



This page describes VBA procedures you can use to sort or order worksheets within a workbook.

---

## *Introduction*

Excel does not provide a mechanism or command to sort or otherwise order worksheets within a workbook. This article provides a number of VBA functions that you can use to sort or reorder worksheets. With these functions, you can

- Sort some or all worksheet by name, in either ascending or descending order.
- Order the worksheets by the names provided in an array.
- Order the worksheets in either ascending or descending order based on a cell reference in each worksheet.
- Group worksheet by tab color (Excel 2002 and later only).
- Order worksheets based on sheet names in a range of cells.

The functions are presented below. You can [download a a bas module file](#) containing all the code on this page. The code on this page consists of functions that take parameters to control their behavior. You should

---

## *Sorting By Name*

This function allows you to sort some or all worksheets based on their names, in either ascending or descending order. The function declaration is:

```
Public Function SortWorksheetsByName(ByVal FirstToSort As Long, ByVal  
    ByVal Ref ErrorText As String, Optional ByVal SortDescending As Boole
```

`FirstToSort` is the index (position) of the first worksheet to sort.

`LastToSort` is the index (position) of the last worksheet to sort. If either both `FirstToSort` and `LastToSort` are less than or equal to 0, all sheets in the workbook are sorted.

`ErrorText` is a variable that will receive the text description of any error that may occur.

`SortDescending` is an optional parameter to indicate that the sheets

should be sorted in descending order. If `True`, the sort is in descending order. If omitted or `False`, the sort is in ascending order.

The function returns `True` if successful. If an error occurs, the function returns `False` and the variable `ErrorMessage` is set to the text of the error message.

The code for `SortWorksheetsByName` is shown below.

```
Public Function SortWorksheetsByName(ByVal FirstToSort As Long, ByVal
    ByVal Ref ErrorMessage As String, Optional ByVal SortDescending As Boolean)
    .....
    ' SortWorksheetsByName
    ' This sorts the worksheets from FirstToSort to LastToSort by name
    ' in either ascending (default) or descending order. If successful,
    ' ErrorMessage is vbNullString and the function returns True. If
    ' unsuccessful, ErrorMessage gets the reason why the function failed
    ' and the function returns False.
    .....

    Dim M As Long
    Dim N As Long
    Dim WB As Workbook
    Dim B As Boolean

    Set WB = Worksheets.Parent
    ErrorMessage = vbNullString

    If WB.ProtectStructure = True Then
        ErrorMessage = "Workbook is protected."
        SortWorksheetsByName = False
    End If

    .....
    ' If First and Last are both 0, sort all sheets.
    .....
    If (FirstToSort = 0) And (LastToSort = 0) Then
        FirstToSort = 1
        LastToSort = WB.Worksheets.Count
    Else
        .....
        ' More than one sheet selected. We
        ' can sort only if the selected
        ' sheet are adjacent.
        .....
        B = TestFirstLastSort(FirstToSort, LastToSort, ErrorMessage)
        If B = False Then
            SortWorksheetsByName = False
            Exit Function
        End If
    End If

    .....
    ' Do the sort, essentially a Bubble Sort.
    .....
    For M = FirstToSort To LastToSort
        For N = M To LastToSort
            If SortDescending = True Then
                If StrComp(WB.Worksheets(N).Name, WB.Worksheets(M).Name,
                    WB.Worksheets(N).Move before:=WB.Worksheets(M)
                End If
            Else
                If StrComp(WB.Worksheets(N).Name, WB.Worksheets(M).Name,
                    WB.Worksheets(N).Move before:=WB.Worksheets(M)
            End If
        End For
    End For
End Function
```

```

        End If
    End If
Next N
Next M

SortWorksheetsByName = True

End Function

```



### *SortWorksheetsByNameArray*

The `SortWorksheetByNameArray` function sorts the worksheets in the order of the names passed as an array. While the individual elements of the array need not refer to adjacent worksheets, the worksheets, taken as a group, named by the values in the array, must be adjacent. You cannot sort non-adjacent sheets. The function returns `True` if successful or `False` if an error occurred. If an error occurred, the variable `ErrorText` will contain the text of the error message.

The declaration of `SortWorksheetsByNameArray` declaration is shown below:

```
Public Function SortWorksheetsByNameArray(NameArray() As Variant, ByRef ErrorText As String) As Boolean
```

`NameArray` is an array containing the worksheet names in the order that they should be ordered.

The code for `SortWorksheetsByNameArray` is shown below:

```
Public Function SortWorksheetsByNameArray(NameArray() As Variant, ByRef
.....
' WorksheetSortByArray
' This procedure sorts the worksheets named in NameArray to the order
' which they appear in NameArray. The adjacent elements in NameArray
' not be adjacent sheets, but the collection of all sheets named in
' NameArray must form a set of adjacent sheets. If successful, return
' True and ErrorText is vbNullString. If failure, returns False and
' ErrorText contains reason for failure.
.....

Dim Arr() As Long
Dim N As Long
Dim M As Long
Dim L As Long
Dim WB As Workbook

ErrorText = vbNullString

.....
' The NameArray need not contain all of the
' worksheets in the workbook, but the sheets
' that it does name together must form a group of
' adjacent sheets. Sheets named in NameArray
' need not be adjacent in the NameArray, only
' that when all sheet taken together, they form an
' adjacent group of sheets
.....

ReDim Arr(LBound(NameArray) To UBound(NameArray))
On Error Resume Next
For N = LBound(NameArray) To UBound(NameArray)

```

```

' .....
```

```

' Ensure all sheets in name array exist
' .....
```

```

Err.Clear
M = Len(WB.Worksheets(NameArray(N)).Name)
If Err.Number <> 0 Then
    ErrorText = "Worksheet does not exist."
    SortWorksheetsByNameArray = False
    Exit Function
End If
' .....
```

```

' Put the index value of the sheet into Arr. Ensure there
' are no duplicates. If Arr(N) is not zero, we've already
' loaded that element of Arr and thus have duplicate sheet
' names.
' .....
```

```

If Arr(N) > 0 Then
    ErrorText = "Duplicate worksheet name in NameArray."
    SortWorksheetsByNameArray = False
    Exit Function
End If

Arr(N) = Worksheets(NameArray(N)).Index
Next N
' .....
```

```

' Sort the sheet indexes. We don't use
' these for the sorting order, but we
' do use them to ensure that the group
' of sheets passed in NameArray are
' together contiguous.
' .....
```

```

For M = LBound(Arr) To UBound(Arr)
    For N = M To UBound(Arr)
        If Arr(N) < Arr(M) Then
            L = Arr(N)
            Arr(N) = Arr(M)
            Arr(M) = L
        End If
    Next N
Next M
' .....
```

```

' Now that Arr is sorted ascending, ensure
' that the elements are in order differing
' by exactly 1. Otherwise, sheet are not
' adjacent.
' .....
```

```

If ArrayElementsInOrder(Arr:=Arr, Descending:=False, Diff:=1) = False
    ErrorText = "Specified sheets are not adjacent."
    SortWorksheetsByNameArray = False
    Exit Function
End If
' .....
```

```

' Now, do the actual move of the sheets.
' .....
```

```

On Error GoTo 0
WB.Worksheets(NameArray(LBound(NameArray))).Move before:=WB.Worksheet
For N = LBound(NameArray) + 1 To UBound(NameArray) - 1
    WB.Worksheets(NameArray(N)).Move before:=WB.Worksheets(NameArray(
Next N

SortWorksheetsByNameArray = True

```

End Function

## *GroupSheetsByColor*

The `GroupSheetsByColor` function groups sheets by their tab color (available only in Excel 2002 and later). You specify in an array the colors and the order in which those colors should appear. The sheets are grouped according to those color indicators. The color indicators are the `ColorIndex` values, not actual RGB colors. The declaration for `GroupSheetsByColor` is shown below:

```
Public Function GroupSheetsByColor(ByVal FirstToSort As Long,
    ByVal LastToSort As Long, _
    ByRef ErrorText As String, ColorArray() As Long) As Boolean
    FirstToSort is the index (position) number of the first sheet to sort.
```

`LastToSort` is the index (position) number of the last sheet to sort. If both `FirstToSort` and `LastSheetToSort` are less than or equal to 0, all sheets are sorted.

`ErrorText` is a variable that will contain the error message if an error occurs.

`ColorArray` is an array of longs indicating the colors and order in which the sheets should be grouped.

The code is shown below:

```
Public Function GroupSheetsByColor(ByVal FirstToSort As Long, ByVal I
    ByRef ErrorText As String, ColorArray() As Long) As Boolean
    .....
    ' GroupSheetsByColor
    ' This groups worksheets by color. The order of the colors
    ' to group by must be the ColorIndex values stored in
    ' ColorsArray.
    .....

    Dim WB As Workbook
    Dim B As Boolean
    Dim N1 As Long
    Dim N2 As Long
    Dim N3 As Long
    Dim CI1 As Long
    Dim CI2 As Long
    Dim CArray As Variant

    Dim CNdx1 As Long
    Dim Cndx2 As Long

    Const MIN_COLOR_INDEX = 1
    Const MAX_COLOR_INDEX = 56

    If IsArrayAllocated(ColorArray) = False Then
        ErrorText = "ColorArray is not a valid, allocated array."
        GroupSheetsByColor = False
        Exit Function
    End If
```

```

Set WB = Worksheets.Parent
ErrorText = vbNullString

.....
' Setup ColorIndex array
.....
If IsMissing(ColorArray) = False Then
    If IsArray(ColorArray) = False Then
        ErrorText = "ColorArray is not an array"
        GroupSheetsByColor = False
        Exit Function
    End If
Else
    .....
    ' Ensure all color indexes are valid.
    .....
    For N1 = LBound(ColorArray) To UBound(ColorArray)
        If (ColorArray(N1) > MAX_COLOR_INDEX) Or (ColorArray(N1) < MI
            ErrorText = "Invalid ColorIndex in ColorArray"
            GroupSheetsByColor = False
            Exit Function
        End If
    Next N1
End If

Set WB = Worksheets.Parent

ErrorText = vbNullString

If (FirstToSort <= 0) And (LastToSort <= 0) Then
    FirstToSort = 1
    LastToSort = WB.Worksheets.Count
End If

B = TestFirstLastSort(FirstToSort, LastToSort, ErrorText)
If B = False Then
    GroupSheetsByColor = False
    Exit Function
End If

For N1 = FirstToSort To LastToSort
    If WB.Worksheets(N1).Tab.ColorIndex = ColorArray(LBound(ColorArra
        WB.Worksheets(N1).Move before:=WB.Worksheets(1)
    Exit For
End If
Next N1
N3 = 1
For N2 = LBound(ColorArray) To UBound(ColorArray)
    For N1 = 2 To LastToSort
        If WB.Worksheets(N1).Tab.ColorIndex = ColorArray(N2) Then
            WB.Worksheets(N1).Move after:=WB.Worksheets(N3)
            N3 = N3 + 1
        End If
    Next N1
Next N2

GroupSheetsByColor = True

End Function

```

## Support Functions

The following functions are used by the primary functions on this page.

```
Private Function ArrayElementsInOrder(Arr As Variant, _
    Optional Descending As Boolean = False, _
    Optional Diff As Integer = 0) As Boolean
.....
' ArrayElementsInOrder
' This function tests an array of integers (Long or Int) to determine
' if they are in order, in ascending or descending sort order, and
' optionally if they all differ by exactly Diff. Diff is the absolute
' value between two adjacent elements. Do not use a negative number
' for a descending sort; Diff should always be greater than 0 to test
' the differences or 0 to ignore differences. The default behavior
' is to test whether the elements are in ascending order with any
' difference between them. Set the Descending and/or Diff parameters
' to change this.
.....
Dim N As Long
For N = LBound(Arr) To UBound(Arr) - 1
    If Descending = False Then
        If Diff > 0 Then
            If Arr(N) <> Arr(N + 1) - Diff Then
                ArrayElementsInOrder = False
                Exit Function
            End If
        Else
            If Arr(N) > Arr(N + 1) Then
                ArrayElementsInOrder = False
                Exit Function
            End If
        End If
    Else
        If Diff > 0 Then
            If Arr(N) <> Arr(N + 1) + Diff Then
                ArrayElementsInOrder = False
                Exit Function
            End If
        Else
            If Arr(N) < Arr(N + 1) Then
                ArrayElementsInOrder = False
                Exit Function
            End If
        End If
    End If
Next N
ArrayElementsInOrder = True
End Function
```

```
Private Function TestFirstLastSort(FirstToSort As Long, LastToSort As
    ByRef ErrorText As String) As Boolean
.....
' TestFirstLastSort
' This ensures FirstToSort and LastToSort are valid values. If succes
' returns True and sets ErrorText to vbNullString. If unsuccessful, r
' False and set ErrorText to the reason for failure.
.....
```

```

ErrorText = vbNullString
If FirstToSort <= 0 Then
    TestFirstLastSort = False
    ErrorText = "FirstToSort is less than or equal to 0."
    Exit Function
End If

If FirstToSort > Worksheets.Count Then
    TestFirstLastSort = False
    ErrorText = "FirstToSort is greater than number of sheets."
    Exit Function
End If

If LastToSort <= 0 Then
    TestFirstLastSort = False
    ErrorText = "LastToSort is less than or equal to 0."
    Exit Function
End If

If LastToSort > Worksheets.Count Then
    TestFirstLastSort = False
    ErrorText = "LastToSort greater than number of sheets."
    Exit Function
End If

If FirstToSort > LastToSort Then
    TestFirstLastSort = False
    ErrorText = "FirstToSort is greater than LastToSort."
    Exit Function
End If

TestFirstLastSort = True

End Function

Private Function IsArrayAllocated(Arr As Variant) As Boolean
' .....
```

' IsArrayAllocated  
' Returns True or False indicating if Arr is an allocated  
' array.  
' .....

```

    On Error Resume Next
    Dim V As Variant
    IsArrayAllocated = True
    V = Arr(LBound(Arr, 1))
    If IsError(V) = True Then
        IsArrayAllocated = False
    End If
    If (UBound(Arr, 1) < LBound(Arr, 1)) Then
        IsArrayAllocated = False
    End If

End Function

Private Function SheetExists(WSName As String, Optional WB As Workboo
' .....
```

' SheetExists  
' Returns True if worksheet named by WSName exists in  
' Workbook WB. If WB is omitted,  
' the ActiveWorkbook is used.  
' .....

```

On Error Resume Next
SheetExists = IsError(IIf(WB Is Nothing, ActiveWorkbook, WB).Workshee
```



End Function

You can [download a a bas module file](#) containing all the code on this page.

This page last updated: 22-September-2007

Created by Chip Pearson at Pearson Software Consulting, LLC  
 Email: [chip@cpearson.com](mailto:chip@cpearson.com) Before emailing me, please read [this page](#).  
<http://www.cpearson.com/excel/sortws.aspx>  
 Copyright © 1997 - 2007, Charles H. Pearson

Subscribe To The CPearson.com Weekly Excel Newsletter

Main Page

Page Index

Topic Index

What's New

Search

Consulting

About This Site

Downloads

Feedback

Legal & Copyright

About This Web Site

Go To Page

0 - Rate This Page

Submit Rating

Essential Tools For Developers

Add-in-Express.com  
 Express your projects!™

Ready

Advertise Your Product On This Site

## Mini-graphs - free trial

Bissantz SparkMaker for Excel dashboards and reports - free trial [www.bissantz.de](http://www.bissantz.de)

### Convert PDF File to EXCEL

Easily Convert PDF files to Microsoft Excel - Free Trial!  
[www.cogniview.com/Version4.2](http://www.cogniview.com/Version4.2)

### End broken links in Excel

Find and fix broken links in Excel files automatically!  
[www.linkfixerplus.com](http://www.linkfixerplus.com)

### Downloadable Macro Books

Get over 1200 Excel visual basic macro examples with explanations  
[www.add-ins.com/macro\\_examples](http://www.add-ins.com/macro_examples)

### Excel Spreadsheet

Run your worksheet Business Intelligence  
[www.jedox.com](http://www.jedox.com)

