2016

VBA Cheat Sheet

V1.0

JAUNNOO MOHAMMAD ZYAAD

Contents

Coding Best Practices

Force variable declarations

```
Option Explicit          '   Always include this at the top each source file
```

Error Handling

```
Public Function Foo(...) As Boolean
Const strPROC_NAME As String = "Foo"

On Error GoTo Error_handler
    ' My code goes here
    ' If everything goes on perfectly, exit the function smoothly
    Foo = True
    Exit Function

Error_handler:
    MsgBox "An error occured ...: " & Err.Description
    Foo = False
    Exit Function
End Function
```

Null values

To check if a value is null, use the IsNull(..) function.

Debug.Assert

Assertions are used in development to check your code as it runs. An Assertion is a statement that evaluates to true or false. If it evaluates to false then the code stops at that line. This is useful as it stops you close to the cause of the error.

```
Debug.Assert 1 = 2
```

The "Not Responding" problem

Reference: https://support.microsoft.com/en-us/kb/118468

When a time consuming program runs, most of the time, Excel will fall in a "Not Responding" state, although the program continues to run in the background. In such situation, we would like to have a kind of progress feedback on the screen so that we are sure the program is not stuck in an infinite loop. In such case, use the command:

```
DoEvents
```

Getting the containing folder of the tool

We need to often output files to a folder at the same level of the tool. It is better NOT to hardcode that path in the code. Instead, use the following command to get the path of the Workbook.

```
ThisWorkbook.Path & "\MyOutputFolder\" & OutputFilename & ".txt"
```

Generating random numbers

Use the function from the Worksheet object to generate random numbers.

```
WorksheetFunction.RandBetween(1, 10000)
```

Object oriented coding style

Class Description

```
'
'   Class       : Robot
'   Description : Generic class for Robot
'
Option Explicit

Private Sub class_initialize()
    '   Constructor
    Debug.Print "Robot initialized"
End Sub

Private Sub class_terminate()
    '   Destructor
    Debug.Print "Robot destroyed"
End Sub
```

Using an instantiated class

```
Option Explicit

Public Sub GO()
    Dim oRobot          As Robot

    '   Launch Robot for the simulation
    Set oRobot = New Robot

    '   Release memory
    Set oRobot = Nothing
End Sub
```

Data Structures

Static array

```
Public Sub DecArrayStatic()
   Dim arrMarks1(0 To 3) As Long     ' Create array with locations 0,1,2,3
   Dim arrMarks2(3) As Long          ' Defaults as 0 to 3 i.e. locations 0,1,2,3
   Dim arrMarks1(1 To 5) As Long     ' Create array with locations 1,2,3,4,5
   Dim arrMarks3(2 To 4) As Long     ' Create array with locations 2,3,4
End Sub
```

Dynamic array

```
Public Sub DecArrayDynamic()
   Dim arrMarks() As Long      ' Declare  dynamic array
   ReDim arrMarks(0 To 5)      ' Set the size of the array when you are ready
End Sub
```

Array keyword

```
Public Sub DeclareArray()
   '   To create and "Array", use the Variant keyword
   Dim arr1 As Variant
   arr1 = Array("Orange", "Peach", "Pear")

   Dim arr2 As Variant
   arr2 = Array(5, 6, 7, 8, 12)
End Sub
```

Create an array using the split keyword

```
public Sub DeclareArrayUsingSplit()
   Dim s As String
   s = "Red,Yellow,Green,Blue"

   Dim arr() As String
   arr = Split(s, ",")
End Sub
```

Looping through an array

```
Public Sub ArrayLoops()
   Dim arrMarks(0 To 5) As Long
   Dim i As Long

   For i = LBound(arrMarks) To UBound(arrMarks)
      arrMarks(i) = 5 * Rnd       ' Fill the array with random numbers
   Next i
End Sub
```

The functions LBound and UBound are very useful. Using them means our loops will work correctly with any array size. The real benefit is that if the size of the array changes we do not have to change the code for printing the values. A loop will work for an array of any size as long as you use these functions.

```
For Each mark In arrMarks
    mark = 5 * Rnd           ' Will not change the array value
Next mark
```

Check if an array is allocated

Sometimes, an array is declared without dimensions and grows dynamically with the ReDim keyword. That array may stay without being re-dimensioned. Using the LBound(..) or UBound(..) function on that array will throw the "Subscript out of range error". A solution is to use the following snippet before using the LBound or UBound functions.

```
Dim myArray() As String          'Declare array without dimensions

If (Not Not myArray) = 0 Then    'Means it is not allocated
.
.
Else
.
.
End if
```

Collections

It is better to use a dictionary rather than a collection, for the following reasons:

- Performance.
- Richer functionalities.
- Everything you can do with a collection, you can do with a dictionary as well.

Reference: https://www.experts-exchange.com/articles/3391/Using-the-Dictionary-Class-in-VBA.html

Dictionaries

```
Option Explicit

'   Add reference: Microsoft Scripting Runtime
Public Sub DictionaryTest()
    Dim oDict As Scripting.Dictionary      '   Early binding

    Set oDict = New Scripting.Dictionary

    oDict("Apple") = 5
    oDict("Orange") = 50
    oDict("Peach") = 44
    oDict("Banana") = 47
    oDict("Plum") = 48
    oDict.Add Key:="Pear", Item:="22"
    Call oDict.Add("Strawberry", 11)

    Debug.Print ("There are " & oDict.Count & " items")
    oDict.Remove "Strawberry"
    Debug.Print ("There are " & oDict.Count & " items")

    '   Checks if an item exists by the key
    If Not oDict.Exists("Grapes") Then
        Debug.Print ("This dictionary does not contain grapes")
    End If

    Set oDict = Nothing
End Sub
```

- Adding the same key more than once, will result in an error.
- If you use the Item property to attempt to set an item for a non-existent key, the Dictionary will implicitly add that item along with the indicated key.
- Similarly, if you attempt to retrieve an item associated with a non-existent key, the Dictionary will add a blank item, associated with that key.
- CompareMode is used to compare the keys ➔ Binary vs Text Compare.

Traversing the Dictionary

```
Dim key As Variant
For Each key In oDict.Keys
      Debug.Print key & " - " & oDict(key)
Next
```

Removing a key

The Remove method removes the item associated with the specified key from the Dictionary, as well as that key.

```
MyDictionary.Remove "SomeKey"
```

Clear the dictionary

```
MyDictionary.RemoveAll
```

Boosting Performance

Speeding the read and write process from cells

- Read data in ranges.
- Turn screen updating off
- Turn calculation off
- Read and write the range at once

```
Sub Datechange()
    On Error GoTo error_handler

    Dim initialMode As Long

    initialMode = Application.Calculation
    Application.Calculation = xlCalculationManual
    Application.ScreenUpdating = False

    Dim data As Variant
    Dim i As Long

    'copy range to an array
    data = Range("D2:D" & Range("D" & Rows.Count).End(xlUp).Row)

    For i = LBound(data, 1) To UBound(data, 1)
        If IsDate(data(i, 1)) Then data(i, 1) = CDate(data(i, 1))
    Next i

    'copy array back to range
    Range("D2:D" & Range("D" & Rows.Count).End(xlUp).Row) = data

exit_door:
    Application.ScreenUpdating = True    Application.Calculation = initialMode
    Exit Sub

error_handler:
    'if there is an error, let the user know
    MsgBox "Error encountered on line " & i + 1 & ": " & Err.Description
    Resume exit_door 'don't forget the exit door to restore the calculation mode
End Sub
```

Clearing Ranges

When clearing cells in Excel and we already know which range needs to be cleared, it is much faster to use the .Clear method on the predefined range, rather than clearing cell by cell.

```
Thisworkbook.Sheets(1).Range("A1:J999").Clear
```

Calculating elapsed time in seconds

```
Private Sub Process()
    Dim tickStart   As Date: tickStart = Now()
    Dim tickEnd     As Date

    '  Processing goes here
    tickEnd = Now()

    MsgBox DateDiff("s", tickStart, tickEnd)
End Sub
```

Binary search the last filled row / column

```
Public Function GetLastFilledRow(ByRef oSheet As Worksheet, ByVal col As Long) As Long
    Dim left As Long, right As Long, mid As Long, best As Long

    left = 1
    right = 1048576 ' Maximum index of a row in Excel
    best = 0

    Do While (left <= right)
        mid = (left + right) \ 2

        If (oSheet.Cells(mid, col) <> "") Then
            best = mid
            left = mid + 1
        Else
            right = mid - 1
        End If
    Loop

    GetLastFilledRow = best
End Function


Public Function GetLastFilledRow(ByRef oSheet As Worksheet, ByVal row As Long) As Long
    Dim left As Long, right As Long, mid As Long, best As Long

    left = 1
    right = 1048576 ' Maximum index of a row in Excel
    best = 0

    Do While (left <= right)
        mid = (left + right) \ 2

        If (oSheet.Cells(row, mid) <> "") Then
            best = mid
            left = mid + 1
        Else
            right = mid - 1
        End If
    Loop

    GetLastFilledRow = best
End Function
```

Sorting: Mergesort

```
Option Explicit

Const MaxN As Long = 100000

Dim a(1 To MaxN) As Long
Dim tmp(1 To MaxN) As Long

Private Sub Mergesort(ByVal l As Long, ByVal r As Long)
    If (r > l) Then
        Dim mid As Long:  mid = (r + l) \ 2

        Call Mergesort(l, mid)
        Call Mergesort(mid + 1, r)

        Dim i As Long, j As Long, k As Long

        i = l
        j = mid + 1
        k = 1
        Do While (i <= mid And j <= r)
            If (a(i) > a(j)) Then
                tmp(k) = a(j)
                j = j + 1
            Else
                tmp(k) = a(i)
                i = i + 1
            End If
            k = k + 1
        Loop

        Do While (i <= mid)
            tmp(k) = a(i)
            i = i + 1
            k = k + 1
        Loop

        Do While (j <= r)
            tmp(k) = a(j)
            j = j + 1
            k = k + 1
        Loop
        For i = 1 To r - l + 1
            a(l + i - 1) = tmp(i)
        Next i
    End If
End Sub
```

```
Public Sub Test()
    Dim i As Long
    Dim tickStart   As Date: tickStart = Now()
    Dim tickEnd     As Date

    For i = 1 To MaxN
        a(i) = Rnd * MaxN
    Next i

    Call Mergesort(1, MaxN)

    For i = 2 To MaxN
        Debug.Assert a(i) >= a(i - 1)
    Next i

    tickEnd = Now()

    Debug.Print "Time taken: " & DateDiff("s", tickStart, tickEnd)
End Sub
```

File Handling

Selecting a file via the File Dialog

The File Dialog is used to select files by browsing the computer. It also allows multiselect, give the possibility to add filters so that we have a choice of which kind of files can be selected, etc...

```
Sub UseFileDialogOpen()
    Dim lngCount As Long

    ' Open the file dialog
    With Application.FileDialog(msoFileDialogOpen)
        ' .AllowMultiSelect = True
        .AllowMultiSelect = False
        .Show
        .Filters.Add "Txt", "*.txt"

        If .SelectedItems.Count = 1 Then
            ThisWorkbook.Sheets("Instructions").Cells(15, 6).Value = .SelectedItems(1)
        Else
            ThisWorkbook.Sheets("Instructions").Range("G15:G15").Clear
        End If

        ' Display paths of each file selected
'        For lngCount = 1 To .SelectedItems.Count
'            MsgBox .SelectedItems(lngCount)
'        Next lngCount
    End With
End Sub
```

Reading from an input file

```
Public Sub ReadFile()
    Dim myfile As String: myfile = "..."
    Dim textline As String
    Dim linecount As Long: linecount = 0

    Close #1
    Open myfile For Input As #1

    Do Until EOF(1)
        Line Input #1, textline
        linecount = linecount + 1
    Loop

    Debug.Print linecount

    Close #1
End Sub
```

Writing to an output file

```
Public Sub WriteToFile()
    Dim myfile As String: myfile = "c:\users\x76544\try.txt"

    Close #1

    Open myfile For Output As #1
    Print #1, "This is a test"      '   Outputs to file without double quotes
```

```
    Write #1, "This is a test"      '   Outputs to file with double quotes

    Close #1
End Sub
```

Getting a file extension

```
Set oFs = New FileSystemObject
.
.
For Each oFile In currentFolder.Files
   .
   .
   Debug.Print oFs.GetExtensionName(oFile.path)
Next
```

Recursively get a list of files

Firstly, we should add a reference to the DLL “Microsoft Scripting Runtime”.
This DLL exposes the “FileSystemObject” class, which will be used for traversing the folders recursively.
The following example traverses a folder, picks up all the .cpp files and count the number of lines each file contains.

```
Sub CountLines(oFile As File)
    Dim oTextStream As TextStream
    Dim lineCount As Long: lineCount = 0

    Set oTextStream = oFile.OpenAsTextStream(ForReading)

    Do While Not (oTextStream.AtEndOfStream)
        oTextStream.ReadLine
        lineCount = lineCount + 1
    Loop

    fileNum = fileNum + 1
End Sub

Sub Traverse(currentFolder As Folder)
    Dim oFile As File
    Dim oFolder As Folder

    '   Gets the list of .cpp files in the current folder
    For Each oFile In currentFolder.Files
        If (oFile.Type = "CPP File") Then
            '   Code goes here...
        End If
    Next

    '   Recurse in each folder
    For Each oFolder In currentFolder.SubFolders
        Call Traverse(oFolder)
    Next
End Sub

Public Sub Test()
    Dim oFS As Scripting.ileSystemObject

    Set oFS = New FileSystemObject
    Call Traverse(oFS.GetFolder("..."))

    Set oFS = Nothing
End Sub
```

Copying files & folders

```
Dim ofs As New FileSystemObject
ofs.CopyFile "Source File", "Destination File"

Set ofs = Nothing
```

The FileSystemObject also exposes other interesting methods like to copy folders, create folders etc.

Connection to Database

Connecting to the local MS Access database in VBA

Reference: https://msdn.microsoft.com/en-us/library/office/ff835631.aspx

```
Dim db      As DAO.Database
Dim rs      As DAO.Recordset
Dim strSQL  As String

'  Use the current db
Set db = CurrentDb

'  Build the sql query
strSQL = "SELECT * FROM Person"

'  Execute the query
Set rs = db.OpenRecordset(strSQL)

'  Traversing the dataset result
Do While Not rs.EOF
  Debug.Print rs!Id & " " & rs!firstname & " " & rs!familyname
  rs.MoveNext
Loop

Debug.Print rs.RecordCount

'  Cleaning up
rs.Close
db.Close
```

Dealing with MS Office & PDF files

Microsoft Excel

Creating an Excel File

```
Dim oXlsxApplication    As Excel.Application
Dim oXlsxWorkbook       As Excel.Workbook
Dim oXlsxWorksheet      As Excel.Worksheet

Set oXlsxApplication   = New Excel.Application
Set oXlsxWorkbook      = oXlsxApplication.Workbooks.Add
Set oXlsxWorksheet     = oXlsxWorkbook.Sheets.Add

' Code goes here
' ...

Set oXlsxApplication = Nothing
Set oXlsxWorkbook    = Nothing
Set oXlsxWorksheet   = Nothing
```

Microsoft Word

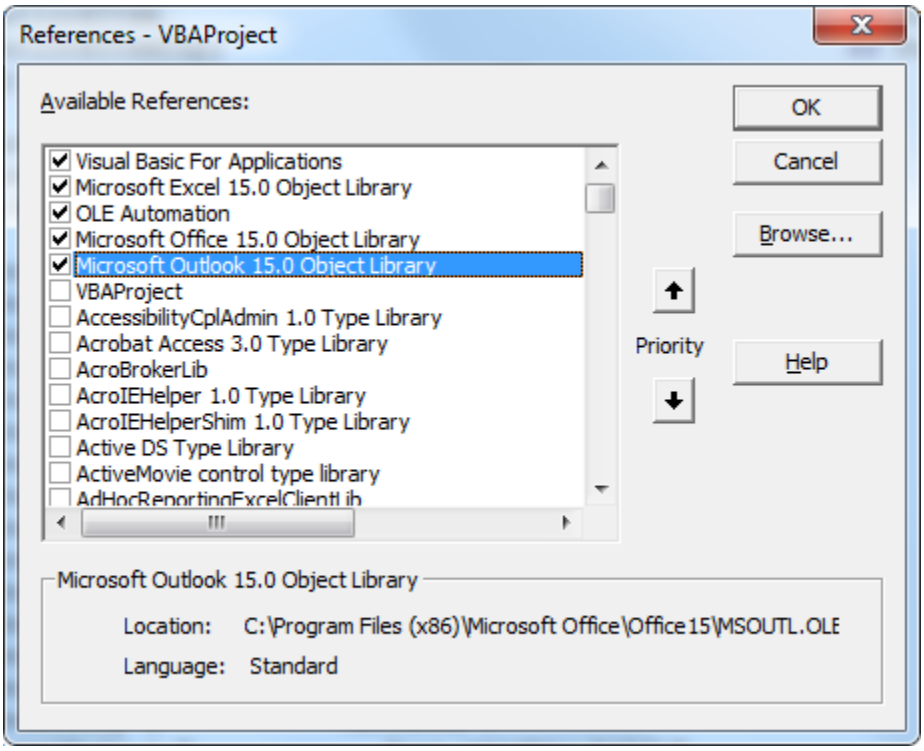
Creating a Word Document

```
Dim oWordApplication    As Word.Application
Dim oWordDocument       As Word.Document

Set oWordApplication = New Word.Application
Set oWordDocument = oWordApplication.Documents.Add

With oWordDocument
    .Content.InsertAfter "This is a test"
End With

oWordApplication.Visible = True
```

Outlook

References

To use the outlook object, make sure the “Microsoft Outlook 15.0 Object Library” is added as reference.



Sending emails via Outlook

```
Dim locObjOutlook            As Outlook.Application
Dim locObjOutlookItem        As Outlook.MailItem
Dim locObjOutlookItemCopy    As Outlook.MailItem
Dim htmlBody                 As String: htmlBody = ""

Set locObjOutlook = New Outlook.Application
Set locObjOutlookItem = locObjOutlook.CreateItem(olMailItem)
locObjOutlookItem.BodyFormat = olFormatHTML

htmlBody = htmlBody & "<html>"
htmlBody = htmlBody & "   <head>"
.
.
htmlBody = htmlBody & "   </head>"
htmlBody = htmlBody & "   <body>"
.
.
htmlBody = htmlBody & "   </body>"
htmlBody = htmlBody & "</html>"

locObjOutlookItem.htmlBody = htmlBody
locObjOutlookItem.Display          ' displays the email first
Set locObjOutlook = Nothing
```

Creating a PDF File

We can simulate the creation of a pdf file by first creating an office file and then using the “Save” command to save it as a pdf.

For saving a file under the pdf format, we use file format = 17.

```
Dim oWordApplication    As Word.Application
Dim oWordDocument       As Word.Document

Set oWordApplication = New Word.Application
Set oWordDocument = oWordApplication.Documents.Add

With oWordDocument
    .Content.InsertAfter "This is a test"
    .SaveAs2 "C:\Users\x76544\" & "myDoc.pdf", FileFormat:=17
End With

oWordDocument.Close

Set oWordApplication = Nothing
```

Internet Explorer Automation

Required References

Add the following references before doing the internet explorer automation.

- Microsoft Internet Controls
- Microsoft HTML Object Library

Windows API

```
Declare Function apiShowWindow Lib "user32" Alias "ShowWindow" (ByVal hwnd As Long, ByVal
nCmdShow As Long) As Long
Declare Function SetForegroundWindow Lib "user32" (ByVal hwnd As Long) As Long
```

Loading a new Internet Explorer Window and navigate to www.google.com

```
Dim ieWindow    As SHDocVw.InternetExplorer

Set ieWindow = New SHDocVw.InternetExplorer
With ieWindow
  .Visible = True
  .navigate "www.google.com"

  While ieWindow.Busy Or ieWindow.readyState <> READYSTATE_COMPLETE
      ' Just wait
      DoEvents    ' gives back control to the OS
  Wend

  MsgBox "Google is loaded!"
  Set ieWindow = Nothing
End With
```

Check if any of the opened Internet Explorer windows is already on a specific page

The following code snippet iterates through all the opened windows and checks if one of the internet explorer windows has already loaded the url passed as parameter.

```
Public Function Start(ByVal url As String) As Boolean
    Dim windows     As SHDocVw.shellWindows: Set windows = New SHDocVw.shellWindows
    Dim ieWindow    As SHDocVw.InternetExplorer
    Dim found       As Boolean: found = False

    For Each ieWindow In windows
        Debug.Print ieWindow.Name & " " & ieWindow.LocationURL
        If ieWindow.Name Like "*Internet Explorer*" Then
            If InStr(ieWindow.LocationURL, url) > 0 Then
                found = True
                Set ie = ieWindow.Application

                Exit For
            End If
        End If
    Next ieWindow

    If found Then
        Call SetForegroundWindow(ie.hwnd)
    End If
```

```
    Set windows = Nothing
    Set ieWindow = Nothing

    Start = found
End Function
```

Document object

The document object of the loaded page can be obtained by:

```
Private ie As InternetExplorer
ByVal htmlDoc As MSHTML.HTMLDocument
.
.
Set htmlDoc = ie.document
```

Searching for an HTML Element by its ID

The "Microsoft HTML Object Library" exposes the different HTML object types like buttons, textboxes, checkboxes etc. To get a reference to an html object, use the following command:

```
Dim htmlInput        As MSHTML.HTMLInputElement

Set htmlInput = htmlDoc.getElementById("MyCheckbox")
```

Common HTML objects

- MSHTML.HTMLInputElement
- MSHTML.HTMLSelectElement
- MSHTML.HTMLButtonElement
- MSHTML.HTMLLinkElement

Searching for HTML elements by its type

Use the IHTMLElementCollection collection interface to retrieve a list of html objects.

```
Dim htmlDoc             As MSHTML.HTMLDocument
Dim htmlLinks           As MSHTML.IHTMLElementCollection
Dim htmlLink            As MSHTML.HTMLLinkElement

Set htmlDoc = ie.document
Set htmlLinks = htmlDoc.getElementsByTagName("a")
For Each htmlLink In htmlLinks
   If htmlLink.id Like "*" & likeId & "*" Then
      .
      .
   End If
Next htmlLink
```

Generic Robot Class

The generic robot class can be used in any VBA project and can be easily extended.

```
Dim oRobot      As Robot
Dim url         As String: url = "ww.myWebsiteURL.com"

Set oRobot = New Robot

If oRobot.Start(url) Then
  Call oRobot.CheckCheckBox("MyCheckBox")
  Call oRobot.SetComboValue("MyCombo", "Value01")
Else
  MsgBox "Please launch " & url
End If

Set oRobot = Nothing
```

Robot.ds

Waiting in the application

Sometimes, it is good to wait for a particular amount of time before executing the next command. This can be done by the following command.

```
Application.Wait (Now + TimeValue("0:00:03"))
```

Force the robot to click on “Yes” on a confirmation window

Sometimes, we have to click a “Yes” or “No” button on a JavaScript window. Unfortunately, this cannot be automated via the DOM components of the Internet Explorer Object.

A workaround is to override the window, so that whenever the confirmation windows is displayed, the “true” value is always returned. The “execScript” command is used.

```
Private Sub OverrideConfirmationWindow()
    Dim htmlDoc             As MSHTML.HTMLDocument

    Set htmlDoc = ie.document
    htmlDoc.parentWindow.execScript "window.confirm = function(){return true;};"
End Sub
```

ActiveX controls

https://msdn.microsoft.com/en-us/library/aa231215(v=vs.60).aspx

Formatting Data

Padding with leading zeros

To add padding leading zeros in front of a number, use the command Format(..). Example

```
Format(198, "0000")      ' output = 0198
```

MS DOS

Getting help for a particular command

```
S:\>help dir

Displays a list of files and subdirectories in a directory.

DIR [drive:][path][filename] [/A[[:]attributes]] [/B] [/C] [/D] [/L] [/N]
  [/O[[:]sortorder]] [/P] [/Q] [/R] [/S] [/T[[:]timefield]] [/W] [/X] [/4]

  [drive:][path][filename]
              Specifies drive, directory, and/or files to list.

  /A          Displays files with specified attributes.
  attributes   D  Directories                R  Read-only files
               H  Hidden files               A  Files ready for archiving
               S  System files               I  Not content indexed files
               L  Reparse Points             -  Prefix meaning not
  /B          Uses bare format (no heading information or summary).
  /C          Display the thousand separator in file sizes.  This is the
              default.  Use /-C to disable display of separator.
  /D          Same as wide but files are list sorted by column.
  /L          Uses lowercase.
  /N          New long list format where filenames are on the far right.
  /O          List by files in sorted order.
  sortorder    N  By name (alphabetic)       S  By size (smallest first)
               E  By extension (alphabetic)  D  By date/time (oldest first)
               G  Group directories first    -  Prefix to reverse order
  /P          Pauses after each screenful of information.
  /Q          Display the owner of the file.
  /R          Display alternate data streams of the file.
  /S          Displays files in specified directory and all subdirectories.
  /T          Controls which time field displayed or used for sorting
  timefield   C  Creation
              A  Last Access
              W  Last Written
  /W          Uses wide list format.
  /X          This displays the short names generated for non-8dot3 file
              names.  The format is that of /N with the short name inserted
              before the long name. If no short name is present, blanks are
              displayed in its place.
  /4          Displays four-digit years

Switches may be preset in the DIRCMD environment variable.  Override
preset switches by prefixing any switch with - (hyphen)--for example, /-W.
```

Accessing a folder on the network

When accessing a network path directly from the command prompt, we have the error **CMD does not support UNC paths as current directories.**

```
S:\>cd "\\network\path"
CMD does not support UNC paths as current directories.
```

To get around this problem, we map the network path to a free drive. This can be done easily with the pushD command. Windows will assign the path to a free drive letter.

```
S:\>pushD “\\network\path”
Z:\network\path>
```

We can then access the folder as we do on the local drive.

To free the drive letter that has been assigned to the network path, use the popD command.

```
Z:\network\path>popD
S:\>
```

Executing a command on selected files

```
S:\>forfiles /?

FORFILES [/P pathname] [/M searchmask] [/S]
         [/C command] [/D [+ | -] {dd/MM/yyyy | dd}]

Description:
    Selects a file (or set of files) and executes a
    command on that file. This is helpful for batch jobs.

Parameter List:
    /P    pathname      Indicates the path to start searching.
                        The default folder is the current working
                        directory (.).

    /M    searchmask    Searches files according to a searchmask.
                        The default searchmask is '*' .

    /S                  Instructs forfiles to recurse into
                        subdirectories. Like "DIR /S".

    /C    command       Indicates the command to execute for each file.
                        Command strings should be wrapped in double
                        quotes.

                        The default command is "cmd /c echo @file".

                        The following variables can be used in the
                        command string:
                        @file    - returns the name of the file.
                        @fname   - returns the file name without
                                   extension.
                        @ext     - returns only the extension of the
                                   file.
                        @path    - returns the full path of the file.
                        @relpath - returns the relative path of the
                                   file.
```

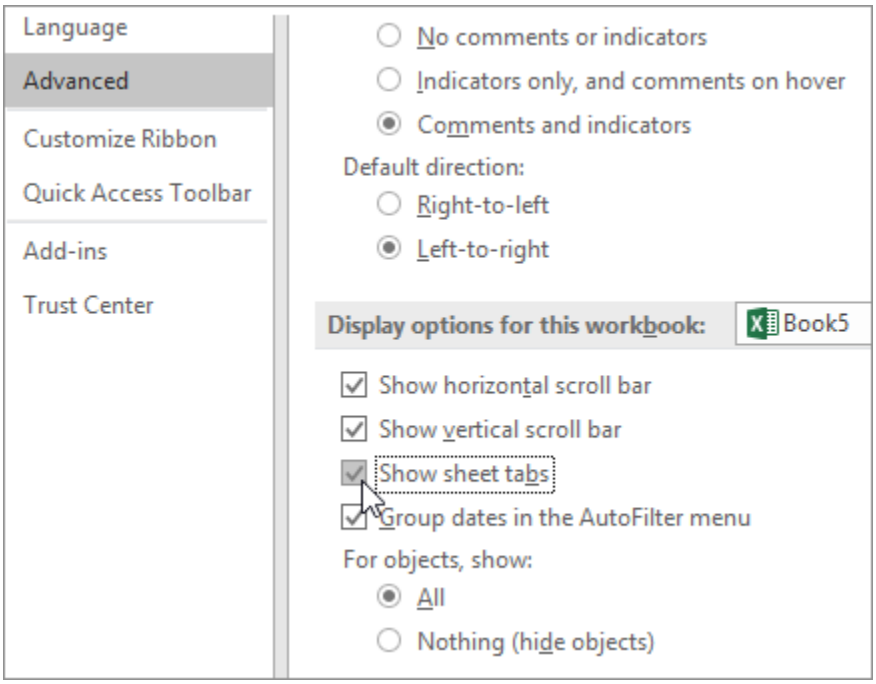
```
                        @isdir   - returns "TRUE" if a file type is
                                   a directory, and "FALSE" for files.
                        @fsize   - returns the size of the file in
                                   bytes.
                        @fdate   - returns the last modified date of the
                                   file.
                        @ftime   - returns the last modified time of the
                                   file.

                        To include special characters in the command
                        line, use the hexadecimal code for the character
                        in 0xHH format (ex. 0x09 for tab). Internal
                        CMD.exe commands should be preceded with
                        "cmd /c".

    /D    date          Selects files with a last modified date greater
                        than or equal to (+), or less than or equal to
                        (-), the specified date using the
                        "dd/MM/yyyy" format; or selects files with a
                        last modified date greater than or equal to (+)
                        the current date plus "dd" days, or less than or
                        equal to (-) the current date minus "dd" days. A
                        valid "dd" number of days can be any number in
                        the range of 0 - 32768.
                        "+" is taken as default sign if not specified.

    /?                  Displays this help message.

Examples:
    FORFILES /?
    FORFILES
    FORFILES /P C:\WINDOWS /S /M DNS*.*
    FORFILES /S /M *.txt /C "cmd /c type @file | more"
    FORFILES /P C:\ /S /M *.bat
    FORFILES /D -30 /M *.exe
             /C "cmd /c echo @path 0x09 was changed 30 days ago"
    FORFILES /D 01/01/2001
             /C "cmd /c echo @fname is new since Jan 1st 2001"
    FORFILES /D +31/1/2019 /C "cmd /c echo @fname is new today"
    FORFILES /M *.exe /D +1
    FORFILES /S /M *.doc /C "cmd /c echo @fsize"
    FORFILES /M *.txt /C "cmd /c if @isdir==FALSE notepad.exe @file"
```

Useful techniques

Hiding sheet tabs

Sometimes, it may prove useful to hide the sheet tabs to prevent “beginner” users from navigating from one sheet to another.

From the Options > Advanced tab, the sheet tabs can be displayed or hidden.



Hiding Row numbers and Column numbers

When programming an end user computing tool, it is sometimes useful to hide the row numbers and the column numbers. It is easily done by unchecking the “Headings” option from the “View” tab.