Excel, OOP and Visual Basic for Applications (VBA)

An Excel Application is a workbook containing graphical user interfaces (GUIs) for interaction with a user. The GUIs contain objects that have their corresponding properties, methods and events. The steps for creating an Excel application include the following:

1. Develop a working spreadsheet model.
2. Design and create the forms for user input and displaying the information.
3. Establish the navigation path for user interaction with the model.
4. Create the worksheet/VB forms (GUIs).
5. Record and link macros and/or program the control objects.
6. Create Open/BeforeClose macros (if needed).

I. OBJECTS
1. An object is an item.
2. In OOP, an object is a self-contained entity that consists of both data and procedures to manipulate the data.
3. In VBA, a User-Form is a container object that holds other objects like buttons, labels, and images.
4. EXAMPLE—a CAR object.

II. PROPERTIES
1. An object’s attributes
2. A set of characteristics that essentially control the objects appearance and behavior
3. EXAMPLE—color, year and make of a car.

III. METHODS
1. An object’s actions
2. In VBA, a subprocedure or macro.
3. EXAMPLE—starting, accelerating, decelerating, braking and steering a car.

IV. EVENTS
1. Each action by the user (i.e., drag, click, dbl-click) causes an event to occur.
2. The application developer adds code to process each important event for the application.

Given a description of the problem scenario for the application, begin choosing the GUI objects based on the principal nouns, verbs, and adjectives in the description. The nouns represent objects for the application. The verbs and adjectives associated with the nouns respectively represent the methods and properties for the corresponding objects.
The following table shows the prefixes used when conforming the standard practice for naming the control objects:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Object Naming Conventions</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbo</td>
<td>Combo list box</td>
</tr>
<tr>
<td>chk</td>
<td>Checkbox</td>
</tr>
<tr>
<td>cht</td>
<td>Chart object</td>
</tr>
<tr>
<td>cmd</td>
<td>Command button</td>
</tr>
<tr>
<td>drp</td>
<td>Drop-down list</td>
</tr>
<tr>
<td>edb</td>
<td>Edit box</td>
</tr>
<tr>
<td>frm</td>
<td>User Form</td>
</tr>
<tr>
<td>grp</td>
<td>Group object or group box</td>
</tr>
<tr>
<td>hsb</td>
<td>Horizontal scroll bar</td>
</tr>
<tr>
<td>lbl</td>
<td>Label</td>
</tr>
<tr>
<td>lin</td>
<td>Line</td>
</tr>
<tr>
<td>lst</td>
<td>List box</td>
</tr>
<tr>
<td>opt</td>
<td>Option button</td>
</tr>
<tr>
<td>pic</td>
<td>Picture</td>
</tr>
<tr>
<td>shp</td>
<td>Shape (oval, rectangle, arc, drawing)</td>
</tr>
<tr>
<td>spn</td>
<td>Spinner</td>
</tr>
<tr>
<td>txt</td>
<td>Text box</td>
</tr>
<tr>
<td>vsb</td>
<td>Vertical scroll bar</td>
</tr>
</tbody>
</table>
CREATING WORKSHEET/USER FORMS

I. MAIN MENU ITEMS
   A. View
      1. Developer Tab

   B. Tools
      1. Macro
         • Macros
         • Record New Macro
         • Visual Basic Editor
      2. Options
         • View
         • Calc
         • Edit
         • General

II. VISUAL BASIC FOR APPLICATIONS
    A. UserForm
    B. Control Toolbox
    C. Project Explorer
    D. Properties Window

Microsoft Visual Basic Help

To install Visual Basic Help, run the setup program that you used to install Microsoft Office or Microsoft Excel. Click Add or Remove Features, and then click next to Office Tools. Click the icon next to Visual Basic Help, click Run from My Computer or Run from Network (as appropriate), and then click Update Now.

To get Help for Visual Basic in Excel, point to Macro on the Tools menu, and then click Visual Basic Editor. Click Microsoft Visual Basic Help, type the method, property, function, statement, or object that you want information about, click Search, and then click the topic you want.
III. EXCEL’s MACRO RECORDING FEATURE
A. Start Recording
   1. Alternative ways to do so
B. Perform all of the required steps in the proper sequence.
C. Stop Recording

EXCEL/VBA CONTROL OBJECTS

About adding buttons, check boxes, and other controls
When you create a custom form by using a worksheet or chart sheet, you can add controls that let the user interact with the form. Controls are graphic objects that you place on a form to display or enter data, perform an action, or make the form easier to read. These objects include text boxes, list boxes, option buttons, command buttons, and others. Controls offer options from which users can select or buttons that users can click that run macros or Web scripts that automate tasks.

MICROSOFT EXCEL PROVIDES TWO WAYS TO ADD CONTROLS TO A WORKSHEET. (1) ActiveX controls in the Control Toolbox allow you to create custom controls and control events. (2) The controls on the Forms toolbar are for use with existing Excel macros.

Changing properties When you add a control, you can change the control's properties. The properties define the control's appearance, the cell or range of cells the control refers to, and the control's state (for example, if a check box is selected or cleared by default).

The Control Toolbox: ActiveX controls for macros and Web scripts
When to use When you need to add a control, use an ActiveX control from the Control Toolbox for most situations, especially when you need to control different events that occur when you use the control. For example, you can add a list box control to a worksheet, and then write a macro to take a different action depending on which choice a user selects from the list.

What the controls can do ActiveX controls are similar to controls you find in programming languages such as Microsoft Visual Basic, and they are the same controls you can add to custom forms you create by using the Visual Basic Editor. When you add an ActiveX control to a worksheet, you write macro code that is stored with the control itself, as opposed to a macro that is assigned to run when you click the control.

What the controls can't do You cannot use ActiveX controls on XLM macro sheets from earlier versions of Microsoft Excel or ActiveX controls on chart sheets. For these situations, use the Forms toolbar controls. If you want to create a control and then attach a macro to run directly from it, use a Forms toolbar control. Or if you want to assign a macro to an existing ActiveX control, you could record or write a macro and then modify the code for the ActiveX control so that the control runs your macro.

ActiveX controls in custom solutions ActiveX controls have properties you can change to customize the control. For example, you can have an option button label wrap onto multiple lines as you resize the control. ActiveX controls have many different actions, or events, that can occur when you use the control. For example, you can determine the type of pointer that appears when you rest the pointer over
the control. You can also use ActiveX controls to create forms and dialog boxes for custom programs created in Visual Basic for Applications.

**ActiveX controls on Web pages** When you want to publish a Web page that includes Excel data, you can make it possible for users to interact with that Excel data by including ActiveX controls. When you include ActiveX controls, you write Web scripts (rather than macro code) to run when the controls are used in the Web browser. You can write the scripts in VBScript or JavaScript.

**The Control Toolbox** The controls in the Control Toolbox are ActiveX controls. Some of these controls look the same as the Forms toolbar controls, and others, such as toggle buttons and image controls, are not available on the Forms toolbar. In addition, custom controls that are installed by other programs are available, such as the control that displays ActiveX movies that is installed by Microsoft Internet Explorer. For more information about the properties for ActiveX controls, see Visual Basic Help.

**Add an ActiveX control from the Control Toolbox**

1. Open the worksheet you want to add an ActiveX control to.
2. If the Control Toolbox is not displayed, point to the Developer Tab and choose Insert.
3. Click the button for the control you want to add.
4. On the worksheet, drag the control to the size you want.

To set the properties for the control, right-click the control, and then click Properties on the shortcut menu.

5. Do one of the following:

   To add macro code for the control so that you can use the control in an Microsoft Excel workbook, right-click the control, and then click View Code on the shortcut menu. In the Visual Basic Editor, write your macro code. For information about writing macro code, see Visual Basic Help. To quit, click Close and Return to Microsoft Excel on the File menu.

   To add a Web script for the control so that you can use the control on a Web page, click the worksheet, point to Macro on the Tools menu, and then click Microsoft Script Editor. In the Script Editor, write your script. For information about developing Web scripts, refer to Microsoft Script Editor Help To return to Excel from the Script Editor, click Exit on the File menu.

6. To quit design mode and enable the ActiveX control, click Exit Design Mode.

**Note** A Web script uses the control ID to handle events for the ActiveX control. Therefore, you can insert a script anywhere in the data that will be published to a Web page.

**The Forms toolbar: controls for worksheet forms**

**When to use** The Forms controls are compatible with earlier versions of Excel, beginning with Excel version 5.0, and can be used on XLM macro sheets. Use a control from Forms when you want to record all of the macros for a form but do not want to write or modify any macro code in Visual Basic for Applications. You can also use Forms controls on chart sheets. These controls are designed for use on
worksheet forms that other users fill in by using Microsoft Excel. You can attach an existing macro to a control or write or record a new macro.

**Note** When you assign a macro to a button on a worksheet, the button is available only when the worksheet is open. If you want to run a macro by using a button that is available regardless of which workbooks are open, you can run the macro from a custom toolbar button.

### Add controls to a sheet by using the Forms toolbar

If you want to run an existing macro from a worksheet control, make sure the workbook that contains the macro is open.

1. Open the worksheet you want to add controls to.
2. Make sure the to the Developer Tab and choose **Insert**.
3. On the **Forms** group, click the button for the control you want to add.
4. On the worksheet, drag the control to the size you want.
5. If you are adding a button, select the macro you want to run when you click the button in the **Macro name** box.

If you are adding a control that is not a button, right-click the control, and then click **Assign Macro** on the shortcut menu.

If the macro does not exist, you can record or write a new macro. Click **Record** to record a new macro, or click **New** to open a Visual Basic for Applications module in which you can write the macro.

I. **Control Objects for User Interaction**
   A. **Form**
      1. Name
      2. Caption
      3. Other properties (BackColor, BorderStyle, Font, Special Effects)
      4. Methods and statements for loading and displaying
         ✦ Show
         ✦ Hide
   B. **Label control**
      1. Name
      2. Caption
      3. Other properties (BackColor, BorderStyle, Font, Special Effects)
   C. **Textbox control**
      1. Name
      2. Text
      3. Other properties (BackColor, BorderStyle, Font, Special Effects)
      4. Checking user input
         ✦ KeyPress event
      5. Val function

***SUPPLEMENTAL NOTES***

✦ **Name property** – used in the code to refer to an object
✦ **Caption property** – specifies what the control displays on the screen to the user
BackStyle property - determines whether a label control is transparent or opaque.

BorderStyle property - determines the style of an object’s border

Appearance property - determines if the control appears flat or three-dimensional

Text property - controls the text that appears inside a text box

To remove the contents of a text box or a label control, assign a zero-length string to it. A zero-length string is a set of quotation marks with nothing between them (“”) and is also referred to as an empty string.

Visual Basic has two statements and two methods that control the loading and displaying, respectively, of forms.

A. Load statement brings a form into memory, but it does not display the form on the screen. Syntax: `Load object`

B. Unload statement removes a form from memory and from the screen. Syntax: `Unload object`

C. Hide method removes a form from the screen, but leaves it in memory. Syntax: `object.Hide`

D. Show method displays a form on the screen. If the form is not already loaded, the Show method loads it before displaying it. Syntax: `object.Show`

Sample code for only accepting numbers from a user via a textbox:

```vba
Private Sub <Textboxname>_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
    If (KeyAscii < 48 Or KeyAscii > 57) And KeyAscii <> 8 Then
        KeyAscii = 0
    End If
End Sub
```

The Val function

A. A function is a predefined procedure that performs a specific task. Unlike a method, a function results in a value.

B. The Val function tells VB to treat a character string as a numeric value and results in a number.

   Its syntax is `Val(StringExpression)`. The `StringExpression` cannot include a letter or a special character, such as the dollar sign, comma, or the percent sign; it can include the period.

C. When VB encounters an invalid character in the Val function’s `StringExpression`, it stops converting the `StringExpression` to a number at that point.
RECOMMENDATIONS FOR DESIGNING A USER INTERFACE

Some of the following recommendations are self-evident and could probably go without mention. Others deal with less obvious issues related to using Excel--issues that often can be well understood only after years of experience.

1. **Use a main control form in your application.** A main control form is one that the user first encounters when running your application. The form should clearly display the name of the application, the purpose of the application, and a means of navigating to the various component parts of the application.

2. **Break up your application into at most four or five main logical components, and build separate forms for accessing each component.** By breaking up an application into logical components, you give more structure to your application. If users associate one main task with one form, the application appears more intuitive, and users learn the application more quickly.

3. **Make the navigation path through your application clear to the user.** The user should never have to wonder how to access a form or a major component of an application; the design of the interface should make this information obvious. Perhaps the best way to achieve this clarity is to place large and intuitive buttons on your application’s forms--buttons that allow users to access the various component parts. Take care in using menus or toolbars as a means of navigating through an application because menuitems and toolbarbuttons are not as readily visible to the user as are control objects places on a form. For example, users might not know to look in a certain menu to access a certain screen.

4. **Make your application as graphical as possible.** The more graphical the presentation of information, the better your user will understand it. For example, if you are creating an application to display sales data, go beyond simply displaying numeric data to display the data graphically in charts or even in geographical maps. Doing so will make the data much easier to understand.

5. **Format numeric data to make it easier to read.** When you create applications that involve the display of numeric data in tables, take advantage of Excel’s built-in automatic formats, which are designed to make data easier to read.

6. **Avoid clutter on forms.** A form should contain no more than 7 to 10 controls, with 5 probably the optimal number. Putting too many controls on a form usually makes it confusing.

7. **For data-centric routines, use control objects on sheets.** For example, if you have a table of data on a worksheet and you’ve written a macro that manipulates the data in some fashion, attach the macro to a control that’s close to the data table. With the control near the data that it affects, the control’s function is more obvious to the user. In such situations, using a control on a sheet is better than using a menu or a dialog box because accessing a routine through a menu or a dialog box might be much less intuitive to the user.

8. **Use control objects on sheets whenever you can.** At the risk of sounding redundant, we’ll mention again that you should use control objects directly on worksheets or charts whenever doing so makes sense. Some developers have a tendency to use controls only in dialog boxes, but the user must complete an extra step to display a dialog box and then access a control. This extra step is not required when the control exists directly on the worksheet or chart.
9. **Create wizards for multi-step processes.** By breaking up complicated tasks into logical component parts and placing them in wizards, you will greatly simplify such tasks for your users.

10. **Use custom menus only when control objects on sheets and dialog boxes are not appropriate.** Placing a command for a routine deep within a menu might make it difficult for the user to discover it. Control objects on worksheets, on the other hand, are much more intuitive and readily accessible. Of course, there are times when using menus is appropriate.

11. **Use the Menu Editor to design your custom menus.** The only way to design custom menus and save them with the workbook file that contains your application is to use the Menu Editor. (Changes made through VBA are not saved.)

12. **Attach toolbars rather than creating them at runtime.** To avoid the performance problems that occur when you create toolbars at runtime, create your toolbars during the development process, and attach them to the workbook file that contains your application. Also delete any toolbars that are associated with your application from the user’s workspace when your application has finished executing—provided that the toolbars will not be used by the user in the absence of your application.

13. **Eliminate all evidence of Excel from your application.** The ultimate goal is to make users think they are running your application and not Excel. If your application does not require Excel’s menus, hide them. You should also hide all built-in Excel toolbars and customize the Application and ActiveWindow captions. If your application does not require the user to interact with any standard built-in Excel functionality, take whatever steps are necessary to prevent the user from breaking out of your application and accessing Excel.

14. **Use Workbook_Open() to make environment settings when your application first starts, and use Workbook_BeforeClose() to restore environment settings when your application has finished executing.** These are ‘Workbook’ events found on the code sheet for the ‘ThisWorkbook’ object. If your application sets any properties of environment objects (Application, Toolbar, Menu, and so on), be sure to restore the original values of those properties after your application has finished executing. Use Workbook_Open() to read the settings of all of the environment properties that will be changed, storing such settings in variables. Workbook_Open() can then make the necessary changes to the properties, and Workbook_BeforeClose() can use these variables to restore the environment properties to their original state. Using these events in this way ensures that your users will still be able to use Excel as a stand-alone spreadsheet application, without being affected by any environment settings left over from running your application.