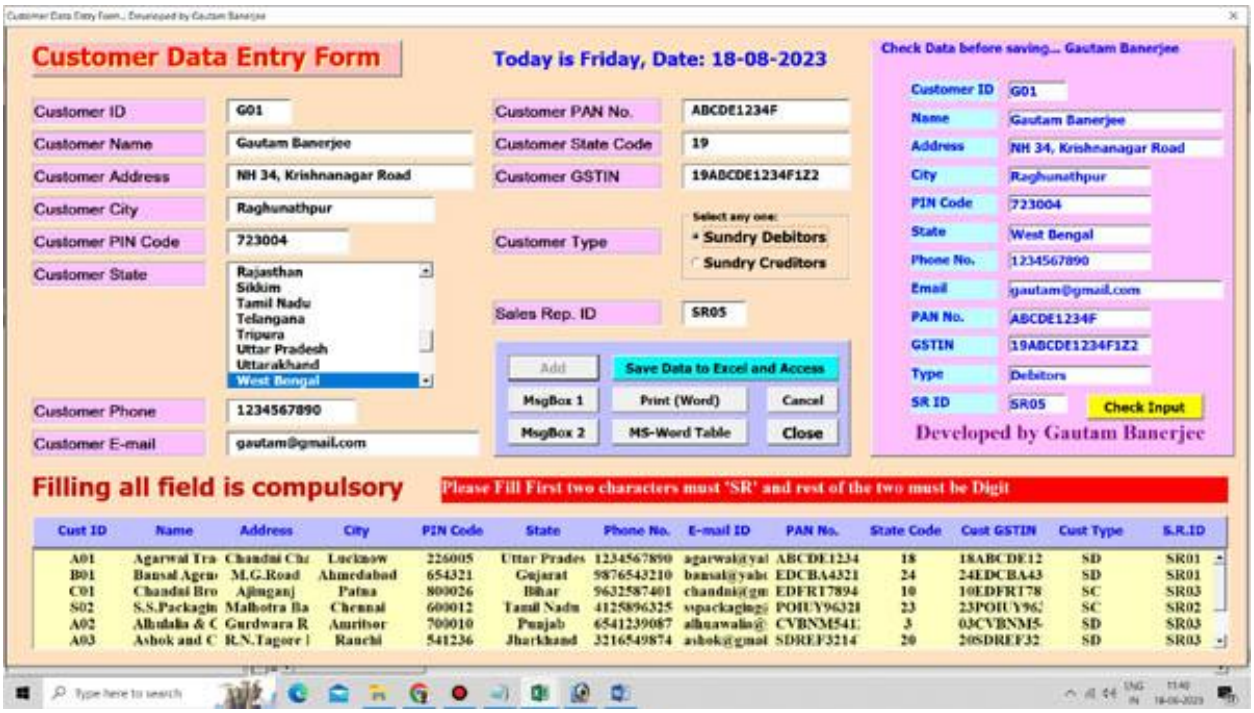


Designing a Customer Master Entry Form: Validation and Duplicate Entry Prevention

Discover how to create a robust Customer Master Entry Form that ensures data accuracy and prevents duplicate entries. In this tutorial, we'll guide you through the process of designing a user-friendly form for entering customer details. Learn how to implement validation rules that validate user input and enforce data integrity. We'll also show you step-by-step how to prevent duplicate entries using smart coding techniques. By the end of this video, you'll have the skills to build an efficient, error-free data entry system. Don't miss this opportunity to enhance your coding and form design skills!

Getting Started with Designing a Customer Master Entry Form

Description: Embarking on the journey of creating a Customer Master Entry Form? This video tutorial is your perfect guide. Learn the foundational steps to design an

effective form for entering customer details. From layout planning to choosing the right input fields, we'll walk you through the essential elements of form design. Get ready to create a user-friendly and visually appealing form that streamlines data entry. Whether you're a beginner or looking to enhance your design skills, this tutorial is your first step towards crafting a professional Customer Master Entry Form.

When designing a form using Excel VBA, you can use a combination of built-in Excel features and the VBA programming environment. Here are the tools you'll need:

Microsoft Excel: Microsoft Excel itself serves as the primary environment for designing and building forms using VBA. It provides the necessary tools for creating user interfaces and working with data.

UserForm Designer: Excel includes a UserForm designer that allows you to create customized forms with various controls such as textboxes, buttons, checkboxes, list boxes, and more. You can access the UserForm designer by opening the Visual Basic for Applications (VBA) editor and inserting a UserForm.

Visual Basic for Applications (VBA) Editor: The VBA editor within Excel is where you write the code that powers your UserForm. It provides a code window for writing event handlers, procedures, and functions that control the behavior of your form.

Controls Palette: The UserForm designer includes a controls palette that allows you to drag and drop controls onto the form. You can customize these controls' properties and link them to VBA code.

Code Editor: The VBA editor provides a code editor where you write VBA code to handle form events, validate input, and perform other actions.

Programming Language Documentation: Access to VBA documentation, tutorials, and forums can provide guidance on writing code and handling form-related tasks.

Testing Environment: Excel itself serves as your testing environment. You can run your UserForm from the VBA editor to see how it behaves and make necessary adjustments.

Graphics Software (Optional): If you want to include custom graphics or icons on your form, you can use graphics software like Adobe Photoshop or similar tools.

With Excel's built-in tools and the VBA programming environment, you have everything you need to design and develop forms for data entry and user interaction.

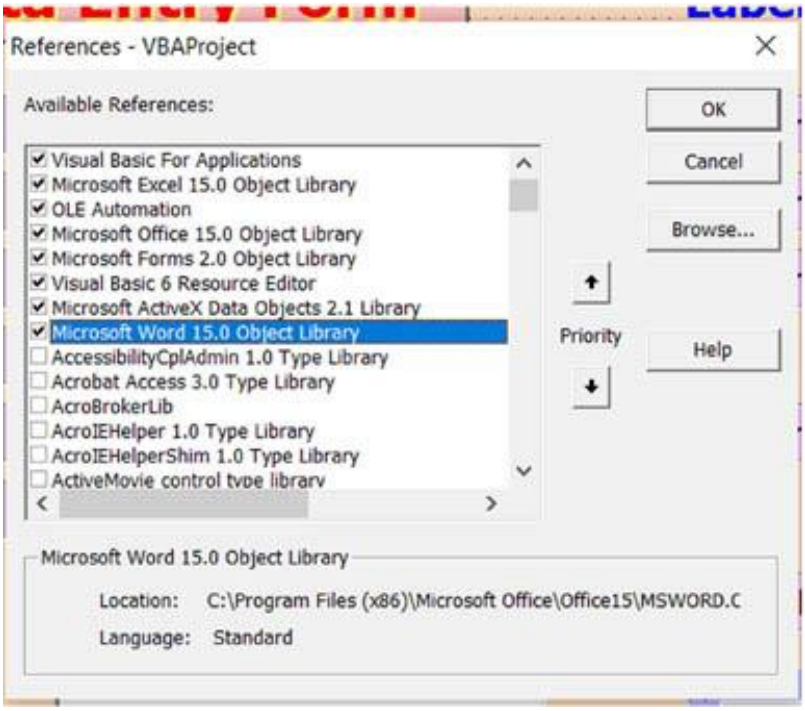
A toolbox is an indispensable resource that empowers developers to create software efficiently, consistently, and with enhanced functionality. It enables them to harness the

power of pre-built components while also fostering creativity and innovation in application development.

Save the Excel workbook as “Excel Macro Enabled Workbook”

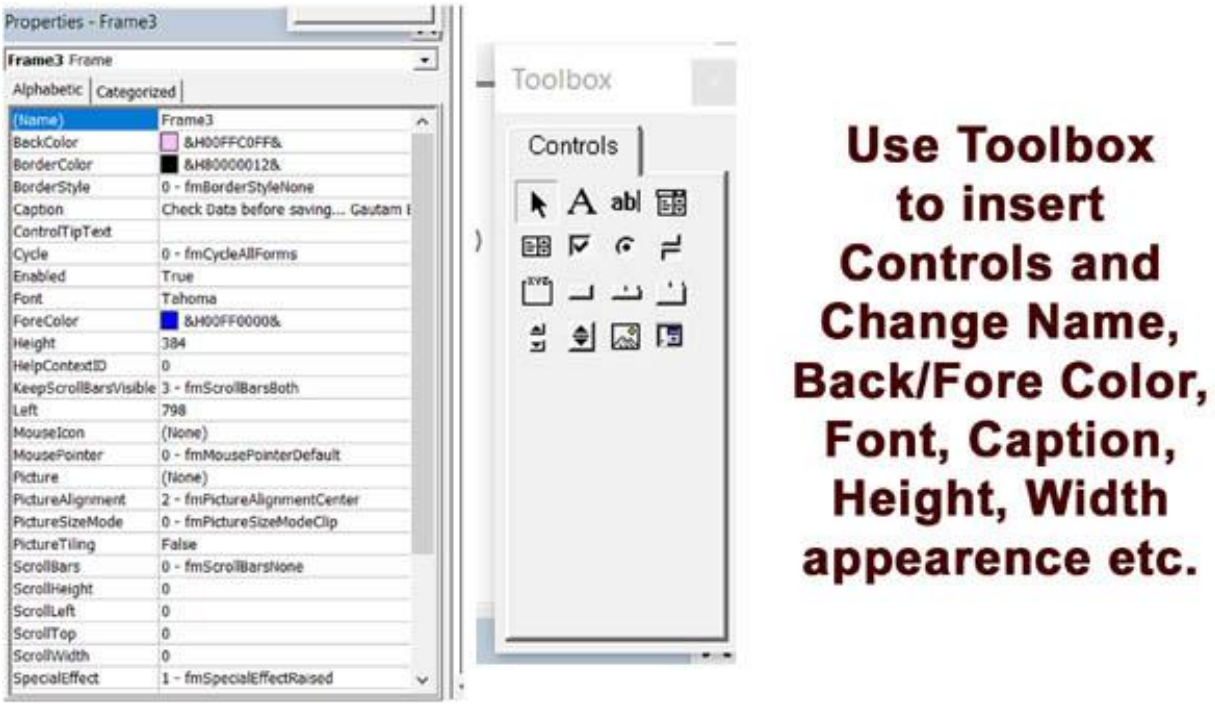
A macro-enabled sheet refers to an Excel workbook that contains macros, which are scripts or recorded actions that automate tasks in Excel. These macros are written in the Visual Basic for Applications (VBA) programming language and can perform a wide range of actions, from automating repetitive tasks to enhancing the functionality of Excel.

A macro-enabled sheet is typically saved with the ".xlsm" file extension. The "m" in ".xlsm" stands for "macro-enabled." When you open a macro-enabled sheet, Excel may prompt you to enable macros if they are disabled in your Excel settings. Enabling macros allows the workbook's code to run and automate various actions.

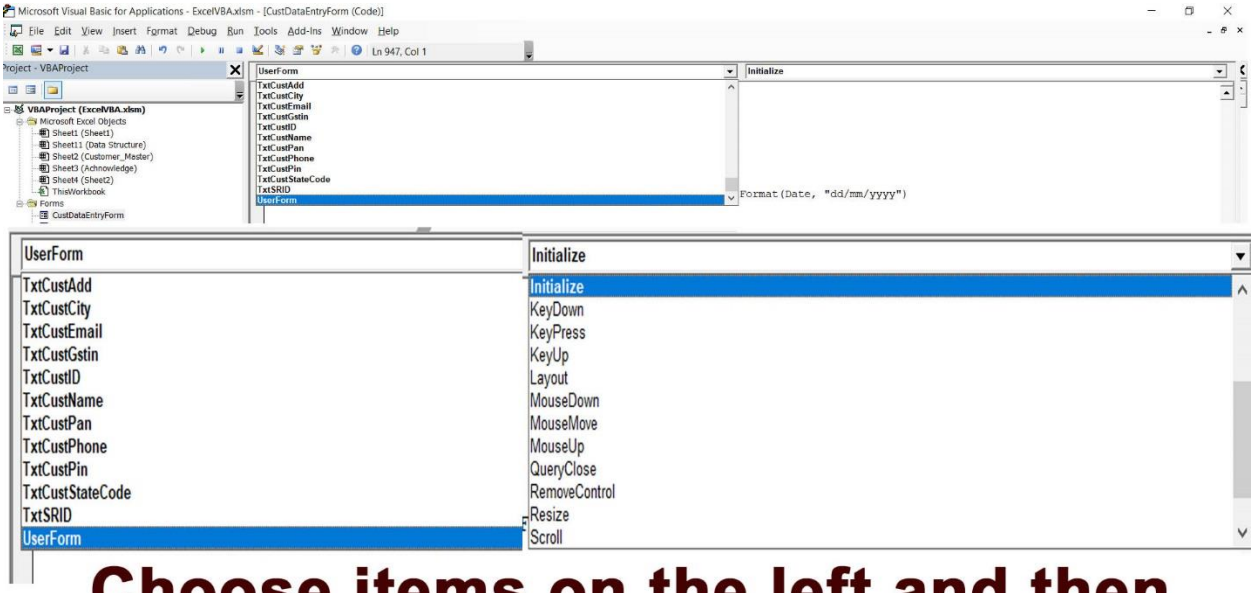


Navigate to the "Tools" menu. Choose "References" from the menu. In the list of available references, mark the checkboxes next to the items as shown in the picture.

Before you start coding, it's important to select the referenced library mentioned above



All the necessary tools are available in the Toolbox window, and make sure to set the properties for the selected tools.



Importance of Naming the tools

Naming your tools, such as TextBoxes, ListBoxes, and CommandButtons, is important for several reasons in programming, including VBA (Visual Basic for Applications) development. Proper naming enhances the readability, maintainability, and overall organization of your code. Here's why naming is important and how to name your tools effectively:

Readability and Understanding: Descriptive names make it clear what each tool is used for. When you or others read the code, it's easier to understand the purpose of each element without needing to refer back to the form or document.

Maintainability: Well-named tools make it easier to maintain and update your code. If you need to make changes or debug an issue, you can quickly identify which element is involved and make modifications accordingly.

Avoiding Confusion: Meaningful names prevent confusion when you have multiple tools on a form or within your project. This is particularly important as projects grow in complexity.

Collaboration: If you're working on a project with others, having clear names ensures everyone is on the same page and understands the structure of the code.

Code Reusability: Good naming conventions help you identify opportunities for code reuse. You can more easily spot patterns and elements that can be reused in different parts of your code.

Tips for Naming Tools:

Use Descriptive Names: Choose names that accurately describe the purpose of the tool. For example, if you have a TextBox for entering customer names, you might name it "TxtCustomerName."

Use Camel Case: Capitalize the first letter of each word in the name (except the first word) to enhance readability. For example, "TxtCustomerName."

Include the Type: Include a prefix that indicates the type of tool. For example, "Txt" for TextBox, "Lst" for ListBox, and "Cmd" for CommandButton.

Be Consistent: Use a consistent naming convention throughout your project. This helps maintain a uniform structure and makes your code more predictable.

Avoid Abbreviations: While concise names are important, avoid excessive abbreviations that might make the name unclear. Strive for a balance between brevity and clarity.

Use Intuitive Names: Choose names that would make sense to someone else reading your code. Avoid using jargon or overly creative names that might confuse others.

Consider Context: If your project involves multiple forms or sections, consider including some context in the name. For example, "FrmCustomerNameTxt" for a TextBox on the "Customer Name" form.

Refactor as Needed: As your project evolves, you might find that certain names need adjustment. Don't hesitate to refactor your code to improve naming clarity.

By following these naming conventions, you'll enhance the readability, maintainability, and overall quality of your VBA code and make it easier for yourself and others to work with the tools and elements in your forms and projects.

Designing a UserForm in VBA

Designing a UserForm in VBA involves creating an interface that is user-friendly, visually appealing, and efficient to navigate. Here are some tips to consider when designing a UserForm:

Plan the Layout: Before you start designing, plan the layout of your UserForm. Decide where each control (TextBoxes, Labels, Buttons, etc.) will be placed and how they will be organized.

Keep It Simple: Avoid cluttering the UserForm with too many controls. Keep the design clean and focused on the essential information and actions.

Consistent Alignment: Maintain a consistent alignment for controls. Align controls vertically and horizontally to create a neat and organized appearance.

Use Grouping: Group related controls together using frames or labels to visually organize the form and make it easier for users to understand the content.

Proper Labeling: Use clear and concise labels for each control. Labels should describe the purpose of the control or provide instructions to the user.

Readable Font and Size: Choose a readable font and an appropriate font size for labels and controls. Avoid using fonts that are too small or difficult to read.

Use White Space: Leave enough white space between controls to prevent a cluttered appearance. White space improves readability and makes the form more visually appealing.

Logical Flow: Arrange controls in a logical order that follows the user's expected interaction flow. For example, place navigation buttons (Next, Previous) in a consistent location.

Use Color Thoughtfully: Use color to draw attention to important controls or sections, but avoid overwhelming the form with too many colors. Stick to a color scheme that is easy on the eyes.

Error Handling: If your form involves user input, consider including error handling mechanisms like error messages or tooltips to guide users in case of input errors.

Responsive Design: Design the form to be responsive, meaning it should look and function well on different screen sizes and resolutions.

Icons and Images: If appropriate, use icons or images to make the form visually appealing and to help users understand the purpose of certain controls.

Use User-Friendly Names: As mentioned earlier, name your controls with user-friendly and descriptive names that convey their purpose.

Test the Form: Test the UserForm as a user would. Ensure that controls are working as intended and that the navigation is smooth.

Feedback and Validation: Provide feedback to users when they perform actions. For instance, show a message when data is successfully submitted or a record is updated.

Focus Control: Set the initial focus on the most relevant control, such as the first input field, for easy navigation.

Resize and Rescale: If your UserForm might be used on different computers or monitors, make sure it resizes and rescales properly to fit various screen sizes.

Remember that UserForm design is not just about aesthetics; it's about creating a functional and user-friendly interface that enhances the user experience. Regularly seek feedback from users to identify areas for improvement and refinement.

Checklist of things to keep in mind

Before creating a Customer Master Entry Form in Excel VBA, there are several important factors to consider. Here's a checklist of things to keep in mind:

Data Requirements: Determine the fields and information you need to collect from customers. This may include customer name, address, contact details, account type, etc.

User Interface: Design a user-friendly interface that makes it easy for users to input and manage customer data. Consider using labels, textboxes, dropdowns, and other controls for a clear layout.

Data Validation: Implement validation for the data entered by users. Ensure that required fields are filled, and provide error messages for incorrect or incomplete entries.

Data Storage: Decide where and how the customer data will be stored. You can use Excel worksheets, Access databases, or other data storage methods.

Data Security: If the customer data contains sensitive information, implement security measures to protect it from unauthorized access.

User Access and Permissions: Determine who will have access to the Customer Master Entry Form and what level of permissions they will have.

Functionality: Define the functionality you want in the form, such as adding new customers, editing existing records, deleting records, and navigating through records.

User Experience: Focus on creating a smooth and intuitive user experience. Ensure that the form is easy to navigate and understand.

Error Handling: Plan for error handling mechanisms to inform users about errors and guide them to correct the issues.

Buttons and Actions: Decide on the necessary buttons and actions for the form. Common buttons include "Add New," "Save," "Edit," "Delete," "Navigate," and "Close."

Data Integrity: Implement measures to ensure data integrity, such as avoiding duplicate entries and maintaining consistent data formats.

Backup and Recovery: Have a backup plan in place in case of data loss. Regularly backup the data to prevent any major disruptions.

Testing: Thoroughly test the form before deploying it. Check all functionalities, data entry scenarios, error handling, and user interactions.

Documentation: Document the form's purpose, functionality, and usage guidelines for future reference.

As for the number of command buttons, it depends on the features and functionality you want to provide in the form. Here are a few common command buttons that could be included:

Add New: Adds a new customer record to the database.

Edit: Allows users to edit an existing customer record.

Save: Saves the data entered or edited.

Delete: Deletes a selected customer record (consider providing a confirmation prompt).

Navigate: Provides navigation options to move through the customer records.

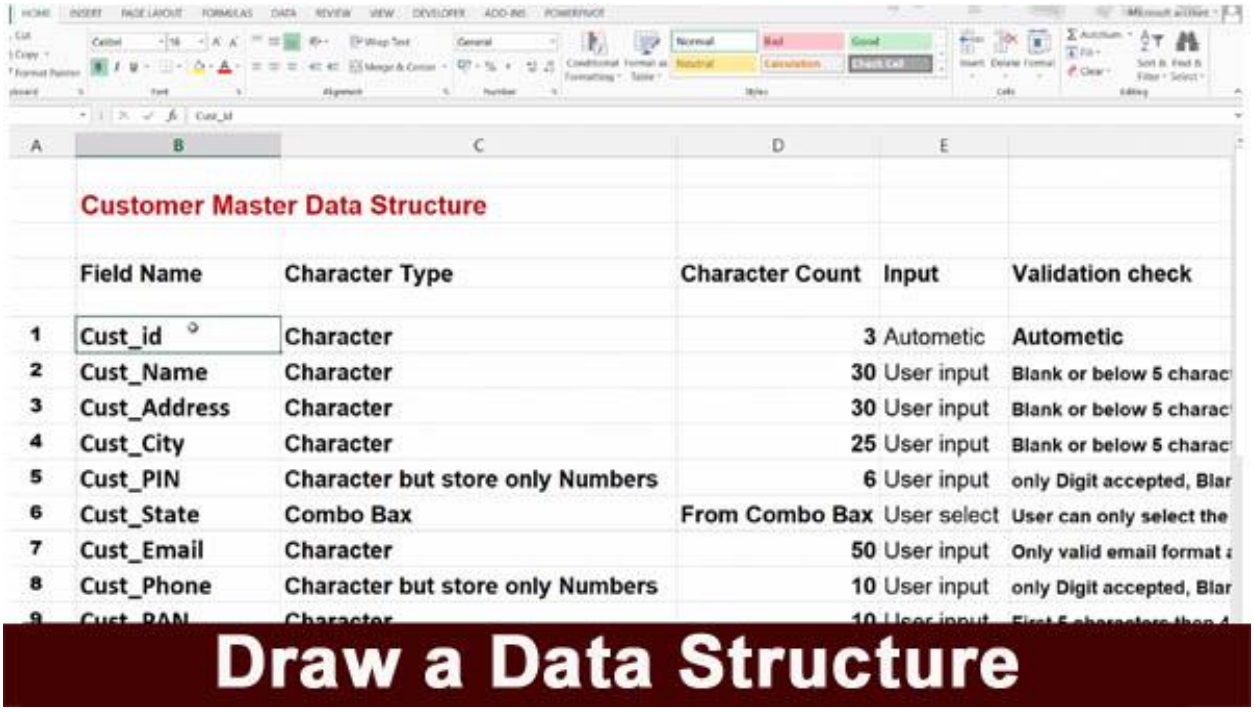
Close: Closes the form.

Search: Enables users to search for specific customer records.

Clear: Clears the form fields for entering new data.

Refresh: Reloads the data or form.

Ultimately, the number and type of buttons will depend on your specific requirements and the functionality you want to offer to users. It's important to strike a balance between providing essential actions and keeping the form uncluttered.



Customer Master Data Structure

	Field Name	Character Type	Character Count	Input	Validation check
1	Cust_id	Character	3	Autometic	Autometic
2	Cust_Name	Character	30	User input	Blank or below 5 charac
3	Cust_Address	Character	30	User input	Blank or below 5 charac
4	Cust_City	Character	25	User input	Blank or below 5 charac
5	Cust_PIN	Character but store only Numbers	6	User input	only Digit accepted, Blar
6	Cust_State	Combo Bax	From Combo Bax	User select	User can only select the
7	Cust_Email	Character	50	User input	Only valid email format a
8	Cust_Phone	Character but store only Numbers	10	User input	only Digit accepted, Blar

Ensure to list the following column names exactly as provided in the 'Customer_Master' sheet to avoid any program error messages.

Cust_ID
Cust_Name
Cust_Address
Cust_City
Cust_PIN
Cust_State

Cust_Phone
Cust_Email
Cust_PAN
Cust_State_Code
Cust_GSTIN
Cust_Type
SR_ID
Cust_Tag

Please make sure to use the exact names for the Textboxes, Listbox, Command Buttons, etc. Otherwise, the program might display an error message.

(Name of Textbox, ListBox and Option Button)

TxtCustID

TxtCustName

TxtCustAdd

TxtCustCity

StatesList (ListBox)

TxtCustPin

TxtCustPhone

TxtCustEmail

TxtCustPan

TxtCustStateCode

TxtCustGstin

CustSD (Option Button)

CustSC (Option Button)

TxtSRID

(Name of Command Button)

Add : CmdAdd

Save : CmdSave

Cancel : CmdCancel

Close : CommandButton1

MsgBox 1 : CmdMsgboxText

Msgbox 2 : CmdDataReview

Print Word : CmdAckPrint

Ms-Word Table : CmdPrint

Writing code involves a systematic process to ensure your code is organized, readable, and functional. Here's a step-by-step procedure to help you get started:

Understand the Objective: Clearly understand what your code needs to achieve. Break down the task into smaller steps if necessary.

Plan Your Logic: Plan the logic of how you'll accomplish the task. Consider the flow, conditions, loops, and variables you'll need.

Open the Code Window: In your development environment (e.g., Excel VBA Editor), open the code window for the object you're working with (e.g., a module, worksheet, or userform).

Start with a Subroutine or Function Declaration: If you're writing a new subroutine (a block of code that performs a specific task), start with a declaration. For example:

Private Sub MySubroutine()

If you're writing a function (returns a value), use:

Private Function MyFunction() As DataType

`Private Sub MySubroutine()`: Add comments to describe what your code will do. Use comments to provide context and explanations for complex parts.

Write the Code: Start writing the actual code based on your planned logic. Make sure to use proper indentation and formatting to improve readability.

Use Variables: Declare and use variables to store and manipulate data. Choose meaningful variable names for clarity.

Implement Control Structures: Use control structures like If...Then...Else, For and Do loops, Select Case, etc., as needed.

Handle Errors: Implement error handling using constructs like On Error Resume Next or On Error GoTo to handle unexpected errors gracefully.

Test Incrementally: Test your code frequently as you write it. Run it step-by-step using breakpoints, and monitor variable values to ensure they're as expected.

Refactor and Optimize: Once your code works, review it for areas where you can make it more efficient or concise. Remove duplicate code and ensure best practices.

Document Your Code: Provide comments to explain complex sections, assumptions, and any dependencies. Good documentation helps others understand your code.

Run and Debug: Run the code in your application. Debug and fix any errors or unexpected behavior. Use breakpoints, watches, and the Immediate window for debugging.

Test Thoroughly: Test your code under various scenarios to ensure it works as expected. Include edge cases and boundary conditions.

Final Review: Review your code one last time for readability, logic, and efficiency. Make any necessary adjustments.

Save and Share: Save your code and back it up. If you're collaborating, share your code with others and ensure they understand it.

Remember that practice makes perfect. The more you write code, the more comfortable you'll become with the process. Additionally, don't hesitate to refer to documentation, tutorials, and examples when you encounter challenges or want to learn new techniques.

Now Design the Form, Copy and Paste the code in the Code Window.

Save the Project (Save the Excel workbook as "Excel Macro Enabled Workbook". Sheet Name should be : "Customer_Master"

Where to save your projects…

Where to save your projects and workbooks depends on your workflow and preferences. Here are some tips to help you decide where to save your projects and how to organize them effectively:

Create a Dedicated Folder: Start by creating a dedicated folder on your computer or a network drive to store all your projects. This helps keep your projects organized and easily accessible.

Subfolders for Each Project: Within your main projects folder, create subfolders for each individual project. This prevents clutter and makes it easy to locate specific projects.

Project Naming Convention: Give your projects meaningful and descriptive names. This helps you quickly identify what each project is about.

Use Version Control: If you're working on larger projects or collaborating with others, consider using version control systems like Git. These systems track changes, help with collaboration, and provide a history of your code.

Backup Regularly: Regularly back up your projects to prevent data loss. Use external drives, cloud storage, or dedicated backup software.

Use Cloud Storage: Cloud storage services like Google Drive, Dropbox, or OneDrive can be useful for storing and syncing your projects across devices.

Save Your Workbook: For Excel projects, save your workbook in the project folder. You can create a subfolder within your project folder to store related files (e.g., data files, images).

Organize Files: Within your project folder, organize files into relevant subfolders. For example, keep code files separate from data files.

Backup Code: Regularly save your code in separate text files (e.g., .txt files) outside the workbook. This ensures you have a backup even if the workbook becomes corrupted.

Use Descriptive File Names: Give your files descriptive names that reflect their content. This makes it easier to find what you need later.

Document Your Projects: Create a README.txt or README.md file in your project folder to document project details, instructions, and dependencies.

Versioning: If your project has multiple versions, consider appending version numbers or dates to the project folder or file names.

Clean Up Unused Files: Regularly review your projects and remove any unused or unnecessary files to keep things organized.

Keep It Consistent: Establish a consistent folder structure and naming convention to make it easier to manage and locate your projects.

Personalize Your Workflow: Ultimately, find an organization method that works best for you and suits your workflow.

Remember that organization is key to maintaining your projects and making them easy to manage and share. Whether you're working on personal projects or collaborating with others, a well-organized file structure can save you time and reduce frustration.

Now Start Coding:

UserForm_Initialize:

UserForm_Initialize is a special event procedure in Excel VBA that is automatically triggered when a UserForm is initialized or loaded. It is a built-in event that allows you to perform specific actions or set initial values when a UserForm is displayed to the user.

When you open or display a UserForm, Excel VBA automatically looks for a procedure named UserForm_Initialize within the code module of that UserForm. If such a procedure exists, it will be executed before the UserForm becomes visible to the user.

How to …. Watch Videos for Live Example

Learn VBA Coding for Excel | How to Start Excel with VBA | Part 01 :

https://youtu.be/dW2Zt--Xddo

The VBA code should always begin with "Private Sub ...()" and end with "End Sub" to define the start and end of a subroutine or event procedure.

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

```
Private Sub UserForm_Initialize()

    CmdSave.Enabled = False
    txtFieldsDisabled
    TxtLength
    TxtBoxBlank
    PopulateStatesList

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


    Dim ws As Worksheet
    Set ws = ThisWorkbook.Worksheets("Customer_Master")

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

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

    ListBox1.Clear
    ListBox1.List = rng.Value

 End Sub
```

The UserForm_Initialize event procedure is a powerful tool within Excel VBA that is automatically activated when a UserForm is loaded. It plays a crucial role in setting up the UserForm's initial state, enhancing user experience, and preparing data for interaction. In this particular example:

CmdSave.Enabled = False ensures that a "Save" button remains disabled initially, allowing the user to fill in required data before enabling the save action.

txtFieldsDisabled presumably handles the disabling of certain text fields to restrict user input until specific conditions are met.

TxtLength and TxtBoxBlank likely enforce validation rules by checking the length and empty status of textboxes respectively, ensuring valid inputs.

PopulateStatesList populates a dropdown or list with states, making it convenient for users to select from predefined options.

lblDate.Caption dynamically generates and displays the current date in a label, enhancing the form's real-time information.

A worksheet named "Customer_Master" is accessed from the workbook.

The code calculates the last used row in column A of the "Customer_Master" sheet, ensuring that the UserForm updates with the latest data.

A range of data (columns A to M) is captured from rows 2 to the last used row, populating a list box (ListBox1) with relevant information.

In essence, this UserForm_Initialize procedure orchestrates a synchronized set of actions as the UserForm is initialized. It ensures the UserForm is armed with the necessary controls, data, and validation mechanisms, guaranteeing a smooth user experience while enhancing data accuracy and integrity.

Call this procedure to UserForm_initialized event (txtFieldsEnabled)

```
Sub txtFieldsEnabled()
    TxtCustID.Enabled = True
    TxtCustName.Enabled = True
    TxtCustAdd.Enabled = True
    TxtCustCity.Enabled = True
    TxtCustPin.Enabled = True
    TxtCustPhone.Enabled = True
    TxtCustEmail.Enabled = True
    TxtCustPan.Enabled = True
    TxtCustStateCode.Enabled = True
```

```
    TxtCustGstin.Enabled = True
    TxtSRID.Enabled = True
    StatesList.Enabled = True
    CustSD.Enabled = True
    CustSC.Enabled = True
End Sub
```

The provided code defines a subroutine named txtFieldsEnabled that seemingly pertains to enabling a set of textboxes and controls within a UserForm. This subroutine is likely utilized in scenarios where user input is permitted or required. Each control mentioned is enabled, granting users the ability to interact with and input data.

Here's a brief description of the code's purpose:

The txtFieldsEnabled subroutine serves to enable multiple textboxes and controls within a UserForm at once. This action allows users to input data or make selections within those controls. It is a convenient way to transition from a state where input is restricted to one where users can interact with the form's elements. This subroutine proves valuable in scenarios where you want to give users the capability to provide information or make selections after certain conditions have been met.

For instance, if the UserForm initially opens with some fields disabled and requires users to perform specific actions before enabling those fields, this subroutine can be invoked to activate those fields once those conditions are satisfied. This enhances user-friendliness by dynamically adapting the UserForm's functionality based on user actions or data availability.

Call this procedure to UserForm_initialized event (TxtLength)

```
Sub TxtLength()

    TxtCustID.MaxLength = 3
    TxtCustName.MaxLength = 25
    TxtCustAdd.MaxLength = 30
    TxtCustCity.MaxLength = 25
```

TxtCustPin.MaxLength = 6

TxtCustPhone.MaxLength = 10

TxtCustEmail.MaxLength = 40

TxtCustPan.MaxLength = 10

TxtCustStateCode.MaxLength = 2

TxtCustGstin.MaxLength = 15

TxtSRID.MaxLength = 4

End Sub

The provided code snippet defines a subroutine named TxtLength that appears to be related to setting the maximum lengths (character limits) for various textboxes within a UserForm. This subroutine is likely intended to enforce character restrictions on user input, preventing entries that exceed predefined lengths. Here's a description of the code's function:

The TxtLength subroutine is responsible for limiting the maximum number of characters that users can input into specific textboxes within a UserForm. Each textbox mentioned in the code snippet has its MaxLength property set to a certain value. This property dictates the maximum number of characters that can be entered into the respective textbox.

For instance, if TxtCustName.MaxLength is set to 25, users will only be able to enter up to 25 characters in the "Customer Name" textbox. Similarly, the code sets character limits for other fields like address, city, PIN code, phone number, email, and more.

By implementing this subroutine, you ensure that users cannot input excessive text that could potentially disrupt the layout of the form or exceed the available space. It helps maintain consistent formatting and aids in data validation by preventing input that doesn't adhere to specified length requirements.

Call this procedure to UserForm_initialized event (TxtBoxBlank)

```
Sub TxtBoxBlank()
    TxtCustID.Text = ""
```

```
    TxtCustName.Text = ""
    TxtCustAdd.Text = ""
    TxtCustCity.Text = ""
    TxtCustPin.Text = ""
    TxtCustPhone.Text = ""
    TxtCustEmail.Text = ""
    TxtCustPan.Text = ""
    TxtCustStateCode.Text = ""
    TxtCustGstin.Text = ""
    TxtSRID.Text = ""

    Label29.Caption = ""
    Label30.Caption = ""
    Label31.Caption = ""
    Label32.Caption = ""
    Label33.Caption = ""
    Label34.Caption = ""
    Label35.Caption = ""
    Label36.Caption = ""
    Label37.Caption = ""
    Label38.Caption = ""
    Label39.Caption = "
End Sub
```

The provided code snippet defines a subroutine named TxtBoxBlank that seems to be related to clearing the content of various textboxes and labels within a UserForm. This subroutine is likely used to reset or initialize the form's input fields and associated labels to empty states. Here's a description of the code's function:

The TxtBoxBlank subroutine serves the purpose of clearing the contents of multiple textboxes and labels within a UserForm. Each textbox specified in the code (TxtCustID,

TxtCustName, etc.) is reset to an empty state using the .Text property, effectively removing any existing input.

In addition to textboxes, the subroutine also seems to clear the captions of various labels (e.g., Label29, Label30, etc.), possibly associated with error messages, warnings, or information display.

By utilizing this subroutine, you provide users with a convenient way to clear all input fields and reset any displayed messages, allowing them to start fresh or make corrections without the need to manually delete each entry. This enhances user experience and can be particularly useful in situations where users want to re-enter data or correct mistakes easily.

Call this procedure to UserForm_initialized event (PopulateStatesList)

```
Sub PopulateStatesList()
    Dim statesArray As Variant
    statesArray = Array("Andhra Pradesh", "Arunachal Pradesh", "Assam", "Bihar", "Chhattisgarh", "Goa", "Gujarat", _
        "Haryana", "Himachal Pradesh", "Jharkhand", "Karnataka", "Kerala", "Madhya Pradesh", "Maharashtra", _
        "Manipur", "Meghalaya", "Mizoram", "Nagaland", "Odisha", "Punjab", "Rajasthan", "Sikkim", _
        "Tamil Nadu", "Telangana", "Tripura", "Uttar Pradesh", "Uttarakhand", "West Bengal", "Andaman and Nicobar Islands", _
        "Chandigarh", "Delhi", "Dadra and Nagar Haveli and Daman and Diu", "Jammu and Kashmir", "Ladakh", "Lakshadweep", "Puducherry")
Me.StatesList.Clear
Me.StatesList.List = statesArray
End Sub
```

The code defines a subroutine named PopulateStatesList that is used to populate a list or dropdown control within a UserForm with a predefined array of Indian states and union territories. Here's a breakdown of the code's functionality:

A variant array named statesArray is created. This array contains the names of Indian states and union territories as individual elements.

The code clears the existing items from the StatesList control within the UserForm. This ensures that the control is empty before populating it with new data.

The List property of the StatesList control is assigned the statesArray. This action populates the control with the elements from the array, displaying the list of states and union territories as options for users to choose from.

This subroutine is useful for providing users with a standardized list of options for selecting their state or union territory. By dynamically loading the list from an array, you ensure that any updates or changes to the list of states are automatically reflected in the UserForm, reducing the need for manual adjustments. This approach enhances user-friendliness and data accuracy when collecting information related to geographic location.

Day and Date Display : (This line already in UserForm Initialize)

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

The provided code sets the caption of a label control (lblDate) within a UserForm to display the current day of the week and the current date in a specific format. Here's a breakdown of the code:

lblDate.Caption: This refers to the caption property of the label control named lblDate.

"Today is ": This is a static text that is displayed at the beginning of the label's caption.

Format(Date, "dddd"): This part of the code uses the Format function to format the current date (Date) as the full name of the day of the week (e.g., "Monday", "Tuesday", etc.). The "dddd" format specifier represents the full day name.

& ", Date: ": This is another static text that is added after the day of the week.

Format(Date, "dd/mm/yyyy"): This part of the code formats the current date in the format "dd/mm/yyyy", where "dd" represents the day, "mm" represents the month, and "yyyy" represents the year.

When this code is executed, the label's caption will be updated to show a message like "Today is Monday, Date: 12/08/2023" (assuming the current date is August 12, 2023, and it's a Monday). This can provide users with real-time information about the current day and date when interacting with the UserForm.

```
Dim ws As Worksheet
    Set ws = ThisWorkbook.Worksheets("Customer_Master")
    Dim lastRow As Long
    lastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row
    Dim rng As Range
    Set rng = ws.Range("A2:M" & lastRow) ' Assuming data starts from A2
    ListBox1.Clear
    ListBox1.List = rng.Value
```

The code snippet performs the following actions within an Excel VBA macro:

It defines a worksheet variable named ws and assigns it the reference to the worksheet named "Customer_Master" in the current workbook (ThisWorkbook).

It calculates the last used row in column A of the "Customer_Master" worksheet using the .Cells(ws.Rows.Count, "A").End(xlUp).Row expression. This determines the upper boundary of the range of data to be processed.

It defines a range variable named rng and sets it to cover a range from cell A2 to the last used row in column M ("A2:M" & lastRow). This range is assumed to include the data, and it will be used to populate the list box.

It clears the items currently present in ListBox1.

It populates ListBox1 with the values from the defined range (rng.Value).

In summary, this code segment retrieves data from the "Customer_Master" worksheet, starting from cell A2 and extending to the last used row in column M. It then clears the existing items in ListBox1 and fills it with the retrieved data, effectively displaying the data in a list format for users to view. This kind of operation is often used to display data stored in Excel worksheets within a user-friendly interface, such as a UserForm's list box, to allow users to interact with and select specific data entries.

Add Button Code : (Copy this Code)

```
Private Sub CmdAdd_Click()
    txtFieldsEnabled
    Me.TxtCustID.SetFocus
```

```
    CmdAdd.Enabled = False
    CmdSave.Enabled = True
End Sub
```

The code snippet defines a subroutine named CmdAdd_Click that appears to be associated with the "Add" button within a UserForm. This subroutine is likely executed when the user clicks the "Add" button. Here's a breakdown of the code's functionality:

txtFieldsEnabled: This subroutine is invoked, enabling various textboxes and controls within the UserForm. This action likely allows the user to start entering data after clicking the "Add" button.

Me.TxtCustID.SetFocus: This line sets the focus to the "TxtCustID" textbox, presumably after enabling the fields. This action ensures that the user's input cursor is placed in the specified textbox, indicating that it's ready to receive user input.

CmdAdd.Enabled = False: This line disables the "Add" button after it has been clicked. This could prevent users from clicking the button multiple times while processing the addition of data.

This code segment is a typical part of a UserForm's interaction logic. It prepares the UserForm for data entry by enabling input fields, setting focus to the appropriate textbox, and managing the button's state. Additionally, the commented-out lines allow for testing with preset data before users start entering their own information.

Call this sub to add button sub : txtFieldsEnabled (**Copy below code**)

```
Sub txtFieldsEnabled()

    TxtCustID.Enabled = True
    TxtCustName.Enabled = True
    TxtCustAdd.Enabled = True
    TxtCustCity.Enabled = True
    TxtCustPin.Enabled = True
    TxtCustPhone.Enabled = True
    TxtCustEmail.Enabled = True
    TxtCustPan.Enabled = True
```

```
    TxtCustStateCode.Enabled = True
    TxtCustGstin.Enabled = True
    TxtSRID.Enabled = True
    StatesList.Enabled = True
    CustSD.Enabled = True
    CustSC.Enabled = True
End Sub
```

The provided code snippet defines a subroutine named txtFieldsEnabled that seems to be related to enabling a set of textboxes, dropdowns, and controls within a UserForm. This subroutine is likely used to activate input fields, allowing users to interact with and provide data. Here's a description of the code's function:

The txtFieldsEnabled subroutine serves the purpose of enabling multiple textboxes, dropdowns, and controls within a UserForm. Each control specified in the code (e.g., TxtCustID, TxtCustName, TxtCustAdd, etc.) is set to an enabled state using the .Enabled property.

By invoking this subroutine, you make the designated controls active and available for user input. This action is particularly useful when transitioning from a state where input is restricted (e.g., when a form is initially displayed) to a state where users are allowed to provide data. It enhances user experience by adapting the form's interface to different stages of interaction, enabling users to enter information seamlessly.

Customer_ID Validation Code : (**Copy below code in TxtCustID Textbox**)

```
Private Sub TxtCustID_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
     ' Restrict Cust_ID to three characters: 1 alphabet + 2 numeric
    If Len(Me.TxtCustID.Value) = 0 Then

        ' Allow only uppercase alphabets as the first character
        If Not (KeyAscii >= 65 And KeyAscii <= 90) Then
            KeyAscii = 0 ' Cancel the keypress
```

```
        MsgBox "Please enter an uppercase alphabet as the first character.", vbExclamation, "Invalid Input...Gautam Banerjee"
    End If
ElseIf Len(Me.TxtCustID.Value) = 1 Or Len(Me.TxtCustID.Value) = 2 Then
    ' Allow only numeric characters for the second and third characters
    If Not (KeyAscii >= 48 And KeyAscii <= 57) Then
        KeyAscii = 0 ' Cancel the keypress
        MsgBox "Please enter numeric digits for the second and third characters.", vbExclamation, "Invalid Input...Gautam Banerjee"
    End If
ElseIf Len(Me.TxtCustID.Value) >= 3 Then
    ' Prevent entering more than 3 characters
    KeyAscii = 0 ' Cancel the keypress
    MsgBox "Cust_ID should be exactly 3 characters.", vbExclamation, "Invalid Input...Gautam Banerjee"
End If
End Sub
```

The code snippet defines an event handler named TxtCustID_KeyPress, which is associated with the KeyPress event of a textbox control named TxtCustID within a UserForm. This event handler is used to validate and control user input as they type characters into the TxtCustID textbox. Here's a breakdown of the code's functionality:

KeyAscii As MSForms.ReturnInteger: The parameter KeyAscii represents the ASCII value of the key being pressed by the user. It's used to control the behavior of the keypress.

The code first checks if the length of the TxtCustID value is zero, indicating that no characters have been entered yet. If this is the case:

It ensures that only uppercase alphabetic characters (A-Z) are allowed as the first character of the TxtCustID. If the pressed key doesn't match this criteria, the KeyAscii value is set to 0 to cancel the keypress, and a message is displayed.

If the length of TxtCustID value is 1 or 2 (i.e., the first character has been entered):

It restricts input to numeric characters (0-9) for the second and third characters. If the pressed key is not a numeric digit, the KeyAscii value is set to 0, and a message is displayed.

If the length of TxtCustID value is 3 or more:

It prevents further characters from being entered, effectively limiting the Cust_ID to exactly 3 characters. If the pressed key is any character, the KeyAscii value is set to 0, and a message is displayed.

In summary, this event handler enforces specific validation rules on the TxtCustID textbox as the user types. It ensures that only uppercase alphabetic characters and numeric digits are entered in the correct sequence, while also limiting the total length to 3 characters. This type of validation enhances data integrity by preventing incorrect or incomplete entries as users input data.

Learn VBA Coding for Excel | How to Start Excel with VBA | Part 01 :

https://youtu.be/dW2Zt--Xddo

Gautam Banerjee

Age: 63

Pay by UPI

9748327614

Customer_Name Validation Code : (**Copy below code in TxtCustName Textbox)**

```
Private Sub TxtCustName_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    Dim CName As String
    CName = Trim(Me.TxtCustName.Text)
    Instruction.Caption = "Name should be minimum 3 Characters"
    If Len(CName) < 3 Then
      MsgBox "Customer name Can not Blank", vbExclamation, "Invalid input"
      Cancel = True
      Me.TxtCustName.SetFocus
    End If
     Label29.Caption = Me.TxtCustID.Text
End Sub
```

The code snippet defines an event handler named TxtCustName_Exit, which is associated with the Exit event of a textbox control named TxtCustName within a UserForm. This event handler is used to validate the customer's name input when the focus leaves the TxtCustName textbox. Here's a breakdown of the code's functionality:

Cancel As MSForms.ReturnBoolean: The parameter Cancel is used to control whether the focus remains on the control after the event handler has executed. If set to True, the focus will stay on the TxtCustName control.

Dim CName As String: Declares a variable named CName to store the trimmed content of the TxtCustName textbox.

CName = Trim(Me.TxtCustName.Text): Trims any leading or trailing spaces from the text in the TxtCustName textbox and assigns the result to the CName variable.

Instruction.Caption = "Name should be minimum 3 Characters": Sets the caption of an unspecified control (likely a label) named Instruction to provide an instruction to the user.

If the length of the trimmed CName is less than 3 characters:

A message box is displayed to inform the user that the customer name cannot be blank.

The Cancel parameter is set to True, preventing the focus from leaving the TxtCustName control.

The focus is set back to the TxtCustName control, ensuring that the user must correct the input.

In summary, this event handler ensures that the customer's name input is valid before allowing the focus to move away from the TxtCustName control. It displays messages to guide the user and prevent them from entering insufficient or invalid data.

Copy below Code:

```
Private Sub TxtCustName_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
    Instruction.Caption = "Name should be minimum 3 Characters"
    Instruction.BackColor = vbBlue
End Sub
```

The code snippet defines an event handler named TxtCustName_KeyPress, which is associated with the KeyPress event of a textbox control named TxtCustName within a UserForm. This event handler is used to provide a visual cue to the user when they press a key while entering data into the TxtCustName textbox. Here's a breakdown of the code's functionality:

KeyAscii As MSForms.ReturnInteger: The parameter KeyAscii represents the ASCII value of the key being pressed by the user. It's used to control the behavior of the keypress.

Instruction.Caption = "Name should be minimum 3 Characters": Sets the caption of an unspecified control (likely a label) named Instruction to provide an instruction to the user.

Instruction.BackColor = vbBlue: Sets the background color of the same Instruction control to blue. This action changes the visual appearance of the control to draw the user's attention.

In summary, this event handler updates the Instruction control's caption to remind the user about the requirement for a minimum of 3 characters in the customer name. Additionally, it changes the background color of the control to blue, creating a visual indicator that helps draw the user's attention to the specific instruction. This kind of feedback can be helpful in guiding users as they input data into the form.

Customer_Address Validation Code : (**Copy below code in TxtCustAdd Textbox)**

```
Private Sub TxtCustAdd_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    Dim address As String
    address = Trim(Me.TxtCustAdd.Text)
    If Len(address) < 5 Then
        MsgBox "Customer Address should be at least 5 characters long.", vbExclamation, "Invalid input... Gautam Banerjee"
        Cancel = True
        Me.TxtCustAdd.SetFocus
    End If
    Label30.Caption = Me.TxtCustName.Text
End Sub
```

The provided code snippet defines an event handler named TxtCustAdd_Exit, which is associated with the Exit event of a textbox control named TxtCustAdd within a UserForm. This event handler is used to validate the customer's address input when the focus leaves the TxtCustAdd textbox. Here's a breakdown of the code's functionality:

Cancel As MSForms.ReturnBoolean: The parameter Cancel is used to control whether the focus remains on the control after the event handler has executed. If set to True, the focus will stay on the TxtCustAdd control.

Dim address As String: Declares a variable named address to store the trimmed content of the TxtCustAdd textbox.

address = Trim(Me.TxtCustAdd.Text): Trims any leading or trailing spaces from the text in the TxtCustAdd textbox and assigns the result to the address variable.

If the length of the trimmed address is less than 5 characters:

A message box is displayed to inform the user that the customer address should be at least 5 characters long.

The Cancel parameter is set to True, preventing the focus from leaving the TxtCustAdd control.

The focus is set back to the TxtCustAdd control, ensuring that the user must correct the input.

In summary, this event handler ensures that the customer's address input is valid before allowing the focus to move away from the TxtCustAdd control. It displays messages to guide the user and prevent them from entering insufficient or invalid data.

Customer_City Validation Code : (**Copy below code in TxtCustCity Textbox)**

```
Private Sub TxtCustcity_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    Dim City As String
    City = Trim(Me.TxtCustCity.Text)
    If Len(City) < 3 Then
      MsgBox "Customer City should be at least 3 characters long.", vbExclamation, "Invalid input... Gautam Banerjee"
      Cancel = True
      Me.TxtCustCity.SetFocus
    End If
    Label31.Caption = Me.TxtCustAdd.Text
End Sub

Private Sub TxtCustCity_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
    Instruction.BackColor = vbYellow
    Instruction.ForeColor = vbBlue
    Instruction.Caption = "City should be minimum 3 Characters"
End Sub
```

The provided code snippet defines an event handler named TxtCustcity_Exit, which is associated with the Exit event of a textbox control named TxtCustCity within a UserForm. This event handler is used to validate the customer's city input when the focus leaves the TxtCustCity textbox. Here's a breakdown of the code's functionality:

Cancel As MSForms.ReturnBoolean: The parameter Cancel is used to control whether the focus remains on the control after the event handler has executed. If set to True, the focus will stay on the TxtCustCity control.

Dim City As String: Declares a variable named City to store the trimmed content of the TxtCustCity textbox.

City = Trim(Me.TxtCustCity.Text): Trims any leading or trailing spaces from the text in the TxtCustCity textbox and assigns the result to the City variable.

If the length of the trimmed City is less than 3 characters:

A message box is displayed to inform the user that the customer city should be at least 3 characters long.

The Cancel parameter is set to True, preventing the focus from leaving the TxtCustCity control.

The focus is set back to the TxtCustCity control, ensuring that the user must correct the input.

In summary, this event handler ensures that the customer's city input is valid before allowing the focus to move away from the TxtCustCity control. It displays messages to guide the user and prevent them from entering insufficient or invalid data.

Customer_PIN Validation Code : (**Copy below code in TxtCustPIN Textbox)**

```
Private Sub TxtCustPin_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    Dim Pin As String
    Pin = Me.TxtCustPin.Text
  If Len(Pin) <> 6 Or Not IsNumeric(Pin) Then
      MsgBox "Customer PIN Code should be exactly 6 digits and all numeric", vbExclamation, "Invalid input... Gautam Banerjee"
      Cancel = True
      Me.TxtCustPin.SetFocus
    End If

  Label32.Caption = Me.TxtCustCity.Text
End Sub

Private Sub TxtCustPin_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
    Instruction.BackColor = vbBlue
    Instruction.ForeColor = vbWhite
```

```
    Instruction.Caption = "PIN code should be minimum 6 Characters long and all must be Digit"

End Sub
```

The code snippet defines an event handler named TxtCustPin_Exit, which is associated with the Exit event of a textbox control named TxtCustPin within a UserForm. This event handler is used to validate the customer's PIN code input when the focus leaves the TxtCustPin textbox. Here's a breakdown of the code's functionality:

Cancel As MSForms.ReturnBoolean: The parameter Cancel is used to control whether the focus remains on the control after the event handler has executed. If set to True, the focus will stay on the TxtCustPin control.

Dim Pin As String: Declares a variable named Pin to store the content of the TxtCustPin textbox.

Pin = Me.TxtCustPin.Text: Assigns the content of the TxtCustPin textbox to the Pin variable.

If the length of the Pin is not equal to 6 digits or if the content of Pin is not entirely numeric:

A message box is displayed to inform the user that the customer PIN code should be exactly 6 digits and consist of numeric characters only.

The Cancel parameter is set to True, preventing the focus from leaving the TxtCustPin control.

The focus is set back to the TxtCustPin control, ensuring that the user must correct the input.

In summary, this event handler ensures that the customer's PIN code input is valid before allowing the focus to move away from the TxtCustPin control. It displays messages to guide the user and prevent them from entering insufficient or invalid data.

The code snippet defines an event handler named TxtCustPin_KeyPress, which is associated with the KeyPress event of a textbox control named TxtCustPin within a UserForm. This event handler is used to provide a visual cue to the user when they press a key while entering data into the TxtCustPin textbox. Here's a breakdown of the code's functionality:

KeyAscii As MSForms.ReturnInteger: The parameter KeyAscii represents the ASCII value of the key being pressed by the user. It's used to control the behavior of the keypress.

Instruction.BackColor = vbBlue: Sets the background color of an unspecified control (likely a label) named Instruction to blue. This action changes the visual appearance of the control to draw the user's attention.

Instruction.ForeColor = vbWhite: Sets the text color of the same Instruction control to white. This further enhances the visibility of the text against the blue background.

Instruction.Caption = "PIN code should be minimum 6 Characters long and all must be Digit": Sets the caption of the Instruction control to provide an instruction to the user regarding the requirements for the PIN code. This text appears with the modified visual appearance (blue background and white text) to ensure that it stands out.

In summary, this event handler updates the Instruction control's appearance and caption to remind the user about the validation rules for the PIN code. The blue background and white text combination is used to draw the user's attention to the instruction while they input data into the TxtCustPin textbox.

State List Validation Code : (**Copy below code in StatesList Listbox)**

```
Private Sub StatesList_Click()
    Dim SelectedStates As String
    SelectedStates = StatesList.Value
    Dim StateCode As String
    StateCode = GetStateGSTIN(SelectedStates)
    TxtCustStateCode.Value = StateCode
    Instruction.BackColor = vbRed
    Instruction.Caption = "Phone should be 10 characters long and must be numeric"

    Label33.Caption = Me.TxtCustPin.Text
End Sub
```

The code snippet defines an event handler named StatesList_Click, which is associated with the Click event of a combo box control named StatesList within a UserForm. This

event handler is used to perform several actions when the user selects a state from the dropdown list. Here's a breakdown of the code's functionality:

Dim SelectedStates As String: Declares a variable named SelectedStates to store the selected state from the StatesList combo box.

SelectedStates = StatesList.Value: Assigns the selected value from the StatesList combo box to the SelectedStates variable.

Dim StateCode As String: Declares a variable named StateCode to store the GSTIN (Goods and Services Tax Identification Number) state code for the selected state.

StateCode = GetStateGSTIN(SelectedStates): Calls a function named GetStateGSTIN and passes the SelectedStates value to it. This function is likely used to retrieve the state code associated with the selected state.

TxtCustStateCode.Value = StateCode: Sets the value of a textbox control named TxtCustStateCode to the value of the StateCode variable. This may be used to display the GSTIN state code.

Instruction.BackColor = vbRed: Sets the background color of an unspecified control (likely a label) named Instruction to red. This action changes the visual appearance of the control to draw the user's attention.

Instruction.Caption = "Phone should be 10 characters long and must be numeric": Sets the caption of the Instruction control to provide an instruction to the user regarding the validation rules for the phone number. This text appears with the modified visual appearance (red background).

In summary, this event handler updates the UI and provides instructions to the user when they select a state from the StatesList combo box. It displays the GSTIN state code and an instruction about the validation rules for the phone number while drawing attention to the instruction using a red background.

Customer Phone No. Validation Code : (**Copy below code in TxtCustPhone Textbox)**

```
Private Sub TxtCustPhone_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    Dim Fone As String
    Fone = Me.TxtCustPhone.Text
    If Len(Fone) <> 10 Or Not IsNumeric(Fone) Then
```

```
        MsgBox "Please enter valid Mobile no. (10 Digit)", vbExclamation, "Invalid input...Gautam Banerjee"
        Cancel = True
        Me.TxtCustPhone.SetFocus
    End If
    Label34.Caption = Me.StatesList.Text
End Sub
```

The provided code snippet defines an event handler named TxtCustPhone_Exit, which is associated with the Exit event of a textbox control named TxtCustPhone within a UserForm. This event handler is used to validate the customer's mobile phone number input when the focus leaves the TxtCustPhone textbox. Here's a breakdown of the code's functionality:

Cancel As MSForms.ReturnBoolean: The parameter Cancel is used to control whether the focus remains on the control after the event handler has executed. If set to True, the focus will stay on the TxtCustPhone control.

Dim Fone As String: Declares a variable named Fone to store the content of the TxtCustPhone textbox.

Fone = Me.TxtCustPhone.Text: Assigns the content of the TxtCustPhone textbox to the Fone variable.

If the length of the Fone is not equal to 10 digits or if the content of Fone is not entirely numeric:

A message box is displayed to inform the user that they need to enter a valid 10-digit mobile phone number.

The Cancel parameter is set to True, preventing the focus from leaving the TxtCustPhone control.

The focus is set back to the TxtCustPhone control, ensuring that the user must correct the input.

In summary, this event handler ensures that the customer's mobile phone number input is valid before allowing the focus to move away from the TxtCustPhone control. It displays messages to guide the user and prevent them from entering insufficient or invalid data.

Customer Email Validation Code : (**Copy below code in TxtCustEmail Textbox)**

```
Private Sub TxtCustEmail_Exit(ByVal Cancel As MSForms.ReturnBoolean)
Dim Email As String
Email = Me.TxtCustEmail.Text
If Not IsValidEmail(Email) Then
    MsgBox "Please enter a valid email address", vbExclamation, "Invalid input...Gautam Banerjee"
    Cancel = True
    Me.TxtCustEmail.SetFocus
End If
Label35.Caption = Me.TxtCustPhone.Text
End Sub

Private Function IsValidEmail(ByVal Email As String) As Boolean
    Dim regex As Object
    Set regex = CreateObject("VbScript.RegExp")
    regex.Pattern = "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"
    IsValidEmail = regex.test(Email)
End Function

Private Sub TxtCustEmail_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
    Instruction.BackColor = vbBlue
    Instruction.Caption = "Email should be right E-mail Format (i.e.gincom1@yahoo.com)"
End Sub
```

The code snippet defines an event handler named TxtCustEmail_Exit, which is associated with the Exit event of a textbox control named TxtCustEmail within a UserForm. This event handler is used to validate the customer's email address input when the focus leaves the TxtCustEmail textbox. It appears to use a separate function named

IsValidEmail to perform the email validation. Here's a breakdown of the code's functionality:

Cancel As MSForms.ReturnBoolean: The parameter Cancel is used to control whether the focus remains on the control after the event handler has executed. If set to True, the focus will stay on the TxtCustEmail control.

Dim Email As String: Declares a variable named Email to store the content of the TxtCustEmail textbox.

Email = Me.TxtCustEmail.Text: Assigns the content of the TxtCustEmail textbox to the Email variable.

If Not IsValidEmail(Email) Then: Calls a function named IsValidEmail and passes the Email value to it. The function is likely used to validate the format of the email address.

If the email address is not valid:

A message box is displayed to inform the user that they need to enter a valid email address.

The Cancel parameter is set to True, preventing the focus from leaving the TxtCustEmail control.

The focus is set back to the TxtCustEmail control, ensuring that the user must correct the input.

In summary, this event handler ensures that the customer's email address input is valid before allowing the focus to move away from the TxtCustEmail control. It uses a separate function to validate the email format and displays messages to guide the user and prevent them from entering invalid email addresses.

The code snippet defines a function named IsValidEmail that is used to validate an email address based on a regular expression pattern. Here's a breakdown of the code's functionality:

Private Function IsValidEmail(ByVal Email As String) As Boolean: This is the function definition. It takes an Email parameter as input and returns a Boolean value (True if the email is valid, False if not).

Dim regex As Object: Declares a variable named regex to hold an instance of the VbScript.RegExp object. This object is used for regular expression pattern matching.

Set regex = CreateObject("VbScript.RegExp"): Creates an instance of the VbScript.RegExp object and assigns it to the regex variable.

regex.Pattern = "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$": Sets the regular expression pattern for validating an email address. The pattern checks for the format of a typical email address. It includes the following components:

[a-zA-Z0-9._%+-]+: Matches the username part of the email address (before the '@' symbol), allowing for letters, digits, dots, underscores, percent signs, plus signs, and hyphens.

@: Matches the literal '@' symbol.

[a-zA-Z0-9.-]+: Matches the domain name part of the email address (between the '@' symbol and the last dot), allowing for letters, digits, dots, and hyphens.

\.: Matches the literal dot '.'.

[a-zA-Z]{2,}: Matches the top-level domain (TLD) part of the email address, requiring at least two letters (e.g., com, org, gov).

IsValidEmail = regex.test(Email): Calls the test method of the regex object to check if the provided Email matches the regular expression pattern. The result of the test (either True or False) is assigned to the IsValidEmail function.

In summary, the IsValidEmail function uses a regular expression pattern to validate whether an input string conforms to the format of a valid email address. It returns True if the email is valid and False if it's not. This function can be used to enhance the validation of email addresses in your VBA UserForm.

The code snippet defines an event handler named TxtCustEmail_KeyPress, which is associated with the KeyPress event of a textbox control named TxtCustEmail within a UserForm. This event handler is used to provide instructions to the user about the expected format of the email address when they are typing in the textbox. Here's a breakdown of the code's functionality:

KeyAscii As MSForms.ReturnInteger: The parameter KeyAscii represents the ASCII code of the key being pressed. However, this event handler does not use the KeyAscii parameter for any processing.

Instruction.BackColor = vbBlue: Sets the background color of a control named Instruction to blue. This control seems to be used to display instructional messages.

Instruction.Caption = "Email should be right E-mail Format (i.e. gincom1@yahoo.com)": Sets the caption or text of the Instruction control to provide guidance to the user. The message informs the user about the expected format of an email address (i.e., "gincom1@yahoo.com").

In summary, this event handler is triggered when the user presses a key in the TxtCustEmail textbox. It changes the background color of an instructional control and

displays a message indicating the expected format for the email address input. This helps guide the user as they type their email address.

Customer PAN No. Validation Code : (**Copy below code in TxtCustPan Textbox)**

```
Private Sub TxtCustPan_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
    Dim validChars As String
    Dim pan As String
    Instruction.BackColor = vbRed
    Instruction.Caption = "PAN Card No. should be right format, All letters must type in Uppercase (i.e ABCDE9999M)"
    ' Handle the Enter key separately
    If KeyAscii = 13 Then
        KeyAscii = 0 ' Cancel the keypress
        Exit Sub
    End If
    validChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"
    pan = UCase(Trim(Me.TxtCustPan.Text)) ' Convert input to uppercase
    ' Check if the entered character is a valid character (A-Z, 0-9)
    If InStr(validChars, Chr(KeyAscii)) = 0 Then
        KeyAscii = 0 ' Cancel the keypress
        MsgBox "PAN No. should only contain uppercase letters (A-Z) and digits (0-9).", vbExclamation, "Invalid Input"
    End If
    ' Validate the PAN No. format after 5 letters and 4 digits
    If Len(pan) < 5 And KeyAscii >= 48 And KeyAscii <= 57 Then
        KeyAscii = 0 ' Cancel the keypress
        MsgBox "Invalid PAN format. The first 5 characters should be uppercase letters (A-Z).", vbExclamation, "Invalid Input"
    ElseIf Len(pan) = 5 And Not (KeyAscii >= 48 And KeyAscii <= 57) Then
```

```
        KeyAscii = 0 ' Cancel the keypress

        MsgBox "Invalid PAN format. The next 4 characters should be digits (0-9).",
vbExclamation, "Invalid Input"

    ElseIf Len(pan) = 9 And (KeyAscii >= 48 And KeyAscii <= 57) Then

        KeyAscii = 0 ' Cancel the keypress

        MsgBox "Invalid PAN format. The last character should be an uppercase letter (A-
Z).", vbExclamation, "Invalid Input"

    End If

    Label36.Caption = Me.TxtCustEmail.Text

End Sub
```

The provided code snippet defines an event handler named TxtCustPan_KeyPress, which is associated with the KeyPress event of a textbox control named TxtCustPan within a UserForm. This event handler is used to validate the format of a PAN (Permanent Account Number) card by restricting the input based on specific rules. Here's a breakdown of the code's functionality:

KeyAscii As MSForms.ReturnInteger: The parameter KeyAscii represents the ASCII code of the key being pressed.

Dim validChars As String: Declares a string variable named validChars to store the valid characters allowed for a PAN card.

Dim pan As String: Declares a string variable named pan to store the current content of the TxtCustPan textbox, converted to uppercase.

Instruction.BackColor = vbRed: Sets the background color of an instructional control to red.

Instruction.Caption = "PAN Card No. should be right format, All letters must type in Uppercase (i.e ABCDE9999M)": Sets the caption of the instructional control to provide guidance on the expected format of a PAN card number.

The section that follows handles the behavior when certain keys are pressed, such as the Enter key. If the Enter key is pressed (KeyAscii = 13), the keypress is canceled, and the event handler exits.

validChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890": Initializes the validChars string with all uppercase letters and digits.

pan = UCase(Trim(Me.TxtCustPan.Text)): Converts the content of the TxtCustPan textbox to uppercase and trims any leading or trailing spaces.

The subsequent code block validates the entered character against the validChars string. If the entered character is not valid, the keypress is canceled, and a message box is displayed to inform the user.

The code then checks for specific formats for the PAN card number, such as requiring the first 5 characters to be letters (A-Z), the next 4 characters to be digits (0-9), and the last character to be a letter. If the entered character violates any of these format rules, the keypress is canceled, and an appropriate error message is shown.

In summary, the event handler TxtCustPan_KeyPress is used to validate the input in the TxtCustPan textbox for PAN card numbers. It enforces the specified format rules and provides user feedback through error messages and instructional captions.

Auto populate State GST Code : Validation Code : (**Copy below code in TxtCustStateCode Textbox)**

```
Private Sub TxtCustStateCode_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    If Not bexitflag Then
        ' Get the values of TxtCustPAN and TxtCustStateCode
        Dim PanNo As String
        Dim StateCode As String
        PanNo = Trim(Me.TxtCustPan.Text)
        StateCode = Trim(Me.TxtCustStateCode.Text)
        ' Combine the values of TxtCustPAN and TxtCustStateCode to form GSTIN
        Dim gstin As String
        gstin = StateCode & PanNo
        ' Update the TxtCustGstin field
       Me.TxtCustGstin.Text = ""
       Me.TxtCustGstin.Text = gstin
        ' Move the cursor to the end of the TxtCustGstin field
        Me.TxtCustGstin.SetFocus
        Me.TxtCustGstin.SelStart = Len(gstin)
    End If
```

```
    ' Reset the flag
    bexitflag = False
    Label38.Caption = Me.TxtCustPan.Text
    Label39.Caption = Me.TxtCustGstin.Text
End Sub


Private Sub TxtCustStateCode_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, ByVal Shift As Integer)
    If KeyCode = vbKeyReturn Then
        bexitflag = True
        Me.TxtCustStateCode.SelStart = Len(Me.TxtCustStateCode.Text)
        Me.TxtCustStateCode.SelLength = 0
    End If
    Instruction.BackColor = vbRed
    Instruction.Caption = "Please fill last 3 Characters in Customer GSTIN field (i.e 1Z2)"
End Sub
```

The provided code snippet is an event handler named TxtCustStateCode_Exit, associated with the Exit event of a textbox control named TxtCustStateCode within a UserForm. This event handler is used to automatically generate a GSTIN (Goods and Services Tax Identification Number) by combining the values of PAN card and state code, and then updating the TxtCustGstin textbox accordingly. Here's an explanation of the code's functionality:

If Not bexitflag Then: This conditional check ensures that the code inside the TxtCustStateCode_Exit event handler is only executed if the bexitflag variable is not True. This flag is used to prevent the code from executing during certain scenarios, as controlled by external logic.

Dim PanNo As String and Dim StateCode As String: These lines declare string variables PanNo and StateCode to store the values of TxtCustPan (PAN card number) and TxtCustStateCode (state code) textboxes, respectively.

gstin = StateCode & PanNo: This line combines the values of StateCode and PanNo to form the GSTIN.

Me.TxtCustGstin.Text = "" and Me.TxtCustGstin.Text = gstin: These lines clear the current content of the TxtCustGstin textbox and then set its content to the calculated GSTIN.

Me.TxtCustGstin.SetFocus: Moves the cursor focus to the TxtCustGstin textbox.

Me.TxtCustGstin.SelStart = Len(gstin): Sets the starting position of the cursor within the TxtCustGstin textbox to the length of the calculated GSTIN. This effectively places the cursor at the end of the textbox content.

The flag bexitflag is then reset to False to ensure that the event handler will work as expected in subsequent interactions.

In summary, the TxtCustStateCode_Exit event handler automatically generates a GSTIN by combining the PAN card number and state code, updates the TxtCustGstin textbox with the generated GSTIN, and positions the cursor at the end of the textbox content. The use of the bexitflag variable ensures that the automatic GSTIN generation only occurs under specific conditions.

The provided code snippet is an event handler named TxtCustStateCode_KeyDown, associated with the KeyDown event of a textbox control named TxtCustStateCode within a UserForm. This event handler captures the Return key (Enter key) press and performs specific actions. Here's an explanation of the code's functionality:

If KeyCode = vbKeyReturn Then: This line checks if the pressed key code corresponds to the Return key (Enter key).

bexitflag = True: Sets the bexitflag variable to True. This flag, when set to True, is used in the TxtCustStateCode_Exit event handler to control whether certain actions are performed or not.

Me.TxtCustStateCode.SelStart = Len(Me.TxtCustStateCode.Text): Positions the cursor at the end of the TxtCustStateCode textbox's content.

Me.TxtCustStateCode.SelLength = 0: Sets the selection length to zero. This effectively deselects any text in the textbox.

Instruction.BackColor = vbRed and Instruction.Caption = "Please fill last 3 Characters in Customer GSTIN field (i.e 1Z2)": These lines set the background color of the Instruction label to red and update its caption to provide a user instruction.

In summary, the TxtCustStateCode_KeyDown event handler captures the Return key press, sets the bexitflag to True (to control the behavior of the TxtCustStateCode_Exit event handler), positions the cursor at the end of the TxtCustStateCode textbox's content, and updates the instruction label to guide the user to fill in the last 3 characters for the GSTIN field.

Write top of the Code window:

```
Option Explicit

Dim bexitflag As Boolean
```

The code snippet includes the declaration of a variable bexitflag and the usage of the Option Explicit statement. Here's an explanation of each part:

Option Explicit: This statement enforces the declaration of all variables before they are used in the code. It helps catch typos and promotes better coding practices by ensuring that all variables are properly defined.

Dim bexitflag As Boolean: This line declares a Boolean variable named bexitflag. Boolean variables can hold either True or False values. This variable is likely intended to be used as a flag to control certain behaviors in different parts of the code.

By using Option Explicit, you ensure that variables are explicitly declared, and the declaration of bexitflag prepares the variable for later use within the code. This approach helps prevent accidental usage of undefined variables and improves the overall reliability and readability of the code.

Auto populate Customer GSTIN : Validation Code : (**Copy below code in TxtCustGSTIN Textbox)**

```
Private Sub TxtCustGstin_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    Instruction.Caption = "Please Fill First two characters must 'SR' and rest of the two must be Digit"
    Label39.Caption = Me.TxtCustGstin.Text
End Sub

Private Sub TxtCustGstin_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
   Instruction.Caption = "Type only last 3 Characters"
End Sub
```

The provided code snippet contains event handlers for the TxtCustGstin control in a VBA UserForm. Here's an explanation of each part:

Private Sub TxtCustGstin_Exit(ByVal Cancel As MSForms.ReturnBoolean): This event handler is triggered when the user exits the TxtCustGstin control (usually by clicking outside the control). It sets the caption of the Instruction label to guide the user about the expected format of the GSTIN. Additionally, it updates the caption of the Label39 label with the content of the TxtCustGstin control.

Private Sub TxtCustGstin_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger): This event handler is triggered when a key is pressed while the focus is on the TxtCustGstin control. It provides guidance to the user by updating the caption of the Instruction label, suggesting that only the last 3 characters should be entered.

These event handlers help provide user guidance and validation as the user interacts with the TxtCustGstin control. They ensure that users are aware of the expected format and help prevent incorrect inputs.

Sales Rep. ID : Validation Code : (**Copy below code in TxtSRID Textbox)**

```
Private Sub TxtSRID_Exit(ByVal Cancel As MSForms.ReturnBoolean)
Dim Srid As String
Srid = UCase(Trim(TxtSRID.Text))
If Len(Srid) <> 4 Or Left(Srid, 2) <> "SR" Or Not IsNumeric(Mid(Srid, 3, 2)) Then
    MsgBox "Invalid SRID format. It should start with 'SR' followed by two numeric characters", vbExclamation, "Invalid input"
    Cancel = True
    Me.TxtSRID.SetFocus
End If
Label37.Caption = Srid
End Sub
```

The provided code snippet contains an event handler for the TxtSRID control in a VBA UserForm. When the user exits the TxtSRID control (usually by clicking outside the control), the code checks the entered value.

It first converts the entered value to uppercase and removes any leading or trailing spaces using the UCase and Trim functions.

The code then performs several checks:

It checks if the length of the value is exactly 4 characters.

It checks if the first two characters of the value are "SR".

It checks if the characters at positions 3 and 4 (after "SR") are numeric.

If any of the checks fail, a message box is displayed to inform the user about the invalid format. The Cancel parameter is set to True, which prevents the focus from leaving the TxtSRID control. This ensures that the user is prompted to correct the input.

Finally, the code sets the focus back to the TxtSRID control.

This event handler helps ensure that the entered SRID follows the required format and contains valid characters.

Cancel Button : (Copy below code)

```
Private Sub CmdCancel_Click()
    TxtBoxBlank
    txtFieldsDisabled
    CmdSave.Enabled = False
    CmdAdd.Enabled = True
End Sub
```

The code snippet contains an event handler for the CmdCancel button in a VBA UserForm. When the user clicks the CmdCancel button, the code is triggered.

The TxtBoxBlank procedure is called, which clears the content of all text boxes and labels on the form.

The txtFieldsDisabled procedure is called, which disables the text boxes and controls that are used to input data.

The CmdSave button is disabled, preventing the user from attempting to save data after clicking the cancel button.

The CmdAdd button is enabled, allowing the user to enter new data after canceling the current operation.

This event handler helps reset the form to a clean state and provides an option to cancel any ongoing data entry or editing operation.

Close Button : (Copy below code)

You should change the name of Command Button (i.e. ClsForm)

```
Private Sub CommandButton1_Click()
    Unload Me
End Sub
```

The code snippet contains an event handler for a CommandButton1 (presumably a Close or Exit button) in a VBA UserForm. When the user clicks the CommandButton1, the code is triggered.

The Unload Me statement is used to close the current UserForm.

This event handler allows the user to close the UserForm by clicking the button, which can be useful for providing an exit option when the form is no longer needed.

Gautam Banerjee

Age: 63

Pay by UPI

9748327614

Display Text based Review message box (Copy below code)

```
Private Sub CmdMsgboxText_Click()
    Dim reviewText As String
    reviewText = "Review the entered data:" & vbNewLine & _
                 "Customer ID: " & TxtCustID.Text & vbNewLine & _
                 "Customer Name: " & TxtCustName.Text & vbNewLine & _
                 "Customer Address: " & TxtCustAdd.Text & vbNewLine & _
                 "Customer City: " & TxtCustCity.Text & vbNewLine & _
                 "Customer State: " & StatesList.Value & vbNewLine & _
                 "Customer PIN: " & TxtCustPin.Text & vbNewLine & _
                 "Customer Phone: " & TxtCustPhone.Text & vbNewLine & _
                 "Customer Email: " & TxtCustEmail.Text & vbNewLine & _
                 "Customer PAN: " & TxtCustPan.Text & vbNewLine & _
                 "Customer State Code: " & TxtCustStateCode.Text & vbNewLine & _
                 "Customer GSTIN: " & TxtCustGstin.Text & vbNewLine & _
                 "SRID: " & TxtSRID.Text
   ' Display the review text in a message box for the user to confirm
   MsgBox reviewText, vbInformation, "Review Data"
End Sub
```

The provided code snippet defines an event handler for a button (presumably named CmdMsgboxText) in a VBA UserForm.

The purpose of this button is to display a message box that shows a summary of the entered customer data for review before taking any further action. The review text is constructed using concatenation (&) and the values from various textboxes and controls in the UserForm.

The constructed review text is displayed in a message box using the MsgBox function with the vbInformation icon and a title of "Review Data."

This allows the user to see a summarized review of the entered customer data before proceeding with any actions, ensuring that the data is accurate before further processing (Like Save etc).

Display UserForm based Review message box (Copy below code)

```
Private Sub CmdDataReview_Click()
    MsgBoxForm
End Sub
```

CLICK 1 save 2

Below code for saving data to an Excel sheet

```
Sub SaveDatatoExcel()

    Dim ws As Worksheet
    Dim lastRow As Long

    Set ws = ThisWorkbook.Sheets("Customer_Master")
    lastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row + 1

    Dim newCustID As String
    newCustID = TxtCustID.Value

    ' Check if the new Cust_ID already exists
    If CustIDExists(newCustID) Then
        MsgBox "Cust_ID already exists. Please enter a unique Cust_ID.", vbExclamation, "Duplicate Cust_ID"
```

```
        Me.TxtCustID.SetFocus
        Exit Sub
    End If


    If Trim(TxtCustID.Text) = "" Or Trim(Me.TxtCustName.Text) = "" Or _
      Trim(Me.TxtCustCity.Text) = "" Or Trim(Me.TxtCustPin.Text) = "" Or _
      Trim(Me.StatesList.Text) = "" Or Trim(Me.TxtCustEmail.Text) = "" Or _
      Trim(Me.TxtCustPhone.Text) = "" Or Trim(Me.TxtCustPan.Text) = "" Or _
      Trim(Me.TxtCustStateCode.Text) = "" Or Trim(Me.TxtCustGstin.Text) = "" Or _
      Trim(Me.TxtSRID.Text) = "" Then

      MsgBox "Please fill in all the required fields... Gautam Banerjee", vbExclamation, "Incomplete Entry... Gautam Banerjee"
      Exit Sub
    End If

    Dim response As VbMsgBoxResult

    response = MsgBox("Are you sure all the entries are correct? Do you want to save the record?", vbYesNo + vbQuestion, "Confirm Save")

    If response = vbYes Then

        ws.Cells(lastRow, "A").Value = TxtCustID.Value
        ws.Cells(lastRow, "B").Value = TxtCustName.Value
        ws.Cells(lastRow, "C").Value = TxtCustAdd.Value
        ws.Cells(lastRow, "D").Value = TxtCustCity.Value
        ws.Cells(lastRow, "E").Value = TxtCustPin.Value
```

```
        ws.Cells(lastRow, "F").Value = StatesList.Value
        ws.Cells(lastRow, "G").Value = TxtCustPhone.Value
        ws.Cells(lastRow, "H").Value = TxtCustEmail.Value
        ws.Cells(lastRow, "I").Value = TxtCustPan.Value
        ws.Cells(lastRow, "J").Value = TxtCustStateCode.Value
        ws.Cells(lastRow, "K").Value = TxtCustGstin.Value

       If CustSD.Value = True Then
          ws.Cells(lastRow, "L").Value = "SD"
       Else
          ws.Cells(lastRow, "L").Value = "SC"
       End If

       ws.Cells(lastRow, "M").Value = TxtSRID.Value
       ws.Cells(lastRow, "N").Value = ""

       MsgBox "Record saved successfully", vbExclamation, "Success...Gautam
Banerjee"

    Else

       MsgBox "Please make the necessary changes and try again", vbInformation,
"Changes Required...Gautam Banerjee"
    End If

    RefreshListBox

End Sub
```

This code defines the behavior of the **"Save Data to Excel & Access"** button in the UserForm. It performs the following steps to save customer data:

Checks if the entered Cust_ID already exists. If it does, displays a message and exits.

Validates that all required fields are filled.

Asks for confirmation before saving.

If confirmed, saves the data in the worksheet.

Displays success or prompt for necessary changes.

Refreshes the ListBox that displays customer data.

Saves the data in Access database.

Disables the "Save" button, enables the "Add" button, and clears input fields.

This ensures that the entered data is valid, and if confirmed by the user, it's saved both in the worksheet and the Access database.

(Note: The provided code snippet is already a description of the functionality. If you need a more concise description, please let me know.)

Below code for saving data to Ms-Access

```
Sub SaveDatatoAccess()

Dim conn As Object
Set conn = CreateObject("ADODB.connection")

conn.ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=D:\AccessData\Customer_Master.accdb"

conn.Open
```

This code establishes a connection to an Access database named "Customer_Master.accdb" located at "D:\AccessData". It uses the "Microsoft.ACE.OLEDB.12.0" provider to establish the connection. This connection will be used to interact with the database, allowing you to save data from the Excel UserForm to the Access database.

```
Dim sql As String

sql = "INSERT INTO Customer_Master (Cust_ID, Cust_Name, Cust_Address, Cust_City, Cust_PIN, Cust_State, " & _
    "Cust_Phone, Cust_Email, Cust_Pan, Cust_GSTIN, Cust_Type, SR_ID) " & _
    "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)" ' Parameterized query
```

This code defines an SQL statement for inserting data into the "Customer_Master" table in the Access database. It's using a parameterized query to ensure safer and more efficient data insertion. The placeholders denoted by "?" will be replaced with actual values when the query is executed. The columns being inserted into are: Cust_ID, Cust_Name, Cust_Address, Cust_City, Cust_PIN, Cust_State, Cust_Phone, Cust_Email, Cust_Pan, Cust_GSTIN, Cust_Type, and SR_ID.

```
Dim cmd As Object

Set cmd = CreateObject("ADODB.command")

Dim newCustID As String
newCustID = TxtCustID.Value

If CustIDExists(newCustID) Then
    MsgBox "Cust_ID already exists. Please enter a unique Cust_ID.", vbExclamation, "Duplicate Cust-ID... Gautam Banerjee"
    Exit Sub
End If
```

In this section of the code, a new ADODB Command object is created to execute the SQL query. Before executing the query, it checks if the entered Cust_ID already exists in the records. If it does, a message box is displayed to inform the user about the duplicate Cust_ID and to prompt them to enter a unique one. The "CustIDExists" function is being called to perform this check. If the Cust_ID is found to be a duplicate, the code exits the subprocedure using the "Exit Sub" statement.

```
    With cmd
        .ActiveConnection = conn
        .CommandText = sql
        .CommandType = 1

            ' Add parameters and set their values
        .Parameters.Append .CreateParameter("@Cust_ID", 200, 1, 255, TxtCustID.Value)
' 200: adVarChar, 255: max length
        .Parameters.Append .CreateParameter("@Cust_Name", 200, 1, 255,
TxtCustName.Value)
        .Parameters.Append .CreateParameter("@Cust_Address", 200, 1, 255,
TxtCustAdd.Value)
        .Parameters.Append .CreateParameter("@Cust_City", 200, 1, 255,
TxtCustCity.Value)
        .Parameters.Append .CreateParameter("@Cust_PIN", 200, 1, 255,
TxtCustPin.Value)
        .Parameters.Append .CreateParameter("@Cust_State", 200, 1, 255,
StatesList.Value)
        .Parameters.Append .CreateParameter("@Cust_Phone", 200, 1, 255,
TxtCustPhone.Value)
        .Parameters.Append .CreateParameter("@Cust_Email", 200, 1, 255,
TxtCustEmail.Value)
        .Parameters.Append .CreateParameter("@Cust_Pan", 200, 1, 255,
TxtCustPan.Value)
```

```
        .Parameters.Append .CreateParameter("@Cust_GSTIN", 200, 1, 255, TxtCustGstin.Value)
        .Parameters.Append .CreateParameter("@Cust_Type", 200, 1, 255, CustSD.Value)
        .Parameters.Append .CreateParameter("@SR_ID", 200, 1, 255, TxtSRID.Value)

        .Execute
    End With
```

In this portion of the code, the ADODB Command object "cmd" is being configured to execute the SQL query using the connection "conn". Parameters are appended to the command to represent the values that need to be inserted into the database. The "CreateParameter" method is used to create each parameter, and their values are assigned based on the corresponding fields in the user form.

After setting up the parameters, the "Execute" method is called on the command object to execute the SQL statement, which results in the insertion of the new customer record into the database table "Customer_Master".

```
    conn.Close

    Set conn = Nothing
    Set cmd = Nothing

    CmdSave.Enabled = True
    'DoCmd.Requery
End Sub
```

In this part of the code, the database connection "conn" is being closed to release the resources associated with it. Additionally, both the "conn" and "cmd" objects are set to "Nothing" to clear their references and release memory.

After these cleanup steps, the "CmdSave" button on the user form is re-enabled, allowing the user to save additional records. The commented out line "DoCmd.Requery" suggests that there might be an intention to refresh the data displayed on the form, but since this

code appears to be in the context of Excel VBA, "DoCmd.Requery" is not applicable. Instead, you might want to consider calling a function to update the list or perform any necessary refresh of the user interface.

Function: To check if newly entered data exists in the database or not?

```
Function CustIDExists(custID As String) As Boolean

    Dim ws As Worksheet
    Set ws = ThisWorkbook.Worksheets("Customer_Master")

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

    Dim rng As Range
    Set rng = ws.Range("A2:A" & lastRow)

    Dim cell As Range
CustIDExists = False

For Each cell In rng

    If cell.Value = custID Then
        CustIDExists = True
        Exit Function
    End If
Next cell

End Function
```

The code defines a function named "CustIDExists" that checks whether a given customer ID (custID) already exists in a specific Excel worksheet ("Customer_Master"). The function iterates through the range of customer IDs in the worksheet and returns a Boolean value indicating whether the provided custID exists in the dataset or not. If the custID is found, the function returns True; otherwise, it returns False. This function is useful for preventing duplicate customer IDs in a database.

Save Button Coding:

```
Private Sub CmdSave_Click()

    Dim newCustID As String
    newCustID = TxtCustID.Value

    If CustIDExists(newCustID) Then
        MsgBox "Cust_ID already exists. Please enter a unique Cust_ID.", vbExclamation, "Duplicate Cust-ID... Gautam Banerjee"
        Exit Sub
    End If

    SaveDatatoExcel
    SaveDatatoAccess

    CmdSave.Enabled = False
    CmdAdd.Enabled = True
    TxtBoxBlank

End Sub
```

The code snippet is for the "CmdSave_Click" (Mentione your CmdBtn name) event in an Excel VBA userform. It performs the following actions when the "Save" button is clicked:

It retrieves the new customer ID (newCustID) from the TxtCustID textbox.

It uses the CustIDExists function to check if the new customer ID already exists in the dataset.

If the customer ID already exists, it displays a message box indicating the duplication and exits the subroutine.

If the customer ID is unique, it calls two functions: SaveDatatoExcel and SaveDatatoAccess to save the entered data to both an Excel worksheet and an Access database.

It disables the "Save" button and enables the "Add" button.

It clears the input fields using the TxtBoxBlank subroutine.

This code ensures that the data is properly validated and saved while preventing duplicate customer IDs in both Excel and Access data sources.

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 Me**" page.

Thank You. See you again!

PLEASE DONATE

Gautam Banerjee

Age: 63

Pay by UPI

9748327614