Video Link :

Last three Videos Link : https://youtu.be/bZuGpHa-pg4

1. https://youtu.be/j4voq5KbPSs
2. https://youtu.be/xJ0_vbWUnEA
3. https://youtu.be/YDWwF6FAIGg

It looks like a comprehensive VBA UserForm here with various ComboBoxes, Labels, and Buttons for different functionalities. Let me summarize and provide some advice on your code:

CmdDeleteItem_Click: This subroutine is designed to delete selected items from your "PictureData" sheet and move them to the "DeletedRecord" sheet. It appears to work correctly as long as you've set up your ComboBox (or other controls) to collect the selected items.

CmdRestoreItem_Click: This subroutine is meant to restore items from the "DeletedRecord" sheet back to the "PictureData" sheet. It seems to work correctly as well.

Image2_Click: This subroutine unloads the UserForm. This is a standard way to close a UserForm.

ComboxAddItemsFromFolder: This subroutine populates ComboBox1 with file names from a folder path. It sets a default value for ComboBox1.

ComboBox1_Change: This subroutine loads images and displays file information when an item is selected from ComboBox1. It also clears labels when no image is found.

GetImageDimensions: This is a custom function that returns the dimensions of an image file.

ComboboxAssItemsFromExcelSheet: This subroutine populates ComboBox2 with customer names from the "PictureData" sheet and displays additional information when a name is selected.
ComboBox2_Change: This subroutine displays image-related information when a customer name is selected from ComboBox2.

ComboBoxDynamic1: This subroutine populates ComboBox3 with unique locations from the "PictureData" sheet. It sets up a "Select Case" structure for further actions.

Combobox3_Change: This subroutine populates ComboBox4 based on the location selected in ComboBox3. It also sets a variable based on the selection for later use.

ComboBox4_Change: This subroutine displays image-related information when a picture name is selected from ComboBox4.

ComboRemove: This subroutine populates ComboBox5 with items from a specified sheet, and it allows you to remove selected items from the ComboBox.

CmdRemoveItem_Click: This subroutine removes selected items from ComboBox5.

ComboMulty: This subroutine initializes a collection (SelectedItems) to manage multiple selections from ComboBox6.

ComboBox6_Change: This subroutine handles the selection and deselection of items in ComboBox6. It also updates a label to display selected items.

UpdateLabel: This subroutine updates a label with the selected items from ComboBox6.

ComboRestoreInitia: Similar to ComboMulty, this subroutine initializes SelectedItems for use with ComboBox7. It also populates ComboBox7 with items from the "DeletedRecord" sheet.

UserForm_Initialize: This subroutine runs when the UserForm is initialized. It sets the current date, initializes ComboBoxes with data, and sets default values.

UserForm Initialize event:

You can use separate subroutines to populate ComboBoxes and then call these subroutines from the UserForm Initialize event. In fact, this is a good practice for keeping your code organized and more readable.
By breaking down the initialization code into smaller, focused subroutines, you make it easier to understand, maintain, and troubleshoot. It also follows the principle of modular programming, which promotes code reusability.
The code you provided appears to be doing just that. You have separate subroutines (ComboxAddItemsFromFolder, ComboboxAssItemsFromExcelSheet, ComboBoxDynamic1, ComboRemove, ComboMulty, and ComboRestoreInitia) responsible for populating specific

ComboBoxes, and you call these subroutines from the UserForm Initialize event. This approach is perfectly fine and can help make your code more manageable.

The code seems well-structured and organized, with clear comments to describe each part. It appears that have implemented various features for managing images and data. If you encounter any issues or need specific assistance with a particular part of the code, feel free to ask me in Comments Section.

"Insert the following controls into the user form: seven combo boxes, several labels, one image control, one image control for a closing icon, and three command buttons. You can customize their properties such as names, captions, fonts, and alignments from the properties window. Additionally, ensure you have an Excel file containing sheets named 'PictureData,' 'DeletedRecord,' and 'TempPictureData' for data handling. To delete some records from 'PictureData,' we recommend saving all records to a temporary sheet to ensure their safety."

This VBA code is designed to work with an Excel UserForm that seems to have various ComboBoxes, Labels, and Buttons for managing images and data. Let's break down the code step by step:

```
Dim SelectedItems As Collection ' Declare the SelectedItems collection at the module level
Dim J As Long, K As Long, L As Long, M As Long
```

Declaring Variables and Collections:

Here, I am declaring some module-level variables and a collection named SelectedItems. Module-level variables are accessible to all procedures within the module. The collection will be used to store selected items.

```
Private Sub CmdDeleteItem_Click()
    Dim SourceSheet As Worksheet
    Dim TargetSheet As Worksheet
    Dim LastRow As Long
    Dim i As Long
    Dim SelectedItem As Variant

    ' Set references to the source and target sheets
```

```
    Set SourceSheet = ThisWorkbook.Sheets("PictureData")
    Set TargetSheet = ThisWorkbook.Sheets("DeletedRecord")

    ' Find the last used row in the source sheet
    LastRow = SourceSheet.Cells(SourceSheet.Rows.Count, "A").End(xlUp).row

    ' Loop through each selected item
    For Each SelectedItem In SelectedItems
        ' Loop through each row in reverse order to avoid issues with row deletion
        For i = LastRow To 2 Step -1 ' Assuming data starts from row 2
            ' Check if the SpotName in the current row matches the selected item
            If SourceSheet.Cells(i, "A").value = SelectedItem Then
                ' Copy the entire row to the target sheet
                SourceSheet.Rows(i).Copy TargetSheet.Cells(TargetSheet.Rows.Count, "A").End(xlUp).Offset(1, 0)

                ' Delete the row from the source sheet
                SourceSheet.Rows(i).Delete
            End If
        Next i
    Next SelectedItem

    ' Clear the SelectedItems collection
    Set SelectedItems = New Collection
End Sub
```

This VBA code is designed to handle the "CmdDeleteItem_Click" event in a workbook, most likely in Excel. It seems to be part of a larger application, possibly for managing picture data. Let's break down this code step by step to explain it to your student:

Private Sub CmdDeleteItem_Click()

This line signifies the beginning of a subroutine that will be executed when a control with the name "CmdDeleteItem" is clicked. Typically, this control could be a button on a user form or a worksheet.

Dim SourceSheet As Worksheet Dim TargetSheet As Worksheet Dim LastRow As Long Dim i As Long Dim SelectedItem As Variant

Here, several variables are declared:

SourceSheet and TargetSheet are variables that will hold references to worksheet objects in the current workbook.

LastRow will store the row number of the last used row in the "PictureData" sheet.

i is a loop counter variable.

SelectedItem will represent each selected item from a collection.

' Set references to the source and target sheets *Set SourceSheet = ThisWorkbook.Sheets("PictureData") Set TargetSheet = ThisWorkbook.Sheets("DeletedRecord")*

These lines set the SourceSheet and TargetSheet variables to refer to specific worksheets in the current workbook ("PictureData" and "DeletedRecord" sheets, respectively).

' Find the last used row in the source sheet *LastRow = SourceSheet.Cells(SourceSheet.Rows.Count, "A").End(xlUp).Row*

This code calculates the last used row in the "PictureData" sheet. It starts from the bottom of column A (assuming that column contains data) and goes up until it finds the last used cell. The row number is then stored in the LastRow variable.

' Loop through each selected item *For Each SelectedItem In SelectedItems*

This line begins a loop that will iterate through each item in a collection called SelectedItems. It's assumed that SelectedItems contains a list of items to be processed.

' Loop through each row in reverse order to avoid issues with row deletion *For i = LastRow To 2 Step -1 ' Assuming data starts from row 2*

This nested loop is used to iterate through each row in the "PictureData" sheet, starting from the last row (LastRow) and moving upwards (step -1). This is done to avoid issues with row deletion, as deleting rows from the top down can cause problems with row references.

' Check if the SpotName in the current row matches the selected item *If SourceSheet.Cells(i, "A").Value = SelectedItem Then*

This code checks if the value in column A of the current row (SourceSheet.Cells(i, "A").Value) in the "PictureData" sheet matches the currently selected item (SelectedItem) from the collection.

' Copy the entire row to the target sheet *SourceSheet.Rows(i).Copy TargetSheet.Cells(TargetSheet.Rows.Count, "A").End(xlUp).Offset(1, 0)*

If there's a match, this line copies the entire row from the "PictureData" sheet to the "DeletedRecord" sheet. It pastes it in the next available row in column A of the "DeletedRecord" sheet.

' Delete the row from the source sheet *SourceSheet.Rows(i).Delete*

After copying the row, it is deleted from the "PictureData" sheet to avoid duplicates or unwanted data.

End If Next i Next SelectedItem

The loops continue until all selected items have been processed.

' Clear the SelectedItems collection *Set SelectedItems = New Collection*

Finally, after all items have been processed and moved to the "DeletedRecord" sheet, the SelectedItems collection is cleared, presumably to prepare for a new selection in the future.

In summary, this code is designed to move rows of data from one worksheet ("PictureData") to another ("DeletedRecord") based on a collection of selected items (SelectedItems). It iterates through the selected items, checks for matches in the "PictureData" sheet, copies matching rows to the "DeletedRecord" sheet, and then deletes them from the source sheet. The code is structured to ensure that it works efficiently and without errors when dealing with row deletions.

CmdDeleteItem_Click() Sub:

This subroutine is triggered when a button named CmdDeleteItem is clicked.

It does the following:

Sets references to source and target worksheets.

Finds the last used row in the source sheet.

Loops through each selected item stored in SelectedItems.

Within the loop, it again loops through each row (in reverse order to avoid issues with row deletion) in the source sheet.

Checks if the "SpotName" in the current row matches the selected item. If so, it copies the entire row to the target sheet and deletes it from the source sheet.

Finally, it clears the SelectedItems collection.

Private Sub CmdRestoreItem_Click()

Dim SourceSheet As Worksheet

```
    Dim TargetSheet As Worksheet
    Dim LastRow As Long
    Dim i As Long
    Dim SelectedItem As Variant

    ' Set references to the source and target sheets
    Set SourceSheet = ThisWorkbook.Sheets("DeletedRecord")
    Set TargetSheet = ThisWorkbook.Sheets("PictureData")

    ' Check if an item is selected in the ComboBoxRestore
    If ComboBox7.ListIndex >= 0 Then
        ' Get the selected item
        SelectedItem = ComboBox7.List(ComboBox7.ListIndex)

        ' Initialize a variable to keep track of the number of rows deleted
        Dim RowsDeleted As Long
        RowsDeleted = 0

        ' Find the last used row in the target sheet
        LastRow = TargetSheet.Cells(TargetSheet.Rows.Count, "A").End(xlUp).row

        ' Loop through each row in the source sheet
        For i = SourceSheet.Cells(SourceSheet.Rows.Count, "A").End(xlUp).row To 2 Step -1 ' Loop in reverse
order
            ' Check if the SpotName in the current row matches the selected item
            If SourceSheet.Cells(i, "A").value = SelectedItem Then
                ' Copy the entire row to the target sheet
                SourceSheet.Rows(i).Copy TargetSheet.Cells(LastRow + RowsDeleted + 1, "A")
```

```
            ' Delete the row from the source sheet
            SourceSheet.Rows(i).Delete
            RowsDeleted = RowsDeleted + 1 ' Increment the counter for rows deleted
        End If
    Next i

    ' Clear the ComboBoxRestore selection
    ComboBox7.ListIndex = -1
  Else
    MsgBox "Please select an item to restore.", vbExclamation
  End If
End Sub
```

This VBA code is designed to handle the "CmdRestoreItem_Click" event in a workbook, most likely in Excel. It appears to be part of a larger application for managing data. Let's break down this code step by step to explain it:

Private Sub CmdRestoreItem_Click()

This line signifies the beginning of a subroutine that will be executed when a control with the name "CmdRestoreItem" is clicked. Typically, this control could be a button on a user form or a worksheet.

Dim SourceSheet As Worksheet Dim TargetSheet As Worksheet Dim LastRow As Long Dim i As Long Dim SelectedItem As Variant

Here, several variables are declared:

SourceSheet and TargetSheet are variables that will hold references to worksheet objects in the current workbook.

LastRow will store the row number of the last used row in the "DeletedRecord" sheet.

i is a loop counter variable.

SelectedItem will represent the item selected in a ComboBox.

' Set references to the source and target sheets *Set SourceSheet = ThisWorkbook.Sheets("DeletedRecord") Set TargetSheet = ThisWorkbook.Sheets("PictureData")*

These lines set the SourceSheet and TargetSheet variables to refer to specific worksheets in the current workbook ("DeletedRecord" and "PictureData" sheets, respectively).

' Check if an item is selected in the *ComboBoxRestore If ComboBox7.ListIndex >= 0 Then*

This part checks if an item is selected in a ComboBox named "ComboBox7." If an item is selected (the ListIndex is greater than or equal to 0), the code proceeds. Otherwise, it shows a message box asking the user to select an item and exits the subroutine.

' Get the selected item *SelectedItem = ComboBox7.List(ComboBox7.ListIndex)*

If an item is selected, it retrieves the selected item from the ComboBox and stores it in the SelectedItem variable.

' Initialize a variable to keep track of the number of rows deleted *Dim RowsDeleted As Long RowsDeleted = 0*

This code initializes a counter variable RowsDeleted to keep track of the number of rows that will be deleted from the source sheet.

' Find the last used row in the target sheet *LastRow = TargetSheet.Cells(TargetSheet.Rows.Count, "A").End(xlUp).Row*

This line calculates the last used row in column A of the "PictureData" sheet and stores it in the LastRow variable.

' Loop through each row in the source sheet *For i = SourceSheet.Cells(SourceSheet.Rows.Count, "A").End(xlUp).Row To 2 Step -1 ' Loop in reverse order*

This loop iterates through each row of the "DeletedRecord" sheet, starting from the last used row and moving upwards in reverse order (from bottom to top).

' Check if the SpotName in the current row matches the selected item *If SourceSheet.Cells(i, "A").Value = SelectedItem Then*

In each iteration, it checks if the value in column A of the current row in the "DeletedRecord" sheet matches the SelectedItem from the ComboBox.

' Copy the entire row to the target sheet *SourceSheet.Rows(i).Copy TargetSheet.Cells(LastRow + RowsDeleted + 1, "A")*

If there's a match, this code copies the entire row from the "DeletedRecord" sheet to the "PictureData" sheet. It pastes it in the next available row in column A of the "PictureData" sheet.

' Delete the row from the source sheet *SourceSheet.Rows(i).Delete*

After copying the row, it is deleted from the "DeletedRecord" sheet to avoid duplicates or unwanted data.

RowsDeleted = RowsDeleted + 1 ' Increment the counter for rows deleted

The RowsDeleted counter is incremented to keep track of how many rows have been deleted and copied.

Next i

The loop continues until all rows in the "DeletedRecord" sheet have been checked.

' Clear the ComboBoxRestore selection *ComboBox7.ListIndex = -1*

After processing, the code clears the selection in "ComboBox7" by setting its ListIndex to -1.

Else *MsgBox "Please select an item to restore.", vbExclamation End If End Sub*

If no item is selected in "ComboBox7," it shows a message box asking the user to select an item.

In summary, this code allows the user to restore previously deleted records from the "DeletedRecord" sheet to the "PictureData" sheet. It checks if an item is selected in a ComboBox, copies the corresponding row from the "DeletedRecord" sheet to the "PictureData" sheet, deletes it from the "DeletedRecord" sheet, and updates counters accordingly. If no item is selected, it displays a message prompting the user to select one.

CmdRestoreItem_Click() Sub:

This subroutine is triggered when a button named CmdRestoreItem is clicked.

It does the following:

Sets references to source and target worksheets (in this case, for restoring deleted records).

Checks if an item is selected in a ComboBox named ComboBox7.

If an item is selected, it gets the selected item and initializes a variable to keep track of the number of rows deleted.

Finds the last used row in the target sheet.

Loops through each row (in reverse order) in the source sheet.

Checks if the "SpotName" in the current row matches the selected item. If so, it copies the entire row to the target sheet, deletes it from the source sheet, and increments the counter for rows deleted.

Finally, it clears the ComboBox selection.

```
Private Sub Image2_Click()
    Unload Me
End Sub
```

Image2_Click() Sub:

This subroutine is triggered when an image (presumably a close button) named Image2 is clicked.

It unloads the UserForm, effectively closing it.

```
Private Sub ComboxAddItemsFromFolder()
    Dim folderPath As String
    Dim fileName As String
```

```
    ' Set the folder path where your picture files are located
    folderPath = "E:\Shanti\"

    ' Loop through each file in the folder
    fileName = Dir(folderPath & "*.jpg") ' Change the file extension as needed

    ' Populate the ListBox with picture file names
    Do While fileName <> ""
        ComboBox1.AddItem fileName
        fileName = Dir ' Get the next file in the folder
    Loop
    ' Set the default value for ComboBox1
    ComboBox1.value = "Picture from Folder" ' Change to the item you want as the default
End Sub
```

This VBA code defines a subroutine called ComboxAddItemsFromFolder, and its purpose is to populate a ComboBox (likely a user interface control in Excel) with the names of image files found in a specific folder. Let's break down the code step by step:

Private Sub ComboxAddItemsFromFolder()

This line signifies the beginning of a subroutine named ComboxAddItemsFromFolder.

Dim folderPath As String Dim fileName As String

Two variables, folderPath and fileName, are declared.

folderPath will store the path to the folder where image files are located.

fileName will be used to temporarily store the names of files found in the folder.

' Set the folder path where your picture files are located *folderPath = "E:\Shanti\"*

This line assigns a specific folder path to the folderPath variable. In this case, it's set to "E:\Shanti". This is the folder where the code will look for image files.

' Loop through each file in the folder *fileName = Dir(folderPath & "*.jpg")* ' Change the file extension as needed

This part initiates a loop that will iterate through all the files in the folder specified by folderPath. It uses the Dir function to retrieve the name of the first file in the folder with a ".jpg" extension. You can change "*.jpg" to a different file extension if your images have a different format.

' Populate the ListBox with picture file names *Do While fileName <> ""*

This line starts a loop that continues as long as the fileName variable contains a valid file name (i.e., it's not an empty string).

ComboBox1.AddItem fileName

Inside the loop, this code adds the current fileName to a ComboBox control named "ComboBox1." This effectively populates the ComboBox with the names of image files found in the folder.

fileName = Dir ' Get the next file in the folder

After adding the current file name to the ComboBox, this line updates the fileName variable to the name of the next file in the folder. The Dir function is used again to retrieve the next file name.

Loop

This Loop statement indicates that the code will keep looping until there are no more valid file names in the folder.

' Set the default value for *ComboBox1 ComboBox1.Value = "Picture from Folder"* ' Change to the item you want as the default End Sub

Finally, after all files have been added to the ComboBox, this code sets the default selected item in the ComboBox to "Picture from Folder." You can change this default value to any other item you prefer.

In summary, this subroutine is used to dynamically populate a ComboBox control with the names of image files found in a specific folder. It's a useful function for creating a user-friendly interface in Excel where users can select image files from a predefined folder.

ComboxAddItemsFromFolder() Sub:

This subroutine populates ComboBox1 with file names from a specified folder path. It uses the Dir function to loop through files with a .jpg extension in the folder and adds them to the ComboBox.

```
Private Sub ComboBox1_Change()
    Label4.Visible = False
    selectedImageName = ComboBox1.value
    Dim imagePath As String
```

```
    imagePath = "E:\Shanti\" & selectedImageName '& ".jpg"
    If Len(Dir(imagePath)) > 0 Then
        ' Get file information
        Dim file As Object
        Set file = CreateObject("Scripting.FileSystemObject").GetFile(imagePath)

        ' Display file information in labels
        Image1.Picture = LoadPicture(imagePath)
        LabelFileSize.caption = "File Size: " & Format(file.Size / 1024, "0.00") & " KB"
        LabelDimensions.caption = "Dimensions: " & GetImageDimensions(imagePath)
        LabelCreationDate.caption = "Creation Date: " & Format(file.DateCreated, "dd-mmm-yyyy")
        LabelFolder.caption = "File Location: " & imagePath
        LabelClickedBy.caption = "Clicked by : Gautam Banerjee"
    Else
        ' Clear labels if the image file does not exist
        'MsgBox "Image not found at the specified location or invalid file format." & _
            vbCrLf & "Please select a valid image file.", vbExclamation, "Error - Gautam Banerjee"
        LabelFileSize.caption = "File Size: N/A"
        LabelDimensions.caption = "Dimensions: N/A"
        LabelCreationDate.caption = "Creation Date: N/A"
        LabelFolder.caption = "Folder Location: N/A"
        LabelClickedBy.caption = "Invalid Picture File Name"
    End If
    Label2.caption = "Santiniketan was established by Debendranath Tagore in 1863, and then developed
by his son, Rabindranath Tagore in West Bengal"
End Sub
```

This VBA code defines a subroutine named ComboBox1_Change. This subroutine appears to be associated with a ComboBox control (likely in an Excel user form) named ComboBox1. Here's an explanation of what this code does:

Private Sub ComboBox1_Change()

This line marks the beginning of the subroutine ComboBox1_Change, which will execute whenever the selected item in ComboBox1 changes.

Label4.Visible = False

This line sets the Visible property of a Label control named Label4 to False, effectively hiding it. This may be used to hide or show certain elements based on user interactions.

selectedImageName = ComboBox1.Value

This line assigns the selected item's value from ComboBox1 to the variable selectedImageName. It appears to be used to store the selected image file's name.

Dim imagePath As String imagePath = "E:\Shanti\" & selectedImageName '& ".jpg"

This code constructs the full path to the selected image file using the selectedImageName and a fixed folder path. It assumes that the image files are located in the "E:\Shanti" directory. You might need to change this path to match the actual location of your image files.

If Len(Dir(imagePath)) > 0 Then

This If statement checks if the image file specified by imagePath exists. It does this by using the Dir function to look for the file. If the length of the result is greater than 0, it means the file exists.

' Get file information

Dim file As Object Set file = CreateObject("Scripting.FileSystemObject").GetFile(imagePath)

This code creates an instance of the Scripting.FileSystemObject and uses it to retrieve information about the selected image file, such as its size, creation date, etc.

' Display file information in labels *Image1.Picture = LoadPicture(imagePath) LabelFileSize.Caption = "File Size: " & Format(file.Size / 1024, "0.00") & " KB" LabelDimensions.Caption = "Dimensions: " & GetImageDimensions(imagePath) LabelCreationDate.Caption = "Creation Date: " & Format(file.DateCreated, "dd-mmm-yyyy") LabelFolder.Caption = "File Location: " & imagePath LabelClickedBy.Caption = "Clicked by: Gautam Banerjee"*

This block of code updates various labels and an image control (Image1) with information about the selected image file. It does the following:

Sets the Picture property of Image1 to display the selected image.

Updates LabelFileSize with the file size in kilobytes (KB).

Calls a function GetImageDimensions (which is assumed to return the image dimensions) and updates LabelDimensions with the dimensions.

Retrieves the file's creation date and updates LabelCreationDate.

Updates LabelFolder with the image file's path.

Sets LabelClickedBy to indicate who clicked the ComboBox.

Else

If the image file specified by imagePath does not exist (the If condition is False), this code block is executed.

' Clear labels if the image file does not exist ' *MsgBox "Image not found at the specified location or invalid file format." & _ vbCrLf & "Please select a valid image file.", vbExclamation, "Error - Gautam Banerjee"*

This section is commented out (notice the ' before MsgBox). It appears to be a placeholder for showing a message box if the selected image file is not found or if there's an invalid file format. You can uncomment and customize this code if you want to display a message box in such cases.

LabelFileSize.Caption = "File Size: N/A"

LabelDimensions.Caption = "Dimensions: N/A"

LabelCreationDate.Caption = "Creation Date: N/A"

LabelFolder.Caption = "Folder Location: N/A"

LabelClickedBy.Caption = "Invalid Picture File Name"

If the image file is not found or is invalid, this code block sets various labels to indicate that the file information is not available (N/A) and sets LabelClickedBy to indicate that the file name is invalid.

End If Label2.Caption = "Santiniketan was established by Debendranath Tagore in 1863, and then developed by his son, Rabindranath Tagore in West Bengal" End Sub

This section marks the end of the If statement, and it sets the caption of Label2 to a specific text. This is likely providing some additional information or context related to the selected image.

In summary, this subroutine is designed to respond to changes in the selected item of ComboBox1. It updates various labels and displays the selected image along with its file information based on the selected item from the ComboBox and the assumed file path. If the file doesn't exist, it displays appropriate error messages in the labels.

ComboBox1_Change() Sub:

This subroutine is triggered when the selection in ComboBox1 changes.

It updates various labels and displays information about the selected image file, such as file size, dimensions, creation date, and file location.

```
Function GetImageDimensions(imagePath As String) As String
    ' This function returns the dimensions of an image in the format "Width x Height"
    Dim img As Object
    Set img = CreateObject("WIA.ImageFile")
    img.LoadFile imagePath
    GetImageDimensions = img.Width & " x " & img.Height
End Function
```

This VBA function, named GetImageDimensions, takes an imagePath as input, which is assumed to be the path to an image file. It's designed to return the dimensions of that image in the format "Width x Height." Here's a breakdown of how this function works:

Function GetImageDimensions(imagePath As String) As String

This line defines a VBA function named GetImageDimensions. It expects a single argument imagePath, which should be a string representing the path to an image file. The function will return a string.

' This function returns the dimensions of an image in the format "Width x Height"

This comment provides a brief description of what the function does for anyone reading the code.

Dim img As Object Set img = CreateObject("WIA.ImageFile")

These lines declare a variable img as an object and then create an instance of the Windows Image Acquisition (WIA) ImageFile object using CreateObject. This object will be used to load and manipulate the image.

img.LoadFile imagePath

This line loads the image located at the specified imagePath into the img object. It essentially opens the image file for processing.

GetImageDimensions = img.Width & " x " & img.Height

This line calculates and assigns the image dimensions to the function's return value. It concatenates the width (img.Width) and height (img.Height) of the image with the "x" separator, resulting in a string in the format "Width x Height."

End Function

This line marks the end of the function.

In summary, this VBA function uses the Windows Image Acquisition (WIA) library to load an image file specified by its path and then extracts its width and height, returning them as a string in

the "Width x Height" format. This function can be called from elsewhere in your VBA code, passing the path to an image file as an argument, and it will provide the dimensions of that image in the desired format.

GetImageDimensions() Function:

This is a custom function that calculates and returns the dimensions (width x height) of an image file.

```
Private Sub ComboboxAssItemsFromExcelSheet()
   Dim ws As Worksheet
   Dim cell As Range

   ' Set the worksheet where customer names are stored
   Set ws = ThisWorkbook.Sheets("PictureData")

   ' Loop through the worksheet to populate the ComboBox
   For Each cell In ws.Range("B2:B" & ws.Cells(ws.Rows.Count, "B").End(xlUp).row)
      ComboBox2.AddItem cell.value
   Next cell
   ' Set the default value for ComboBox1
   ComboBox2.value = "Picture from Sheet" ' Change to the item you want as the default
End Sub
```

This VBA subroutine, named ComboboxAssItemsFromExcelSheet, populates a ComboBox (ComboBox2) with items retrieved from a specified worksheet in an Excel workbook. Here's an explanation of how this code works:

Private Sub ComboboxAssItemsFromExcelSheet()

This line defines a subroutine named ComboboxAssItemsFromExcelSheet.

Dim ws As Worksheet Dim cell As Range

These lines declare two variables: ws to represent a worksheet object and cell to represent a cell within the worksheet.

' Set the worksheet where customer names are stored *Set ws = ThisWorkbook.Sheets("PictureData")*

This comment explains that the ws variable is set to reference a worksheet named "PictureData" within the current workbook (ThisWorkbook).

' Loop through the worksheet to populate the ComboBox *For Each cell In ws.Range("B2:B" & ws.Cells(ws.Rows.Count, "B").End(xlUp).Row)*

This comment indicates that the code is about to loop through a specific range of cells in the worksheet. It starts from cell B2 and goes up to the last used cell in column B.

ComboBox2.AddItem cell.Value

Inside the loop, this line adds the value of each cell in the specified range to ComboBox2. This effectively populates the ComboBox with the values from column B of the "PictureData" worksheet.

Next cell

This line marks the end of the loop through the worksheet cells.

' Set the default value for ComboBox1 *ComboBox2.Value = "Picture from Sheet" ' Change to the item you want as the default End Sub*

This comment indicates that the code is setting a default value for ComboBox2, which is "Picture from Sheet." You can change this default value to any other item that you want to be preselected when the ComboBox is initially displayed.

In summary, this VBA subroutine reads data from a specific column (column B) in a worksheet named "PictureData" and populates ComboBox2 with these values. It also sets a default value for the ComboBox, which will be displayed when the user interacts with it. This can be useful for providing a default selection or prompt in the ComboBox.

ComboboxAssItemsFromExcelSheet() Sub:

This subroutine populates ComboBox2 with items from a specified Excel worksheet column (B2:B).

```
Private Sub ComboBox2_Change()

Label4.Visible = True

   selectedImageName = ComboBox2.value

   imagePath = "E:\Chennai_Tours\Excel_Picture\" & selectedImageName & ".jpg"
```

```
    Dim ws As Worksheet
    Set ws = ThisWorkbook.Sheets("PictureData") ' Replace with your actual worksheet name
    If Len(Dir(imagePath)) > 0 Then
        ' Get file information


        ' Display file information in labels
        Image1.Picture = LoadPicture(imagePath)
        Label2.caption = ws.Cells(ComboBox2.ListIndex + 2, "D").value
        Label4.caption = ws.Cells(ComboBox2.ListIndex + 2, "E").value
    Else
        ' Clear labels if the image file does not exist
        Image1.Picture = LoadPicture("")
        'MsgBox "Image not found at the specified location or invalid file format." & _
            vbCrLf & "Please select a valid image file.", vbExclamation, "Error - Gautam Banerjee"


    End If
End Sub
```

This VBA subroutine, named ComboBox2_Change, is associated with a ComboBox (ComboBox2). It triggers when the selected item in ComboBox2 is changed. Here's an explanation of how this code works:

Private Sub ComboBox2_Change()

This line defines a subroutine named ComboBox2_Change. This subroutine is executed when the selected item in ComboBox2 is changed by the user.

Label4.Visible = True

This line sets the Visible property of a label named Label4 to True. This probably makes Label4 visible to the user when an item is selected in ComboBox2. The purpose of this label and its visibility may be specific to your application's user interface.

selectedImageName = ComboBox2.Value

This line retrieves the currently selected item from ComboBox2 and stores it in the selectedImageName variable.

imagePath = "E:\Chennai_Tours\Excel_Picture\" & selectedImageName & ".jpg"

This line constructs a file path (imagePath) by appending the selectedImageName to a fixed directory path. It assumes that image files are located in a folder named "Excel_Picture" under the "E:\Chennai_Tours" directory, and the file extension is ".jpg".

Dim ws As Worksheet *Set ws = ThisWorkbook.Sheets("PictureData")* ' Replace with your actual worksheet name

These lines declare a variable ws as a reference to a specific worksheet in the current workbook. The Set statement assigns this reference to the "PictureData" worksheet.

If Len(Dir(imagePath)) > 0 Then

This conditional statement checks if a file exists at the imagePath. The Dir function is used to determine whether a file exists at the specified location. If the file exists, the condition evaluates to True.

' Get file information

This comment indicates that the code is about to retrieve information about the image file located at imagePath.

' Display file information in labels *Image1.Picture = LoadPicture(imagePath)*

This line loads and displays the image located at imagePath in an image control named Image1.

Label2.Caption = ws.Cells(ComboBox2.ListIndex + 2, "D").Value

Label4.Caption = ws.Cells(ComboBox2.ListIndex + 2, "E").Value

These lines set the captions of labels Label2 and Label4 to values from cells in the "PictureData" worksheet. The specific cells being referred to are determined by the selected item in ComboBox2. It appears that the code is using ComboBox2.ListIndex to calculate the row number and then fetching data from columns "D" and "E" of that row.

Else

This Else statement is executed if the file specified by imagePath does not exist.

' Clear labels if the image file does not exist *Image1.Picture = LoadPicture("") 'MsgBox "Image not found at the specified location or invalid file format." & _ vbCrLf & "Please select a valid image file.", vbExclamation, "Error - Gautam Banerjee"*

These lines clear the image displayed in Image1 by loading an empty picture. They also handle the scenario where the image file is not found. A message box (MsgBox) could be displayed to inform the user, but it's currently commented out (preceded by an apostrophe '). You can uncomment it if you want to display an error message when the image file is not found.

In summary, this VBA subroutine updates the user interface when an item is selected in ComboBox2. It displays an image, updates label captions, and handles cases where the selected

image file is not found. The specific labels, image controls, and file paths used in this code may vary depending on your Excel application's design.

ComboBox2_Change() Sub:

This subroutine is triggered when the selection in ComboBox2 changes.

It updates labels and displays information based on the selected item, presumably an image file.

```
Private Sub ComboBoxDynamic1()

    Dim ws As Worksheet
    Set ws = ThisWorkbook.Sheets("PictureData")

    Dim cell As Range
    Dim uniqueLocations As Collection
    Set uniqueLocations = New Collection

    On Error Resume Next
    For Each cell In ws.Range("A2:A" & ws.Cells(ws.Rows.Count, "A").End(xlUp).row)
        uniqueLocations.Add cell.value, CStr(cell.value)
    Next cell
    On Error GoTo 0

    Dim Loc As Variant ' Declare Loc as a variant

    For Each Loc In uniqueLocations
        ComboBox3.AddItem Loc
    Next Loc
    ComboBox3.value = "First Select Tourist Spot" ' Change to the item you want as the default
End Sub
```

This VBA subroutine, named ComboBoxDynamic1, seems to be designed to populate ComboBox3 with unique values from a specific column in the "PictureData" worksheet. Let's break down how this code works:

Private Sub ComboBoxDynamic1()

This line defines a subroutine named ComboBoxDynamic1.

Dim ws As Worksheet *Set ws = ThisWorkbook.Sheets("PictureData")*

These lines declare a variable ws to represent a specific worksheet in the current workbook. The Set statement assigns this variable to the "PictureData" worksheet.

Dim cell As Range Dim uniqueLocations As Collection Set uniqueLocations = New Collection

These lines declare two variables, cell and uniqueLocations.

cell is used to loop through the cells in a specific column of the worksheet.

uniqueLocations is a collection object that will store unique values from that column.

On Error Resume Next

This line turns on error handling, which means that if an error occurs during the execution of the subsequent code, it will be ignored, and the code will continue to run.

For Each cell In ws.Range("A2:A" & ws.Cells(ws.Rows.Count, "A").End(xlUp).Row)

This is a loop that iterates through each cell in column "A" of the "PictureData" worksheet, starting from row 2 and continuing until the last used row in column "A" is reached.

ws.Range("A2:A" & ws.Cells(ws.Rows.Count, "A").End(xlUp).Row) defines the range of cells to loop through.

uniqueLocations.Add cell.Value, CStr(cell.Value)

Inside the loop, this line adds the value of the current cell (i.e., cell.Value) to the uniqueLocations collection. This is done using the Add method, which ensures that only unique values are added. The CStr function is used to convert the cell value to a string, which is used as the key for uniqueness in the collection.

Next cell

This line indicates the end of the loop through the cells in column "A."

On Error GoTo 0

This line turns off error handling, meaning that any errors that occur after this point will be treated as normal and will not be ignored.

Dim Loc As Variant ' Declare Loc as a variant

This line declares a variable Loc as a variant. This variable will be used to iterate through the unique values stored in the uniqueLocations collection.

For Each Loc In uniqueLocations

This is a loop that iterates through each unique location in the uniqueLocations collection.

ComboBox3.AddItem Loc

Inside the loop, this line adds each unique location (stored in the Loc variable) to ComboBox3. This populates ComboBox3 with the unique values from column "A" of the "PictureData" worksheet.

Next Loc

This line indicates the end of the loop through the unique locations.

ComboBox3.Value = "First Select Tourist Spot" ' Change to the item you want as the default

This line sets the default selected item in ComboBox3 to "First Select Tourist Spot." You can change this to another default item if needed.

In summary, this VBA subroutine reads unique values from column "A" of the "PictureData" worksheet and populates them into ComboBox3. It also sets a default item in the ComboBox. This is useful for creating dynamic ComboBoxes based on the data in your worksheet, ensuring that users can select from a list of unique values.

ComboBoxDynamic1() Sub:

This subroutine populates ComboBox3 with unique values from a specified Excel worksheet column (A2:A).

```
Private Sub Combobox3_Change()
    Dim selectedValue As String
    Dim ws As Worksheet
    Dim i As Long


    ' Clear the list box
    ComboBox4.Clear
```

```
' Get the selected item from the combo box
selectedValue = ComboBox3.value

' Set your worksheet (change the sheet name if needed)
Set ws = ThisWorkbook.Sheets("PictureData")

' Loop through the data in the worksheet
For i = 2 To ws.Cells(ws.Rows.Count, "A").End(xlUp).row
    If ws.Cells(i, 1).value = selectedValue Then
        ' Add matching items to the list box
        ComboBox4.AddItem ws.Cells(i, 2).value
    End If
Next i

Select Case ComboBox3
    Case "Chennai Tourist Spot"
    J = 2

    Case "Rameswaram Tourist Spot"
    J = 15

    Case "Pondichery Mahabalipuram"
    J = 25

    Case "Kodaikanal Tourist Spot"
    J = 35

    Case "ECO Park Kolkata"
    J = 45
```

End Select

End Sub

This VBA subroutine, named Combobox3_Change, appears to be associated with an event handler for a ComboBox named ComboBox3. Let's break down how this code works:

Private Sub Combobox3_Change()

This line defines a subroutine named Combobox3_Change, which is triggered when the value in ComboBox3 changes (i.e., when a user selects a different item).

Dim selectedValue As String Dim ws As Worksheet Dim i As Long

These lines declare three variables:

selectedValue: This variable will store the currently selected value in ComboBox3.

ws: This variable represents a worksheet in the current workbook.

i: This variable will be used as a counter in a loop.

' Clear the list box ComboBox4.Clear

This line clears the contents of ComboBox4, presumably to remove any previous selections or items.

' Get the selected item from the combo box *selectedValue = ComboBox3.Value*

These lines get the currently selected item in ComboBox3 and store it in the selectedValue variable.

' Set your worksheet (change the sheet name if needed) *Set ws = ThisWorkbook.Sheets("PictureData")*

These lines set the ws variable to represent a specific worksheet named "PictureData" in the current workbook. You can change the sheet name if your data is on a different worksheet.

' Loop through the data in the worksheet *For i = 2 To ws.Cells(ws.Rows.Count, "A").End(xlUp).Row*

This line starts a loop that will go through each row of data in the "PictureData" worksheet. It starts from row 2 and goes up to the last used row in column "A."

If ws.Cells(i, 1).Value = selectedValue Then

Inside the loop, this line checks if the value in column "A" of the current row (ws.Cells(i, 1).Value) is equal to the selectedValue from ComboBox3.

' Add matching items to the list box ComboBox4.AddItem ws.Cells(i, 2).Value

If there's a match, it adds the value from column "B" of the same row (ws.Cells(i, 2).Value) to ComboBox4. This presumably populates ComboBox4 with items related to the selection in ComboBox3.

End If Next i

This line marks the end of the loop. It will continue checking each row in the worksheet until it reaches the last used row.

Select Case ComboBox3

This line starts a Select Case statement, which allows different actions to be taken based on the value of ComboBox3.

Case "Chennai Tourist Spot" J = 2

In this part of the Select Case, if ComboBox3 contains the text "Chennai Tourist Spot," it sets the variable J to 2. This appears to be setting some kind of identifier for Chennai in your code.

Case "Rameswaram Tourist Spot" J = 15

Similarly, if the value is "Rameswaram Tourist Spot," it sets J to 15, presumably as an identifier for Rameswaram.

' Additional cases for other locations

The code continues with additional Case statements for other tourist spots, each setting a different value for J depending on the selected location.

The purpose of this subroutine seems to be to update ComboBox4 based on the selection in ComboBox3 and set a corresponding value for J based on the selected location. The specific use of J is not clear from this code snippet, but it appears to be related to identifying or configuring different locations in your application.

Combobox3_Change() Sub:

This subroutine is triggered when the selection in ComboBox3 changes.

It clears another ComboBox named ComboBox4 and populates it with items that match the selected value in ComboBox3. The selected value determines which items are shown.

It also sets a module-level variable J based on the selected value.

```
Private Sub Combobox4_Change()
    Label4.Visible = True
    selectedImageName = ComboBox4.value
    imagePath = "E:\Chennai_Tours\Excel_Picture\" & selectedImageName & ".jpg"



    Dim ws As Worksheet
    Set ws = ThisWorkbook.Sheets("PictureData") ' Replace with your actual worksheet name
    If Len(Dir(imagePath)) > 0 Then
        ' Get file information


        ' Display file information in labels
        Image1.Picture = LoadPicture(imagePath)
        Label2.caption = ws.Cells(ComboBox4.ListIndex + J, "D").value
        Label4.caption = ws.Cells(ComboBox4.ListIndex + J, "E").value
    Else
        ' Clear labels if the image file does not exist
        Image1.Picture = LoadPicture("")
        'MsgBox "Image not found at the specified location or invalid file format." & _
            vbCrLf & "Please select a valid image file.", vbExclamation, "Error - Gautam Banerjee"


    End If
End Sub
```

This VBA subroutine, named Combobox4_Change, appears to be associated with an event handler for a ComboBox named ComboBox4. Let's break down how this code works:

Private Sub Combobox4_Change()

This line defines a subroutine named Combobox4_Change, which is triggered when the value in ComboBox4 changes (i.e., when a user selects a different item).

Label4.Visible = True

This line sets the Visible property of Label4 to True. This probably means that Label4 was initially hidden, and now it's being made visible.

selectedImageName = ComboBox4.Value

This line gets the currently selected item in ComboBox4 and stores it in the selectedImageName variable. This variable seems to hold the name of an image file.

imagePath = "E:\Chennai_Tours\Excel_Picture\" & selectedImageName & ".jpg"

This line constructs a file path (imagePath) by concatenating the selected image name from ComboBox4 with a base path. It's assumed that the images are located in the folder "E:\Chennai_Tours\Excel_Picture" and have a ".jpg" file extension. This path represents the location of the image file on your system.

Dim ws As Worksheet Set ws = ThisWorkbook.Sheets("PictureData") ' Replace with your actual worksheet name

These lines set the ws variable to represent a specific worksheet named "PictureData" in the current workbook. You should replace "PictureData" with the actual name of your worksheet.

If Len(Dir(imagePath)) > 0 Then

This line checks if the file located at imagePath exists. The Dir function is used to check if a file exists at a specified path.

' Get file information ' Display file information in labels *Image1.Picture = LoadPicture(imagePath)*

Label2.Caption = ws.Cells(ComboBox4.ListIndex + J, "D").Value

Label4.Caption = ws.Cells(ComboBox4.ListIndex + J, "E").Value

If the file exists, this block of code does the following:

Loads and displays the image located at imagePath in Image1.

Retrieves and displays information from the worksheet:

Label2 is updated with a value from column "D" of the selected row in ComboBox4.

Label4 is updated with a value from column "E" of the selected row in ComboBox4.

Else

If the file doesn't exist at imagePath, this block of code is executed.

Image1.Picture = LoadPicture("")

It clears the image in Image1, essentially making it empty.

'MsgBox "Image not found at the specified location or invalid file format." & _ ' vbCrLf & "Please select a valid image file.", vbExclamation, "Error - Gautam Banerjee"

This part seems to be commented out but appears to be part of an error-handling mechanism. It suggests displaying a message box with an error message if the image file is not found. You can uncomment and customize this code if you want to display an error message.

In summary, this subroutine is responsible for updating the interface with information about the selected image when the user changes their selection in ComboBox4. It also handles the case where the selected image file doesn't exist, clearing the image and potentially displaying an error message. The specific data displayed on the interface comes from the worksheet "PictureData."

Combobox4_Change() Sub:

This subroutine is triggered when the selection in ComboBox4 changes.

It updates labels and displays information based on the selected item from ComboBox4.

```
Private Sub ComboRemove()
   Dim ws As Worksheet
   Dim LastRow As Long
   Dim cell As Range

   ' Set the worksheet containing your data
   Set ws = ThisWorkbook.Sheets("NewSheet") ' Replace with your sheet name

   ' Find the last row with data in the worksheet
   LastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).row

   ' Loop through the data and add each item to the ComboBox
   For Each cell In ws.Range("A2:A" & LastRow) ' Assuming your data starts from A2
      ComboBox5.AddItem cell.value
   Next cell
   ComboBox5.value = "You can Remove Item"
End Sub
```

This VBA subroutine, named ComboRemove, appears to be responsible for populating items into a ComboBox named ComboBox5. Let's break down how this code works:

Private Sub ComboRemove()

This line defines a subroutine named ComboRemove.

Dim ws As Worksheet

This declares a variable ws to represent a worksheet.

Dim LastRow As Long

This declares a variable LastRow to store the last row with data in the worksheet.

Dim cell As Range

This declares a variable cell to represent a cell within the worksheet.

' Set the worksheet containing your data *Set ws = ThisWorkbook.Sheets("NewSheet")* ' Replace with your sheet name

This sets the ws variable to represent a specific worksheet named "NewSheet" in the current workbook. You should replace "NewSheet" with the actual name of the worksheet containing your data.

' Find the last row with data in the worksheet *LastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row*

These lines calculate the LastRow by finding the last used row in column "A" of the worksheet. It starts from the bottom (the last row) and goes upwards until it encounters a non-empty cell.

' Loop through the data and add each item to the ComboBox *For Each cell In ws.Range("A2:A" & LastRow)* ' Assuming your data starts from A2 ComboBox5.AddItem cell.Value Next cell

This loop iterates through each cell in the range "A2" to the last used row in column "A" and adds the value of each cell to ComboBox5. This is a way to populate ComboBox5 with the values from the specified range.

ComboBox5.Value = "You can Remove Item"

This line sets the default value of ComboBox5 to "You can Remove Item."

In summary, this subroutine is used to populate ComboBox5 with values from a specific column in a worksheet named "NewSheet." It assumes that the data starts from cell "A2" and goes down to the last used row in column "A." The default value for the ComboBox is set to "You can Remove Item."

ComboRemove() Sub:

This subroutine populates ComboBox5 with items from a specified Excel worksheet (NewSheet).

```
Private Sub CmdRemoveItem_Click()
   Dim SelectedItem As String
   Dim i As Long

   ' Get the selected item from the ComboBox
   SelectedItem = ComboBox5.value

   ' Find the index of the selected item in the ComboBox
   For i = 0 To ComboBox5.ListCount - 1
      If ComboBox5.List(i) = SelectedItem Then
         ' Remove the selected item
         ComboBox5.RemoveItem i
         Exit For
      End If

   Next i
   If ComboBox5.ListCount = 0 Then
      ComboBox5.value = ""
      MsgBox "Nothing to Remove item...Please Quit !", vbInformation, "Gautam Banerjee"
   End If
End Sub
```

This VBA subroutine, named CmdRemoveItem_Click, seems to be associated with a button click event, likely named CmdRemoveItem. It appears to handle the removal of items from a ComboBox named ComboBox5. Let's break down how this code works:

Private Sub CmdRemoveItem_Click()

This line defines a subroutine named CmdRemoveItem_Click, which is associated with the click event of a button named CmdRemoveItem.

Dim SelectedItem As String

This declares a variable SelectedItem to store the selected item from ComboBox5.

Dim i As Long

This declares a variable i to be used as a counter for looping through the items in ComboBox5.

' Get the selected item from the *ComboBox SelectedItem = ComboBox5.Value*

This line retrieves the currently selected item in ComboBox5 and stores it in the SelectedItem variable.

' Find the index of the selected item in the *ComboBox For i = 0 To ComboBox5.ListCount - 1 If ComboBox5.List(i) = SelectedItem Then* ' Remove the selected item ComboBox5.RemoveItem i Exit For End If Next i

This loop iterates through the items in ComboBox5 to find the index of the selected item (SelectedItem). When a match is found, it removes the selected item using ComboBox5.RemoveItem i. The loop exits as soon as a match is found, using Exit For.

If ComboBox5.ListCount = 0 Then ComboBox5.Value = "" MsgBox "Nothing to Remove item...Please Quit !", vbInformation, "Gautam Banerjee" End If

After removing the item, it checks if there are any items left in ComboBox5. If there are no items (ComboBox5.ListCount = 0), it sets the value of ComboBox5 to an empty string, and it displays a message box indicating that there is nothing left to remove.

In summary, this subroutine allows the user to select an item from ComboBox5 and remove it by clicking a button (CmdRemoveItem). If there are no items left in the ComboBox after removal, it shows a message box indicating this.

CmdRemoveItem_Click() Sub:

This subroutine is triggered when a button named CmdRemoveItem is clicked.

It removes the selected item from ComboBox5 and displays a message if there are no items left.

Private Sub ComboMulty()

ComboBox6.AddItem "Chennai Tourist Spot"

ComboBox6.AddItem "Rameswaram Tourist Spot"

ComboBox6.AddItem "Pondichery Mahabalipuram"

```
    ComboBox6.AddItem "Kodaikanal Tourist Spot"
    ComboBox6.AddItem "ECO Park Kolkata"
End Sub
```

This VBA subroutine, named ComboMulty, seems to be responsible for populating items into a ComboBox named ComboBox6. Let's break down how this code works:

Private Sub ComboMulty()

This line defines a subroutine named ComboMulty.

ComboBox6.AddItem "Chennai Tourist Spot"

ComboBox6.AddItem "Rameswaram Tourist Spot"

ComboBox6.AddItem "Pondichery Mahabalipuram"

ComboBox6.AddItem "Kodaikanal Tourist Spot"

ComboBox6.AddItem "ECO Park Kolkata"

These lines add multiple items to the ComboBox ComboBox6. Each ComboBox6.AddItem line adds a new item to the ComboBox. In this case, it's adding five different tourist spot names as items.

In summary, this subroutine ComboMulty is used to populate ComboBox6 with a predefined list of tourist spot names. This list will be displayed as options in ComboBox6 when the user interacts with it.

ComboMulty() Sub:

This subroutine populates ComboBox6 with predefined items (tourist spot names).

```
Private Sub ComboBox6_Change()
    Dim SelectedItem As String
    Dim i As Long
    Dim isSelected As Boolean

    ' Get the selected item from the ComboBox
    SelectedItem = ComboBox6.value
```

```
    ' Check if the item is already selected
    isSelected = False
    For i = 1 To SelectedItems.Count
        If SelectedItems.Item(i) = SelectedItem Then
            ' Item is already selected, so deselect it
            SelectedItems.Remove i
            isSelected = True
            Exit For
        End If
    Next i

    ' If the item is not already selected, add it to the collection
    If Not isSelected Then
        SelectedItems.Add SelectedItem
    End If

    ' Update the label to display selected items
    UpdateLabel
End Sub
```

This VBA subroutine, named ComboBox6_Change, appears to be associated with the change event of a ComboBox, likely named ComboBox6. Let's break down how this code works:

Private Sub ComboBox6_Change()

This line defines a subroutine named ComboBox6_Change, which will be executed when the selected item in ComboBox6 is changed by the user.

Dim SelectedItem As String Dim i As Long Dim isSelected As Boolean

These lines declare some variables:

SelectedItem will store the currently selected item in ComboBox6.

i is used as a loop counter for iterating through a collection.

isSelected is a boolean variable that will help determine if an item is already selected.

SelectedItem = ComboBox6.value

This line gets the currently selected item from ComboBox6 and stores it in the SelectedItem variable.

For i = 1 To SelectedItems.Count

If SelectedItems.Item(i) = SelectedItem Then

SelectedItems.Remove i isSelected = True

Exit For

End If

Next i

This loop checks if the SelectedItem is already present in a collection named SelectedItems. If it's found in the collection, it removes it (deselects it), sets isSelected to True, and exits the loop. This essentially allows the user to toggle the selection of an item in ComboBox6. If it's not found, it proceeds with the next part.

If Not isSelected Then

SelectedItems.Add SelectedItem

End If

Here, if the isSelected variable is False, it means the selected item wasn't found in the collection, so it adds (selects) the SelectedItem to the SelectedItems collection. This allows the user to select or deselect items in the ComboBox by clicking on them.

UpdateLabel

This line calls another subroutine named UpdateLabel. Presumably, this subroutine updates some label or display element to reflect the currently selected items in ComboBox6.

In summary, this code allows the user to select or deselect items in ComboBox6, and it keeps track of the selected items in the SelectedItems collection. It also updates a label (likely displaying the selected items) by calling the UpdateLabel subroutine.

ComboBox6_Change() Sub:

This subroutine is triggered when the selection in ComboBox6 changes.

It manages a collection (SelectedItems) to store selected items and updates a label (MultyLabel) to display selected items.

```
Private Sub UpdateLabel()
    Dim selectedText As String
    selectedText = "Selected Items: "

    ' Concatenate the selected items
    For Each Item In SelectedItems
        selectedText = selectedText & Item & ", "
    Next Item

    ' Display the selected items in the label
    MultyLabel.caption = Left(selectedText, Len(selectedText) - 2) ' Remove the trailing comma and space
End Sub
```

This VBA subroutine, named UpdateLabel, is responsible for updating the caption of a label, presumably named MultyLabel. Here's a breakdown of how it works:

Private Sub UpdateLabel()

This line defines a subroutine named UpdateLabel.

Dim selectedText As String selectedText = "Selected Items: "

These lines declare a variable named selectedText as a string and initialize it with the text "Selected Items: ". This is the initial text that will appear on the label.

' Concatenate the selected items *For Each Item In SelectedItems selectedText = selectedText & Item & ", " Next Item*

This block of code uses a loop to concatenate the items in the SelectedItems collection to the selectedText string. It appears to assume that SelectedItems is a collection containing selected items. Each item is separated by a comma and a space. This loop effectively builds a string that lists all the selected items.

' Display the selected items in the label *MultyLabel.caption = Left(selectedText, Len(selectedText) - 2)* ' Remove the trailing comma and space

This part sets the caption (text) of the MultyLabel to the value of selectedText. However, it also removes the trailing comma and space from the end of the selectedText string. This is done using the Left function with the length of the string reduced by 2.

In summary, this subroutine constructs a text string that lists the selected items and then displays this text in a label (MultyLabel). It ensures that there is no trailing comma and space in the displayed text, making it a clean and user-friendly representation of the selected items. This code is often used when you have a multi-selection control (like a list or combo box) and you want to display the selected items in a more readable format.

This code defines a subroutine in VBA (Visual Basic for Applications) named UpdateLabel(). Let's break down this code step by step and explain it :

Private Sub UpdateLabel()

This line signifies the beginning of a subroutine named UpdateLabel(). Subroutines are blocks of code that can be called to perform specific tasks or actions. In this case, UpdateLabel() is designed to update the text displayed in a label control on a UserForm.

Dim selectedText As String selectedText = "Selected Items: "

Here, a new variable named selectedText is declared as a string. This variable will be used to build a text string that represents selected items.

selectedText is initialized with the text "Selected Items: ". This is the starting text that will be displayed in the label.

' Concatenate the selected items *For Each Item In SelectedItems selectedText = selectedText & Item & ", " Next Item*

This section of the code is a loop that iterates through each item in a collection called SelectedItems. Collections are like lists that can hold multiple items.

For Each Item In SelectedItems starts the loop. It means that for each item (referred to as Item) in the SelectedItems collection, the code inside the loop will execute.

Inside the loop, selectedText is updated. The code *selectedText = selectedText & Item & ", "* appends the current Item to the end of selectedText. It also adds a comma and a space to separate items. This effectively builds a string of all selected items separated by commas and spaces.

The loop continues until it has processed all items in the SelectedItems collection.

' Display the selected items in the label *MultyLabel.Caption = Left(selectedText, Len(selectedText) - 2)* ' Remove the trailing comma and space End Sub

Finally, this part of the code assigns the generated selectedText to the Caption property of a label control named MultyLabel. The Caption property determines what text is displayed in the label on a UserForm.

Before assigning the selectedText to the label, it uses the Left function to remove the trailing comma and space. The *Len(selectedText) - 2* calculates the length of the string minus the last two characters (which are the comma and space), effectively removing them.

In summary, the UpdateLabel() subroutine builds a text string by concatenating items from a collection (SelectedItems) and then displays this text in a label (MultyLabel) on a UserForm. This label is likely used to show the user what items they have selected in a user interface. The code is written in a way that dynamically updates the label text whenever items in the collection change, ensuring that the label always displays an accurate list of selected items.

```
Private Sub ComboRestoreInitia()

    ' Initialize the SelectedItems collection
   Set SelectedItems = New Collection

   ' Populate the ComboBox with items from your "Data" sheet
   Dim ws As Worksheet
   Dim LastRow As Long
   Dim cell As Range

   ' Set the worksheet containing your data
   Set ws = ThisWorkbook.Sheets("DeletedRecord") ' Replace with your sheet name

   ' Find the last row with data in the worksheet
   LastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).row

   ' Loop through the data and add each item to the ComboBox
   For Each cell In ws.Range("A2:A" & LastRow) ' Assuming your data starts from A2
      ComboBox7.AddItem cell.value
   Next cell
End Sub
```

This VBA subroutine, named ComboRestoreInitia, is responsible for initializing a ComboBox (ComboBox7) with items from a worksheet ("DeletedRecord") and initializing a collection (SelectedItems) to manage selected items. Here's a breakdown of how it works:

Private Sub ComboRestoreInitia()

This line defines a subroutine named ComboRestoreInitia.

' Initialize the SelectedItems collection *Set SelectedItems = New Collection*

This part initializes a collection named SelectedItems. Collections are objects in VBA that can store a collection of values, similar to an array. In this case, it's intended to store selected items.

' Populate the ComboBox with items from your "Data" sheet *Dim ws As Worksheet, Dim LastRow As Long, Dim cell As Range* ' Set the worksheet containing your data *Set ws = ThisWorkbook.Sheets("DeletedRecord")* ' Replace with your sheet name

These lines declare variables and set references to the "DeletedRecord" worksheet in your Excel workbook.

' Find the last row with data in the worksheet *LastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).row*

This line calculates the last used row in column A of the "DeletedRecord" worksheet. It finds the last row with data by starting from the bottom of the column and moving upward until it encounters a non-empty cell.

' Loop through the data and add each item to the ComboBox *For Each cell In ws.Range("A2:A" & LastRow)* ' Assuming your data starts from A2 ComboBox7.AddItem cell.value

Next cell

This block of code uses a loop to iterate through the cells in column A (starting from A2 to the last used row) of the "DeletedRecord" worksheet. It adds the value of each cell as an item to ComboBox7. Essentially, it's populating the ComboBox with items from the specified worksheet.

In summary, this subroutine is typically called when initializing a user form or a worksheet. It sets up an empty collection (SelectedItems) to manage selected items and populates a ComboBox (ComboBox7) with items from the "DeletedRecord" worksheet. This is useful when you want to provide users with a list of choices to restore items, and then you can use the SelectedItems collection to keep track of their selections.

ComboRestoreInitia() Sub:

This subroutine populates ComboBox7 with items from a specified Excel worksheet (DeletedRecord).

```
Private Sub UserForm_Initialize()
    Label9.caption = "Today is " & Format(Date, "dddd") & ", Date: " & Format(Date, "dd/mm/yyyy")
    Dim GB As String
```

GB = " Developed by Gautam Banerjee"

Label10.caption = "It looks like a comprehensive VBA UserForm here with various ComboBoxes, Labels, and Buttons for different functionalities. The code seems well-structured and organized, with clear comments to describe each part. It appears that have implemented various features for managing images and data." & GB

Label4.Visible = False

ComboxAddItemsFromFolder

ComboboxAssItemsFromExcelSheet

ComboBoxDynamic1

ComboRemove

ComboMulty

ComboRestoreInitia

End Sub

This VBA subroutine, named UserForm_Initialize, is typically called when initializing a user form in Excel. Its purpose is to set up and configure various elements of the user form, such as labels and combo boxes. Let's break down what this subroutine does:

Private Sub UserForm_Initialize()

This line defines a subroutine named UserForm_Initialize, which gets executed when the user form is initialized.

Label9.caption = "Today is " & Format(Date, "dddd") & ", Date: " & Format(Date, "dd/mm/yyyy")

This line sets the caption property of Label9 to display the current day of the week and the date in the format "Today is [day], Date: [date]." It uses the Format function to format the date.

Dim GB As String GB = " Developed by Gautam Banerjee" Label10.caption = "It looks like a comprehensive VBA UserForm here with various ComboBoxes, Labels, and Buttons for different functionalities. The code seems well-structured and organized, with clear comments to describe each part. It appears that you have implemented various features for managing images and data." & GB

These lines set the caption property of Label10 to display a message. It combines a description of the user form's functionality with the name of the developer, Gautam Banerjee.

Label4.Visible = False

This line hides Label4 by setting its Visible property to False. This can be used to initially hide a label on the user form.

ComboxAddItemsFromFolder

ComboboxAssItemsFromExcelSheet

ComboBoxDynamic1

ComboRemove

ComboMulty

ComboRestoreInitia

These lines call various subroutines that populate combo boxes and set up other elements on the user form. These subroutines were discussed in previous interactions, and they typically populate combo boxes with data from folders or worksheets and initialize collections or other variables.

In summary, the UserForm_Initialize subroutine is responsible for setting up the initial state of the user form when it is loaded. It sets labels, populates combo boxes, and performs other necessary initialization tasks to ensure that the user form is ready for interaction.

UserForm_Initialize() Sub:

This subroutine initializes the UserForm and sets the initial values and labels. It also calls several of the above subroutines to populate ComboBoxes and perform initializations.

In summary, this code appears to be part of an Excel UserForm designed to manage images and data in an organized manner, with various functionalities such as deleting, restoring, and displaying information about images and data from different sources. It uses ComboBoxes, Labels, and Buttons to interact with the user. The code is well-commented, making it easier for you to understand and explain its functionality.

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!

Gautam Banerjee

Age: 63

Pay by Google Pay

9748327614

Copy and Paste Below code :

```
Dim SelectedItems As Collection ' Declare the SelectedItems collection at the module level
Dim J As Long, K As Long, L As Long, M As Long

Private Sub CmdDeleteItem_Click()
    Dim SourceSheet As Worksheet
    Dim TargetSheet As Worksheet
    Dim LastRow As Long
    Dim i As Long
    Dim SelectedItem As Variant
```

```
    ' Set references to the source and target sheets
    Set SourceSheet = ThisWorkbook.Sheets("PictureData")
    Set TargetSheet = ThisWorkbook.Sheets("DeletedRecord")

    ' Find the last used row in the source sheet
    LastRow = SourceSheet.Cells(SourceSheet.Rows.Count, "A").End(xlUp).row

    ' Loop through each selected item
    For Each SelectedItem In SelectedItems
        ' Loop through each row in reverse order to avoid issues with row deletion
        For i = LastRow To 2 Step -1 ' Assuming data starts from row 2
            ' Check if the SpotName in the current row matches the selected item
            If SourceSheet.Cells(i, "A").value = SelectedItem Then
                ' Copy the entire row to the target sheet
                SourceSheet.Rows(i).Copy TargetSheet.Cells(TargetSheet.Rows.Count,
"A").End(xlUp).Offset(1, 0)

                ' Delete the row from the source sheet
                SourceSheet.Rows(i).Delete
            End If
        Next i
    Next SelectedItem

    ' Clear the SelectedItems collection
    Set SelectedItems = New Collection
End Sub
```

```
Private Sub CmdRestoreItem_Click()
    Dim SourceSheet As Worksheet
    Dim TargetSheet As Worksheet
    Dim LastRow As Long
    Dim i As Long
    Dim SelectedItem As Variant

    ' Set references to the source and target sheets
    Set SourceSheet = ThisWorkbook.Sheets("DeletedRecord")
    Set TargetSheet = ThisWorkbook.Sheets("PictureData")

    ' Check if an item is selected in the ComboBoxRestore
    If ComboBox7.ListIndex >= 0 Then
        ' Get the selected item
        SelectedItem = ComboBox7.List(ComboBox7.ListIndex)

        ' Initialize a variable to keep track of the number of rows deleted
        Dim RowsDeleted As Long
        RowsDeleted = 0

        ' Find the last used row in the target sheet
        LastRow = TargetSheet.Cells(TargetSheet.Rows.Count, "A").End(xlUp).row

        ' Loop through each row in the source sheet
        For i = SourceSheet.Cells(SourceSheet.Rows.Count, "A").End(xlUp).row To 2 Step -1 ' Loop in reverse order
            ' Check if the SpotName in the current row matches the selected item
            If SourceSheet.Cells(i, "A").value = SelectedItem Then
                ' Copy the entire row to the target sheet
```

```
            SourceSheet.Rows(i).Copy TargetSheet.Cells(LastRow + RowsDeleted + 1, "A")

            ' Delete the row from the source sheet
            SourceSheet.Rows(i).Delete
            RowsDeleted = RowsDeleted + 1 ' Increment the counter for rows deleted
        End If
    Next i

    ' Clear the ComboBoxRestore selection
    ComboBox7.ListIndex = -1
  Else
    MsgBox "Please select an item to restore.", vbExclamation
  End If
End Sub

Private Sub Image2_Click()
  Unload Me
  MainMenuForm.Show
End Sub

Private Sub ComboxAddItemsFromFolder()
  Dim folderPath As String
  Dim fileName As String

  ' Set the folder path where your picture files are located
  folderPath = "E:\Shanti\"

  ' Loop through each file in the folder
  fileName = Dir(folderPath & "*.jpg") ' Change the file extension as needed
```

```
    ' Populate the ListBox with picture file names
    Do While fileName <> ""
        ComboBox1.AddItem fileName
        fileName = Dir ' Get the next file in the folder
    Loop
    ' Set the default value for ComboBox1
    ComboBox1.value = "Picture from Folder" ' Change to the item you want as the default
End Sub

Private Sub ComboBox1_Change()
    Label4.Visible = False
    selectedImageName = ComboBox1.value
    Dim imagePath As String
    imagePath = "E:\Shanti\" & selectedImageName '& ".jpg"
    If Len(Dir(imagePath)) > 0 Then
        ' Get file information
        Dim file As Object
        Set file = CreateObject("Scripting.FileSystemObject").GetFile(imagePath)

        ' Display file information in labels
        Image1.Picture = LoadPicture(imagePath)
        LabelFileSize.caption = "File Size: " & Format(file.Size / 1024, "0.00") & " KB"
        LabelDimensions.caption = "Dimensions: " & GetImageDimensions(imagePath)
        LabelCreationDate.caption = "Creation Date: " & Format(file.DateCreated, "dd-mmm-yyyy")
        LabelFolder.caption = "File Location: " & imagePath
        LabelClickedBy.caption = "Clicked by : Gautam Banerjee"
    Else
        ' Clear labels if the image file does not exist
```

```
        'MsgBox "Image not found at the specified location or invalid file format." & _
            vbCrLf & "Please select a valid image file.", vbExclamation, "Error - Gautam Banerjee"
        LabelFileSize.caption = "File Size: N/A"
        LabelDimensions.caption = "Dimensions: N/A"
        LabelCreationDate.caption = "Creation Date: N/A"
        LabelFolder.caption = "Folder Location: N/A"
        LabelClickedBy.caption = "Invalid Picture File Name"
    End If
    Label2.caption = "Santiniketan was established by Debendranath Tagore in 1863, and then developed by his son, Rabindranath Tagore in West Bengal"
End Sub

Function GetImageDimensions(imagePath As String) As String
    ' This function returns the dimensions of an image in the format "Width x Height"
    Dim img As Object
    Set img = CreateObject("WIA.ImageFile")
    img.LoadFile imagePath
    GetImageDimensions = img.Width & " x " & img.Height
End Function

Private Sub ComboboxAssItemsFromExcelSheet()

    Dim ws As Worksheet
    Dim cell As Range

    ' Set the worksheet where customer names are stored
    Set ws = ThisWorkbook.Sheets("PictureData")

    ' Loop through the worksheet to populate the ComboBox
```

```
    For Each cell In ws.Range("B2:B" & ws.Cells(ws.Rows.Count, "B").End(xlUp).row)
        ComboBox2.AddItem cell.value
    Next cell
    ' Set the default value for ComboBox1
    ComboBox2.value = "Picture from Sheet" ' Change to the item you want as the default
End Sub

Private Sub ComboBox2_Change()
    Label4.Visible = True
    selectedImageName = ComboBox2.value
    imagePath = "E:\Chennai_Tours\Excel_Picture\" & selectedImageName & ".jpg"

    Dim ws As Worksheet
    Set ws = ThisWorkbook.Sheets("PictureData") ' Replace with your actual worksheet name
    If Len(Dir(imagePath)) > 0 Then
        ' Get file information

        ' Display file information in labels
        Image1.Picture = LoadPicture(imagePath)
        Label2.caption = ws.Cells(ComboBox2.ListIndex + 2, "D").value
        Label4.caption = ws.Cells(ComboBox2.ListIndex + 2, "E").value
    Else
        ' Clear labels if the image file does not exist
        Image1.Picture = LoadPicture("")
        'MsgBox "Image not found at the specified location or invalid file format." & _
            vbCrLf & "Please select a valid image file.", vbExclamation, "Error - Gautam Banerjee"

    End If
```

```
End Sub

Private Sub ComboBoxDynamic1()

    Dim ws As Worksheet
    Set ws = ThisWorkbook.Sheets("PictureData")

    Dim cell As Range
    Dim uniqueLocations As Collection
    Set uniqueLocations = New Collection

    On Error Resume Next
    For Each cell In ws.Range("A2:A" & ws.Cells(ws.Rows.Count, "A").End(xlUp).row)
        uniqueLocations.Add cell.value, CStr(cell.value)
    Next cell
    On Error GoTo 0

    Dim Loc As Variant ' Declare Loc as a variant

    For Each Loc In uniqueLocations
        ComboBox3.AddItem Loc
    Next Loc
    ComboBox3.value = "First Select Tourist Spot" ' Change to the item you want as the default
 End Sub

Private Sub Combobox3_Change()
    Dim selectedValue As String
    Dim ws As Worksheet
    Dim i As Long
```

```
' Clear the list box
ComboBox4.Clear

' Get the selected item from the combo box
selectedValue = ComboBox3.value

' Set your worksheet (change the sheet name if needed)
Set ws = ThisWorkbook.Sheets("PictureData")

' Loop through the data in the worksheet
For i = 2 To ws.Cells(ws.Rows.Count, "A").End(xlUp).row
   If ws.Cells(i, 1).value = selectedValue Then
      ' Add matching items to the list box
      ComboBox4.AddItem ws.Cells(i, 2).value
   End If
Next i

Select Case ComboBox3
   Case "Chennai Tourist Spot"
   J = 2

   Case "Rameswaram Tourist Spot"
   J = 15

   Case "Pondichery Mahabalipuram"
   J = 25
```

```
    Case "Kodaikanal Tourist Spot"
    J = 35

    Case "ECO Park Kolkata"
    J = 45

  End Select

End Sub

Private Sub Combobox4_Change()
  Label4.Visible = True
  selectedImageName = ComboBox4.value
  imagePath = "E:\Chennai_Tours\Excel_Picture\" & selectedImageName & ".jpg"


  Dim ws As Worksheet
  Set ws = ThisWorkbook.Sheets("PictureData") ' Replace with your actual worksheet name
  If Len(Dir(imagePath)) > 0 Then
    ' Get file information

    ' Display file information in labels
    Image1.Picture = LoadPicture(imagePath)
    Label2.caption = ws.Cells(ComboBox4.ListIndex + J, "D").value
    Label4.caption = ws.Cells(ComboBox4.ListIndex + J, "E").value
  Else
    ' Clear labels if the image file does not exist
    Image1.Picture = LoadPicture("")
    'MsgBox "Image not found at the specified location or invalid file format." & _
```

```
        vbCrLf & "Please select a valid image file.", vbExclamation, "Error - Gautam Banerjee"

    End If
End Sub

Private Sub ComboRemove()
    Dim ws As Worksheet
    Dim LastRow As Long
    Dim cell As Range

    ' Set the worksheet containing your data
    Set ws = ThisWorkbook.Sheets("NewSheet") ' Replace with your sheet name

    ' Find the last row with data in the worksheet
    LastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).row

    ' Loop through the data and add each item to the ComboBox
    For Each cell In ws.Range("A2:A" & LastRow) ' Assuming your data starts from A2
        ComboBox5.AddItem cell.value
    Next cell
    ComboBox5.value = "You can Remove Item"
End Sub

Private Sub CmdRemoveItem_Click()
    Dim SelectedItem As String
    Dim i As Long

    ' Get the selected item from the ComboBox
    SelectedItem = ComboBox5.value
```

```
    ' Find the index of the selected item in the ComboBox
    For i = 0 To ComboBox5.ListCount - 1
        If ComboBox5.List(i) = SelectedItem Then
            ' Remove the selected item
            ComboBox5.RemoveItem i
            Exit For
        End If

    Next i
    If ComboBox5.ListCount = 0 Then
        ComboBox5.value = ""
        MsgBox "Nothing to Remove item...Please Quit !", vbInformation, "Gautam Banerjee"
    End If
End Sub

Private Sub ComboMulty()
    ComboBox6.AddItem "Chennai Tourist Spot"
    ComboBox6.AddItem "Rameswaram Tourist Spot"
    ComboBox6.AddItem "Pondichery Mahabalipuram"
    ComboBox6.AddItem "Kodaikanal Tourist Spot"
    ComboBox6.AddItem "ECO Park Kolkata"
End Sub

Private Sub ComboBox6_Change()
    Dim SelectedItem As String
    Dim i As Long
    Dim isSelected As Boolean
```

```
    ' Get the selected item from the ComboBox
    SelectedItem = ComboBox6.value

    ' Check if the item is already selected
    isSelected = False
    For i = 1 To SelectedItems.Count
        If SelectedItems.Item(i) = SelectedItem Then
            ' Item is already selected, so deselect it
            SelectedItems.Remove i
            isSelected = True
            Exit For
        End If
    Next i

    ' If the item is not already selected, add it to the collection
    If Not isSelected Then
        SelectedItems.Add SelectedItem
    End If

    ' Update the label to display selected items
    UpdateLabel
End Sub

Private Sub UpdateLabel()
    Dim selectedText As String
    selectedText = "Selected Items: "

    ' Concatenate the selected items
    For Each Item In SelectedItems
```

```
        selectedText = selectedText & Item & ", "
    Next Item

    ' Display the selected items in the label
    MultyLabel.caption = Left(selectedText, Len(selectedText) - 2) ' Remove the trailing comma and space
End Sub

Private Sub ComboRestoreInitia()

        ' Initialize the SelectedItems collection
    Set SelectedItems = New Collection

    ' Populate the ComboBox with items from your "Data" sheet
    Dim ws As Worksheet
    Dim LastRow As Long
    Dim cell As Range

    ' Set the worksheet containing your data
    Set ws = ThisWorkbook.Sheets("DeletedRecord") ' Replace with your sheet name

    ' Find the last row with data in the worksheet
    LastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).row

    ' Loop through the data and add each item to the ComboBox
    For Each cell In ws.Range("A2:A" & LastRow) ' Assuming your data starts from A2
        ComboBox7.AddItem cell.value
    Next cell
End Sub
```

```
Private Sub UserForm_Initialize()

    Label9.caption = "Today is " & Format(Date, "dddd") & ", Date: " & Format(Date, "dd/mm/yyyy")

    Dim GB As String

    GB = " Developed by Gautam Banerjee"

    Label10.caption = "It looks like a comprehensive VBA UserForm here with various ComboBoxes, Labels, and Buttons for different functionalities. The code seems well-structured and organized, with clear comments to describe each part. It appears that have implemented various features for managing images and data." & GB


    Label4.Visible = False

    ComboxAddItemsFromFolder

    ComboboxAssItemsFromExcelSheet

    ComboBoxDynamic1

    ComboRemove

    ComboMulty

    ComboRestoreInitia

End Sub
```