Creating Purchase Orders

This excel VBA is designed to read purchase order data from a source worksheet and export it into separate, styled Excel workbooks for each unique store. The goal is to automate the generation of individual purchase order documents for further processing or distribution.

1. **Initialization and Setup**
   - Declares variables for workbooks, worksheets, store IDs, counters, paths, and flags.
   - Creates a dated folder name and output path to prevent overwriting files.
   - Checks if the output folder exists; creates it if not.
   - Disables screen updating and alerts to speed up the process and suppress prompts.
   - Checks if the destination folder exists using Dir; creates it with MkDir if missing to prevent errors.
   - Temporarily disables screen updating and alerts (ScreenUpdating = False, DisplayAlerts = False) to boost performance and suppress prompts.
   - Uses "Sheet2" as the source worksheet.
   - Finds the last data row in column B (store IDs) to set loop boundaries.
   - Initializes currentStoreID, prevStoreID, and hasData to manage data processing state.
   - Clears wbNew to avoid carrying over any previous workbook

```
    Dim wsSource As Worksheet
    Dim wbNew As Workbook
    Dim wsNew As Worksheet
    Dim currentStoreID As Variant
    Dim prevStoreID As Variant

    Dim lastRow As Long
    Dim i As Long
    Dim outputRow As Long
    Dim hasData As Boolean
    Dim outputPath As String
    Dim todayDate As String

    'outputPath = "C:\Users\PC\Desktop\GITHUB\inventory\purchase\"
    todayDate = Format(Date, "yyyy-mm-dd")

    Dim basePath As String
    basePath = "C:\Users\PC\Desktop\GITHUB\inventory\purchase\"

    Dim newFolderName As String
    newFolderName = Format(Now, "yyyy-mm-dd_hh-nn-ss") ' e.g. 2025-08-21_14-45-00

    outputPath = basePath & newFolderName & "\"

' Create the folder if it doesn't exist
If Dir(outputPath, vbDirectory) = "" Then
    MkDir outputPath
End If


    Application.ScreenUpdating = False
    Application.DisplayAlerts = False

    Set wsSource = ThisWorkbook.Sheets("Sheet2")
    lastRow = wsSource.Cells(wsSource.Rows.Count, "B").End(xlUp).Row

    currentStoreID = ""
    prevStoreID = ""
    Set wbNew = Nothing
    hasData = False
```

2. Processing Each Store and Workbook Creation

- The macro loops through each row of the source worksheet starting from row 4. For each row:
    - It **checks if column B has a Store ID**.
        - If **yes**, it means a new store's data starts here:
            - If a previous workbook (wbNew) exists and contains data (hasData = True), it applies the layout, saves, and closes that workbook.
            - Then it **creates a new workbook** for the current store.
            - Sets up headers and resets variables (outputRow, hasData).
- Whether column B has a new store or not, it then checks if column C has a **Product ID**:
    - If there is a product, and a workbook is open (wsNew is not Nothing), it copies product details (Product ID, Price) into the new workbook.
    - It sets quantity (hardcoded as 100, no data from dataset), adds a formula to calculate the total price, formats numbers, and increments the output row.
    - Sets hasData = True to mark that this workbook has product data.

```
For i = 4 To lastRow

    If Trim(wsSource.Cells(i, "B").Value) <> "" Then
        ' Save previous if it had data
        If Not wbNew Is Nothing Then
            If hasData Then
                Call ApplySimpleLayout(wsNew, outputRow, todayDate, prevStoreID)
                wbNew.SaveAs fileName:=outputPath & "Store " & prevStoreID & ".xlsx", FileFormat:=51
            End If
            wbNew.Close False
        End If

        ' New workbook
        currentStoreID = wsSource.Cells(i, "B").Value
        Set wbNew = Workbooks.Add(xlWBATWorksheet)
        Set wsNew = wbNew.Sheets(1)

        ' Set up headers
        wsNew.Range("C8").Value = "Product ID"
        wsNew.Range("D8").Value = "Price"
        wsNew.Range("E8").Value = "Qty"
        wsNew.Range("F8").Value = "Total"
        outputRow = 9
        hasData = False
        prevStoreID = currentStoreID
    End If

    ' Add product rows
    If Trim(wsSource.Cells(i, "C").Value) <> "" And Not wsNew Is Nothing Then
        wsNew.Cells(outputRow, "C").Value = wsSource.Cells(i, "C").Value
        wsNew.Cells(outputRow, "D").Value = wsSource.Cells(i, "D").Value
        wsNew.Cells(outputRow, "D").NumberFormat = "0.00"
        wsNew.Cells(outputRow, "E").Value = 100
        wsNew.Cells(outputRow, "F").FormulaR1C1 = "=RC[-2]*RC[-1]"
        wsNew.Cells(outputRow, "F").NumberFormat = "0.00"
        outputRow = outputRow + 1
        hasData = True
    End If
Next i
```

3. Finalizing Export

- After the loop ends, saves and closes the last workbook if it contains data.
- Restores screen updating and alerts to normal.
- Displays a message box confirming export completion.

```
' Final file
If Not wbNew Is Nothing Then
    If hasData Then
        Call ApplySimpleLayout(wsNew, outputRow, todayDate, currentStoreID)
        wbNew.SaveAs fileName:=outputPath & "Store " & currentStoreID & ".xlsx", FileFormat:=51
    End If
    wbNew.Close False
End If

Application.DisplayAlerts = True
Application.ScreenUpdating = True

MsgBox "Clean purchase orders exported!", vbInformation
```

4. Formatting the Exported Workbook

- Adds a merged, bold, centered title "Purchase Order" at the top.
- Labels and populates purchase date, store ID, purchase order number, and shipping code in rows 3 to 6.
- Creates address and receiver fields with merged cells and bottom borders at rows 27 and 30.
- Styles header row (C8:F8) with bold text, background color, and borders.
- Adds borders to product rows if present.
- Inserts a total row summing the "Total" column, styled with background color and bold font.
- Auto-fits columns C to F for neat appearance.

```
With ws.Range("A1:H1")
    .Merge
    .Value = "Purchase Order"
    .Font.Size = 22
    .Font.Bold = True
    .HorizontalAlignment = xlCenter
    .VerticalAlignment = xlCenter
End With

' Labels and values with underlines
With ws.Range("C3")
    .Value = "Purchase Date:"
    .Font.Size = 12
    .Font.Bold = True
End With

With ws.Range("D3")
    .Value = docDate
    .Font.Size = 12
End With

With ws.Range("C4")
    .Value = "Store ID:"
    .Font.Size = 12
    .Font.Bold = True
End With

With ws.Range("D4")
    .Value = storeID
    .Font.Size = 12
End With

With ws.Range("C5")
    .Value = "Purchase Order No.:"
    .Font.Size = 12
    .Font.Bold = True
End With

With ws.Range("D5")
    .Value = ""
    .Font.Underline = xlUnderlineStyleSingle
End With

With ws.Range("C6")
    .Value = "Shipping Code:"
    .Font.Size = 12
    .Font.Bold = True
End With

With ws.Range("D6")
    .Value = ""
    .Font.Underline = xlUnderlineStyleSingle
End With
```

```
' Address and Receiver fields
With ws.Range("B27")
    .Value = "Address:"
    .Font.Size = 12
    .Font.Bold = True
End With

With ws.Range("C27:F27")
    .Merge
    .Value = ""
    .Borders(xlEdgeBottom).LineStyle = xlContinuous
End With

With ws.Range("B30")
    .Value = "Receiver:"
    .Font.Size = 12
    .Font.Bold = True
End With

With ws.Range("C30:E30")
    .Merge
    .Value = ""
    .Borders(xlEdgeBottom).LineStyle = xlContinuous
End With

' Header row styling (C8:F8)
With ws.Range("C8:F8")
    .Font.Bold = True
    .Interior.Color = RGB(230, 240, 255)
    .Borders.LineStyle = xlContinuous
End With

' Product table borders
If outputRow > 9 Then
    With ws.Range("C8:F" & (outputRow - 1))
        .Borders.LineStyle = xlContinuous
    End With
End If

' Total amount row
With ws.Range("E" & totalRow)
    .Value = "Total Amount"
    .Font.Bold = True
    .Interior.Color = RGB(255, 255, 200)
End With

With ws.Range("F" & totalRow)
    .Formula = "=SUM(F9:F" & (totalRow - 1) & ")"
    .Font.Bold = True
    .Interior.Color = RGB(255, 255, 200)
    .NumberFormat = "#,##0.00"
End With

' AutoFit
ws.Columns("C:F").AutoFit
```

5. Print Area and Page Setup

- Sets the print area to cover the entire purchase order form.
- Configures page setup for A4 size, portrait orientation, and fitting to one page wide.
- Adjusts margins, header/footer spacing, and centers the printout horizontally.
- Disables printing of gridlines and headings for a clean print.

```
' --- Print Area Setup for A4 ---
lastRow = totalRow
lastCol = ws.Range("F1").Column

ws.PageSetup.PrintArea = ws.Range(ws.Cells(1, "A"), ws.Cells(lastRow + 15, lastCol + 2)).Addre

With ws.PageSetup
    .Orientation = xlPortrait
    .PaperSize = xlPaperA4
    .Zoom = False
    .FitToPagesWide = 1
    .FitToPagesTall = False
    .LeftMargin = Application.InchesToPoints(0.5)
    .RightMargin = Application.InchesToPoints(0.5)
    .TopMargin = Application.InchesToPoints(0.75)
    .BottomMargin = Application.InchesToPoints(0.75)
    .HeaderMargin = Application.InchesToPoints(0.3)
    .FooterMargin = Application.InchesToPoints(0.3)
    .CenterHorizontally = True
    .CenterVertically = False
    .PrintGridlines = False
    .PrintHeadings = False
End With
```

Print All Purchase Orders

This Excel VBA macro is designed to automate the printing of purchase order files. It reads all .xlsx files in a selected folder and prints the first worksheet of each, streamlining batch printing for physical distribution or documentation.

- Prompts the user to select a folder that contains the purchase order Excel files.
- Verifies that the selected folder exists; exits if not found.
- Turns off screen updating and display alerts to improve performance and avoid prompts during execution.
- Loops through each .xlsx file in the selected folder:
    - Opens the workbook as **read-only**.
    - Attempts to print the **first worksheet**.
    - Closes the workbook without saving changes.
- Restores screen updating and alert settings after all files are processed.
- Displays a message box confirming successful printing of all purchase orders.

```
Dim folderPath As String
Dim fileName As String
Dim wb As Workbook

' --- Ask user to pick the folder only once ---
With Application.FileDialog(msoFileDialogFolderPicker)
    .Title = "Select the folder with purchase order files"
    If .Show <> -1 Then Exit Sub ' User cancelled
    folderPath = .SelectedItems(1) & "\"
End With

' --- Check folder exists ---
If Dir(folderPath, vbDirectory) = "" Then
    MsgBox "Selected folder does not exist.", vbExclamation
    Exit Sub
End If

' --- Turn off flicker and alerts ---
Application.ScreenUpdating = False
Application.DisplayAlerts = False

' --- Loop through all .xlsx files in the folder ---
fileName = Dir(folderPath & "*.xlsx")
Do While fileName <> ""

    Set wb = Workbooks.Open(folderPath & fileName, ReadOnly:=True)

    On Error Resume Next
    wb.Sheets(1).PrintOut ' Print first sheet
    On Error GoTo 0

    wb.Close SaveChanges:=False
    fileName = Dir ' Get next file
Loop

' --- Restore settings ---
Application.DisplayAlerts = True
Application.ScreenUpdating = True

MsgBox "All purchase orders have been printed successfully.", vbInformation
```

Convert All Excel Files to PDF

This Excel VBA macro converts purchase order workbooks into PDF format. It processes all .xlsx files in a selected folder, exporting the first worksheet of each file as a PDF and saving them into a dedicated PDF_Output folder for easy sharing or archiving.

- Prompts the user to select a folder containing Excel .xlsx files.
- Constructs a subfolder named **PDF_Output** within the selected folder to store the converted PDF files.
- Uses FileSystemObject to check if the PDF_Output folder exists; creates it if it doesn't.
- Turns off screen updating and alerts for smoother and silent execution.
- Loops through all .xlsx files in the selected folder:
    - Opens each workbook as **read-only**.
    - Attempts to export the **first worksheet** as a PDF using ExportAsFixedFormat.
    - Saves the PDF with the same name as the Excel file (but with .pdf extension) in the PDF_Output folder.
    - Closes the workbook without saving changes.
- Restores application display settings after processing all files.
- Displays a message box indicating successful conversion and the location of the saved PDFs.

```
Dim folderPath As String
Dim pdfFolderPath As String
Dim fileName As String
Dim wb As Workbook
Dim fso As Object

' Ask user to select the folder containing Excel files
With Application.FileDialog(msoFileDialogFolderPicker)
    .Title = "Select the folder containing Excel files"
    If .Show <> -1 Then Exit Sub ' User cancelled
    folderPath = .SelectedItems(1) & "\"
End With

' Create FileSystemObject to create folder if needed
Set fso = CreateObject("Scripting.FileSystemObject")
pdfFolderPath = folderPath & "PDF_Output\"

If Not fso.FolderExists(pdfFolderPath) Then
    fso.CreateFolder pdfFolderPath
End If

Application.ScreenUpdating = False
Application.DisplayAlerts = False

' Loop through all .xlsx files in the folder
fileName = Dir(folderPath & "*.xlsx")

Do While fileName <> ""
    Set wb = Workbooks.Open(folderPath & fileName, ReadOnly:=True)

    ' Export first worksheet as PDF
    On Error Resume Next
    wb.Sheets(1).ExportAsFixedFormat Type:=xlTypePDF, _
        fileName:=pdfFolderPath & Left(fileName, Len(fileName) - 5) & ".pdf
        Quality:=xlQualityStandard, IncludeDocProperties:=True, _
        IgnorePrintAreas:=False, OpenAfterPublish:=False
    On Error GoTo 0

    wb.Close SaveChanges:=False
    fileName = Dir ' next file
Loop

Application.DisplayAlerts = True
Application.ScreenUpdating = True

MsgBox "Conversion complete! PDFs saved in:" & vbCrLf & pdfFolderPath, vbIn
```