Video Link (Current) :

Video Link (Previous) :

It's important to note that this code relies on specific worksheets, labels, list boxes, and folder paths. Make sure these elements exist and are correctly named in your Excel workbook for this code to work as expected. Additionally, the code uses Microsoft Word for generating reminder letters, so you need to have Word installed on your system for that functionality to work.

Code Analysis : (SRSalesProdPopulate)

```
Private Sub SRSalesProdPopulate()
    Dim i As Long
    Dim ProductSales As Double
    Dim ProductID As String
```

```
    Dim ProductName As Variant
    Dim ProductValue As Double

    Set wsData = ThisWorkbook.Sheets("Data")
    Set wsProductMaster = ThisWorkbook.Sheets("Product_Master")

    ' Clear the ProductListBox or other controls where you displayed product sales details
    ProductListBox.Clear

    ' Dictionary to store product-wise total sales
    Dim ProductSalesDict As Object
    Set ProductSalesDict = CreateObject("Scripting.Dictionary")

    ' Loop through the "Data" sheet to calculate product-wise total sales
    For i = 2 To wsData.Cells(wsData.Rows.Count, "A").End(xlUp).row
        If wsData.Cells(i, "J").value = SRIDLbl.Caption Then
            'If wsData.Cells(i, "I").value = "Y" Then
                ' Retrieve the Product ID for the sale
                ProductID = wsData.Cells(i, "G").value

                ' Lookup the Product details from the Product Master sheet
                Dim productRow As Long
                On Error Resume Next
                productRow = Application.WorksheetFunction.Match(ProductID, wsProductMaster.Range("A:A"), 0)
```

```
            On Error GoTo 0

            If productRow > 0 Then
                ProductName = wsProductMaster.Cells(productRow,
"B").value ' Product Name
                ProductValue = wsData.Cells(i, "E").value ' Product Value
            Else
                ProductName = "N/A" ' Product not found
                ProductValue = 0
            End If

            ' Update the product-wise total sales in the dictionary
            If Not ProductSalesDict.Exists(ProductName) Then
                ProductSalesDict(ProductName) = 0
            End If
            ProductSalesDict(ProductName) =
ProductSalesDict(ProductName) + ProductValue

            ' Calculate and display total sales and other values as needed
            ' ...
        'End If
    End If
Next i

' Display the product-wise total sales summary
Dim summaryText As String
```

```
    summaryText = "  Product-wise Total Sales for " & SRNameLbl.Caption
& ":" & vbCrLf & vbCrLf

    For Each ProductName In ProductSalesDict.Keys

        summaryText = summaryText & "  " & ProductName & " : Rs." &
Format(ProductSalesDict(ProductName), "#0.00") & vbCrLf

    Next ProductName


    ' Display the summary in the ProductSummaryLabel or another control

    ProductSummaryLabel.Caption = summaryText

End Sub
```

This VBA code snippet appears to be part of a larger Excel macro designed to populate a summary of product-wise sales for a particular Sales Representative (SR). Let's break down how this code works:

Private Sub SRSalesProdPopulate()

This line defines the start of a VBA subroutine named SRSalesProdPopulate. Subroutines in VBA are blocks of code that can be executed when called.

Dim i As Long

Dim ProductSales As Double

Dim ProductID As String

Dim ProductName As Variant

Dim ProductValue As Double

These lines declare several variables that will be used in the code:

i is declared as a Long, which is typically used for loop counters or row numbers.

ProductSales and ProductValue are declared as Double, which is used for storing numeric values with decimal points.

ProductID is declared as a String, which is used for storing text.

ProductName is declared as a Variant, which can store different data types (text, numbers, etc.).

Set wsData = ThisWorkbook.Sheets("Data")

Set wsProductMaster = ThisWorkbook.Sheets("Product_Master")

These lines set two worksheet objects, wsData and wsProductMaster, to refer to specific worksheets in the current workbook. This allows the code to work with data on these sheets.

ProductListBox.Clear

This line clears the ProductListBox control. It's preparing the control to display new data. Any existing content in the ProductListBox is removed.

Dim ProductSalesDict As Object

Set ProductSalesDict = CreateObject("Scripting.Dictionary")

These lines create a dictionary object named ProductSalesDict. This dictionary is used to store product-wise total sales. A dictionary is a data structure that associates keys (in this case, product names) with values (total sales).

****

What is Scripting Dictionary in VBA ?

A Scripting Dictionary in VBA (Visual Basic for Applications) is a powerful data structure that allows you to store and manipulate data in the form of key-

value pairs. It's a versatile tool for working with collections of data, and it offers several advantages:

Key-Value Storage: Each item in a Scripting Dictionary is a pair of a unique key and its associated value. You use the key to access the corresponding value. This makes it easy to retrieve data based on specific criteria.

Fast Lookup: Scripting Dictionaries are optimized for quick lookups. They can efficiently retrieve values associated with a key, even when working with large datasets.

Dynamic Sizing: Dictionaries automatically adjust their size as you add or remove items, so you don't need to specify the number of items in advance.

Flexibility: Values in a Scripting Dictionary can be of various data types, including numbers, text, objects, or even other dictionaries. This flexibility allows you to handle diverse datasets.

Here's a breakdown of how Scripting Dictionaries work:

Creating a Dictionary: To create a Scripting Dictionary, you typically use the CreateObject function, as shown in your code:

```
Set ProductSalesDict = CreateObject("Scripting.Dictionary")
```

This line creates a new instance of a Scripting Dictionary and assigns it to the ProductSalesDict variable.

Adding Items: You can add key-value pairs to the dictionary using the Add method:

```
ProductSalesDict.Add Key, Value
```

Key: The unique identifier for the item.

Value: The data associated with the key.

Accessing Items: To retrieve a value from the dictionary, you use the key:

```
Value = ProductSalesDict(Key)
```

Key: The key associated with the value you want to retrieve.

Value: The data associated with the key.

Checking if a Key Exists: You can check if a key exists in the dictionary using the Exists method:

If ProductSalesDict.Exists(Key) Then ' Key exists in the dictionary End If

Removing Items: To remove an item from the dictionary, you use the Remove method:

ProductSalesDict.Remove Key

Iterating Through Keys: You can loop through all the keys in the dictionary using a For Each loop:

For Each Key In ProductSalesDict.Keys ' Access each key and its associated value Value = ProductSalesDict(Key) Next Key

In your code snippet, ProductSalesDict is used to accumulate and store product-wise sales data for later display. It allows you to organize this data efficiently, making it easier to calculate and present summaries or reports.

****

For i = 2 To wsData.Cells(wsData.Rows.Count, "A").End(xlUp).Row

This line starts a loop that iterates through rows in the "Data" worksheet, starting from row 2 and continuing until it reaches the last row with data in column A.

If wsData.Cells(i, "J").Value = SRIDLbl.Caption Then

This line checks if the value in column J of the current row matches the caption of an object named SRIDLbl. This appears to be a conditional check to determine if the row corresponds to the selected Sales Representative.

ProductID = wsData.Cells(i, "G").Value

This line retrieves the product ID from column G of the current row.

Dim productRow As Long

On Error Resume Next

productRow = Application.WorksheetFunction.Match(ProductID, wsProductMaster.Range("A:A"), 0)

On Error GoTo 0

These lines attempt to find the ProductID in column A of the "Product_Master" worksheet using the Match function. If a match is found, the row number is stored in the productRow variable. If not, productRow remains 0.

If productRow > 0 Then

ProductName = wsProductMaster.Cells(productRow, "B").Value ProductValue = wsData.Cells(i, "E").Value Else ProductName = "N/A" ProductValue = 0

End If

This block of code checks if a matching product was found in the "Product_Master" worksheet. If a match is found (productRow > 0), it retrieves the product name from column B and the product value from column E. If no match is found, it sets ProductName to "N/A" and ProductValue to 0.

If Not ProductSalesDict.Exists(ProductName) Then

ProductSalesDict(ProductName) = 0

End If

ProductSalesDict(ProductName) = ProductSalesDict(ProductName) + ProductValue

These lines update the ProductSalesDict dictionary with the product name as the key and the product's cumulative sales as the value. If the product name doesn't exist in the dictionary yet, it initializes it with a value of 0 before adding the current product's sales.

Dim summaryText As String

summaryText = " Product-wise Total Sales for " & SRNameLbl.Caption & ":" & vbCrLf & vbCrLf

For Each ProductName In ProductSalesDict.Keys summaryText = summaryText & " " & ProductName & " : Rs." & Format(ProductSalesDict(ProductName), "#0.00") & vbCrLf

Next ProductName

These lines create a summary text that will be displayed in the ProductSummaryLabel or another control. It iterates through the keys (product names) in the ProductSalesDict dictionary and formats the summary text to include product names and their respective total sales.

ProductSummaryLabel.Caption = summaryText

This line sets the caption of the ProductSummaryLabel (or another control) to the summaryText, effectively displaying the product-wise total sales summary.

In summary, this code is part of a larger VBA macro designed to populate a summary of product-wise sales for a selected Sales Representative based on data in the "Data" and "Product_Master" worksheets. It uses a dictionary to aggregate and store the data before displaying it in a user interface element (likely a label or text box).

Code Analysis : (SRSalesProdPopulate1)

Gautam Banerjee

“Helping beginners learn something new is a great way to share your knowledge and make a positive impact”.

Email: gincom1@yahoo.com

If you have any queries, please visit our "Contact Us" page.

Thank You. See you again!

PLEASE DONATE

Gautam Banerjee

Age: 63

Pay by UPI

9748327614

Code Analysis : (SRSalesProdPopulate1)

```
Private Sub SRSalesProdPopulate1()
    Dim i As Long
    Dim ProductSales As Double
    Dim ProductID As String
    Dim ProductNamm As String
```

```
    Dim ProductValue As Double
    Dim TotDueValue As Double

    Set wsData = ThisWorkbook.Sheets("Data")
    Set wsProductMaster = ThisWorkbook.Sheets("Product_Master")

    ' Clear the ProductListBox or other controls where you want to display product sales
    ProductListBox.Clear

    ' Loop through the "Data" sheet to calculate and display product details
    For i = 2 To wsData.Cells(wsData.Rows.Count, "A").End(xlUp).row
        If wsData.Cells(i, "J").value = SRIDLbl.Caption Then
            If wsData.Cells(i, "I").value = "N" Then
                ' Retrieve the Product ID for the sale
                ProductID = wsData.Cells(i, "G").value

                ' Lookup the Product details from the Product Master sheet
                Dim productRow As Long
                On Error Resume Next
                productRow = Application.WorksheetFunction.Match(ProductID, wsProductMaster.Range("A:A"), 0)
                On Error GoTo 0

                If productRow > 0 Then
```

```
                ProductName = wsProductMaster.Cells(productRow, "B").value ' Product Name
                ProductValue = wsData.Cells(i, "E").value ' Product Value
                TotDueValue = TotDueValue + ProductValue
            Else
                ProductName = "N/A" ' Product not found
                ProductValue = 0
            End If

            ' Add the Product Name and Value to the ProductListBox
            ProductListBox.AddItem " Product Name : " & ProductName & " | Product Value: Rs." & Format(ProductValue, "#0.00")

            ' Calculate and display total sales and other values as needed
            ' ...
        End If
    End If
Next i

 TotProdDues.Caption = "Following Products are unpaid as on " & Format(Date, "dd-mmm-yyyy") & "  Rs. " & Format(TotDueValue, "#0.00")
End Sub
```

This VBA code snippet is part of a procedure named SRSalesProdPopulate1. It appears to be designed to populate a list (possibly a list box) with product sales details for a specific Sales Representative (SR) and calculate some summary information. Let's break down how this code works:

Private Sub SRSalesProdPopulate1()

This line defines the start of a VBA subroutine named SRSalesProdPopulate1. Subroutines in VBA are blocks of code that can be executed when called.

Dim i As Long

Dim ProductSales As Double

Dim ProductID As String

Dim ProductName As String

Dim ProductValue As Double

Dim TotDueValue As Double

These lines declare several variables that will be used in the code:

i is declared as a Long, typically used for loop counters or row numbers.

ProductSales and ProductValue are declared as Double, used for storing numeric values with decimal points.

ProductID is declared as a String, used for storing text.

ProductName is declared as a String, used for storing product names.

TotDueValue is declared as a Double, used to accumulate the total due value for unpaid products.

Set wsData = ThisWorkbook.Sheets("Data")

Set wsProductMaster = ThisWorkbook.Sheets("Product_Master")

These lines set two worksheet objects, wsData and wsProductMaster, to refer to specific worksheets in the current workbook. This allows the code to work with data on these sheets.

ProductListBox.Clear

This line clears the ProductListBox control or any other controls where product sales details will be displayed. It ensures that the control is empty before adding new data.

For i = 2 To wsData.Cells(wsData.Rows.Count, "A").End(xlUp).Row

This line starts a loop that iterates through rows in the "Data" worksheet, starting from row 2 and continuing until it reaches the last row with data in column A.

If wsData.Cells(i, "J").Value = SRIDLbl.Caption Then

This line checks if the value in column J of the current row matches the caption of an object named SRIDLbl. This appears to be a conditional check to determine if the row corresponds to the selected Sales Representative.

If wsData.Cells(i, "I").Value = "N" Then

This line checks if the value in column I of the current row is equal to "N". This condition is used to identify unpaid products.

ProductID = wsData.Cells(i, "G").Value

This line retrieves the product ID from column G of the current row.

Dim productRow As Long

On Error Resume Next

productRow = Application.WorksheetFunction.Match(ProductID, wsProductMaster.Range("A:A"), 0)

On Error GoTo 0

These lines attempt to find the ProductID in column A of the "Product_Master" worksheet using the Match function. If a match is found, the row number is stored in the productRow variable. If not, productRow remains 0.

If productRow > 0 Then

ProductName = wsProductMaster.Cells(productRow, "B").Value

ProductValue = wsData.Cells(i, "E").Value

TotDueValue = TotDueValue + ProductValue Else ProductName = "N/A"

ProductValue = 0

End If

This block of code checks if a matching product was found in the "Product_Master" worksheet. If a match is found (productRow > 0), it retrieves the product name from column B and the product value from column E. If no match is found, it sets ProductName to "N/A" and ProductValue to 0.

ProductListBox.AddItem " Product Name : " & ProductName & " | Product Value: Rs." & Format(ProductValue, "#0.00")

This line adds an item to the ProductListBox. It displays the product name and value in a specific format.

TotProdDues.Caption = "Following Products are unpaid as on " & Format(Date, "dd-mmm-yyyy") & " Rs. " & Format(TotDueValue, "#0.00")

This line sets the caption of an object named TotProdDues. It appears to display a message indicating unpaid products along with the total due value for these products, formatted with the date.

```
Private Sub SRNameList_Click()
   Dim selectedSRRow As Long
    selectedSRRow = SRNameList.ListIndex + 1

    ' Get SRID and SRName based on the selected row
    SRID = wsSR.Cells(selectedSRRow + 1, 1).value
    SRName = wsSR.Cells(selectedSRRow + 1, 2).value

    ' Update labels with SR information
    SRIDLbl.Caption = SRID
    SRNameLbl.Caption = SRName

    ' Create the full path to the picture based on the SR name and folder
    picturePath = "D:\VBAExcel\" & SRName & ".jpg" ' Adjust the folder path
    If Dir(picturePath) <> "" Then ' Check if the picture file exists
        SRImage.Picture = LoadPicture(picturePath)
    Else
        SRImage.Picture = LoadPicture("") ' Clear the image if no picture found
    End If

    ' Populate SR sales information including product sales
    SRSalesPopulate
    SRSalesProdPopulate
    SRSalesProdPopulate1
```

End Sub

This VBA code is part of a procedure named SRNameList_Click, which appears to be associated with a user interface event in Excel, likely a button click or list selection. It is used to perform various actions when a Sales Representative (SR) is selected from a list.

Here's a breakdown of how this code works:

Private Sub SRNameList_Click()

This line defines the start of a VBA subroutine named SRNameList_Click. Subroutines in VBA are blocks of code that can be executed when an associated event occurs, in this case, when a user clicks on an item in a list named SRNameList.

Dim selectedSRRow As Long

selectedSRRow = SRNameList.ListIndex + 1

These lines declare a variable selectedSRRow as a Long and assign it the value of the selected index in the SRNameList plus 1. The ListIndex property indicates the index of the selected item in the list.

' Get SRID and SRName based on the selected row SRID = wsSR.Cells(selectedSRRow + 1, 1).Value SRName = wsSR.Cells(selectedSRRow + 1, 2).Value

These lines retrieve the Sales Representative's ID (SRID) and name (SRName) from the corresponding columns in the wsSR worksheet based on the selected row. The +1 is used because Excel row and column indices are 1-based, while the list index is 0-based.

' Update labels with SR information SRIDLbl.Caption = SRID

SRNameLbl.Caption = SRName

These lines update labels (SRIDLbl and SRNameLbl) with the Sales Representative's ID and name, respectively, based on the values retrieved in the previous step. This likely provides a visual indication of the selected Sales Representative.

' Create the full path to the picture based on the SR name and folder
picturePath = "D:\VBAExcel\" & SRName & ".jpg"

' Adjust the folder path

If Dir(picturePath) <> "" Then

' Check if the picture file exists SRImage.Picture =
LoadPicture(picturePath) Else SRImage.Picture = LoadPicture("")

' Clear the image if no picture found End If

These lines construct a file path (picturePath) based on the Sales Representative's name and a folder path. It assumes that pictures of Sales Representatives are stored as JPG files in a specific folder. It then checks if the picture file exists using the Dir function. If the file exists, it loads the picture into an object named SRImage using the LoadPicture function. If no picture is found, it clears the image.

' Populate SR sales information including product sales SRSalesPopulate

SRSalesProdPopulate

SRSalesProdPopulate1

These lines call three other procedures (SRSalesPopulate, SRSalesProdPopulate, and SRSalesProdPopulate1) to populate Sales Representative sales information, including product sales. These procedures are likely responsible for gathering and displaying data related to the selected Sales Representative.

In summary, the SRNameList_Click subroutine is triggered when a Sales Representative is selected from a list. It updates labels with the Sales Representative's information, loads a picture of the Sales Representative (if available), and calls other procedures to populate sales-related data. This

code is part of a user interface designed to interact with and display information about Sales Representatives in an Excel workbook.

Gautam Banerjee

“Helping beginners learn something new is a great way to share your knowledge and make a positive impact”.

Email: gincom1@yahoo.com

If you have any queries, please visit our "Contact Us" page.

Thank You. See you again!

PLEASE DONATE

Gautam Banerjee

Age: 63

Pay by UPI

9748327614

Summary :

The above VBA code appears to be part of a larger Excel workbook with user forms and procedures to manage Sales Representatives (SRs), their sales

data, and generate reminder letters for SRs with outstanding dues. Here's a breakdown of the key components:

ListBoxSRPopulate Subroutine:

This subroutine populates a list box (likely named SRNameList) with the names of Sales Representatives (SRs) from a worksheet called "SR_Master." It retrieves the SR names from the second column of the "SR_Master" sheet and adds them to the list box.

SRNameList_Click Subroutine:

This subroutine is triggered when a user clicks on an item in the SRNameList list box.

It retrieves the selected SR's ID and name based on the clicked item and updates labels (SRIDLbl and SRNameLbl) with this information.

It attempts to load an image of the selected SR based on their name from a specific folder path.

It calls three other procedures (SRSalesPopulate, SRSalesProdPopulate, and SRSalesProdPopulate1) to populate sales-related data.

SRSalesPopulate Subroutine:

This subroutine calculates and displays SR sales information.

It calculates total sales, total paid, and total dues for the selected SR based on data in a worksheet named "Data." The results are displayed in labels (SalesValueLbl, Label1, and Label2).

SRSalesProdPopulate and SRSalesProdPopulate1 Subroutines:

These subroutines calculate and display product-wise sales for the selected SR based on data in the "Data" sheet and a "Product_Master" sheet.

SRSalesProdPopulate seems to handle paid sales, while SRSalesProdPopulate1 handles unpaid sales. Both populate a list box (likely named ProductListBox) with product details.

GenerateReminderLetters Subroutine:

This subroutine generates reminder letters for SRs with outstanding dues.

It uses Microsoft Word (if installed) to create individualized reminder letters for each SR.

Data for the letters is sourced from worksheets "Data," "SR_Master," and "Customer_Master."

The letters are based on a template ("SRDueLetterwithTable.docx") and include details of outstanding dues for each SR.

The letters are saved in a folder called "Reminder_Letters."

UserForm_Initialize Subroutine:

This subroutine is executed when the user form initializes.

It calls ListBoxSRPopulate to populate the SRNameList list box with SR names.

It's important to note that this code relies on specific worksheets, labels, list boxes, and folder paths. Make sure these elements exist and are correctly named in your Excel workbook for this code to work as expected. Additionally, the code uses Microsoft Word for generating reminder letters, so you need to have Word installed on your system for that functionality to work.