDateTime(cDestFile)
End Sub
```

STATUS

======

Microsoft has confirmed this to be a bug in the products listed above. We are researching this bug and will post new information here in the Microsoft Knowledge Base as it becomes available.

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

Steps to Reproduce Problem
----

1. Start Excel, or from the File menu, choose New (ALT, F, N) if Excel is already running.

2. Insert a New module. From the Insert Menu, choose Macro Module (ALT, I, M, M). Module1 is created by default.

3. Insert the following code into Module1:

```
Sub TestFileCopy()
   Const cSourceFile = "C:\autoexec.bat"
   Const cDestFile = "C:\autoexec.old"
   FileCopy cSourceFile, cDestFile
   MsgBox "Source Date = " & FileDateTime(cSourceFile) & Chr(13) & _
      Chr(10) & "Destination Date = " & FileDateTime(cDestFile)
End Sub
```

4. Run the macro. From the Tools Menu, choose Macro (ALT, T, M). From the Macro dialog, select the macro TestFileCopy. Then click the Run button.

The time stamps for the Source and Destination files differ. You would expect them to be the same. Now try the workaround routine, you will see the Source and Destination files are the same.

Additional reference words: 1.00 buglist1.00
KBCategory:
KBSubcategory: IAPVBA

BUG: VBA Shell Returns Invalid Proc Call Even If App Starts

Article ID: Q114001

This information in this article applies to:

- Microsoft Visual Basic, Applications Edition, version 1.0
- Microsoft Excel, version 5.0
- Microsoft Project, version 4.0

SYMPTOMS

The Shell statement can cause a Invalid Procedure Call even though the Shell statement successfully started the desired application.

WORKAROUND

To work around this bug, turn error trapping on and trap the Invalid Procedure Call error as in this example:

```
On Error Resume Next
Shell("C:\WINDOWS\NOTEPAD.EXE")

' Invalid Procedure Call:

If Err <> 5  Then
   ' Insert code to handle other errors here.
   ' Exit if fatal
End if
```

STATUS

Microsoft has confirmed this to be a bug in Microsoft Visual Basic, Applications Edition, version 1.0 that comes with Microsoft Excel version 5.0 and Microsoft Project version 4.0. We are researching this bug and will post new information here in the Microsoft Knowledge Base as it becomes available.

Additional reference words: 1.00 buglist1.00
KBCategory:
KBSubcategory: IAPVBA

"Invalid Data Format" Referencing File that Contains Procedure

Article ID: Q114226

The information in this article applies to:

- Microsoft Visual Basic Programming System, Applications Edition, version 1.0

SYMPTOMS

In a Visual Basic module, when you use the Browse dialog box to reference a file created in another application that contains a Visual Basic procedure, you receive the following error message:

```
Invalid data format
```

CAUSE

This is by design. You cannot, for example, reference a Microsoft Excel workbook file that contains a Visual Basic module, from a Visual Basic module in a Microsoft Project project file.

WORKAROUND

To run a procedure from a Visual Basic module in one application (App1) that is contained in a file created by another application (App2), you can use the methods and properties of the object library exposed by the application that was used to create the Visual Basic procedure (App2).

For example, to run a macro from a Visual Basic module in Microsoft Excel that is contained in a Microsoft Project project file, do the following:

1. In a new module in Microsoft Excel, choose References from the Tools menu.

2. From the Available References list, select the Microsoft Project 4.0 Object Library check box, and choose OK.

3. In the new module, enter the following:

```
Sub Run_Proj_Macro()
   ' Dimension variable Proj as object type
   Dim Proj As Object
   ' Set Proj equal to Microsoft Project application object
   Set Proj = MSProject.Application
   ' Open project file that contains macro
   ' Note this command is not necessary to run a global macro
   Proj.FileOpen "C:\WINPROJ\PROJECT1.MPP"
   ' Run macro
   ' Replace MACRONAME with the name of the macro you want to run
   ' PROJECT1 and Module1 are not necessary but distinguish
```

```
    ' Between multiple macros with the same name
    Proj.Macro "[PROJECT1]Module1!MACRONAME"
    ' Quit Microsoft Project
    Proj.Quit
End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided "as is" and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the Visual Basic "User's Guide."

MORE INFORMATION

In a Visual Basic module in Microsoft Excel, you can use the Browse dialog box to reference another workbook file (*.XLS) or an add-in file, (*.XLA) that contains a Visual Basic procedure.

In a Visual Basic module in Microsoft Project, you can use the Browse dialog box to reference another project file (*.MPP) that contains a Visual Basic procedure.

You can reference a toolbar file, (*.TLB) or an object library file (*.OLB) from a Visual Basic module in either Microsoft Excel or Microsoft Project.

Additional reference words: 1.00 4.00 5.00

How to Dimension a Variable as Name Type

Article ID: Q114287

The information in this article applies to:

- Microsoft Visual Basic Programming System, Applications Edition, version 1.0

SYMPTOMS

In Microsoft Visual Basic Programming System, Applications Edition, if you declare a variable of Name type in your module, as in the following example

```
Dim MyVar as Name
```

you receive a syntax error when you run the code.

CAUSE

In Microsoft Visual Basic in Microsoft Excel, Name is a reserved keyword that conflicts with the Name object of the Names collection.

WORKAROUND

To declare a variable of Name type, explicitly reference the object, as in the following example:

```
Dim MyVar as Excel.Name
```

MORE INFORMATION

The Dim statement in Visual Basic, Applications Edition, is used to declare variables and allocate storage space. The syntax is as follows

```
Dim varname As <type>
```

where <type> is the data type of the variable. This data type can be an object type, such as Name. An object type is a type of object exposed by an application through OLE Automation. An example of using the Dim statement to declare a variable as Name type is as follows:

```
Sub List_Names()
    ' Dimension variable MyName as Name type
   Dim MyName As Excel.Name
   ' Display reference for each name in the names collection
   For Each MyName In ActiveWorkbook.Names
     Count = Count + 1
     MsgBox ActiveWorkbook.Names(Count)
   Next MyName
End Sub
```

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided "as is" and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the Visual Basic "User's Guide."

REFERENCES
==========

For more information about the Dim Statement, choose the Search button in the Visual Basic Reference and type:

    Dim

Additional reference words: 1.00 5.00

Using Square Brackets in a Visual Basic Procedure

Article ID: Q114320

The information in this article applies to:

- Microsoft Visual Basic Programming System, Applications Edition, version 1.0

SUMMARY

Square brackets ([]) are used in Visual Basic 3.0 so that the contents of the brackets are ignored when the code is run. This is useful when you are using OLE Automation statements because the brackets allow a manual override of the syntax checker.

Microsoft Visual Basic Programming System, Applications Edition also uses a syntax checker. When you use brackets in your procedure, they are sometimes removed after you run the procedure. In most cases, this does not cause a problem. However, if the statement needs to be recompiled, for example, if you copy the statement that no longer contains the brackets, and paste it to another location in a module, a syntax error appears when you run the pasted statement.

In most cases, there is another way to reference an object or method with the correct syntax without using brackets. The following is an example.

Example

The Microsoft Excel Name object conflicts with a reserved word.

The following statement works correctly in Visual Basic, Applications Edition:

```
    Dim xl as [Name]
```

When you run the procedure that contains this statement, however, the brackets are removed. To use this statement without brackets, and without receiving an error message, use the following:

```
    Dim xl as Excel.Name
```

MORE INFORMATION

In Visual Basic version 3.0, type declaration characters are not allowed in procedures. However, they are allowed in Visual Basic, Applications Edition. The following is the required syntax in Visual Basic 3.0:

```
   Dim Word As Object
   Set Word = CreateObject("Word.Basic")
   MsgBox Word.[GetBookmark$](Word.[BookmarkName$](1))
```

In Visual Basic, Applications Edition, you can use the following:

```
MsgBox = Word.GetBookmark$(Word.BookmarkName$(1))
```

Note that in this example, the CreateObject function is used to create an object-linking-and-embedding (OLE) Automation object using the Microsoft Word for Windows macro language. When you call a function in the Microsoft Word for Windows macro language, the brackets are generally not required, as shown in the example above. However, there are at least two functions that you must enclose in brackets in order to use in a Visual Basic, Applications Edition macro. These are Language$() and Font$(). The following is an example of how to use the Language$() function in a Visual Basic, Applications Edition macro:

```
Dim Word As Object
Set Word = CreateObject("Word.Basic")
MsgBox Word.[Language$](0)
```

When you run a Visual Basic, Applications Edition procedure that uses a Microsoft Word for Windows function and you use the brackets, the brackets are not removed after you run the commands.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. This Visual Basic procedure is provided "as is" and Microsoft does not guarantee that it can be used in all situations. Microsoft does not support modifications of this procedure to suit customer requirements for a particular purpose. Note that a line that is preceded by an apostrophe introduces a comment in the code--comments are provided to explain what the code is doing at a particular point in the procedure. Note also that an underscore character (_) indicates that code continues from one line to the next. You can type lines that contain this character as one logical line or you can divide the lines of code and include the line continuation character. For more information about Visual Basic for Applications programming style, see the "Programming Style in This Manual" section in the "Document Conventions" section of the Visual Basic "User's Guide."

Additional reference words: 1.00 5.00 6.00 6.00a w4wmacro wordbasic

BUG: SendKeys Fails to Send Right Brace Character in Excel VBA

Article ID: Q114343

The information in this article applies to:

- Visual Basic, Applications Edition, version 1.0
- Microsoft Excel for Windows, version 5.0
- Microsoft Office for Windows, version 4.0

SYMPTOMS

The SendKeys command in Visual Basic, Applications Edition in Microsoft Excel allows you to simulate keystrokes under program control. If a right brace character is part of the string sent, Visual Basic, Applications Edition generates a run-time error 5, "Invalid procedure call."

STATUS

Microsoft has confirmed this to be a bug in Visual Basic, Applications Edition that ships with Microsoft Excel version 5.0 and Microsoft Office version 4.0. We are researching this problem and will post new information here in here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

Steps to Reproduce Problem

1. Start Excel version 5.0.

2. From the File menu, choose New to create a new Book.

3. From the Insert menu, choose Macro Module.

4. Add the following code to the new module.

```
Sub Main ()
   SendKeys "{}}"
End Sub
```

5. Click the Tab for Sheet1 at the bottom of the Excel window.

6. From the Tools menu, choose Macro Run; then select the macro 'Main' that you just created.

The program will quit and give you error 5.

NOTE: The following equivalent code works in Visual Basic for Windows Programming System, version 3.0.

```
Sub Form_Click ()
```

```
   SendKeys "{}}"
End Sub
```

Additional reference words: buglist1.00 1.00 4.00 5.00
KBCategory: IAP
KBSubcategory: IAPVBA

BUG: Is Operator in VBA Incorrectly Evaluates Excel Objects

Article ID: Q114345

The information in this article applies to:

- Microsoft Visual Basic, Applications Edition, version 1.0
- Microsoft Excel, version 5.0

SYMPTOMS

Using the Is operator to evaluate whether or not two object variables reference the same Excel object, incorrectly evaluates to False.

WORKAROUND

Using a property of an object, such as Name, correctly evaluates whether or not the objects are the same. For example:

```
If xlObject1.Name = xlObject2.Name then
   ' Code when the objects refer to our code
End If
```

STATUS

Microsoft has confirmed this to be a bug in Microsoft Visual Basic, Applications Edition, version 1.0 that ships with Microsoft Excel version 5.0. 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 Excel, or from the File menu, choose New (ALT, F, N) if Excel is already running.

2. Insert a New module. From the Insert Menu, choose Macro Module (ALT, I, M, M). Module1 is created by default.

3. Insert the following code into Module1:

```
Sub TestXLObject()
   dim xlObject as Object
   Worksheets(1).Select
   set XlObject = Selection
   If xlObject Is xlObject then beep
End sub
```

4. Run the macro. From the Tools Menu, choose Macro (press ALT, T, M).

From the Macro dialog, select the macro TestXLObject, and press the Run button.

You would expect to hear a beep to indicate expression 'xlObject Is xlObject' is true, but no beep sounds.

Additional reference words: 1.00 5.00 buglist1.00
KBCategory: IAP
KBSubcategory: IAPVBA