# Introduction

The four-part series Controlling the Printer from Word VBA by MVP Jonathan West is the most comprehensive treatment of this topic I have seen. It was originally published in 2002-2003 on the TechTrax web site run by MVP Dian Chapman. That site is no longer available, but its content is stored at the Internet Archive Wayback Machine (<https://web.archive.org/>), from which I created this document.

Jay Freedman

Microsoft Word MVP

# Controlling the Printer from Word VBA

**by Jonathan West, MVP**

## Part 1: Using VBA to Select the Paper Tray

This is the first part of what will, hopefully, be a multi-part article on controlling the printer properties from Word VBA.

### Introduction

There's a very strange thing about Word—in all its versions, since it was introduced on Windows. Word is a word-processor, designed for output to the printed page. The versions for Windows have always had a scripting language (WordBasic up to Word 95, VBA since Word 97). However, the scripting language has never been able to control the properties of the printer, such as whether to print in color or monochrome; or whether or not to print on both sides of the paper; or provide information about the printer, such as what paper trays it has; or what sizes of paper it can take.

For some time, Visual Basic has had the Printers collection and the Printer object, which allowed this kind of control over the printer for applications written in Visual Basic. In Office XP, Access 2002 has acquired the same objects, but still nothing in Word.

It is quite possible that they will get around to it for the next version of Office. That's all very well, but it doesn't help us right now!

### Current Capabilities of Word VBA

In Word VBA, dealing with paper trays is a horrible mess. The Word object model offers the DefaultTrayID, FirstPageTray and OtherPagesTray properties. The first one is a property of the Options object, and defines the default tray used when printing from Word. The other two are properties of the PageSetup object, and are document-specific.

The VBA Help for Word lists a number of constants which it suggests should be used with the DefaultTrayID, FirstPageTray and OtherPagesTray properties. These are as follows.

| Value Word | Constant Name |
| --- | --- |
| 0 |  wdPrinterDefaultBin |
| 1 |  wdPrinterOnlyBin |
| 1 |  wdPrinterUpperBin |
| 2 |  wdPrinterLowerBin |
| 3 |  wdPrinterMiddleBin |
| 4 |  wdPrinterManualFeed |
| 5 |  wdPrinterEnvelopeFeed |
| 6 |  wdPrinterManualEnvelopeFeed |
| 7 |  wdPrinterAutomaticSheetFeed |
| 8 |  wdPrinterTractorFeed |
| 9 |  wdPrinterSmallFormatBin |
| 10 |  wdPrinterLargeFormatBin |
| 11 |  wdPrinterLargeCapacityBin |
| 14 |  wdPrinterPaperCassette |
| 15 |  wdPrinterFormSource |

Unfortunately, no two printers use quite the same names and numbers for their paper trays, and most of them do not use the numbers defined by the Word constants. Take for instance these two printers, and the paper tray names and numbers they use.

HP LaserJet 4Si/Si MX PS Tektronix Phaser 850P

|  |  |
| --- | --- |
| ID | Name |
| 15 |  Automatically Select |
| 256 |  Upper Tray |
| 257 |  Lower Tray |
| 258 |  Envelope Feeder |
| 4 |  Manual Feed |

|  |  |
| --- | --- |
| ID | Name |
| 15 |  Automatically Select |
| 257 |  Paper |
| 258 |  Transparency |
| 259 |  Upper Tray |
| 260 |  Middle Tray |
| 261 |  Lower Tray |
| 262 |  Manual Feed Paper |
| 263 |  Manual Feed Transparenc |

(No, that last item on the Tektronix list isn't a typo, that's what the printer driver actually returns!)

If you use the Word constants when trying to set the paper trays for these printers, in most cases absolutely nothing will happen—the tray won't change. The printer will simply ignore a request to change to a tray number that is not available.

### Getting the Available Paper Bin Names and Numbers

So, we need a way of finding out what paper trays are actually available for the printer you want to use, and what their numbers are. Word VBA doesn't give you direct access to this information, but the Windows API does allow you to obtain this information from the printer driver. With careful programming, the Windows API is accessible from VBA.

The following code provides a means of getting the list of the paper bin names and numbers for the current printer. Paste it into a fresh module. Each function returns a Variant containing an array. GetBinNumbers lists the numbers, and GetBinNames lists the equivalent names for the paper bins. The code is commented so you can see what is happening at each step. If you are not familiar with VB programming of the Windows API, then it will not be at all obvious how it all works even with the comments, but I promise you, it does work!

Warning! This code makes use of a Windows API function to gain access to the printer information. Unless you are confident that you know what you are doing, messing about with the Windows API from VB or VBA is dangerous. Making a mistake in ordinary VBA will just crash your macro. Making a mistake with an API call will often bring down the whole of Word, and in a bad case even the whole of Windows, requiring a reboot. If you want to modify this code in any way, make sure you save everything first. Don't say I didn't warn you!

Option Explicit

Private Const DC\_BINS = 6

Private Const DC\_BINNAMES = 12

Private Declare Function DeviceCapabilities Lib "winspool.drv" \_

 Alias "DeviceCapabilitiesA" (ByVal lpDeviceName As String, \_

 ByVal lpPort As String, ByVal iIndex As Long, lpOutput As Any, \_

 ByVal dev As Long) As Long

Public Function GetBinNumbers() As Variant

 'Code adapted from Microsoft KB article Q194789

 'HOWTO: Determine Available PaperBins with DeviceCapabilities API

 Dim iBins As Long

 Dim iBinArray() As Integer

 Dim sPort As String

 Dim sCurrentPrinter As String

 'Get the printer & port name of the current printer

 sPort = Trim$(Mid$(ActivePrinter, InStrRev(ActivePrinter, " ") + 1))

 sCurrentPrinter = Trim$(Left$(ActivePrinter, \_

 InStr(ActivePrinter, " on ")))

 'Find out how many printer bins there are

 iBins = DeviceCapabilities(sCurrentPrinter, sPort, \_

 DC\_BINS, ByVal vbNullString, 0)

 'Set the array of bin numbers to the right size

 ReDim iBinArray(0 To iBins - 1)

 'Load the array with the bin numbers

 iBins = DeviceCapabilities(sCurrentPrinter, sPort, \_

 DC\_BINS, iBinArray(0), 0)

 'Return the array to the calling routine

 GetBinNumbers = iBinArray

End Function

Public Function GetBinNames() As Variant

 'Code adapted from Microsoft KB article Q194789

 'HOWTO: Determine Available PaperBins with DeviceCapabilities API

 Dim iBins As Long

 Dim ct As Long

 Dim sNamesList As String

 Dim sNextString As String

 Dim sPort As String

 Dim sCurrentPrinter As String

 Dim vBins As Variant

 'Get the printer & port name of the current printer

 sPort = Trim$(Mid$(ActivePrinter, InStrRev(ActivePrinter, " ") + 1))

 sCurrentPrinter = Trim$(Left$(ActivePrinter, \_

 InStr(ActivePrinter, " on ")))

 'Find out how many printer bins there are

 iBins = DeviceCapabilities(sCurrentPrinter, sPort, \_

 DC\_BINS, ByVal vbNullString, 0)

 'Set the string to the right size to hold all the bin names

 '24 chars per name

 sNamesList = String(24 \* iBins, 0)

 'Load the string with the bin names

 iBins = DeviceCapabilities(sCurrentPrinter, sPort, \_

 DC\_BINNAMES, ByVal sNamesList, 0)

 'Set the array of bin names to the right size

 ReDim vBins(0 To iBins - 1)

 For ct = 0 To iBins - 1

 'Get each bin name in turn and assign to the next item in the array

 sNextString = Mid(sNamesList, 24 \* ct + 1, 24)

 vBins(ct) = Left(sNextString, InStr(1, sNextString, Chr(0)) - 1)

 Next ct

 'Return the array to the calling routine

 GetBinNames = vBins

End Function

### Using the Code

Fortunately, you don't need to know all the details of how that code works in order to be able to use it! It has been designed so that minimal additional code is needed when you want to manipulate the paper bins.

If you want to give the user of a VBA macro the choice of which paper bin to use, then it is necessary to display the list of bins. This is quite straightforward. Create a UserForm, and include a ListBox on it (call it ListBox1). To put the list of bin names into the ListBox, just use the following code in the UserForm\_Initialize event, so that the ListBox is filled with the list of paper trays when the UserForm is first displayed.

 ListBox1.List = GetBinNames

Later, if the user has selected a bin, and you now want to assign the selection to the current document, the following code could be used.

Dim vBinNumbers as Variant

If ListBox1.ListIndex >= 0 Then

 vBinNumbers = GetBinNumbers

 ActiveDocument.PageSetup.OtherPagesTray = \_

 vBinNumbers(ListBox1.ListIndex)

Else

 MsgBox "No paper tray has been selected."

End If

That's all there is to it!

## Part 2: Using VBA to control Duplex, Color Mode and Print Quality

In last month's article, the Windows API was used to get information, from the printer, about the paper trays available in the printer. However, everything that was done in that article to control the printer was done through normal Word properties. However, there are no properties in Word to control the duplex, color mode or print quality, so we are going to have to control the printer for that, using more Windows API calls.

The main code routine that implements these calls is listed at the end of the article.

### Printing in Color or Monochrome

If you have a color inkjet printer, it may often happen that you want to save money by printing in monochrome most of the time, and economize on expensive color ink. If you are printing a draft to check layout, you normally don't need color even if the document contains it. So, it would be nice to have a quick way of checking whether the color mode is set to color or monochrome. The main code routine at the end of this article has a GetColorMode function and a SetColorMode subroutine.

The color mode can take two possible values, as shown in the following table:

|  |  |
| --- | --- |
| Value | Meaning |
| 1 | Monochrome |
| 2 | Color |

The following code is an example of how to use the routines:

Sub AskBeforePrintingInColor()

 Dim iColor As Long

 iColor = GetColorMode

 If iColor = 2 Then

 If MsgBox("Do you really want to print in color?", \_

 vbYesNo) = vbNo Then

 SetColorMode 1

 End If

 End If

 ActiveDocument.PrintOut Background:=False

 SetColorMode iColor

End Sub

This routine checks the color mode. If the color mode is currently color, it asks if you really want to print in color and changes to monochrome before printing if you answer no. The code restores the original color mode after printing.

**Important Note!** For all the printer properties set by the routines described in this article, if you change a setting, it is changed as the default printer setting for all applications (not just Word) until you set it back.

### Printing Duplex

Duplex printing works in a very similar fashion to setting the color property, except that there are three possible values, as show in the table below:

|  |  |
| --- | --- |
| Value | Meaning |
| 1 | Single-sided printing |
| 2 | Duplex printing using a horizontal binding |
| 3 | Duplex printing using a vertical binding |

The **vertical binding** is the more commonly used duplex setting. With this setting, if you turn the page to the left, the text is the same way up on the other side of the page. In other words, if you bind the pages after printing, you would have a vertical binding on the left-hand side. You would use this setting for booklet printing.

**Horizontal binding** is for use when you want the binding horizontally on the top edge of the page. This is sometimes called tablet style.

The following diagram illustrates the two kinds of duplex printing.



The following code will set the printer to duplex and print a document. This can be particularly useful in Word 2002 for a document set up as a booklet using the **Bookfold** option in **File/Page Setup**. The pages can then be stapled in the middle and folded to make the booklet.

Sub PrintDuplexBooklet()

 Dim iDuplex As Long

 iDuplex = GetDuplex 'save the current setting

 SetDuplex 3 'set for vertical binding

 ActiveDocument.PrintOut Background:=False

 SetDuplex iDuplex 'restore the original setting

End Sub

### Setting the Print Quality

The possible values for the Print quality are show in the table below.

| Value | Meaning |
| --- | --- |
| -1 | Draft resolution |
| -2 | Low resolution |
| -3 | Medium resolution |
| -4 | High resolution |
| Any positive value | The printer resolution in dots per inch (dpi) |

The meaning of **draft**, **low**, **medium** and **high** resolution varies from printer to printer. In some cases it is simply an indication of varying resolution. In others, such as inkjet printers, lower-quality printing indicates that less ink is being used to print the page.

Some printers will return a negative value for the property, others will return a positive number giving the resolution, as measured in dots per inch.

In some cases the printer will accept being set with either positive or negative values of the property, others will ignore any requests they don't understand. The only way to know is to test with the specific printer you want to control.

The following code will print any document of more than 10 pages in draft mode

 Public Sub PrintLongDocsDraft()

 Dim iQuality As Long

 ActiveDocument.Repaginate

 If ActiveDocument.Range.Information(wdNumberOfPagesInDocument) > 10 Then

 iQuality = GetPrintQuality 'save the current setting

 SetPrintQuality -1

 ActiveDocument.PrintOut Background:=False

 SetPrintQuality iQuality 'restore the original setting

 End If

End Sub

### Main Code for the Article

The following code should be pasted into a separate module. It contains the GetColorMode, SetColorMode, GetDuplex, SetDuplex, GetPrintQuality and SetPrintQuality routines that are used in the code samples above.

**Important Note!** Same warning as last month. Unless you are confident you know what you are doing, don't alter this code, just use it. Bugs in Windows API code don't just stop a macro, they can bring down Word or even Windows. If you're really keen to know how the code does what it does, I've put in plenty of comments so you can look through it.

Option Explicit

Private Type PRINTER\_DEFAULTS

 pDatatype As Long

 pDevmode As Long

 DesiredAccess As Long

End Type

Private Type PRINTER\_INFO\_2

 pServerName As Long

 pPrinterName As Long

 pShareName As Long

 pPortName As Long

 pDriverName As Long

 pComment As Long

 pLocation As Long

 pDevmode As Long ' Pointer to DEVMODE

 pSepFile As Long

 pPrintProcessor As Long

 pDatatype As Long

 pParameters As Long

 pSecurityDescriptor As Long ' Pointer to SECURITY\_DESCRIPTOR

 Attributes As Long

 Priority As Long

 DefaultPriority As Long

 StartTime As Long

 UntilTime As Long

 Status As Long

 cJobs As Long

 AveragePPM As Long

End Type

Private Type DEVMODE

 dmDeviceName As String \* 32

 dmSpecVersion As Integer

 dmDriverVersion As Integer

 dmSize As Integer

 dmDriverExtra As Integer

 dmFields As Long

 dmOrientation As Integer

 dmPaperSize As Integer

 dmPaperLength As Integer

 dmPaperWidth As Integer

 dmScale As Integer

 dmCopies As Integer

 dmDefaultSource As Integer

 dmPrintQuality As Integer

 dmColor As Integer

 dmDuplex As Integer

 dmYResolution As Integer

 dmTTOption As Integer

 dmCollate As Integer

 dmFormName As String \* 32

 dmUnusedPadding As Integer

 dmBitsPerPel As Integer

 dmPelsWidth As Long

 dmPelsHeight As Long

 dmDisplayFlags As Long

 dmDisplayFrequency As Long

 dmICMMethod As Long

 dmICMIntent As Long

 dmMediaType As Long

 dmDitherType As Long

 dmReserved1 As Long

 dmReserved2 As Long

End Type

Private Const DM\_ORIENTATION = &H1

Private Const DM\_PAPERSIZE = &H2

Private Const DM\_PAPERLENGTH = &H4

Private Const DM\_PAPERWIDTH = &H8

Private Const DM\_DEFAULTSOURCE = &H200

Private Const DM\_PRINTQUALITY = &H400

Private Const DM\_COLOR = &H800

Private Const DM\_DUPLEX = &H1000

Private Const DM\_IN\_BUFFER = 8

Private Const DM\_OUT\_BUFFER = 2

Private Const PRINTER\_ACCESS\_USE = &H8

Private Const STANDARD\_RIGHTS\_REQUIRED = &HF0000

Private Const PRINTER\_NORMAL\_ACCESS = (STANDARD\_RIGHTS\_REQUIRED Or \_

 PRINTER\_ACCESS\_USE)

Private Const PRINTER\_ENUM\_CONNECTIONS = &H4

Private Const PRINTER\_ENUM\_LOCAL = &H2

Private Declare Function ClosePrinter Lib "winspool.drv" \_

 (ByVal hPrinter As Long) As Long

Private Declare Function DocumentProperties Lib "winspool.drv" \_

 Alias "DocumentPropertiesA" (ByVal hwnd As Long, \_

 ByVal hPrinter As Long, ByVal pDeviceName As String, \_

 ByVal pDevModeOutput As Long, ByVal pDevModeInput As Long, \_

 ByVal fMode As Long) As Long

Private Declare Function GetPrinter Lib "winspool.drv" Alias \_

 "GetPrinterA" (ByVal hPrinter As Long, ByVal Level As Long, \_

 pPrinter As Byte, ByVal cbBuf As Long, pcbNeeded As Long) As Long

Private Declare Function OpenPrinter Lib "winspool.drv" Alias \_

 "OpenPrinterA" (ByVal pPrinterName As String, phPrinter As Long, \_

 pDefault As PRINTER\_DEFAULTS) As Long

Private Declare Function SetPrinter Lib "winspool.drv" Alias \_

 "SetPrinterA" (ByVal hPrinter As Long, ByVal Level As Long, \_

 pPrinter As Byte, ByVal Command As Long) As Long

Private Declare Function EnumPrinters Lib "winspool.drv" \_

 Alias "EnumPrintersA" \_

 (ByVal flags As Long, ByVal name As String, ByVal Level As Long, \_

 pPrinterEnum As Long, ByVal cdBuf As Long, pcbNeeded As Long, \_

 pcReturned As Long) As Long

Private Declare Function PtrToStr Lib "kernel32" Alias "lstrcpyA" \_

 (ByVal RetVal As String, ByVal Ptr As Long) As Long

Private Declare Function StrLen Lib "kernel32" Alias "lstrlenA" \_

 (ByVal Ptr As Long) As Long

Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" \_

 (pDest As Any, pSource As Any, ByVal cbLength As Long)

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Private Declare Function DeviceCapabilities Lib "winspool.drv" \_

 Alias "DeviceCapabilitiesA" (ByVal lpDeviceName As String, \_

 ByVal lpPort As String, ByVal iIndex As Long, lpOutput As Any, \_

 ByVal dev As Long) As Long

Public Sub SetColorMode(iColorMode As Long)

 SetPrinterProperty DM\_COLOR, iColorMode

End Sub

Public Function GetColorMode() As Long

 GetColorMode = GetPrinterProperty(DM\_COLOR)

End Function

Public Sub SetDuplex(iDuplex As Long)

 SetPrinterProperty DM\_DUPLEX, iDuplex

End Sub

Public Function GetDuplex() As Long

 GetDuplex = GetPrinterProperty(DM\_DUPLEX)

End Function

Public Sub SetPrintQuality(iQuality As Long)

 SetPrinterProperty DM\_PRINTQUALITY, iQuality

End Sub

Public Function GetPrintQuality() As Long

 GetPrintQuality = GetPrinterProperty(DM\_PRINTQUALITY)

End Function

Private Function SetPrinterProperty(ByVal iPropertyType As Long, \_

 ByVal iPropertyValue As Long) As Boolean

 'Code adapted from Microsoft KB article Q230743

 Dim hPrinter As Long 'handle for the current printer

 Dim pd As PRINTER\_DEFAULTS

 Dim pinfo As PRINTER\_INFO\_2

 Dim dm As DEVMODE

 Dim sPrinterName As String

 Dim yDevModeData() As Byte 'Byte array to hold contents

 'of DEVMODE structure

 Dim yPInfoMemory() As Byte 'Byte array to hold contents

 'of PRINTER\_INFO\_2 structure

 Dim iBytesNeeded As Long

 Dim iRet As Long

 Dim iJunk As Long

 Dim iCount As Long

 On Error GoTo cleanup

 'Get the name of the current printer

 sPrinterName = Trim$(Left$(ActivePrinter, \_

 InStr(ActivePrinter, " on ")))

 pd.DesiredAccess = PRINTER\_NORMAL\_ACCESS

 iRet = OpenPrinter(sPrinterName, hPrinter, pd)

 If (iRet = 0) Or (hPrinter = 0) Then

 'Can't access current printer. Bail out doing nothing

 Exit Function

 End If

 'Get the size of the DEVMODE structure to be loaded

 iRet = DocumentProperties(0, hPrinter, sPrinterName, 0, 0, 0)

 If (iRet < 0) Then

 'Can't access printer properties.

 GoTo cleanup

 End If

 'Make sure the byte array is large enough

 'Some printer drivers lie about the size of the DEVMODE structure they

 'return, so an extra 100 bytes is provided just in case!

 ReDim yDevModeData(0 To iRet + 100) As Byte

 'Load the byte array

 iRet = DocumentProperties(0, hPrinter, sPrinterName, \_

 VarPtr(yDevModeData(0)), 0, DM\_OUT\_BUFFER)

 If (iRet < 0) Then

 GoTo cleanup

 End If

 'Copy the byte array into a structure so it can be manipulated

 Call CopyMemory(dm, yDevModeData(0), Len(dm))

 If dm.dmFields And iPropertyType = 0 Then

 'Wanted property not available. Bail out.

 GoTo cleanup

 End If

 'Set the property to the appropriate value

 Select Case iPropertyType

 Case DM\_ORIENTATION

 dm.dmOrientation = iPropertyValue

 Case DM\_PAPERSIZE

 dm.dmPaperSize = iPropertyValue

 Case DM\_PAPERLENGTH

 dm.dmPaperLength = iPropertyValue

 Case DM\_PAPERWIDTH

 dm.dmPaperWidth = iPropertyValue

 Case DM\_DEFAULTSOURCE

 dm.dmDefaultSource = iPropertyValue

 Case DM\_PRINTQUALITY

 dm.dmPrintQuality = iPropertyValue

 Case DM\_COLOR

 dm.dmColor = iPropertyValue

 Case DM\_DUPLEX

 dm.dmDuplex = iPropertyValue

 End Select

 'Load the structure back into the byte array

 Call CopyMemory(yDevModeData(0), dm, Len(dm))

 'Tell the printer about the new property

 iRet = DocumentProperties(0, hPrinter, sPrinterName, \_

 VarPtr(yDevModeData(0)), VarPtr(yDevModeData(0)), \_

 DM\_IN\_BUFFER Or DM\_OUT\_BUFFER)

 If (iRet < 0) Then

 GoTo cleanup

 End If

 'The code above \*ought\* to be sufficient to set the property

 'correctly. Unfortunately some brands of Postscript printer don't

 'seem to respond correctly. The following code is used to make

 'sure they also respond correctly.

 Call GetPrinter(hPrinter, 2, 0, 0, iBytesNeeded)

 If (iBytesNeeded = 0) Then

 'Couldn't access shared printer settings

 GoTo cleanup

 End If

 'Set byte array large enough for PRINTER\_INFO\_2 structure

 ReDim yPInfoMemory(0 To iBytesNeeded + 100) As Byte

 'Load the PRINTER\_INFO\_2 structure into byte array

 iRet = GetPrinter(hPrinter, 2, yPInfoMemory(0), iBytesNeeded, iJunk)

 If (iRet = 0) Then

 'Couldn't access shared printer settings

 GoTo cleanup

 End If

 'Copy byte array into the structured type

 Call CopyMemory(pinfo, yPInfoMemory(0), Len(pinfo))

 'Load the DEVMODE structure with byte array containing

 'the new property value

 pinfo.pDevmode = VarPtr(yDevModeData(0))

 'Set security descriptor to null

 pinfo.pSecurityDescriptor = 0

 'Copy the PRINTER\_INFO\_2 structure back into byte array

 Call CopyMemory(yPInfoMemory(0), pinfo, Len(pinfo))

 'Send the new details to the printer

 iRet = SetPrinter(hPrinter, 2, yPInfoMemory(0), 0)

 'Indicate whether it all worked or not!

 SetPrinterProperty = CBool(iRet)

cleanup:

 'Release the printer handle

 If (hPrinter <> 0) Then Call ClosePrinter(hPrinter)

 'Flush the message queue. If you don't do this,

 'you can get page fault errors when you try to

 'print a document immediately after setting a printer property.

 For iCount = 1 To 20

 DoEvents

 Next iCount

 End Function

Private Function GetPrinterProperty(ByVal iPropertyType As Long) As Long

 'Code adapted from Microsoft KB article Q230743

 Dim hPrinter As Long

 Dim pd As PRINTER\_DEFAULTS

 Dim dm As DEVMODE

 Dim sPrinterName As String

 Dim yDevModeData() As Byte

 Dim iRet As Long

 On Error GoTo cleanup

 'Get the name of the current printer

 sPrinterName = Trim$(Left$(ActivePrinter, \_

 InStr(ActivePrinter, " on ")))

 pd.DesiredAccess = PRINTER\_NORMAL\_ACCESS

 'Get the printer handle

 iRet = OpenPrinter(sPrinterName, hPrinter, pd)

 If (iRet = 0) Or (hPrinter = 0) Then

 'Couldn't access the printer

 Exit Function

 End If

 'Find out how many bytes needed for the printer properties

 iRet = DocumentProperties(0, hPrinter, sPrinterName, 0, 0, 0)

 If (iRet < 0) Then

 'Couldn't access printer properties

 GoTo cleanup

 End If

 'Make sure the byte array is large enough, including the

 '100 bytes extra in case the printer driver is lying.

 ReDim yDevModeData(0 To iRet + 100) As Byte

 'Load the printer properties into the byte array

 iRet = DocumentProperties(0, hPrinter, sPrinterName, \_

 VarPtr(yDevModeData(0)), 0, DM\_OUT\_BUFFER)

 If (iRet < 0) Then

 'Couldn't access printer properties

 GoTo cleanup

 End If

 'Copy the byte array to the DEVMODE structure

 Call CopyMemory(dm, yDevModeData(0), Len(dm))

 If Not dm.dmFields And iPropertyType = 0 Then

 'Requested property not available on this printer.

 GoTo cleanup

 End If

 'Get the value of the requested property

 Select Case iPropertyType

 Case DM\_ORIENTATION

 GetPrinterProperty = dm.dmOrientation

 Case DM\_PAPERSIZE

 GetPrinterProperty = dm.dmPaperSize

 Case DM\_PAPERLENGTH

 GetPrinterProperty = dm.dmPaperLength

 Case DM\_PAPERWIDTH

 GetPrinterProperty = dm.dmPaperWidth

 Case DM\_DEFAULTSOURCE

 GetPrinterProperty = dm.dmDefaultSource

 Case DM\_PRINTQUALITY

 GetPrinterProperty = dm.dmPrintQuality

 Case DM\_COLOR

 GetPrinterProperty = dm.dmColor

 Case DM\_DUPLEX

 GetPrinterProperty = dm.dmDuplex

 End Select

cleanup:

 'Release the printer handle

 If (hPrinter <> 0) Then Call ClosePrinter(hPrinter)

End Function

## Part 3: Dealing with Different Paper Sizes

### Tidying Up Loose Ends

Before I get on to dealing with paper sizes, I just want to tidy up an issue that arose in correspondence following last month's article.

Andrew Hosking emailed me saying he had tried out the code to control the duplex setting and found that it just wouldn't work for him. We exchanged a few emails before we discovered the cause. It was a limitation that I knew about, but had neglected to mention in the article.

The code will work perfectly happily with a networked printer, but only if the printer driver is installed on the local machine. If printing is relying on the printer driver on the printer server, then the calls to the Windows API do not work. The workaround is to install a copy of the appropriate printer driver on the local machine—pointing to the network printer.

If you are working in a corporate networked environment, you will probably need the assistance and agreement of the system administrator for this (unless you are the system administrator!)

### Paper Sizes

So now to the main business. In Part 1 of this series, I dealt with selecting paper trays, and pointed out that there are significant inconsistencies between the way Word deals with them and the way Windows does. If you thought that was a mess, you ain't seen nothin' yet! The handling of paper trays is a model of logic and consistency compared to how Word deals with paper sizes.

### Why Do We Need to Know About Paper Sizes in Code?

If you are in the US and have received a Word document from Europe, the document is probably formatted A4, while your printer is set up for US Letter.

The larger types of laser printer are sometimes too clever for their own good. On receiving a request to print an A4 document when they are set up for Letter, they usually sit there flashing an "I need help" message on their LCD display. In fact, all that is usually needed is to press the "Continue" button on the printer and it will happily get on with the task. Because US Letter is shorter and wider than A4, you might have a large right margin and lose a bit of the page footer text on the printout.

One way to deal with this is to reformat documents so that they match the paper size you actually have in the printer. That is the ideal, but it isn't always practicable. The way in which Word documents are laid out varies a great deal from person to person, and simply changing the paper size in the Page Setup dialog can result in the most incredible mess, especially if manual page breaks have been used to make sure that related information is all grouped on the same page.

An alterative is to fool the printer into thinking that it has A4 paper when it is actually loaded with Letter. Then the print job will go through uninterrupted.

You can do this manually by going to the Printer Properties dialog and changing the paper size there. (Remember to change it back again afterwards!) If you have a program that is printing a large batch of documents, you need to be able to do the same thing in code.

In the Part 1 article on paper trays, there was no need to have Windows API code to get and set the default paper tray in the printer, because Word already has methods in the Options object to do that for you. Unfortunately, the same cannot be said for getting and setting the current paper size. Therefore, we need some routines that will communicate with the printer driver to do this. The code below allows you to get and set the paper width and height. The values returned are in points (1/72") for compatibility with Word's PageSetup object, even though the printer driver itself returns the height and width in units of a tenth of a millimeter. Paste the following code into the end of the same module that you created for the Part 2 article. (A complete listing of all the code for all three parts of the article is available from the TechTrax Library, ready to be imported into your project.)

Public Function GetPaperHeight() As Single

 'Windows API returns page height in 1/10ths of a millimeter.

 'Value converted to points for compatibility

 'with the PageHeight property

 GetPaperHeight = CSng(GetPrinterProperty(DM\_PAPERLENGTH)) \* 72 / 254

End Function

Public Sub SetPaperHeight(sngHeight As Single)

 Dim lHeight As Long

 'Windows API sets page height in 1/10ths of a millimeter.

 'Value converted from points for compatibility

 'with the PageHeight property

 lHeight = CLng(sngHeight \* 254 / 72)

 SetPrinterProperty DM\_PAPERLENGTH, lHeight

End Sub

Public Function GetPaperWidth() As Single

 'Windows API returns paper width in 1/10ths of a millimeter.

 'Value converted to points for compatibility

 'with the PageWidth property

 GetPaperWidth = CSng(GetPrinterProperty(DM\_PAPERWIDTH)) \* 72 / 254

End Function

Public Sub SetPaperWidth(sngWidth As Single)

 Dim lWidth As Long

 'Windows API sets page height in 1/10ths of a millimeter.

 'Value converted from points for compatibility

 'with the PageWidth property

 lWidth = CLng(sngWidth \* 254 / 72)

 SetPrinterProperty DM\_PAPERWIDTH, lWidth

End Sub

Public Function GetPaperSize() As Long

 GetPaperSize = GetPrinterProperty(DM\_PAPERSIZE)

End Function

Public Sub SetPaperSize(iPaperSize As Long)

 SetPrinterProperty DM\_PAPERSIZE, iPaperSize

End Sub

The code above makes use of the SetPrinterProperty and GetPrinterProperty routines that I listed in last month's Part 2 article.

### Getting a List of Available Paper Sizes

The GetPaperSize and SetPaperSize routines listed above return an ID number which represents a standard paper size. Of course, you need to know what each code means.

It is possible to get a list of available paper size numbers and names using code very similar to the code I described in Part 1 for getting paper trays. The code is in fact so similar, that I am not going to list it here because only a couple of lines would be different in each routine compared to the Part 1 code. Instead, the library file that goes with this article contains a complete set of all the routines for all three parts of the article, with code duplication eliminated by calling common routines.

Taking the same two printers that I mentioned in Part 1, here is a list of the names and ID numbers for the paper sizes that they each support.

|  |  |
| --- | --- |
| ID | Name |
| 1 | Letter |
| 7 | Executive |
| 9 | A4 |
| 11 | A5 |
| 126 | # 10 Envelope |
| 127 | Monarch Envelope |
| 128 | # 6 3/4 Envelope |
| 129 | DL Envelope |
| 130 | C5 Envelope |
| 131 | Choukei 3 Envelope |
| 132 | Choukei 4 Envelope |

 HP LaserJet 4/4Si MX PS Tektronix Phaser 850DP

| ID | Name |
| --- | --- |
| 1 | Letter |
| 2 | Letter Small |
| 5 | Legal |
| 6 | Statement |
| 7 | Executive |
| 9 | A4 |
| 10 | A4 Small |
| 11 | A5 |
| 13 | B5 (JIS) |
| 14 | Folio |
| 15 | Quarto |
| 18 | Note |
| 19 | Envelope # 9 |
| 20 | Envelope # 10 |
| 21 | Envelope # 11 |
| 22 | Envelope # 12 |
| 23 | Envelope # 14 |
| 27 | Envelope DL |
| 28 | Envelope C5 |
| 31 | Envelope C6 |
| 32 | Envelope C65 |
| 34 | Envelope B5 |
| 35 | Envelope B6 |
| 36 | Envelope |
| 37 | Envelope Monarch |
| 38 | 6 3/4 Envelope |
| 40 | German Std Fanfold |
| 41 | German Legal Fanfold |
| 43 | Japanese Postcard |
| 48 | Reserved48 |
| 49 | Reserved49 |
| 54 | Letter Transverse |
| 55 | A4 Transverse |
| 59 | Letter Plus |
| 60 | A4 Plus |
| 61 | A5 Transverse |
| 62 | B5 (JIS) Transverse |
| 64 | A5 Extra |
| 65 | B5 (ISO) Extra |
| 69 | Japanese Double Postcard |
| 70 | A6 |
| 73 | Japanese Envelope Chou # 3 |
| 74 | Japanese Envelope Chou # 4 |
| 78 | A5 Rotated |
| 81 | Japanese Postcard Rotated |
| 82 | Double Japan Postcard Rotated |
| 83 | A6 Rotated |
| 87 | Japan Envelope Chou # 4 Rotated |
| 88 | B6 (JIS) |
| 89 | B6 (JIS) Rotated |
| 91 | Japan Envelope You # 4 |
| 93 | PRC 16K |
| 94 | PRC 32K |
| 95 | PRC 32K(Big) |
| 96 | PRC Envelope # 1 |
| 97 | PRC Envelope # 2 |
| 98 | PRC Envelope # 3 |
| 99 | PRC Envelope # 4 |
| 100 | PRC Envelope # 5 |
| 101 | PRC Envelope # 6 |
| 102 | PRC Envelope # 7 |
| 103 | PRC Envelope # 8 |
| 107 | PRC 32K Rotated |
| 108 | PRC 32K(Big) Rotated |
| 109 | PRC Envelope # 1 Rotated |
| 110 | PRC Envelope # 2 Rotated |
| 111 | PRC Envelope # 3 Rotated |
| 112 | PRC Envelope # 4 Rotated |
| 119 | Letter 8 1/2 x 11 in |
| 120 | Legal 8 1/2 x 14 in |
| 121 | A4 210 x 297 mm |
| 122 | Executive 7 1/4 x 10 1/2 in |
| 123 | Env Comm10 4 1/8 x 9 1/2 in |
| 124 | Env Monarch 3 7/8 x 7 1/2 in |
| 125 | Env DL 110 x 220 mm |

Looking at these codes and similar lists for other printers, there is some good news, some not-quite-so-good news and some really, really bad news.

First the good news. The lower-numbered codes (up to 41) and their associated names are common between all the printers. Therefore, Letter paper uses code 1 and has the name "Letter" for any printer that supports it.

The not-quite-so-good news is that above code 41, there are lots of custom paper sizes, and the codes, names and paper sizes vary between printers. Some printers support custom sizes, and some don't. For those that support custom sizes, there is no way of knowing from the name and number what that size actually is. To find out, it would be necessary to set that size using the SetPaperSize routine, and then find out the actual page height and width using the GetPaperHeight and GetPaperWidth functions.

The really bad news is that the standard code numbers up to 41 don't align with the values of the wdPaperSize constants used to set the PaperSize property of Word's PageSetup object. Arrgghh!

The following table shows how Word's standard constants and the Windows API code values map onto each other.

| Paper Size | Win API Value | Word Constant Name | Constant Value |
| --- | --- | --- | --- |
| Letter | 1 | wdPaperLetter | 2 |
| Letter Small | 2 | wdPaperLetterSmall | 3 |
| Tabloid | 3 | wdPaperTabloid | 23 |
| Ledger | 4 | wdPaperLedger | 19 |
| Legal | 5 | wdPaperLegal | 4 |
| Statement | 6 | wdPaperStatement | 22 |
| Executive | 7 | wdPaperExecutive | 5 |
| A3 | 8 | wdPaperA3 | 6 |
| A4 | 9 | wdPaperA4 | 7 |
| A4 Small | 10 | wdPaperA4Small | 8 |
| A5 | 11 | wdPaperA5 | 9 |
| B4 | 12 | wdPaperB4 | 10 |
| B5 | 13 | wdPaperB5 | 11 |
| Folio | 14 | wdPaperFolio | 18 |
| Quarto | 15 | wdPaperQuarto | 21 |
| 10 x 14 in | 16 | wdPaper10x14 | 0 |
| 11 x 17 in | 17 | wdPaper11x17 | 1 |
| Note | 18 | wdPaperNote | 20 |
| Envelope # 9 | 19 | wdPaperEnvelope9 | 24 |
| Envelope # 10 | 20 | wdPaperEnvelope10 | 25 |
| Envelope # 11 | 21 | wdPaperEnvelope11 | 26 |
| Envelope # 12 | 22 | wdPaperEnvelope12 | 27 |
| Envelope # 14 | 23 | wdPaperEnvelope14 | 28 |
| C size sheet | 24 | wdPaperCSheet | 12 |
| D size sheet | 25 | wdPaperDSheet | 13 |
| E size sheet | 26 | wdPaperESheet | 14 |
| Envelope DL | 27 | wdPaperEnvelopeDL | 37 |
| Envelope C5 | 28 | wdPaperEnvelopeC5 | 34 |
| Envelope C3 | 29 | wdPaperEnvelopeC3 | 32 |
| Envelope C4 | 30 | wdPaperEnvelopeC4 | 33 |
| Envelope C6 | 31 | wdPaperEnvelopeC6 | 35 |
| Envelope C65 | 32 | wdPaperEnvelopeC65 | 36 |
| Envelope B4 | 33 | wdPaperEnvelopeB4 | 29 |
| Envelope B5 | 34 | wdPaperEnvelopeB5 | 30 |
| Envelope B6 | 35 | wdPaperEnvelopeB6 | 31 |
| Envelope | 36 | wdPaperEnvelopeItaly | 38 |
| Envelope Monarch | 37 | wdPaperEnvelopeMonarch | 39 |
| 6 3/4 Envelope | 38 | wdPaperEnvelopePersonal | 40 |
| U.S. Standard Fanfold | 39 | wdPaperFanfoldUS | 17 |
| German Standard Fanfold | 40 | wdPaperFanfoldStdGerman | 16 |
| German Legal Fanfold | 41 | wdPaperFanfoldLegalGerman | 15 |
| User-defined | 256 | wdPaperCustom | 41 |

Quite frankly, this is a horrid mess, and I cannot imagine why Microsoft chose to implement a set of paper size codes in Word that is different from those already standardised for use in Windows.

Still, we have to make the best of it we can. You can set the paper size for the printer by setting the PaperSize property to one of the supported codes. Generally, it is a good idea to make sure that the PaperSize of the printer is equivalent to the PaperSize property of the PageSetup object. For instance, if the ActiveDocument.PageSetup.PaperSize is wdPaperLetter, the you would have to ensure that you use the SetPaperSize subroutine to set the printer's current paper size to 1, and quietly forget about the fact that the value for wdPaperLetter is actually 2! A routine that will check whether a document is A4 size and set the page size accordingly is given below.

Public Sub CheckA4BeforePrinting()

Dim iCurrentPaperSize As Long

If ActiveDocument.PageSetup.PaperSize = wdPaperA4 Then

 'Save current paper size so it can be restored afterwards

 iCurrentPageSize = GetPaperSize

 'Set printer to A4 (value 9 in Win API codes) and print

 SetPaperSize 9

 ActiveDocument.PrintOut Background:=False

 'Restore the original paper size

 SetPaperSize iCurrentPaperSize

Else

 'No need for special action, just print

 ActiveDocument.PrintOut Background:=False

End If

End Sub

## Part 4: Getting printer driver details

### More Information Needed?

In the feedback section of this month's issue, Carol Baxter asked "It would be really useful to know how to capture the printer driver into a variable in word. I work for a large firm and all our network printers are called lp(then a number) so the active printer command will not tell me the name of the printer eg HP Laserjet 4. We have macros for printing."

Carol—this article is for you! (I hope that others will find it useful as well.)

### Printer Information Available

The code provided with this article will allow you to get the following information about a printer.

**ServerName**—the name of the printer server it is attached to (if any)

**ShareName**—if the printer is shared, the share name as broadcast to the network

**PortName**—the name of the port the printer is connected to

**DriverName**—the name of the printer driver

**Comment**—any comments that are listed for the printer in the Printer Properties dialog

**Location**—the location as given in the Printer Properties dialog

**SepFile**—the name of the file that defines the separator page for the printer (is an empty string if no separator file is defined)

**PrintProcessor**—the name of the print processor for the printer

**Datatype**—the format in which the printer files are spooled.

**Parameters**—any parameters of the print processor command

**Status**—the current status of the printer, e.g. "Ready", "Paused" etc.

**Jobs**—the number of print jobs currently in the queue for the printer.

The DriverName is what Carol is after, but the other information might also be useful!

### How to Use the Code

The code listed at the end of the article has a single routine, called GetPrinterDetails. It returns a user-defined type including all of the parameters I have described above. To find out the driver name of the current printer is as simple as this:

MsgBox "Driver name is " & \_

 GetPrinterDetails.DriverName

If you want to get several parameters in one go (to reduce the time spent making calls to the routine) and then use them later in your code, you can do something like this:

Dim pInfo as PrinterInfo

pInfo = GetPrinterDetails

MsgBox "Port name is " & pInfo.PortName

MsgBox "Printer status is " & pInfo.Status

Also, if you want to get the printer details for a printer other than the current printer, then you can do so, by including the printer name, like this:

MsgBox "HP DeskJet 540 status " & \_

 GetPrinterDetails("HP DeskJet 540").Status

By the way, if you want to get a full list of the printers available on your system, this article by Astrid Zeelenberg tells you how.

**Getting Names of Available Printers**

<http://www.mvps.org/word/FAQs/MacrosVBA/AvailablePrinters.htm>

That article includes a routine that returns an array of the available printers. Any one of the items in that array can be used by the GetPrinterDetails routine.

### Why Use the Code?

There are a number of possible reasons

* Carol's reason - you need to know the current printer type so that you can decide which printer tray to use for printing.
* You have several printers available, and want to check their status before printing - no point in printing to a printer that is paused or offline.
* You want to distribute print jobs among a number of printers, and want to find out which one is least heavily loaded.

An example of the second reason might work like this...

Suppose you have three printers (called Printer 1, Printer 2 and Printer 3) that you could use for printing the current job, and you want to be sure that you print to a printer that is ready, or will be reasonably soon. The following code could be used.

Dim PrinterList as Variant

Dim i as Long

Dim pInfo as PrinterInfo

PrinterList = Array("Printer 1", "Printer 2", "Printer 3")

For i = LBound(PrinterList) to UBound(PrinterList)

 pInfo = GetPrinterInfo(PrinterList(i))

 Select Case pInfo.Status

 Case "Ready", "Printing", "Processing Job", "Power Save Mode"

 ActivePrinter = PrinterList(i)

 ActiveDocument.PrintOut

 MsgBox "Job printed to " & PrinterList(i)

 Exit For

 Case Else

 End Select

Next i

If i > UBound(PrinterList) Then

 MsgBox "No printers are available at present"

End If

This code does checks each printer in turn, and if the status indicates that the printer is OK, it prints the job there and tells the user where to find the printout. Otherwise, it goes on to the next printer on the list. If no printers are available, it tells the user so.

### Setting the ActivePrinter in Excel

I mainly do Word VBA, but I like to keep aware of uses for my code in the other Office applications. In this case, there is a particular use in Excel.

In Word, you can use just the printer name to set the ActivePrinter, even though the name doesn't include the port name. So something like this in Word will successfully change the printer.

ActivePrinter = "HP LaserJet 5Si"

If you try that code in Excel, you will get an error, because Excel must have the port name in the string that defines the printer. (Why this is needed by Excel and not Word is one of the mysteries of life!) This can be a bit of a nuisance if you have used Astrid's article (see above) to get a list of the available printers, as you can't then use it in Excel to set the printer. But with this routine, you can! Suppose the name of the printer you want is loaded into the variable NewPrinter. To change the printer in Excel, the following line of code will work fine.

ActivePrinter = NewPrinter & " on " & \_

 GetPrinterInfo(NewPrinter).PortName

### Main Code for the Article

The following code should be pasted into a separate module.

**Important Note!** Same warning as usual. Unless you are confident you know what you are doing, don't alter this code, just use it. Bugs in Windows API code don't just stop a macro, they can bring down Word or even Windows.

Option Explicit

' Win32 API declares

Private Declare Function OpenPrinter Lib "winspool.drv" \_

 Alias "OpenPrinterA" (ByVal pPrinterName As String, \_

 phPrn As Long, pDefault As Any) As Long

Private Declare Function ClosePrinter Lib "winspool.drv" \_

 (ByVal hPrn As Long) As Long

Private Declare Function GetPrinter Lib "winspool.drv" \_

 Alias "GetPrinterA" (ByVal hPrinter As Long, \_

 ByVal Level As Long, pPrinter As Any, \_

 ByVal cbBuf As Long, pcbNeeded As Long) As Long

Private Declare Function SetPrinter Lib "winspool.drv" \_

 Alias "SetPrinterA" (ByVal hPrinter As Long, \_

 ByVal Level As Long, pPrinter As Any, \_

 ByVal Command As Long) As Long

Private Declare Sub CopyMemory Lib "kernel32" \_

 Alias "RtlMoveMemory" (Destination As Any, \_

 Source As Any, ByVal Length As Long)

Private Declare Function lstrlenA Lib "kernel32" \_

 (ByVal lpString As Long) As Long

Private Declare Function FormatMessage Lib "kernel32" \_

 Alias "FormatMessageA" (ByVal dwFlags As Long, \_

 lpSource As Any, ByVal dwMessageId As Long, \_

 ByVal dwLanguageId As Long, ByVal lpBuffer As String, \_

 ByVal nSize As Long, Arguments As Long) As Long

' The data area passed to a system call is too small.

Private Const ERROR\_INSUFFICIENT\_BUFFER As Long = 122

' Printer status flags used with PRINTER\_INFORMATION\_2

Private Const PRINTER\_STATUS\_READY As Long = &H0

Private Const PRINTER\_STATUS\_PAUSED As Long = &H1

Private Const PRINTER\_STATUS\_ERROR As Long = &H2

Private Const PRINTER\_STATUS\_PENDING\_DELETION As Long = &H4

Private Const PRINTER\_STATUS\_PAPER\_JAM As Long = &H8

Private Const PRINTER\_STATUS\_PAPER\_OUT As Long = &H10

Private Const PRINTER\_STATUS\_MANUAL\_FEED As Long = &H20

Private Const PRINTER\_STATUS\_PAPER\_PROBLEM As Long = &H40

Private Const PRINTER\_STATUS\_OFFLINE As Long = &H80

Private Const PRINTER\_STATUS\_IO\_ACTIVE As Long = &H100

Private Const PRINTER\_STATUS\_BUSY As Long = &H200

Private Const PRINTER\_STATUS\_PRINTING As Long = &H400

Private Const PRINTER\_STATUS\_OUTPUT\_BIN\_FULL As Long = &H800

Private Const PRINTER\_STATUS\_NOT\_AVAILABLE As Long = &H1000

Private Const PRINTER\_STATUS\_WAITING As Long = &H2000

Private Const PRINTER\_STATUS\_PROCESSING As Long = &H4000

Private Const PRINTER\_STATUS\_INITIALIZING As Long = &H8000

Private Const PRINTER\_STATUS\_WARMING\_UP As Long = &H10000

Private Const PRINTER\_STATUS\_TONER\_LOW As Long = &H20000

Private Const PRINTER\_STATUS\_NO\_TONER As Long = &H40000

Private Const PRINTER\_STATUS\_PAGE\_PUNT As Long = &H80000

Private Const PRINTER\_STATUS\_USER\_INTERVENTION As Long = &H100000

Private Const PRINTER\_STATUS\_OUT\_OF\_MEMORY As Long = &H200000

Private Const PRINTER\_STATUS\_DOOR\_OPEN As Long = &H400000

Private Const PRINTER\_STATUS\_SERVER\_UNKNOWN As Long = &H800000

Private Const PRINTER\_STATUS\_POWER\_SAVE As Long = &H1000000

' Used to retrieve last API error text.

Private Const FORMAT\_MESSAGE\_FROM\_SYSTEM As Long = &H1000

' VBA-friendly structure used to return the printer info.

Public Type PrinterInfo

 ServerName As String

 ShareName As String

 PortName As String

 DriverName As String

 Comment As String

 Location As String

 SepFile As String

 PrintProcessor As String

 Datatype As String

 Parameters As String

 Status As String

 Jobs As Long

End Type

' Structure used to obtain the data from Windows.

Private Type PRINTER\_INFO\_2

 pServerName As Long

 pPrinterName As Long

 pShareName As Long

 pPortName As Long

 pDriverName As Long

 pComment As Long

 pLocation As Long

 pDevMode As Long 'DEVMODE

 pSepFile As Long

 pPrintProcessor As Long

 pDatatype As Long

 pParameters As Long

 pSecurityDescriptor As Long 'SECURITY\_DESCRIPTOR

 Attributes As Long

 Priority As Long

 DefaultPriority As Long

 StartTime As Long

 UntilTime As Long

 Status As Long

 cJobs As Long

 AveragePPM As Long

 End Type

Public Function GetPrinterDetails(Optional ByVal PrinterName As Variant) As PrinterInfo

 Dim pi2 As PRINTER\_INFO\_2

 Dim pi2\_output As PrinterInfo

 Dim hPrn As Long

 Dim Buffer() As Byte

 Dim BytesNeeded As Long

 Dim BytesUsed As Long

 Dim slash As Long

 Dim DispName As String

 Dim PrinterErrorCode As Long

 Dim StatusCode As Long

 'Use default printer if none specified

 If IsMissing(PrinterName) Then

 PrinterName = ActivePrinter

 PrinterName = Left$(PrinterName, InStr(PrinterName, " on ") - 1)

 End If

 ' Get handle to printer.

 Call OpenPrinter(PrinterName, hPrn, ByVal 0&)

 If hPrn Then

 ' Call once to get proper buffer size.

 Call GetPrinter(hPrn, 2, ByVal 0&, 0, BytesNeeded)

 If Err.LastDllError = ERROR\_INSUFFICIENT\_BUFFER Then

 ' Size buffer and get printer data.

 ReDim Buffer(0 To BytesNeeded - 1) As Byte

 If GetPrinter(hPrn, 2, Buffer(0), BytesNeeded, BytesUsed) Then

 ' Fill local structure with data/pointers.

 Call CopyMemory(pi2, Buffer(0), Len(pi2))

 ' Transfer string data to output structure.

 pi2\_output.ServerName = PointerToStringA(pi2.pServerName)

 pi2\_output.ShareName = PointerToStringA(pi2.pShareName)

 pi2\_output.PortName = PointerToStringA(pi2.pPortName)

 pi2\_output.DriverName = PointerToStringA(pi2.pDriverName)

 pi2\_output.Comment = PointerToStringA(pi2.pComment)

 pi2\_output.Location = PointerToStringA(pi2.pLocation)

 pi2\_output.SepFile = PointerToStringA(pi2.pSepFile)

 pi2\_output.PrintProcessor = PointerToStringA(pi2.pPrintProcessor)

 pi2\_output.Datatype = PointerToStringA(pi2.pDatatype)

 pi2\_output.Parameters = PointerToStringA(pi2.pParameters)

 Call CopyMemory(StatusCode, Buffer(72), 4)

 Call CopyMemory(pi2\_output.Jobs, Buffer(76), 4)

 End If

 PrinterErrorCode = 0 'clear error value

 Else

 PrinterErrorCode = Err.LastDllError

 End If

 pi2\_output.Status = StatusText(StatusCode, PrinterErrorCode)

 Call ClosePrinter(hPrn)

 End If

 GetPrinterDetails = pi2\_output

End Function

Private Function PointerToStringA(ByVal lpStringA As Long) As String

 Dim Buffer() As Byte

 Dim nLen As Long

 If lpStringA Then

 nLen = lstrlenA(ByVal lpStringA)

 If nLen Then

 ReDim Buffer(0 To (nLen - 1)) As Byte

 CopyMemory Buffer(0), ByVal lpStringA, nLen

 PointerToStringA = StrConv(Buffer, vbUnicode)

 End If

 End If

End Function

Private Function StatusText(StatusCode As Long, ErrorCode As Long) As String

 If ErrorCode Then

 StatusText = ApiErrorText(ErrorCode)

 Else

 Select Case StatusCode

 Case PRINTER\_STATUS\_READY

 StatusText = "Ready"

 Case PRINTER\_STATUS\_PAUSED

 StatusText = "Paused"

 Case PRINTER\_STATUS\_ERROR

 StatusText = "Error"

 Case PRINTER\_STATUS\_PENDING\_DELETION

 StatusText = "Deleting..."

 Case PRINTER\_STATUS\_PAPER\_JAM

 StatusText = "Paper Jam"

 Case PRINTER\_STATUS\_PAPER\_OUT

 StatusText = "Paper Out"

 Case PRINTER\_STATUS\_MANUAL\_FEED

 StatusText = "Manual Feed Required"

 Case PRINTER\_STATUS\_PAPER\_PROBLEM

 StatusText = "Paper Problem"

 Case PRINTER\_STATUS\_OFFLINE

 StatusText = "Offline"

 Case PRINTER\_STATUS\_IO\_ACTIVE

 StatusText = "Downloading Job"

 Case PRINTER\_STATUS\_BUSY

 StatusText = "Busy"

 Case PRINTER\_STATUS\_PRINTING

 StatusText = "Printing"

 Case PRINTER\_STATUS\_OUTPUT\_BIN\_FULL

 StatusText = "Output Bill Full"

 Case PRINTER\_STATUS\_NOT\_AVAILABLE

 StatusText = "Not Available"

 Case PRINTER\_STATUS\_WAITING

 StatusText = "Waiting"

 Case PRINTER\_STATUS\_PROCESSING

 StatusText = "Processing Job"

 Case PRINTER\_STATUS\_INITIALIZING

 StatusText = "Initializing"

 Case PRINTER\_STATUS\_WARMING\_UP

 StatusText = "Warming Up"

 Case PRINTER\_STATUS\_TONER\_LOW

 StatusText = "Toner Low"

 Case PRINTER\_STATUS\_NO\_TONER

 StatusText = "Toner Out"

 Case PRINTER\_STATUS\_PAGE\_PUNT

 StatusText = "Page too Complex"

 Case PRINTER\_STATUS\_USER\_INTERVENTION

 StatusText = "User Intervention Required"

 Case PRINTER\_STATUS\_OUT\_OF\_MEMORY

 StatusText = "Out of Memory"

 Case PRINTER\_STATUS\_DOOR\_OPEN

 StatusText = "Door Open"

 Case PRINTER\_STATUS\_SERVER\_UNKNOWN

 StatusText = "Unable to connect"

 Case PRINTER\_STATUS\_POWER\_SAVE

 StatusText = "Power Save Mode"

 Case Else

 StatusText = Hex$(StatusCode)

 End Select

 End If

End Function

Private Function ApiErrorText(ByVal ErrNum As Long) As String

 Dim msg As String

 Dim nRet As Long

 msg = Space$(1024)

 nRet = FormatMessage(FORMAT\_MESSAGE\_FROM\_SYSTEM, ByVal 0&, ErrNum, 0&, msg, Len(msg), ByVal 0&)

 If nRet Then

 ApiErrorText = Left$(msg, nRet - 2) ' account for Cr/Lf

 Else

 ApiErrorText = "Error (" & ErrNum & ") not defined."

 End If

End Function

### Library Code

I have provided a module which can be imported directly into your Word VBA project which includes all the routines described in all three parts of this article. ~~Click here to download it.~~

[See file <https://jay-freedman.info/printers.zip>]

### Acknowledgements

I'm a great one for never writing Windows API code myself from scratch if I can avoid it. This month's article is no exception. The code here is adapted (with permission) from a very extensive set of VB class modules for printer information and control, written by Karl E. Peterson, VB/MVP. There's far too much code in his samples to be able to do it justice here, but you can see his original code at his site [~~www.mvps.org/vb/~~](http://www.mvps.org/vb/) <https://classicvb.net/samples/PrnInfo/> . ~~To see his printer sample code, click the Samples link on the left of the page, and then scroll down to the section PrnInfo.zip.~~

One thing I like about the samples on Karl's page is that he creates entire modules and class modules. If you have Office 2000 or later, this usually means you can import the code directly into your VBA project without modification, and the code just works. There's no development quite so rapid as being able to use somebody else's already tested and working code!

Note that you can't import forms (.frm modules) in this way because VB Forms are quite different from VBA UserForms.