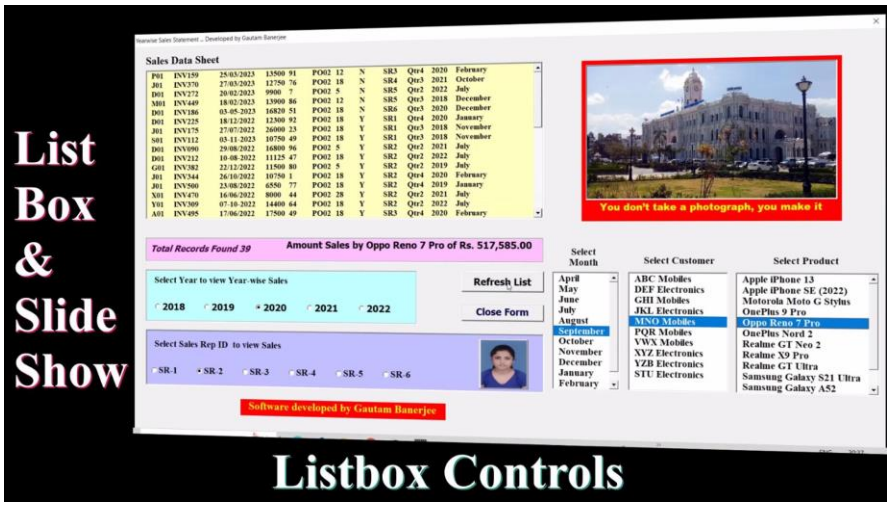

Video Link : https://youtu.be/eVcUatKVG1o

Designed to perform various functions related (List Box) to data analysis and presentation.

The user interface components that are being created or utilized in a VBA userform or similar context. Here's a breakdown of the components mentioned:

2 Listboxes for showing Data: These are likely Listbox controls used to display data. You have two of them, which can be used to show different sets of data.

3 Listboxes for Month Name, Customer Name, and Product Name: These three Listboxes are probably used for selecting or displaying month names, customer names, and product names, respectively.

Some Labels: Labels are used to display text or descriptions on the userform. They provide information to the user.

Few Option Buttons inside 2 Frames: Option Buttons (also known as Radio Buttons) are used for making selections within a set of mutually exclusive options. Placing them inside Frames can help organize them visually.

2 Command Buttons: Command Buttons are typically used for triggering actions, such as submitting a form or running a specific function.

Image Control: This is likely used to display images, such as pictures or icons.

In summary, a user interface layout that includes various controls like Listboxes, Labels, Option Buttons, Command Buttons, and an Image Control, all organized within Frames. These components collectively make up the user interface of the application or userform and allow users to interact with and view data.

The key aspects related to customizing and working with controls in a user interface, as well as the requirement for specific Excel worksheets. Here's a breakdown:

Changing Control Properties: You can modify various aspects of controls, including:

Name of the Controls: This refers to the unique identifier for each control. Changing control names can make it easier to reference them in code.

Captions: Captions are the text displayed on controls like Labels and Command Buttons. You can customize this text to make your interface user-friendly.

Font Formatting: You can adjust the font type, size, color, and other formatting properties for text-based controls like Labels.

Background Color or Transparency: Depending on the control type, you can change the background color or even make it transparent to blend with the form.

Special Effects: Some controls might support special visual effects, which can enhance the user interface.

Excel Worksheets: The need for specific Excel worksheets, including:

Customer Master: This worksheet likely contains data related to customers.

Product Master: This worksheet might store information about products.

Sales Rep Master: This could be a worksheet with details about sales representatives.

Sales Data Sheet: This worksheet probably holds the primary sales data that the application or userform interacts with.

In a VBA (Visual Basic for Applications) context, these worksheets would be used for data storage and retrieval, and the controls mentioned in the first part of the sentence would be used to create a user-friendly interface for interacting with this data.

Overall, the outlines the customization possibilities for user interface controls and highlights the importance of specific Excel worksheets in the context of the application or userform.

Plan for Coding: Code appears to be VBA (Visual Basic for Applications) code written for use in Microsoft Excel. This code is associated with a userform in Excel and is designed to perform various functions related to data analysis and presentation. Here's a breakdown of some of the controls and their purposes:

01. Dim Statements: These lines declare variables that will be used throughout the code to store data or references to objects. For example, Dim SalesYear As String declares a variable named SalesYear to store a string.

02. Private Sub: These are event handler procedures. They are triggered when specific events occur, such as clicking a button or initializing a userform. For example, Private Sub CmdRefresh_Click() is executed when a button with the name CmdRefresh is clicked.

03. ListBox: ListBox controls are used to display a list of items from which users can make selections. In this code, ListBox2_Click(), ListBox3_Click(), and other similar procedures handle events when items in ListBoxes are clicked.

04. UserForm_Initialize(): This is an event handler that runs when the userform is initialized. It's used to populate ListBoxes and perform other setup tasks when the userform is opened.

05. Function Contains(): This is a custom function used to check if an item exists in a collection. Collections are used to store and manage lists of items in VBA.

06. Select Case Statement: The Select Case statement is used to evaluate an expression against a list of possible values. In this code, it's used to change the displayed image (ImageProgress.Picture) based on the value of x.

07. Image: Images are loaded into an Image control (ImageProgress) to display pictures dynamically based on user actions.

08. Setting Values: Various parts of the code involve setting and updating values of controls, labels, and variables based on user interactions or data processing.

09. Loading Pictures: Images are loaded into an Image control (ImageProgress) based on certain conditions or events.

10. File Paths: File paths to images and other resources are specified in the code. For example, "E:\Chennai_Tours\Excel_Picture\Chennai_01.jpg" is a file path to an image.

11. Worksheet References: The code references specific worksheets within the Excel workbook to retrieve and manipulate data.

12. Collections: Collections are used to store and manage lists of items. In this code, collections like AddedItems are used to keep track of added items in ListBoxes to ensure uniqueness.

13. String Manipulation: The code includes string manipulation, such as joining strings and using string values to determine the course of action.

14. Updating Labels: Labels are updated with text to provide information to the user.

15. Event Handling: Event handlers like ListBox_Click() and Button_Click() respond to user interactions with controls on the userform.

16. Loading Data: There are comments indicating that data is being loaded or processed. This might include retrieving data from worksheets and displaying it in ListBoxes or other controls.

This code appears to be part of an Excel userform used for data analysis and presentation. It involves working with data from various worksheets, displaying images, and responding to user interactions. Please note that to fully understand and utilize this code, you would need to have a good understanding of VBA programming within Excel.

<u>Line by line Coding descriptions:</u> (Copy and Paste)

```
Dim SalesYear As String

Dim SalesSr As String

Dim NameofSR As String
```

```
Dim MonthName As String
Dim CustName As String
Dim ProdSales As String
Dim ProdName As String
Dim ProductDictionary As Object
Dim SelectedProductID As String
Dim ItemListClick As String
Dim wsItem As Worksheet
Dim WsItemLastRow As Long
Dim WsColRef As String
```

These lines of code are declaring variables in VBA (Visual Basic for Applications), each with a specific data type and purpose. Let's break down what each of these declarations means:

Dim SalesYear As String: This declares a variable named SalesYear with a data type of String. This variable is likely intended to store a year related to sales data.

Dim SalesSr As String: Similar to the previous line, this declares a variable named SalesSr as a String. It probably stores a sales-related identifier or code.

Dim NameofSR As String: This declares a variable called NameofSR as a String. It appears to be intended to store the name of a sales representative.

Dim MonthName As String: MonthName is declared as a String. It is likely used to store the name of a month.

Dim CustName As String: CustName is another String variable. It probably stores the name of a customer.

Dim ProdSales As String: This variable, named ProdSales, is declared as a String. It may store information related to product sales.

Dim ProdName As String: ProdName is declared as a String. It is likely intended to store the name of a product.

Dim ProductDictionary As Object: This declares a variable called ProductDictionary as an Object. In VBA, an Object can represent various types of objects, including collections and dictionaries. This variable is likely used to store data in a dictionary-like structure.

Dim SelectedProductID As String: SelectedProductID is declared as a String. It is probably used to store an identifier related to a selected product.

Dim ItemListClick As String: ItemListClick is a String variable. It likely stores the name or identifier of an item clicked in a list.

Dim wsItem As Worksheet: wsItem is declared as a Worksheet object. It is used to reference a specific worksheet in an Excel workbook. This variable allows you to interact with and manipulate data on that worksheet.

Dim WsItemLastRow As Long: WsItemLastRow is declared as a Long data type. It is used to store the last row number of data on a worksheet. This can be useful for looping through rows of data.

Dim WsColRef As String: WsColRef is declared as a String. It is likely used to store a reference to a specific column on a worksheet, such as "A" for column A or "B" for column B.

In summary, these variable declarations set aside memory space to store various types of data, such as text, numbers, and references to objects. They are an essential part of VBA programming as they allow you to work with and manipulate data within your code.

<u>Line by line Coding descriptions:</u> (Copy and Paste)

```
Private Sub CmdRefresh_Click()
    ListPopulate
End Sub

Private Sub LblClose_Click()
    Unload Me
    MainMenuForm.Show
End Sub

Private Sub ListBox2_Click()
    PictureLoad
End Sub
```

Here's a description of the provided VBA code:

Private Sub CmdRefresh_Click()

This is an event handler procedure that runs when a control with the name CmdRefresh is clicked. Typically, CmdRefresh is associated with a command button in a userform or worksheet.

When this button is clicked, it calls the ListPopulate procedure, which presumably populates or refreshes a list or data in the user interface.

Private Sub LblClose_Click()

This is an event handler procedure that runs when a control with the name LblClose is clicked. Normally, LblClose refers to a label that acts like a clickable button.

When this label is clicked, it executes the following actions:

Unload Me: This unloads (closes) the userform or window where this code resides.

MainMenuForm.Show: It shows a userform named MainMenuForm. This typically opens another form or menu interface.

Private Sub ListBox2_Click()

This event handler procedure runs when a ListBox control with the name ListBox2 is clicked.

When this ListBox is clicked, it calls the PictureLoad procedure. Presumably, this procedure is responsible for loading or displaying images based on the item selected in ListBox2.

These event handler procedures are part of a VBA userform or worksheet code module. They define what should happen when specific user interface elements are interacted with, such as clicking a button, label, or selecting an item in a ListBox.

<u>Line by line Coding descriptions:</u> (Copy and Paste)

```
Private Sub ListBox3_Click()
    WsColRef = "L"
    ItemListClick = ListBox3.Text
    SalesAsItemList
End Sub
```

```
Private Sub ListBox4_Click()
    WsColRef = "M"
    ItemListClick = ListBox4.Text
    SalesAsItemList
End Sub
```

Here's a description of the provided VBA code:

Private Sub ListBox3_Click()

This is an event handler procedure that runs when a ListBox control with the name ListBox3 is clicked.

When ListBox3 is clicked, it performs the following actions:

WsColRef = "L": It assigns the value "L" to the variable WsColRef. This variable is likely used to reference a specific column in an Excel worksheet.

ItemListClick = ListBox3.Text: It assigns the text of the selected item in ListBox3 to the variable ItemListClick.

SalesAsItemList: It calls the SalesAsItemList procedure. This procedure likely uses the WsColRef and ItemListClick values to perform some operation related to sales data.

Private Sub ListBox4_Click()

This is another event handler procedure that runs when a ListBox control with the name ListBox4 is clicked.

When ListBox4 is clicked, it performs similar actions to the previous code:

WsColRef = "M": It assigns the value "M" to the variable WsColRef, indicating a different column in the worksheet.

ItemListClick = ListBox4.Text: It assigns the text of the selected item in ListBox4 to the variable ItemListClick.

SalesAsItemList: It calls the SalesAsItemList procedure with these updated values.

These event handler procedures are likely used to capture user selections from ListBox3 and ListBox4 and then pass this information to the SalesAsItemList procedure for further processing. The choice of column reference ("L" or "M") appears to be determined by which ListBox is clicked.

Line by line Coding descriptions: (Copy and Paste)

```
Private Sub ListBox5_Click()
    ProdSales = ListBox5.Text

    ' Get the selected Product Name from the ListBox
    SelectedProductName = ListBox5.value

    ' Check if the Product Name exists in the dictionary
    If ProductDictionary.Exists(SelectedProductName) Then
        ' Retrieve the corresponding Product ID

        SelectedProductID = ProductDictionary(SelectedProductName)

        ' Store the Product ID in a variable or use it as needed
        ' For example, you can assign it to a module-level variable
        ' to make it accessible throughout the userform code.
        ' Example:
        ' ModuleLevelVariable = SelectedProductID

        ' Display the selected Product ID (optional)
        MsgBox "Selected Product ID: " & SelectedProductID
    Else
        ' Handle the case where the Product Name doesn't exist in the dictionary
        MsgBox "Product Name not found in the dictionary."
    End If
    'YearlyProdSales
    YearlyProdSales
```

End Sub

This VBA code is associated with a ListBox control (presumably named ListBox5) and is executed when an item in that ListBox is clicked. Here's what it does:

ProdSales = ListBox5.Text: It assigns the text of the selected item in ListBox5 to the variable ProdSales. This variable likely holds the name or description of a product.

SelectedProductName = ListBox5.Value: It retrieves and assigns the value (usually text) of the selected item in ListBox5 to the variable SelectedProductName. This variable is used to store the selected product name.

It checks if the selected product name exists in a dictionary called ProductDictionary. A dictionary is a data structure that maps keys to values.

If the selected product name exists in the dictionary:

It retrieves the corresponding Product ID associated with the selected product name and stores it in the variable SelectedProductID.

Optionally, it displays a message box with the selected Product ID.

If the selected product name is not found in the dictionary, it displays a message box indicating that the product name is not in the dictionary.

Finally, it calls a procedure named **YearlyProdSales**, which presumably performs some action related to the yearly sales of the selected product.

This code appears to be part of a user interface for selecting a product, retrieving its associated ID from a dictionary, and then triggering some action related to the selected product's sales data.

Line by line Coding descriptions: (Copy and Paste)

```
Private Sub UserForm_Initialize()
    ListPopulate
    MonthListPopulate
    CusomerNamePopulate
    ListPopulateProd
```

```
    ImageProgress.Picture = LoadPicture("E:\Chennai_Tours\Excel_Picture\Chennai_01.jpg")

End Sub
```

This code is part of the initialization process for a UserForm in VBA (Visual Basic for Applications). It is executed automatically when the UserForm is loaded or initialized. Here's what each line does:

ListPopulate: This line calls a procedure or function named ListPopulate. The purpose of this procedure is to populate a ListBox (or similar control) with data. It might load data from a worksheet or another data source and display it in a ListBox.

MonthListPopulate: This line calls a procedure or function named MonthListPopulate. This procedure is likely responsible for populating another ListBox with a list of months. It adds the names of months (e.g., "January," "February") to the ListBox.

CustomerNamePopulate: This line calls a procedure or function named CustomerNamePopulate. It is probably responsible for populating a ListBox or similar control with customer names. It retrieves the customer names from a data source and displays them in the ListBox.

ListPopulateProd: This line calls a procedure or function named ListPopulateProd. Similar to ListPopulate, this procedure likely populates a ListBox with data related to products. It could be retrieving product names from a worksheet or another data source and displaying them in the ListBox.

ImageProgress.Picture = LoadPicture("E:\Chennai_Tours\Excel_Picture\Chennai_01.jpg"): This line sets the picture property of an Image control named ImageProgress. It loads an image file located at the specified file path ("E:\Chennai_Tours\Excel_Picture\Chennai_01.jpg") and displays it within the Image control on the UserForm. This is often used to show a default or initial image when the UserForm loads.

In summary, the **UserForm_Initialize** event handler is responsible for setting up or initializing various aspects of the UserForm when it is loaded. This includes populating lists, loading images, or performing other setup tasks necessary for the functionality of the form.

<u>Line by line Coding descriptions:</u> (Copy and Paste)

```
Private Sub MonthListPopulate()
    ListBox3.AddItem "April"
```

```
    ListBox3.AddItem "May"
    ListBox3.AddItem "June"
    ListBox3.AddItem "July"
    ListBox3.AddItem "August"
    ListBox3.AddItem "September"
    ListBox3.AddItem "October"
    ListBox3.AddItem "November"
    ListBox3.AddItem "December"
    ListBox3.AddItem "January"
    ListBox3.AddItem "February"
    ListBox3.AddItem "March"
End Sub
```

The above VBA code is part of a subroutine named **MonthListPopulate**. This subroutine is responsible for populating a ListBox control, likely named ListBox3, with a list of month names. Here's a breakdown of what this code does:

Private Sub MonthListPopulate(): This line defines the start of a private subroutine named MonthListPopulate. Private subroutines can only be accessed and executed from within the module in which they are defined. They are typically used for encapsulating specific tasks.

ListBox3.AddItem "April" to ListBox3.AddItem "March": These lines add individual items to the ListBox3 control. Each line corresponds to adding a month name to the ListBox. It starts with "April" and goes through all the months of the year, ending with "March."

ListBox3.AddItem: This is a method used to add an item to a ListBox control. In this case, it adds the specified month name as an item in ListBox3.

As a result, when the **MonthListPopulate** subroutine is executed, it populates ListBox3 with the names of the months from "April" to "March." This can be helpful for allowing users to select a specific month from the ListBox in the user interface of the application.

Line by line Coding descriptions: (Copy and Paste)

```
Private Sub ListpopulateProd1()
    Dim LastRow As Long
    Dim i As Long
    Set ws = ThisWorkbook.Sheets("Product_Master")
    LastRow = ws.Cells(ws.Rows.Count, "B").End(xlUp).row
    For i = 2 To LastRow ' Assuming the data starts from row 2
        ListBox5.AddItem ws.Cells(i, 2).value
    Next i
End Sub
```

The above VBA code is part of a subroutine named **ListpopulateProd1**. This subroutine is responsible for populating a ListBox control, likely named ListBox5, with data from an Excel worksheet named "Product_Master." Here's a breakdown of what this code does:

Private Sub ListpopulateProd1(): This line defines the start of a private subroutine named ListpopulateProd1. Private subroutines can only be accessed and executed from within the module in which they are defined. They are typically used for encapsulating specific tasks.

Dim LastRow As Long: This line declares a variable named LastRow as a Long data type. This variable will be used to store the last row number with data in the worksheet.

Dim i As Long: This line declares a loop counter variable i as a Long data type. It will be used to iterate through rows in the worksheet.

Set ws = ThisWorkbook.Sheets("Product_Master"): This line sets the ws variable to refer to a specific worksheet in the current workbook. It specifies the worksheet named "Product_Master."

LastRow = ws.Cells(ws.Rows.Count, "B").End(xlUp).Row: This line calculates the last row with data in column "B" of the "Product_Master" worksheet. Here's a breakdown of this line:

ws.Rows.Count returns the total number of rows in the worksheet.

ws.Cells(ws.Rows.Count, "B") refers to the cell in the last row of column "B."

.End(xlUp) is used to navigate from the last cell in column "B" to the first non-empty cell in that column, effectively finding the last row with data in column "B."

.Row retrieves the row number of the last cell found, which is stored in the LastRow variable.

For i = 2 To LastRow: This line starts a For loop, where i will be used to iterate from 2 (assuming that the data starts from row 2) to LastRow, which is the last row with data in column "B" of the "Product_Master" worksheet.

ListBox5.AddItem ws.Cells(i, 2).Value: Inside the loop, this line adds an item to ListBox5. It retrieves the value from the cell in the i-th row and the 2nd column (column "B") of the "Product_Master" worksheet and adds it as an item to the ListBox.

As a result, when the **ListpopulateProd1** subroutine is executed, it populates ListBox5 with the data from column "B" of the "Product_Master" worksheet, starting from row 2 and continuing until the last row with data in that column. This allows users to select a product name from the ListBox in the user interface of the application.

Line by line Coding descriptions: (Copy and Paste)

```
Private Sub ListPopulateProd()
    ' Initialize the dictionary
    Set ProductDictionary = CreateObject("Scripting.Dictionary")

    ' Populate the dictionary with data from the Product_Master sheet
    Dim LastRow As Long
    Dim i As Long
    Dim ws As Worksheet
    Set ws = ThisWorkbook.Sheets("Product_Master")
    LastRow = ws.Cells(ws.Rows.Count, "B").End(xlUp).row

    For i = 2 To LastRow ' Assuming the data starts from row 2
        Dim ProductName As String
        Dim ProductID As String

        ' Assuming Product_ID is in Column A and Product_Name is in Column B
        ProductID = ws.Cells(i, 1).value
        ProductName = ws.Cells(i, 2).value
```

```
        ' Add the mapping to the dictionary
        ProductDictionary(ProductName) = ProductID

        ' Add the Product Name to the ListBox
        ListBox5.AddItem ProductName
    Next i
End Sub
```

The VBA code is part of a subroutine named **ListPopulateProd**. This subroutine serves the purpose of populating a dictionary (Scripting.Dictionary) with data from an Excel worksheet named "Product_Master" and simultaneously adding the product names from that worksheet to a ListBox, likely named ListBox5. Here's a breakdown of what this code does:

Private Sub ListPopulateProd(): This line defines the start of a private subroutine named ListPopulateProd. This subroutine initializes a dictionary and populates it with data from an Excel worksheet.

' Initialize the dictionary: This comment provides a brief description indicating that a dictionary is being initialized. Dictionaries are data structures that store key-value pairs.

Set ProductDictionary = CreateObject("Scripting.Dictionary"): This line initializes a new Scripting.Dictionary object and assigns it to the variable ProductDictionary. This dictionary will be used to store mappings between product names and their corresponding IDs.

' Populate the dictionary with data from the Product_Master sheet: This comment provides an overview of the next steps, which involve filling the dictionary with data from an Excel worksheet.

Dim LastRow As Long: This line declares a variable named LastRow as a Long data type. It will be used to store the last row number with data in the worksheet.

Dim i As Long: This line declares a loop counter variable i as a Long data type. It will be used to iterate through rows in the worksheet.

Set ws = ThisWorkbook.Sheets("Product_Master"): This line sets the ws variable to refer to a specific worksheet in the current workbook. It specifies the worksheet named "Product_Master."

LastRow = ws.Cells(ws.Rows.Count, "B").End(xlUp).Row: This line calculates the last row with data in column "B" of the "Product_Master" worksheet. It uses the .Cells property to access cells in the worksheet, .Rows.Count to get the total number of rows, and .End(xlUp) to navigate from the last cell in column "B" to the first non-empty cell in that column. Finally, .Row retrieves the row number of the last cell found, which is stored in the LastRow variable.

For i = 2 To LastRow: This line starts a For loop, where i will be used to iterate from 2 (assuming that the data starts from row 2) to LastRow, which is the last row with data in column "B" of the "Product_Master" worksheet.

Dim ProductName As String and Dim ProductID As String: These lines declare two variables, ProductName and ProductID, both as String data types. These variables will be used to temporarily store values retrieved from the worksheet.

ProductID = ws.Cells(i, 1).Value and ProductName = ws.Cells(i, 2).Value: These lines extract the values from columns "A" and "B" of the current row (i) in the "Product_Master" worksheet and store them in the respective variables.

ProductDictionary(ProductName) = ProductID: This line adds an entry to the ProductDictionary. It uses the ProductName as the key and ProductID as the corresponding value. This effectively creates a mapping between product names and their IDs in the dictionary.

ListBox5.AddItem ProductName: This line adds the ProductName to the ListBox5. This ListBox will display a list of product names to the user.

In summary, the ListPopulateProd subroutine initializes a dictionary, reads data from the "Product_Master" worksheet, creates mappings between product names and IDs, and adds the product names to a ListBox for user selection. This is a common approach when you need to display a user-friendly list of items while maintaining a mapping to more detailed data.

<u>Line by line Coding descriptions:</u> (Copy and Paste)

```
Private Sub CusomerNamePopulate()
    Dim wsCust As Worksheet
    Set wsCust = ThisWorkbook.Worksheets("Customer_Master")

    Dim LastRow As Long
    LastRow = wsCust.Cells(wsCust.Rows.Count, "A").End(xlUp).row
```

```
    Dim rng As Range
    Set rng = wsCust.Range("B2:B" & LastRow) ' Assuming data starts from A2

    ListBox4.Clear
    ListBox4.List = rng.value
End Sub
```

The above VBA code is part of a subroutine named **CusomerNamePopulate**. This subroutine is responsible for populating a ListBox, likely named ListBox4, with customer names from an Excel worksheet named "Customer_Master." Here's a breakdown of what this code does:

Private Sub CusomerNamePopulate(): This line defines the start of a private subroutine named CusomerNamePopulate. This subroutine is responsible for populating a ListBox with customer names.

Dim wsCust As Worksheet: This line declares a variable named wsCust as a reference to a worksheet. It's set to refer to the "Customer_Master" sheet in the current workbook.

Set wsCust = ThisWorkbook.Worksheets("Customer_Master"): This line assigns the "Customer_Master" worksheet to the wsCust variable, allowing us to work with that specific sheet.

Dim LastRow As Long: This line declares a variable named LastRow as a Long data type. It will be used to store the last row number with data in column A of the "Customer_Master" sheet.

LastRow = wsCust.Cells(wsCust.Rows.Count, "A").End(xlUp).Row: This line calculates the last row with data in column A of the "Customer_Master" sheet. It starts from the bottom of the worksheet, moves upward until it finds the first non-empty cell, and then retrieves the row number. The result is stored in the LastRow variable.

Dim rng As Range: This line declares a variable named rng as a Range data type. This variable will represent the range of customer names in column B of the worksheet.

Set rng = wsCust.Range("B2:B" & LastRow): This line sets the rng variable to represent the range of cells from "B2" to the last row in column B where customer names are stored. It assumes that the data starts from row 2 in column B.

ListBox4.Clear: This line clears any existing items in ListBox4. This is important if you want to refresh the ListBox with updated data.

ListBox4.List = rng.Value: This line populates ListBox4 with the values from the range rng. In other words, it loads the customer names into the ListBox for user selection.

In summary, the **CusomerNamePopulate** subroutine initializes a reference to the "Customer_Master" worksheet, determines the last row with data in column A, defines a range representing customer names in column B, clears any existing items in ListBox4, and populates ListBox4 with the customer names from the specified range, making them available for selection in the user interface.

Line by line Coding descriptions: (Copy and Paste)

```
Private Sub ListPopulate()
    Dim ws As Worksheet
    Set ws = ThisWorkbook.Worksheets("Data")

    Dim LastRow As Long
    LastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).row

    Dim rng As Range
    Set rng = ws.Range("A1:M" & LastRow) ' Assuming data starts from A2

    ListBox1.Visible = False
    ListBox2.Visible = True
    ListBox2.Clear
    ListBox2.List = rng.value

    Label2.caption = "Total Records : " & LastRow - 1

    Dim TotalY, TotalN, TotalYN
    Dim i As Integer
    Dim YN As String
```

```
    DueYN = "Y"


    For i = 2 To LastRow
        If ws.Cells(i, 9).value = DueYN Then
         TotalY = TotalY + ws.Cells(i, "E").value
        Else
        TotalN = TotalN + ws.Cells(i, "E").value
         End If
        TotalYN = TotalYN + ws.Cells(i, "E").value
        'LblDataCustID = WsData.Cells(i, "A").value
        'Label34.Caption = LblDataCustID
    Next i


    'AmtRecd.caption = "Rs. " & Format(TotalY, "###,##0.00")
    'AmtDues.caption = "Rs. " & Format(TotalN, "###,##0.00")
    TotAmt.caption = "Total Sales Rs. " & Format(TotalYN, "###,##0.00")
End Sub
```

The above VBA code is part of a subroutine named **ListPopulate**. This subroutine appears to be responsible for populating a ListBox (ListBox2), displaying some statistics in labels, and performing calculations based on data from an Excel worksheet named "Data." Let's break down what this code does step by step:

Setting Worksheet Reference: The code starts by setting a reference to the "Data" worksheet within the current workbook.

Finding the Last Row: It calculates the last row with data in column A of the "Data" sheet and stores it in the LastRow variable.

Defining the Data Range: A range variable rng is defined to represent the data range from cell A1 to the last row (assuming data starts from row 2).

Working with ListBoxes: It hides ListBox1 and shows ListBox2. Then, it clears any existing items in ListBox2 and populates it with the data from the specified range.

Label2 Caption: The caption of Label2 is set to display the total number of records minus one. The subtraction is likely because the code assumes the first row contains headers, so it's subtracting one from the total row count.

Calculating Totals: Several variables (TotalY, TotalN, and TotalYN) are declared to keep track of totals. It loops through the data rows (starting from row 2), checks a condition based on column I (9th column), and calculates totals accordingly.

Updating Labels: There are some commented lines that appear to update labels (AmtRecd, AmtDues, and TotAmt) with calculated values. These lines are currently commented out, so they won't execute. You might want to uncomment and customize them if needed.

In summary, this subroutine initializes a reference to the "Data" worksheet, populates ListBox2 with data from the worksheet, displays the total number of records in Label2, and calculates totals based on specific conditions in the data, although it currently doesn't update labels with these calculated totals.

Line by line Coding descriptions: (Copy and Paste)

```
Private Sub YearlySales()
    ' This event handler runs when an Option Button is clicked.

    ' Define variables
    Dim ws As Worksheet
    Dim LastRow As Long
    Dim ListBoxData As Object
    Dim AddedItems As Collection
    Dim TotalRecordsYear As Long
    Dim TotalAmountYear As Double

    ListBox2.Visible = False
    ListBox1.Visible = True
    ' Set the worksheet and last row
    Set ws = ThisWorkbook.Worksheets("TempData")
```

```
    LastRow = ws.Cells(ws.Rows.Count, "I").End(xlUp).row

    ' Clear the ListBox
    Set ListBoxData = Me.ListBox1
    ListBoxData.Clear

    ' Create a collection to keep track of added items
    Set AddedItems = New Collection

    TotalRecordsYear = 0
    TotalAmountYear = 0

    ' Loop through the data and populate the ListBox with unique rows where Column I is "No"
    Dim i As Long
    Dim RowData As Range

    For i = 1 To LastRow
        If ws.Cells(i, "K").value = SalesYear Then ' Assuming Column I contains "Yes" or "No"
            ' Define a fixed range for RowData that includes all columns you want to display
            Set RowData = ws.Range("A" & i & ":L" & i) ' Adjust the range as needed

            ' Convert the 2D array to a 1D array of strings
            Dim rowDataArray() As Variant
            Dim j As Long
            ReDim rowDataArray(1 To 1, 1 To RowData.Columns.Count)

            For j = 1 To RowData.Columns.Count
```

```
            rowDataArray(1, j) = RowData.Cells(1, j).value
        Next j

        ' Join the data and add it to the ListBox
        Dim ItemKey As String
        ItemKey = Join(Application.index(rowDataArray, 1, 0), vbTab)

        If Not Contains(AddedItems, ItemKey) Then
            ListBoxData.AddItem ItemKey
            AddedItems.Add ItemKey, ItemKey ' Add the item to the collection
            TotalRecordsYear = TotalRecordsYear + 1
            TotalAmountYear = TotalAmountYear + ws.Cells(i, "D").value ' Assuming Amount is in column E
        End If

    End If
  Next i

   ' Update labels with the total counts and amounts
  Label2.caption = "Total Records Found " & TotalRecordsYear
  TotAmt.caption = "Amount Sales for " & SalesYear & " of Rs. " & Format(TotalAmountYear, "###,##0.00")

End Sub
```

This subroutine, named **YearlySales**, is responsible for populating ListBox1 with unique rows of data based on certain criteria and updating labels to display total counts and amounts. Here's a description and analysis of this code:

Variable Definitions: Several variables are declared at the beginning of the subroutine. These include references to worksheets (ws), a variable to store the last row with data in

column I (LastRow), an object variable for ListBox1 (ListBoxData), a collection to keep track of added items (AddedItems), and variables to store the total number of records and total amount for the selected year (TotalRecordsYear and TotalAmountYear, respectively).

ListBox Visibility: Initially, ListBox2 is hidden (ListBox2.Visible = False), and ListBox1 is made visible (ListBox1.Visible = True). This suggests that this code is related to switching between different views in a user interface.

Clearing the ListBox: The ListBox (ListBox1) is cleared to remove any existing items before populating it with new data.

Loop Through Data: The code then loops through the data in the worksheet named "TempData" (assuming this is the data source). The loop goes from 1 to the last row (LastRow) of data.

Conditional Check: Within the loop, there is a conditional check that verifies if the value in column K for the current row matches the value stored in the variable SalesYear. This condition determines whether a row of data should be considered for inclusion in ListBox1.

Handling Unique Rows: If the condition is met, the code processes the row data. It creates a range object (RowData) that spans the columns from A to L for the current row. It then converts this row of data into a 1D array of strings (rowDataArray) and joins the array elements into a single string (ItemKey) separated by tab characters. This allows for displaying the entire row as a single item in ListBox1.

Checking for Uniqueness: Before adding the ItemKey to ListBox1, the code checks whether it already exists in the AddedItems collection. If it's not already in the collection, it's added to both ListBox1 and the AddedItems collection. This ensures that only unique rows are displayed in ListBox1.

Counting Records and Calculating Amount: As the code processes each eligible row, it increments the TotalRecordsYear count and adds the value from column D (assuming it's the amount) to TotalAmountYear. These variables are used to keep track of the total records and total amount for the selected year.

Updating Labels: Finally, the code updates two labels (Label2 and TotAmt) with the total records count and the total sales amount for the selected year. The Format function is used to format the amount with commas and decimal places.

In summary, this code is part of a user interface where data from a worksheet is displayed in a ListBox (ListBox1). It ensures that only unique rows of data, meeting specific criteria, are shown in the ListBox, and it provides summary information about the displayed data in the form of label captions.

Set AddedItems = New Collection

AddedItems is declared as a collection earlier in the code. Collections are a type of data structure in VBA that can store a collection of items. In this case, it's used to keep track of items that have already been added to ListBox1 to ensure that only unique items are displayed.

New Collection is used to create a new instance of the Collection object and assign it to the AddedItems variable. This effectively initializes the collection so that it can be used to store items.

ItemKey = Join(Application.Index(rowDataArray, 1, 0), vbTab)

rowDataArray is a 2D array that represents a row of data in the worksheet. Each element of this array corresponds to a cell in the row.

Application.Index is a function in VBA used to extract specific rows or columns from a 2D array. In this case, it's used to extract all columns (represented by 1) for a single row (0) from rowDataArray.

Join is a VBA function used to concatenate an array of strings into a single string with a specified delimiter. In this case, the delimiter is specified as vbTab, which represents a tab character.

The result of this line of code is that it takes all the values in the row represented by rowDataArray, joins them together into a single string separated by tab characters, and assigns this concatenated string to the ItemKey variable.

Putting it all together, these lines of code are used to create a unique identifier (ItemKey) for each row of data in the worksheet. This identifier is then checked against the AddedItems collection to ensure that only unique rows are added to ListBox1. If a row's ItemKey is not in the collection, it means that the row hasn't been added to the ListBox yet, and it can be added along with its ItemKey to keep track of uniqueness.

Dim rowDataArray() As Variant

This line declares a variable named rowDataArray as an array of type Variant. Variant is a versatile data type in VBA that can hold various types of data, including numbers, text, and objects. In this case, it's used to store the values from a row of data.

Dim j As Long

This line declares a variable j as a Long data type. It's a common practice to use j, i, or similar variable names as loop counters.

ReDim rowDataArray(1 To 1, 1 To RowData.Columns.Count)

ReDim is short for "Re-dimension," and it's used to change the dimensions of an array or create a new array dynamically. In this case, it's used to create a new 2D array named rowDataArray.

rowDataArray is declared as a 2D array with one row and a number of columns equal to the number of columns in the RowData range. The RowData.Columns.Count part determines the number of columns. This is done to match the size of the array to the number of columns in the data row because each column value will be stored in a separate element of this array.

So, if RowData.Columns.Count is, for example, 5 (meaning there are 5 columns in the RowData range), then rowDataArray would be created as a 2D array with dimensions (1 To 1, 1 To 5), which essentially means it can hold the values of all 5 columns in the data row.

To summarize, these lines of code prepare rowDataArray as a 2D array capable of holding the values from a single row of data. The number of columns in the array is determined dynamically based on the number of columns in the RowData range. This array will be used to temporarily store the values from a row before they are joined into a single string for display in the list box.

Line by line Coding descriptions: (Copy and Paste)

```
Private Sub YearlySRSales()
    ' This event handler runs when an Option Button is clicked.
    ' Define variables
    Dim ws As Worksheet
    Dim LastRow As Long
    Dim ListBoxData As Object
    Dim AddedItems As Collection
    Dim TotalRecordsYear As Long
    Dim TotalAmountYear As Double

    ListBox2.Visible = False
    ListBox1.Visible = True
    ' Set the worksheet and last row
```

```
    Set ws = ThisWorkbook.Worksheets("TempData")
    LastRow = ws.Cells(ws.Rows.Count, "I").End(xlUp).row

    ' Clear the ListBox
    Set ListBoxData = Me.ListBox1
    ListBoxData.Clear

    ' Create a collection to keep track of added items
    Set AddedItems = New Collection

    TotalRecordsYear = 0
    TotalAmountYear = 0

    ' Loop through the data and populate the ListBox with unique rows where Column I is "No"
    Dim i As Long
    Dim RowData As Range

    For i = 1 To LastRow
        If ws.Cells(i, "I").value = SalesSr Then ' Assuming Column I contains "Yes" or "No"
            ' Define a fixed range for RowData that includes all columns you want to display
            Set RowData = ws.Range("A" & i & ":L" & i) ' Adjust the range as needed

            ' Convert the 2D array to a 1D array of strings
            Dim rowDataArray() As Variant
            Dim j As Long
            ReDim rowDataArray(1 To 1, 1 To RowData.Columns.Count)

            For j = 1 To RowData.Columns.Count
```

```
            rowDataArray(1, j) = RowData.Cells(1, j).value
        Next j

        ' Join the data and add it to the ListBox
        Dim ItemKey As String
        ItemKey = Join(Application.index(rowDataArray, 1, 0), vbTab)

        If Not Contains(AddedItems, ItemKey) Then
            ListBoxData.AddItem ItemKey
            AddedItems.Add ItemKey, ItemKey ' Add the item to the collection
            TotalRecordsYear = TotalRecordsYear + 1
            TotalAmountYear = TotalAmountYear + ws.Cells(i, "D").value ' Assuming Amount is in column E
        End If

    End If
  Next i

   ' Update labels with the total counts and amounts
  Label2.caption = "Total Records Found " & TotalRecordsYear
  TotAmt.caption = "Amount Sales by " & NameofSR & " of Rs. " & Format(TotalAmountYear, "###,##0.00")

  Dim picturePath As String
  picturePath = "D:\VBAExcel\" & NameofSR & ".jpg" ' Adjust the folder path
  Image1.Picture = LoadPicture(picturePath)

End Sub
```

Event Handler: This code is meant to run when an Option Button is clicked. It responds to a specific user interaction.

Variables: It defines several variables, including ws for a worksheet, LastRow for the last row with data, ListBoxData to handle a list box, AddedItems as a collection to keep track of unique items, and variables for total records and amounts.

List Box Visibility: It changes the visibility of two list boxes (ListBox1 and ListBox2). ListBox2 is set to be invisible (False), and ListBox1 is set to be visible (True).

Data Processing Loop: It loops through rows of data in a worksheet ("TempData"). For each row where the value in Column I matches SalesSr, it does the following:

Defines a range (RowData) for the columns you want to display.

Converts this row's data into a 1D array of strings (rowDataArray) where each column value is separated by a tab character.

Checks if this item is already in the collection (AddedItems). If not, it adds the item to the ListBox, updates counters, and adds the item to the collection.

Label Updates: It updates the captions of two labels (Label2 and TotAmt) with information about the total records found and the total amount of sales.

Image Loading: It loads an image into an image control (Image1) based on a file path (picturePath) constructed using the NameofSR variable.

Overall, this code is processing data in a worksheet, creating a unique list of items in a list box, updating labels with summary information, and displaying an image. It's often used in Excel VBA userforms to provide dynamic user interfaces for data analysis or presentation.

Line by line Coding descriptions: (Copy and Paste)

Only Change the Excel sheet Col Reference

```
Private Sub YearlyProdSales()
    ' This event handler runs when an Option Button is clicked.

    ' Define variables
    Dim ws As Worksheet
    Dim LastRow As Long
```

```
    Dim ListBoxData As Object
    Dim AddedItems As Collection
    Dim TotalRecordsYear As Long
    Dim TotalAmountYear As Double

    ListBox2.Visible = False
    ListBox1.Visible = True
    ' Set the worksheet and last row
    Set ws = ThisWorkbook.Worksheets("TempData")
    LastRow = ws.Cells(ws.Rows.Count, "M").End(xlUp).row

    ' Clear the ListBox
    Set ListBoxData = Me.ListBox1
    ListBoxData.Clear

    ' Create a collection to keep track of added items
    Set AddedItems = New Collection

    TotalRecordsYear = 0
    TotalAmountYear = 0

    ' Loop through the data and populate the ListBox with unique rows where Column I is
"No"
    Dim i As Long
    Dim RowData As Range

    For i = 1 To LastRow
            If ws.Cells(i, "F").value = SelectedProductID Then
        Set RowData = ws.Range("A" & i & ":L" & i) ' Adjust the range as needed
```

```
            ' Convert the 2D array to a 1D array of strings
            Dim rowDataArray() As Variant
            Dim j As Long
            ReDim rowDataArray(1 To 1, 1 To RowData.Columns.Count)

            For j = 1 To RowData.Columns.Count
               rowDataArray(1, j) = RowData.Cells(1, j).value
            Next j

            ' Join the data and add it to the ListBox
            Dim ItemKey As String
            ItemKey = Join(Application.index(rowDataArray, 1, 0), vbTab)

            If Not Contains(AddedItems, ItemKey) Then
               ListBoxData.AddItem ItemKey
               AddedItems.Add ItemKey, ItemKey ' Add the item to the collection
               TotalRecordsYear = TotalRecordsYear + 1
               TotalAmountYear = TotalAmountYear + ws.Cells(i, "D").value ' Assuming Amount is in column E
            End If

         End If
      Next i

       ' Update labels with the total counts and amounts
      Label2.caption = "Total Records Found " & TotalRecordsYear
      TotAmt.caption = "Amount Sales by " & ProdSales & " of Rs. " & Format(TotalAmountYear, "###,##0.00")
```

```
End Sub


Function Contains(col As Collection, key As Variant) As Boolean
    ' Check if the collection contains a given key
    On Error Resume Next
    col.Item key
    Contains = (Err.Number = 0)
    Err.Clear
    On Error GoTo 0
End Function
```

The Contains function is a custom VBA function that is designed to check if a given key exists in a collection. Here's a breakdown of how it works:

Function Contains(col As Collection, key As Variant) As Boolean: This line defines the function named Contains. It takes two parameters: col, which is a collection, and key, which is a variant (a flexible data type in VBA that can hold various types of data, including strings, numbers, and objects). The function returns a Boolean value (True if the key is found in the collection, False if it's not found).

On Error Resume Next: This statement is used to handle runtime errors. It essentially tells VBA to continue executing the code if an error occurs instead of halting the program.

col.Item key: This line tries to access an item in the collection (col) using the provided key (key). If the key exists in the collection, this line will execute successfully; otherwise, it will generate an error.

Contains = (Err.Number = 0): After attempting to access the item, the code checks if an error occurred (Err.Number). If there was no error (meaning the key exists in the collection), it sets Contains to True. If an error occurred (meaning the key doesn't exist), it sets Contains to False.

Err.Clear: This line clears any error information that may have been generated during the execution of the code.

On Error GoTo 0: This statement resets the error handling behavior to its default state, which means that errors will once again halt the program rather than being ignored.

In summary, this Contains function allows you to check if a particular key exists in a collection without causing an error if the key is not found. It's a useful utility function for working with collections in VBA, especially when you want to avoid runtime errors related to missing keys.

Line by line Coding descriptions: (Copy and Paste)

```
Private Sub Year2018_Click()
    SalesYear = "2018"
    YearlySales
End Sub

Private Sub Year2019_Click()
    SalesYear = "2019"
    YearlySales
End Sub

Private Sub Year2020_Click()
    SalesYear = "2020"
    YearlySales
End Sub

Private Sub Year2021_Click()
    SalesYear = "2021"
    YearlySales
End Sub

Private Sub Year2022_Click()
    SalesYear = "2022"
```

```
    YearlySales
End Sub
```

These code snippets are event handlers for various buttons, most likely used in a user interface, where each button corresponds to a specific year (e.g., 2018, 2019, etc.). When a button is clicked, it sets the SalesYear variable to the corresponding year and then calls the YearlySales subroutine or function. Here's what each of these code snippets does:

Private Sub Year2018_Click(): This is an event handler for a button labeled "Year2018." When this button is clicked, it sets the SalesYear variable to "2018" and then calls the YearlySales subroutine or function.

Private Sub Year2019_Click(): Similar to the previous one, but it sets SalesYear to "2019."

Private Sub Year2020_Click(): Sets SalesYear to "2020" when the button is clicked.

Private Sub Year2021_Click(): Sets SalesYear to "2021" when the button is clicked.

Private Sub Year2022_Click(): Sets SalesYear to "2022" when the button is clicked.

These event handlers are typically used in a graphical user interface (GUI) application, such as a VBA UserForm in Microsoft Excel, to allow the user to select a specific year of interest. When a year button is clicked, it triggers an action (in this case, calling the YearlySales subroutine) that typically updates some display or performs calculations related to the selected year. The exact functionality of the YearlySales subroutine would depend on the rest of your VBA code.

Line by line Coding descriptions: (Copy and Paste)

```
Private Sub SR1_Click()
    SalesSr = "SR1"
    NameofSR = "Sandeep Sarvahi"
    YearlySRSales
End Sub

Private Sub SR2_Click()
    SalesSr = "SR2"
    NameofSR = "Poonam Dixit"
```

```
    YearlySRSales
End Sub
Private Sub SR3_Click()
    SalesSr = "SR3"
    NameofSR = "Anjali Rathore"
    YearlySRSales
End Sub
Private Sub SR4_Click()
    SalesSr = "SR4"
    NameofSR = "Rafikul Ahamed"
    YearlySRSales
End Sub
Private Sub SR5_Click()
    SalesSr = "SR5"
    NameofSR = "Manoj Dubay"
    YearlySRSales
End Sub
Private Sub SR6_Click()
    SalesSr = "SR6"
    NameofSR = "K.K.Krishnamurthy"
    YearlySRSales
End Sub
```

These code snippets appear to be event handlers for buttons or controls associated with different sales representatives (SRs). When one of these buttons is clicked, it sets the SalesSr and NameofSR variables to specific values and then calls the YearlySRSales subroutine or function. Here's what each of these code snippets does:

Private Sub SR1_Click(): This is an event handler for a button associated with SR1 (Sales Representative 1). When this button is clicked, it sets the SalesSr variable to "SR1"

and the NameofSR variable to "Sandeep Sarvahi," and then calls the YearlySRSales subroutine or function.

Private Sub SR2_Click(): Similar to the previous one, but it sets SalesSr to "SR2" and NameofSR to "Poonam Dixit."

Private Sub SR3_Click(): This one is for SR3, setting SalesSr to "SR3" and NameofSR to "Anjali Rathore."

Private Sub SR4_Click(): For SR4, setting SalesSr to "SR4" and NameofSR to "Rafikul Ahamed."

Private Sub SR5_Click(): This one is for SR5, setting SalesSr to "SR5" and NameofSR to "Manoj Dubay."

Private Sub SR6_Click(): For SR6, setting SalesSr to "SR6" and NameofSR to "K.K.Krishnamurthy."

These event handlers are typically used in a graphical user interface (GUI) application, such as a VBA UserForm in Microsoft Excel. They allow the user to select a specific sales representative, and when a sales representative button is clicked, it triggers an action (in this case, calling the YearlySRSales subroutine) that likely displays or analyzes sales data specific to the selected sales representative. The exact functionality of the YearlySRSales subroutine would depend on the rest of your VBA code.

Line by line Coding descriptions: (Copy and Paste)

Only Change the Excel sheet Col Reference

```
Private Sub SalesAsItemList()
    ' This event handler runs when an Option Button is clicked.

    ' Define variables
    'Dim ws As Worksheet
    'Dim LastRow As Long
    Dim ListBoxData As Object
    Dim AddedItems As Collection
    Dim TotalRecordsYear As Long
    Dim TotalAmountYear As Double
```

```
ListBox2.Visible = False
ListBox1.Visible = True
' Set the worksheet and last row
Set ws = ThisWorkbook.Worksheets("TempData")
LastRow = ws.Cells(ws.Rows.Count, "L").End(xlUp).row

' Clear the ListBox
Set ListBoxData = Me.ListBox1
ListBoxData.Clear

' Create a collection to keep track of added items
Set AddedItems = New Collection

TotalRecordsYear = 0
TotalAmountYear = 0

' Loop through the data and populate the ListBox with unique rows where Column I is "No"
Dim i As Long
Dim RowData As Range

For i = 1 To LastRow
   If ws.Cells(i, WsColRef).value = ItemListClick Then ' Assuming Column I contains "Yes" or "No"
      ' Define a fixed range for RowData that includes all columns you want to display
      Set RowData = ws.Range("A" & i & ":L" & i) ' Adjust the range as needed

      ' Convert the 2D array to a 1D array of strings
      Dim rowDataArray() As Variant
      Dim j As Long
```

```
        ReDim rowDataArray(1 To 1, 1 To RowData.Columns.Count)

        For j = 1 To RowData.Columns.Count
           rowDataArray(1, j) = RowData.Cells(1, j).value
        Next j

        ' Join the data and add it to the ListBox
        Dim ItemKey As String
        ItemKey = Join(Application.index(rowDataArray, 1, 0), vbTab)

        If Not Contains(AddedItems, ItemKey) Then
           ListBoxData.AddItem ItemKey
           AddedItems.Add ItemKey, ItemKey ' Add the item to the collection
           TotalRecordsYear = TotalRecordsYear + 1
           TotalAmountYear = TotalAmountYear + ws.Cells(i, "D").value ' Assuming Amount is in column E
        End If

      End If
   Next i

    ' Update labels with the total counts and amounts
   Label2.caption = "Total Records Found " & TotalRecordsYear
   TotAmt.caption = "Amount Sales for " & ItemListClick & " of Rs. " & Format(TotalAmountYear,
"###,##0.00")
End Sub
```

Line by line Coding descriptions: (Copy and Paste)

```
Private Sub PictureLoad()


    Dim x As Long
    For x = 1 To 220000
        DoEvents ' Allow the userform to update
        ' Check if i is a multiple of 50,000
        If x Mod 20000 = 0 Then
            ' Change the picture based on the current value of i
            Select Case x
                Case 20000
                    ImageProgress.Picture = LoadPicture("E:\Chennai_Tours\Excel_Picture\Chennai_02.jpg")
                    Frame4.BackColor = vbMagenta
                    Label10.BackColor = vbMagenta
                    Label10.ForeColor = vbBlack
                Case 40000
                    ImageProgress.Picture = LoadPicture("E:\Chennai_Tours\Excel_Picture\Chennai_03.jpg")
                    Label10.caption = "Photography is the story I fail to put into words"
                    Frame4.BackColor = vbPurple
                    Label10.BackColor = vbPurple
                    Label10.ForeColor = vbYellow
                Case 60000
                    ImageProgress.Picture = LoadPicture("E:\Chennai_Tours\Excel_Picture\Chennai_04.jpg")
                    Frame4.BackColor = vbRed
                    Label10.BackColor = vbRed
                    Label10.ForeColor = vbYellow
                Case 80000
                    ImageProgress.Picture = LoadPicture("E:\Chennai_Tours\Excel_Picture\Chennai_05.jpg")
                    Label10.caption = "Today everything exists to end in a photograph"
                    Frame4.BackColor = vbBlack
```

```
    Label10.BackColor = vbBlack
    Label10.ForeColor = vbWhite
Case 100000
    ImageProgress.Picture = LoadPicture("E:\Chennai_Tours\Excel_Picture\Chennai_06.jpg")
    Frame4.BackColor = vbWhite
    Label10.BackColor = vbWhite
    Label10.ForeColor = vbBlack
Case 120000
    ImageProgress.Picture = LoadPicture("E:\Chennai_Tours\Excel_Picture\Chennai_07.jpg")
    Label10.caption = "Photography is a love affair with life"
    Frame4.BackColor = vbRed
    Label10.BackColor = vbRed
    Label10.ForeColor = vbYellow
Case 140000
    ImageProgress.Picture = LoadPicture("E:\Chennai_Tours\Excel_Picture\Chennai_08.jpg")
    Frame4.BackColor = vbBlue
    Label10.BackColor = vbBlue
    Label10.ForeColor = vbWhite
Case 160000
    ImageProgress.Picture = LoadPicture("E:\Chennai_Tours\Excel_Picture\Chennai_09.jpg")
    Label10.caption = "The best camera is the one that’s with you"
    Frame4.BackColor = vbGreen
    Label10.BackColor = vbGreen
    Label10.ForeColor = vbBlue
Case 180000
    ImageProgress.Picture = LoadPicture("E:\Chennai_Tours\Excel_Picture\Chennai_10.jpg")
    Frame4.BackColor = vbYellow
    Label10.BackColor = vbYellow
    Label10.ForeColor = vbRed
```

```
            Case 200000
                ImageProgress.Picture = LoadPicture("E:\Chennai_Tours\Excel_Picture\Chennai_11.jpg")
                Label10.caption = "You don’t take a photograph, you make it"
                Frame4.BackColor = vbRed
                Label10.BackColor = vbRed
                Label10.ForeColor = vbYellow
            Case Else
                ' Handle other cases if necessary
        End Select
    End If
Next x

ImageProgress.Picture = LoadPicture("E:\Chennai_Tours\Excel_Picture\Chennai_01.jpg")

End Sub
```

This code defines a subroutine called PictureLoad, which seems to be used for loading and displaying a series of images in a user interface, possibly in a VBA UserForm. Let's break down what this code does step by step:

Loop Through a Range: The code starts with a For loop that runs from 1 to 220,000. Inside this loop, there's a DoEvents statement. DoEvents is used to allow the user interface to update while the loop is running, which is useful for preventing the interface from becoming unresponsive during long operations.

Check for Specific Values: Within the loop, there's an If statement that checks if the loop variable x is a multiple of 20,000 (e.g., 20,000, 40,000, 60,000, and so on). When x meets this condition, it enters a Select Case statement.

Select Case Statement: Depending on the value of x, a specific set of actions is taken. Here's a breakdown of what happens in each case:

For example, when x is 20,000, it changes the picture displayed in an ImageProgress control to "Chennai_02.jpg" and sets background and text colors for some other controls (Frame4 and Label10).

Similarly, for other cases, it changes the displayed image, and in some cases, it also updates the caption and colors of other controls.

Final Image Setting: After the loop completes (when x reaches 220,000), it sets the ImageProgress control's picture back to "Chennai_01.jpg".

This code appears to create a visual effect where a series of images are displayed sequentially, along with changes in background and text colors for certain controls. The actual images and color changes are based on the value of x. The specific purpose and context of this code would depend on the overall design and functionality of the user interface it's a part of.

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 Google Pay

9748327614

Summary:

This project appears to have significant learning value, especially for individuals looking to improve their VBA (Visual Basic for Applications) programming skills for Excel. Here's a breakdown of the learning value and what you can potentially learn from this project:

UserForm Interaction: The project uses a UserForm, which is a fundamental concept in VBA for creating custom Excel interfaces. You can learn how to create UserForms, add controls (like buttons, list boxes, labels, etc.), and respond to events triggered by these controls.

Data Handling: It involves working with Excel data. You can learn how to read data from worksheets, populate ListBox controls with data from Excel, and manipulate data based on user interactions.

Working with Collections: The code uses collections to manage data efficiently. You can learn how to use collections for storing and managing data, which is a crucial skill for optimizing VBA code.

Dictionary Object: The code uses a Scripting.Dictionary object to create a key-value data structure. You can learn how to use dictionaries for data lookup and storage.

Conditional Logic: There are numerous conditional statements (If-Then-Else, Select Case) used to make decisions based on certain conditions. This is essential for controlling the flow of your VBA code.

Error Handling: It includes error handling using On Error Resume Next and On Error GoTo 0. Error handling is crucial for making your code robust and handling unexpected issues gracefully.

File Operations: The code loads pictures from specific file paths. You can learn about file operations in VBA, including loading images into UserForms.

Looping: The code uses loops (For...Next) to perform repetitive tasks efficiently. Understanding loops is essential for processing large amounts of data.

Event Handling: The code responds to various events triggered by UserForm controls, such as button clicks and list box selections. Learning how to handle events is a key aspect of UserForm design.

String Manipulation: The code performs string manipulation operations, such as joining strings and checking for substrings. These skills are handy for text processing.

Data Filtering: The project filters and displays data based on various criteria (e.g., year, sales representative, product). You can learn how to filter and display specific data from a larger dataset.

Message Boxes: Message boxes are used to provide feedback to the user. You can learn how to display messages and information to users.

Dynamic Image Loading: The project demonstrates how to dynamically load and change images in a UserForm. This can be useful for creating dynamic interfaces.

Modular Code: The code is modular, with separate procedures for specific tasks. This promotes code organization and reusability.

Variable Handling: It uses different types of variables (String, Long, Object) for storing and manipulating data. You can learn about variable types and their usage.

Comments: The code includes comments that explain the purpose of functions and procedures, making it more readable and understandable. You can learn the importance of code documentation.

In summary, this project provides a practical example of building a user-friendly Excel interface using VBA. By studying and modifying this code, you can gain valuable insights into creating interactive Excel applications and enhance your VBA programming skills, especially in areas like UserForms, data manipulation, and event handling.