More Excel Notes for Using VBA to Create Customized Tools:

User Form Example 1 (Loan Analysis Worksheet):

(UserForms / For Each…Next Loops)

In this example we are to complete an Excel-based system for analyzing and amortizing loans. Figure 1 shows the main interface for this system. After entering data for the purchase price, down payments, other credits, term of the loan, and interest rate, the system automatically calculates the monthly payments, total payments and finance charge for the loan. The system should also be capable of producing an amortization schedule for the loan (see Figure 2 on the next page).

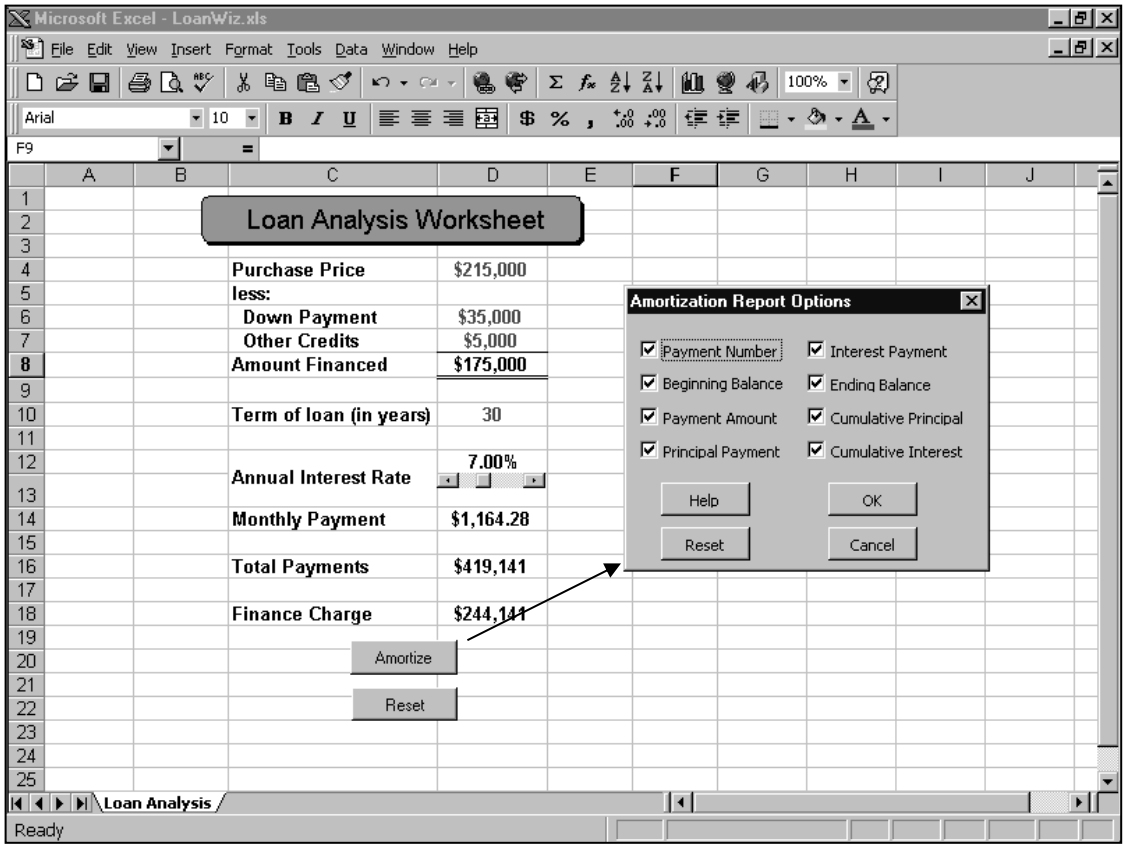


Figure 1

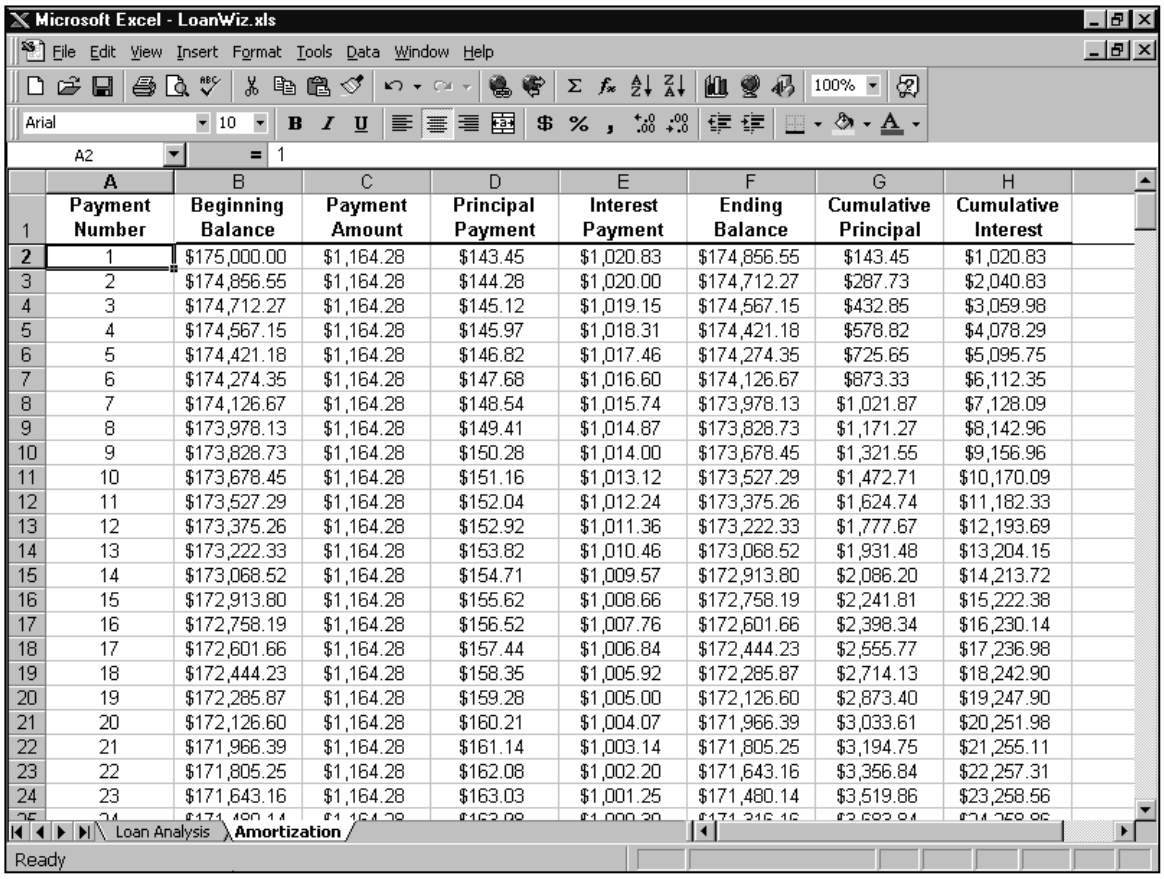
As shown in Figure 1, when the user clicks the Amortize button, a dialog box should pop up listing the various items (columns) that can be included on the amortization schedule. Your job in this assignment is twofold:

(1) You need to create the dialog box:

You are responsible for inserting the UserForm object (frmOptions) into your project and placing the appropriate controls on it. On the last page of this assignment, I have given you the subroutines that will have to be placed on its associated code module. Be sure that you name the CommandButtons appropriately so that the event handling routines will fire when the buttons are clicked. The CheckBoxes can be given any names you choose, with the exception that the Payment Number checkbox must be named chkPaymentNumber.

(2) You need to write the code to make the appropriate items appear in the amortization report:

There is a code module in the file that contains a subroutine called CreateAmortizationTable. This subroutine is only partially completed. By carefully inspecting all the code in the application, you should be able to get some good ideas about how to finish it. This is the <u>only</u> subroutine that you need to change or add to in order to get the application to work correctly.



	A	B	C	D	E	F	G	H
1	Payment Number	Beginning Balance	Payment Amount	Principal Payment	Interest Payment	Ending Balance	Cumulative Principal	Cumulative Interest
2	1	$175,000.00	$1,164.28	$143.45	$1,020.83	$174,856.55	$143.45	$1,020.83
3	2	$174,856.55	$1,164.28	$144.28	$1,020.00	$174,712.27	$287.73	$2,040.83
4	3	$174,712.27	$1,164.28	$145.12	$1,019.15	$174,567.15	$432.85	$3,059.98
5	4	$174,567.15	$1,164.28	$145.97	$1,018.31	$174,421.18	$578.82	$4,078.29
6	5	$174,421.18	$1,164.28	$146.82	$1,017.46	$174,274.35	$725.65	$5,095.75
7	6	$174,274.35	$1,164.28	$147.68	$1,016.60	$174,126.67	$873.33	$6,112.35
8	7	$174,126.67	$1,164.28	$148.54	$1,015.74	$173,978.13	$1,021.87	$7,128.09
9	8	$173,978.13	$1,164.28	$149.41	$1,014.87	$173,828.73	$1,171.27	$8,142.96
10	9	$173,828.73	$1,164.28	$150.28	$1,014.00	$173,678.45	$1,321.55	$9,156.96
11	10	$173,678.45	$1,164.28	$151.16	$1,013.12	$173,527.29	$1,472.71	$10,170.09
12	11	$173,527.29	$1,164.28	$152.04	$1,012.24	$173,375.26	$1,624.74	$11,182.33
13	12	$173,375.26	$1,164.28	$152.92	$1,011.36	$173,222.33	$1,777.67	$12,193.69
14	13	$173,222.33	$1,164.28	$153.82	$1,010.46	$173,068.52	$1,931.48	$13,204.15
15	14	$173,068.52	$1,164.28	$154.71	$1,009.57	$172,913.80	$2,086.20	$14,213.72
16	15	$172,913.80	$1,164.28	$155.62	$1,008.66	$172,758.19	$2,241.81	$15,222.38
17	16	$172,758.19	$1,164.28	$156.52	$1,007.76	$172,601.66	$2,398.34	$16,230.14
18	17	$172,601.66	$1,164.28	$157.44	$1,006.84	$172,444.23	$2,555.77	$17,236.98
19	18	$172,444.23	$1,164.28	$158.35	$1,005.92	$172,285.87	$2,714.13	$18,242.90
20	19	$172,285.87	$1,164.28	$159.28	$1,005.00	$172,126.60	$2,873.40	$19,247.90
21	20	$172,126.60	$1,164.28	$160.21	$1,004.07	$171,966.39	$3,033.61	$20,251.98
22	21	$171,966.39	$1,164.28	$161.14	$1,003.14	$171,805.25	$3,194.75	$21,255.11
23	22	$171,805.25	$1,164.28	$162.08	$1,002.20	$171,643.16	$3,356.84	$22,257.31
24	23	$171,643.16	$1,164.28	$163.03	$1,001.25	$171,480.14	$3,519.86	$23,258.56

Figure 2

Code on the module for the Loan Analysis sheet:

```
Private Sub cmdAmortize_Click()
   If Range("MonthlyPayment") <> 0 Then
      frmOptions.Show
      If frmOptions.Tag = vbOK Then
         CreateAmortizationTable
      End If
   Else
      MsgBox "There's nothing to amortize!", vbCritical, "Error"
   End If
End Sub

Private Sub cmdReset_Click()
   Range("PurchasePrice") = 0
   Range("DownPayment") = 0
   Range("OtherCredits") = 0
   Range("Term") = 5
   hsbInterestRate = 800
End Sub

Private Sub hsbInterestRate_Change()
   Range("InterestRate") = Format(hsbInterestRate / 10000, "00.00%")
End Sub

Private Sub hsbInterestRate_Scroll()
   hsbInterestRate_Change
End Sub
```

Code on the 'generic' module:

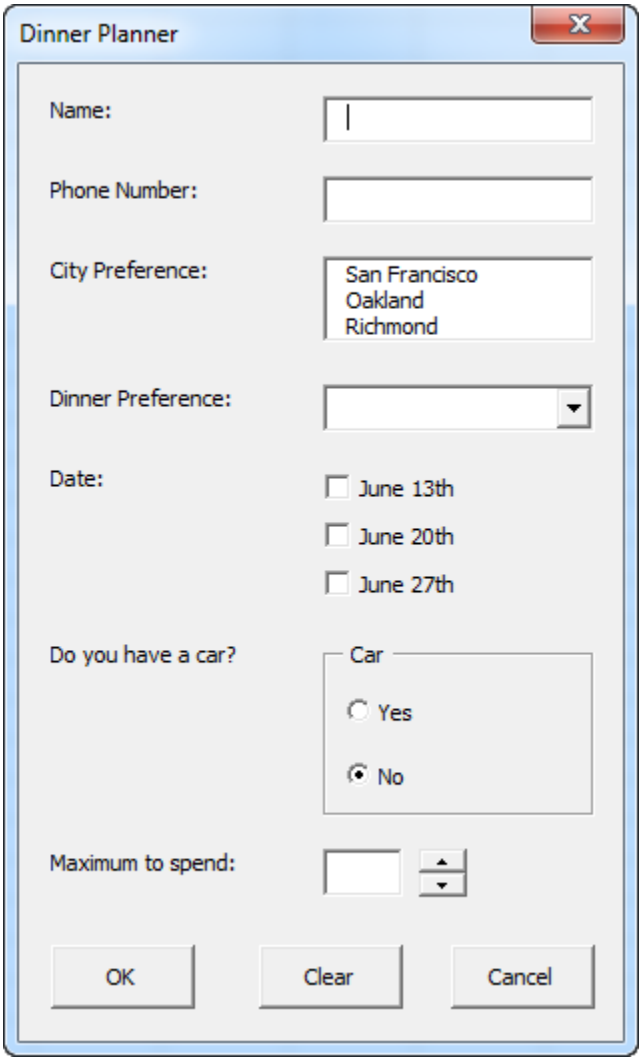
Option Explicit

```
Public Sub CreateAmortizationTable()
   Dim wrksheet As Worksheet, ctl As Control
   Dim row As Integer, col As Integer
   Dim nPayments As Integer, InterestRate As Single
   Dim MonthlyPayment As Currency
   Dim PrincipalPayment As Currency, CumPrincipal As Currency
   Dim InterestPayment As Currency, CumInterest As Currency
   Dim BegBalance As Currency, EndBalance As Currency

   Application.ScreenUpdating = False

 ' Suppress warning messages (about deleting sheets)
   Application.DisplayAlerts = False

   ' Delete any sheet named Amortization
   For Each wrksheet In Worksheets
      If wrksheet.Name = "Amortization" Then wrksheet.Delete
   Next

   ' Add a new sheet named Amortization after the active sheet
   Worksheets.Add(after:=ActiveSheet).Name = "Amortization"

   With Worksheets("Amortization")
      'Setup column Headings
      col = 0   ' Indicates column #
      For Each ctl In frmOptions.Controls
         If TypeName(ctl) = "CheckBox" And ctl.Value = True Then
            col = col + 1
            .Cells(1, col).Value = ctl.Caption
         End If
      Next

      ' Format column headings
      With .Range("A1").CurrentRegion
         .WrapText = True
         .Font.Bold = True
         With .Borders(xlEdgeBottom)
            .LineStyle = xlContinuous
            .Weight = xlMedium
         End With
      End With
      '  Format columns
      With .Columns
         .ColumnWidth = 12
         .HorizontalAlignment = xlCenter
      End With
   End With

   'Get data from the Loan Analysis sheet
   With Worksheets("Loan Analysis")
      nPayments = .Range("Term") * 12
      BegBalance = .Range("AmountFinanced")
```

```
        MonthlyPayment = .Range("MonthlyPayment")
        InterestRate = .Range("InterestRate")
    End With
    CumPrincipal = 0
    CumInterest = 0

    With Worksheets("Amortization")
        ' Fill in rows
        For row = 2 To nPayments + 1
            'Calculate values
            InterestPayment = BegBalance * InterestRate / 12
            PrincipalPayment = MonthlyPayment - InterestPayment
            CumPrincipal = CumPrincipal + PrincipalPayment
            CumInterest = CumInterest + InterestPayment
            EndBalance = BegBalance - PrincipalPayment
            ' Fill in columns of current row
            col = 0

              ' ----------------------------------------
              '  You figure out what goes here! (Hint:  it is in the Excel file.)
              ' ----------------------------------------

            BegBalance = EndBalance
        Next
    End With

    Worksheets("Amortization").Range("A2").Select
    ActiveWindow.FreezePanes = True
    ActiveWindow.Zoom = 75
    Application.DisplayAlerts = True
    Application.ScreenUpdating = True

End Sub
```

User Form Example 2:
This Userform Example is From: http://www.excel-easy.com/vba/userform.html

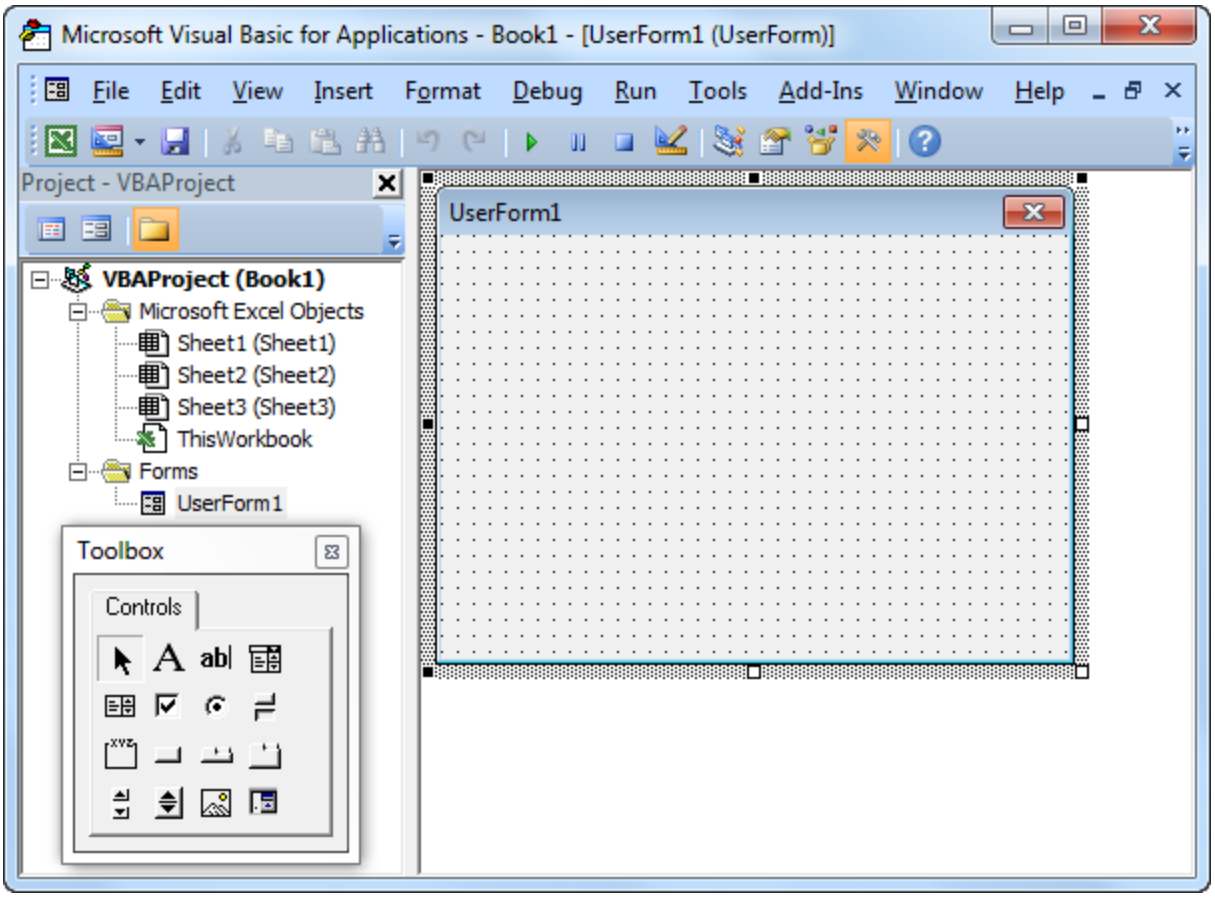
Userform



Add the Controls

To add the controls to the Userform, execute the following steps.

1. Open the Visual Basic Editor. If the Project Explorer is not visible, click View, Project Explorer.
2. Click Insert, Userform. If the Toolbox does not appear automatically, click View, Toolbox. Your screen should be set up as below.



3. Add the controls listed in the table below. Once this has been completed, the result should be consistent with the picture of the Userform shown earlier. For example, create a text box control by clicking on TextBox from the Toolbox. Next, you can drag a text box on the Userform. When you arrive at the Car frame, remember to draw this frame first before you place the two option buttons in it.

4. Change the names and captions of the controls according to the table below. Names are used in the Excel VBA code. Captions are those that appear on your screen. It is good practice to change the names of controls. This will make your code easier to read. To change the names and captions of the controls, click View, Properties Window and click on each control.

Control	Name	Caption
Userform	DinnerPlannerUserForm	Dinner Planner
Text Box	NameTextBox	
Text Box	PhoneTextBox	
List Box	CityListBox	
Combo Box	DinnerComboBox	
Check Box	DateCheckBox1	June 13th
Check Box	DateCheckBox2	June 20th
Check Box	DateCheckBox3	June 27th
Frame	CarFrame	Car
Option Button	CarOptionButton1	Yes
Option Button	CarOptionButton2	No
Text Box	MoneyTextBox	
Spin Button	MoneySpinButton	
Command Button	OKButton	OK
Command Button	ClearButton	Clear
Command Button	CancelButton	Cancel
7 Labels	No need to change	Name:, Phone Number:, etc.

Note: a combo box is a drop-down list from where a user can select an item or fill in his/her own choice. Only one of the option buttons can be selected.

Show the Userform

To show the Userform, place a command button on your worksheet and add the following code line:

```
Private Sub CommandButton1_Click()

DinnerPlannerUserForm.Show

End Sub
```

We are now going to create the Sub UserForm_Initialize. When you use the Show method for the Userform, this sub will automatically be executed.

1. Open the Visual Basic Editor.
2. In the Project Explorer, right click on DinnerPlannerUserForm and then click View Code.
3. Choose Userform from the left drop-down list. Choose Initialize from the right drop-down list.
4. Add the following code lines:

```
Private Sub UserForm_Initialize()
'Empty NameTextBox
NameTextBox.Value = ""

'Empty PhoneTextBox
PhoneTextBox.Value = ""

'Empty CityListBox
CityListBox.Clear

'Fill CityListBox
With CityListBox
   .AddItem "San Francisco"
   .AddItem "Oakland"
   .AddItem "Richmond"
End With

'Empty DinnerComboBox
DinnerComboBox.Clear

'Fill DinnerComboBox
With DinnerComboBox
   .AddItem "Italian"
   .AddItem "Chinese"
   .AddItem "Frites and Meat"
End With

'Uncheck DataCheckBoxes
DateCheckBox1.Value = False
DateCheckBox2.Value = False
DateCheckBox3.Value = False

'Set no car as default
CarOptionButton2.Value = True

'Empty MoneyTextBox
MoneyTextBox.Value = ""

'Set Focus on NameTextBox
NameTextBox.SetFocus
End Sub
```

Explanation: text boxes are emptied, list boxes and combo boxes are filled, check boxes are unchecked, etc.

Assign the Macros

We have now created the first part of the Userform. Although it looks neat already, nothing will happen yet when we click the command buttons on the Userform.

1. Open the Visual Basic Editor.
2. In the Project Explorer, double click on DinnerPlannerUserForm.
3. Double click on the Money spin button.
4. Add the following code line:

```
Private Sub MoneySpinButton_Change()

MoneyTextBox.Text = MoneySpinButton.Value

End Sub
```

Explanation: this code line updates the text box when you use the spin button.

5. Double click on the OK button.
6. Add the following code lines:

```
Private Sub OKButton_Click()

Dim emptyRow As Long

'Make Sheet1 active
Sheet1.Activate

'Determine emptyRow
emptyRow = WorksheetFunction.CountA(Range("A:A")) + 1

'Transfer information
Cells(emptyRow, 1).Value = NameTextBox.Value
Cells(emptyRow, 2).Value = PhoneTextBox.Value
Cells(emptyRow, 3).Value = CityListBox.Value
Cells(emptyRow, 4).Value = DinnerComboBox.Value

If DateCheckBox1.Value = True Then Cells(emptyRow, 5).Value = DateCheckBox1.Caption

If DateCheckBox2.Value = True Then Cells(emptyRow, 5).Value = Cells(emptyRow, 5).Value & " " &
DateCheckBox2.Caption

If DateCheckBox3.Value = True Then Cells(emptyRow, 5).Value = Cells(emptyRow, 5).Value & " " &
DateCheckBox3.Caption

If CarOptionButton1.Value = True Then
   Cells(emptyRow, 6).Value = "Yes"
Else
   Cells(emptyRow, 6).Value = "No"
End If
```

```
Cells(emptyRow, 7).Value = MoneyTextBox.Value

End Sub
```

Explanation: first, we activate Sheet1. Next, we determine emptyRow. The variable emptyRow is the first empty row and increases every time a record is added. Finally, we transfer the information from the Userform to the specific columns of emptyRow.

7. Double click on the Clear button.
8. Add the following code line:

```
Private Sub ClearButton_Click()

Call UserForm_Initialize

End Sub
```

Explanation: this code line calls the Sub UserForm_Initialize when you click on the Clear button.

9. Double click on the Cancel Button.
10. Add the following code line:

```
Private Sub CancelButton_Click()

Unload Me

End Sub
```

Explanation: this code line closes the Userform when you click on the Cancel button.

Test the Userform

Exit the Visual Basic Editor, enter the labels shown below into row 1 and test the Userform.

Result:

	A	B	C	D	E	F	G
1	**Name**	**Phone Number**	**City**	**Dinner**	**Date**	**Car**	**Maximum to spend**
2	Niels	070 540 546	Oakland	Chinese	June 13th	No	30
3	Bregje	070 748 847	San Francisco	Italian	June 13th June 20th	Yes	40
4							
5							

Making a new menu in Excel

1. Select *View* → *Toolbars* → *Customize* OR *Tools* → *Customize*…

2. On the "Toolbars" tab on the form that comes up, click New, then type in a new Name for the Commandbar. Your new toolbar should show up under the Toolbars tab, be selected, and be checked. You should also have a small, new form created with the name of your menu on its title bar.

3. With the selection in step 2, click the Commands tab. Then in the Categories window at the bottom (scroll down) select New Menu. This should put the words "New Menu" in the right window on the Commands tab.

4. Click "New Menu" in the right window and drag it onto the small form that is your customized menu in progress. This will put the words "New menu" with a down "arrow" on the small form with a rectangle around them.

5. Right click the words "New Menu" inside the rectangle and type in the Text (say, &File) you wish to appear. Note that an ampersand ("&") before a letter will cause it to be underlined and become a shortcut key (**<u>F</u>ile** in the example above).

6. If you left-click on the text (say, <u>F</u>ile) you just created, a new blank bar will appear. At this point you may (repetitively) add either a *Commands* → *Categories* → *Macros* → *Custom Menu Item* or a *Commands* → *Categories* → *Macros* → *Custom Button* or a *Commands* → *Categories* → *New Menu* by clicking on the appropriate choices and dragging and dropping to the proper place on your new tool bar.

7. If you right-click on a given one of these custom menu items or custom buttons (or new menus) while still in design mode, you can assign a macro (e.g., "mnuSelectBalancesSheet") that will be run when the DSS user chooses that menu item or button (or menu).

Customizing Excel With CommandBars

(Adapted from: Microsoft Excel97 Developer's Handbook, by Eric Wells and Steve Harshbarger, MicrosoftPress 1997, pp. 297-317)

It is often desirable to create custom command bars (toolbars or menu bars) for an application in Excel. This gives you the ability to: (1) remove functionality built into Excel that is not relevant for an application and, (2) create custom menu items specific to your application.

Menus are typically made up of items in a Controls collection of a CommandBar object in Excel. Although you can create Controls programmatically using VBA, it is usually easier to create custom CommandBars manually using the Tools, Customize commands in Excel. (I will demonstrate this process in class.) It is then fairly simple to use the custom CommandBar directly or transfer controls from a custom CommandBar onto one of Excel's built-in CommandBar objects.

NOTE: It is important to to remember that custom CommandBars attach themselves to the CommandBars collection of the Application object in Excel. Even when a custom CommandBar is 'attached' to an Excel workbook, it attaches automatically to the Application.CommandBars collection when the file containing the custom CommandBar is opened. However, it is ***not*** automatically removed from the Application.CommandBars collection when the file containing the custom CommandBar is closed. We must take care of this detail ourselves.

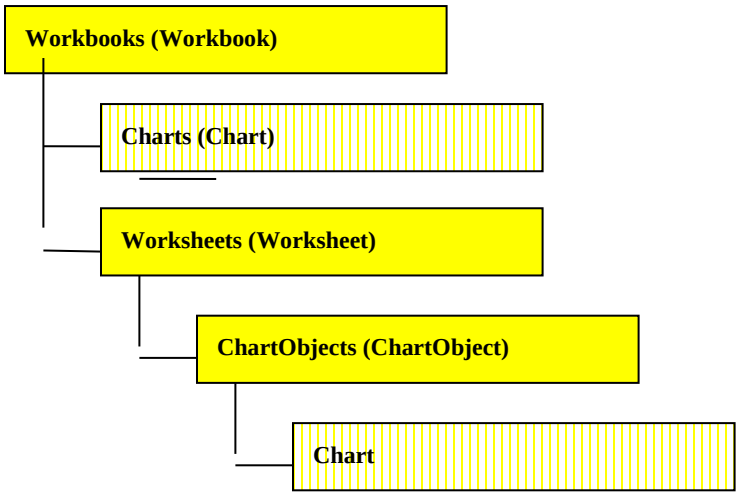
The following code is from the ThisWorkbook object of an Excel application and illustrates various techniques associated with using CommandBars. A custom CommandBar named *Custom1* must be available.

```
Private vcb( ) As String                    ' Array that will be used to store names of Visible CommandBars
Private FormulaBarStatus As Boolean         ' Used to track the FormulaBar's original visible status
Private StatusBarStatus As Boolean          ' Used to track the StatusBar's original visible status

Private Sub Workbook_Open()
  SetUp
End Sub

Private Sub SetUp()
  Dim cb As CommandBar
  Dim ctl As Object
  Dim counter As Integer
  Dim mybar As CommandBar, mainbar As CommandBar

  counter = 0
  On Error Resume Next
  ' Create a list of the visible commandbars & make them invisible
  For Each cb In Application.CommandBars
    If cb.Visible = True Then
      counter = counter + 1
      ReDim Preserve vcb(counter)
      vcb(counter) = cb.Name
```

```
        cb.Visible = False
      End If
    Next
    On Error GoTo 0

    ' Hide all controls in the main menu bar
    Set mainbar = Application.CommandBars("Worksheet Menu Bar")
    For Each ctl In mainbar.Controls
      If ctl.Visible = True Then ctl.Visible = False
    Next

    ' Copy custom controls into the main menu bar
    Set mybar = Application.CommandBars("Custom1")
    For Each ctl In mybar.Controls
      ctl.Copy mainbar
    Next


    With Application
      FormulaBarStatus = Application.DisplayFormulaBar
      StatusBarStatus = Application.DisplayStatusBar
      .DisplayFormulaBar = False
      .DisplayStatusBar = False
      .ActiveWindow.Caption = ""   ' Suppress display of filename
      .Caption = "Super Project"
    End With
End Sub

Private Sub FinishUp()
    Dim cb As Variant
    Dim ctl As Object
    Dim mybar As CommandBar, mainbar As CommandBar

    On Error Resume Next
    ' Make original commandbars visible again
    For Each cb In vcb
      Application.CommandBars(cb).Visible = True
    Next

    Set mainbar = Application.CommandBars("Worksheet Menu Bar")

    ' Restore controls in main menu bar
    Application.CommandBars("Worksheet Menu Bar").Reset

    With Application
      .DisplayFormulaBar = FormulaBarStatus
      .DisplayStatusBar = StatusBarStatus
      .ActiveWindow.Caption = ThisWorkbook.Name
```

```
    .Caption = ""
    .CommandBars("Custom1").Delete  ' Never leave custom bars in the application!!!
  End With
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
  Dim msg As String

  msg = "Do you want to save the changes you made?"

  Select Case MsgBox(msg, vbInformation + vbYesNoCancel, "Super Project")
  Case vbYes
    ThisWorkbook.Save
    FinishUp
  Case vbNo
    FinishUp
    ThisWorkbook.Saved = True  ' Makes Excel think the file was saved/clean
  Case vbCancel
    Cancel = True
  End Select
End Sub
```

Charts



Some of the main objects / methods associated with a **Chart**:

SeriesCollection (Series) - each Series object (within the collection SeriesCollection) represents a plotted data series

ChartGroups (ChartGroup) - each ChartGroup represents a group of data series with the same chart type (can have multiple chart types plotted within the same Chart)

Axes (Axis) - represents the various axes of the Chart object:
- the **category** axis (x)
- the **value** axis (y)
- the **series** axis (z)

You cannot use the Axis object to change the values displayed on an axis.

ChartWizard - a method of the Chart object – it is the primary tool for creating a Chart using VBA (or Excel, for that matter)

Two types of objects can represent charts in Excel:

The ChartObject object: The ChartObject object is a floating graphical image of a chart that can exist on a worksheet. Chartobjects are linked to worksheet ranges; whenever the numbers in the range to which a chartobject is linked are updated, the chartobject is updated as well.

The Chart object: The Chart object exists on a separate sheet by itself. Charts are linked to ranges in the same manner as chartobjects.

A ChartObject object is really just a container for a single Chart object. For example, a chart on a separate sheet can be referred to in code as follows:

Workbooks(1).Charts(1)

A chart embedded on a worksheet, however, is referenced as follows:

Workbooks(1).Worksheets(1).ChartObjects(1).Chart

The ChartObject object has its own set of properties that generally control the position and appearance of the chartobject on the sheet. Left, Top, Height, and Width are examples of these properties.

<u>To create a chart</u>:

The **ChartWizard** method is typically used to create a chart from scratch in Excel

<u>To change source data</u>:

(<u>Option 1</u>) Change the range of data that is plotted by an individual data series by setting the Values property of the Series object

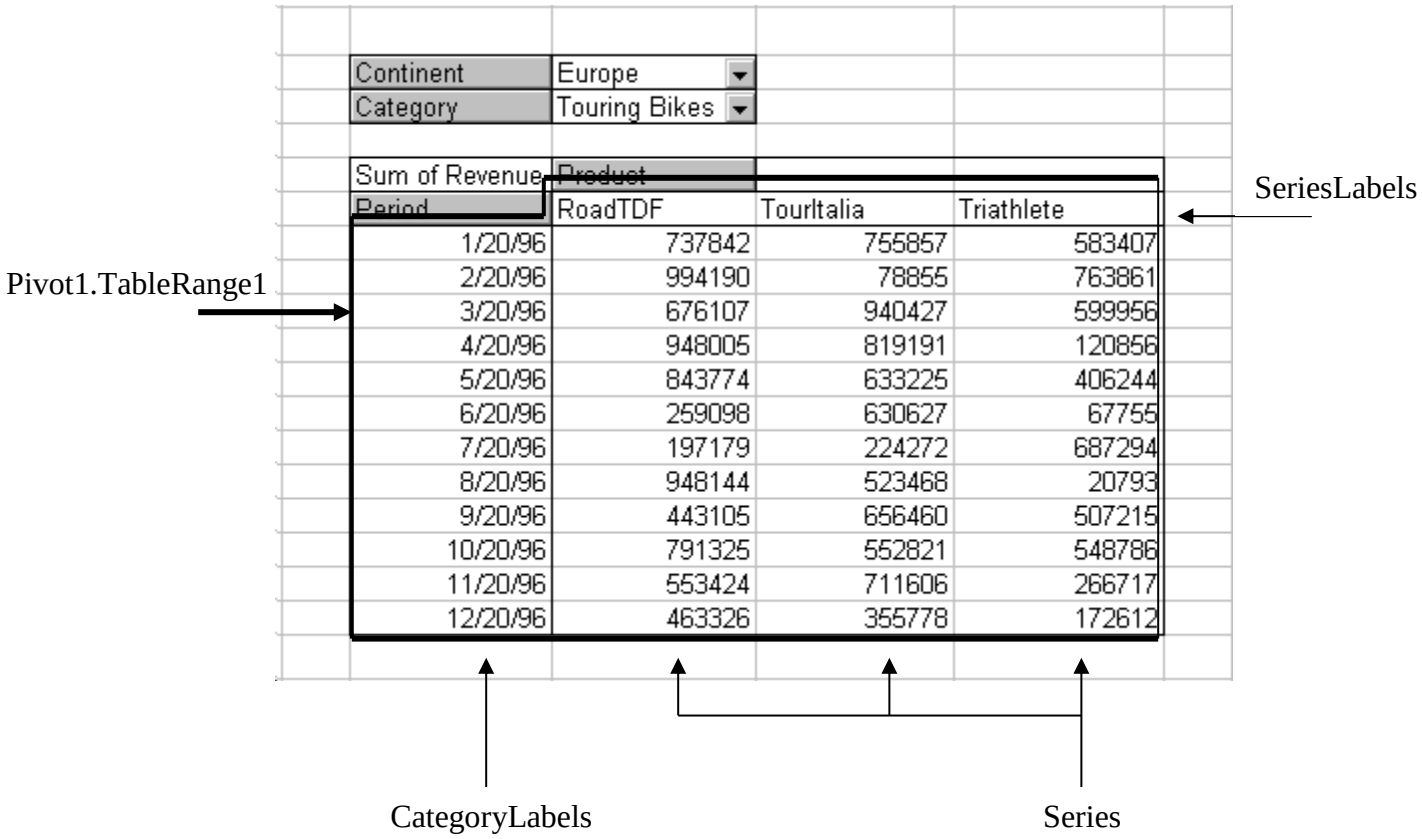
EX: *Worksheets(1).ChartObjects(1).Chart.SeriesCollection(1).Values = Range("C4:C13")*

(<u>Option 2</u>) Use the ChartWizard method to change the data source

EX: *Set DataRange = Range("A3:B15")*
Charts(1).ChartWizard Source:="DataRange", CategoryLabels:=1, SeriesLabels:=1

(<u>Option 3</u>) ***(PivotTables)*** By linking the chart to the data area (the TableRange1 object) of a PivotTable, any changes in that data area will be automatically reflected in the chart.

EX:
```
Dim ChartRange As Range,  Pivot1 As PivotTable,  Chart2 As Chart
Set Pivot1 = PivotTables(1)
Pivot1.RowGrand = False          'Keeps the row totals from being included in the chart
Pivot1.ColumnGrand = False  'Keeps the column totals from being included in the chart

Set ChartRange = Pivot1.TableRange1  'TableRange1 contains the Row, Column, and _ Data
                                      areas (but not the Page area)
Worksheets(2).Select
Set Chart2 = Charts.Add
Chart2.ChartWizard Source:=ChartRange, Gallery:=xl3DColumn, Format:=6, _
        PlotBy:=xlColumns, CategoryLabels:=1, SeriesLabels:=1, HasLegend:=True, _
        Title:="Sales"
```



Some primary ChartWizard arguments: (see Visual Basic Help for more information)

Source: The range that contains the source data for the new chart. If this argument is omitted, Microsoft Excel edits the active chart sheet or the selected chart on the active worksheet.

Gallery: The chart type. Can be one of the following XlChartType constants: xlArea, xlBar, xlColumn, xlLine, xlPie, xlRadar, xlXYScatter, xlCombination, xl3DArea, xl3DBar, xl3DColumn, xl3DLine, xl3DPie, xl3DSurface, xlDoughnut, or xlDefaultAutoFormat.

Format: The option number for the built-in autoformats. Can be a number from 1 through 10, depending on the gallery type. If this argument is omitted, Microsoft Excel chooses a default value based on the gallery type and data source.

PlotBy: Specifies whether the data for each series is in rows or columns. Can be one of the following XlRowCol constants: xlRows or xlColumns.

CategoryLabels: An integer specifying the number of rows or columns within the source range that contain category labels. Legal values are from 0 (zero) through one less than the maximum number of the corresponding categories or series.

SeriesLabels: An integer specifying the number of rows or columns within the source range that contain series labels. Legal values are from 0 (zero) through one less than the maximum number of the corresponding categories or series.

HasLegend: True to include a legend.

Title: The chart title text.

CategoryTitle: The category axis title text.

ValueTitle: The value axis title text.

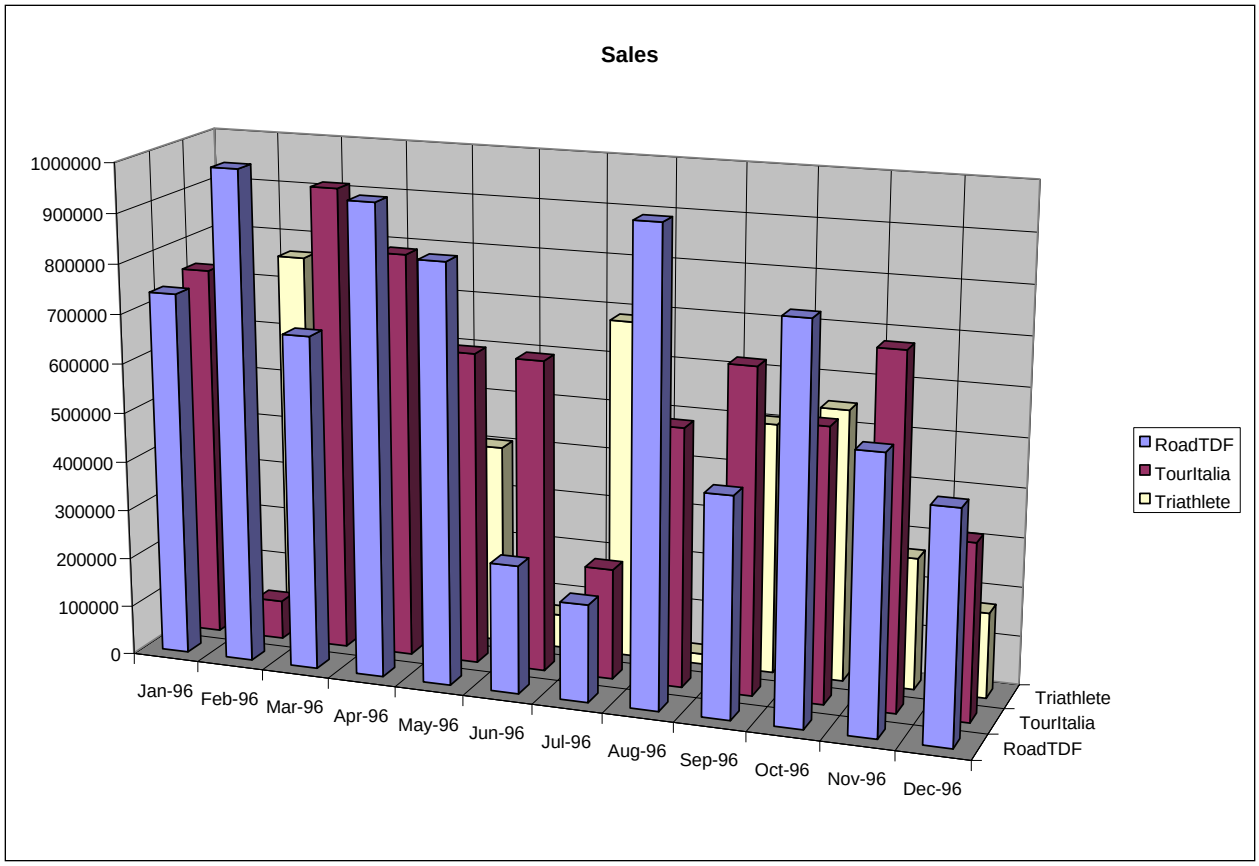
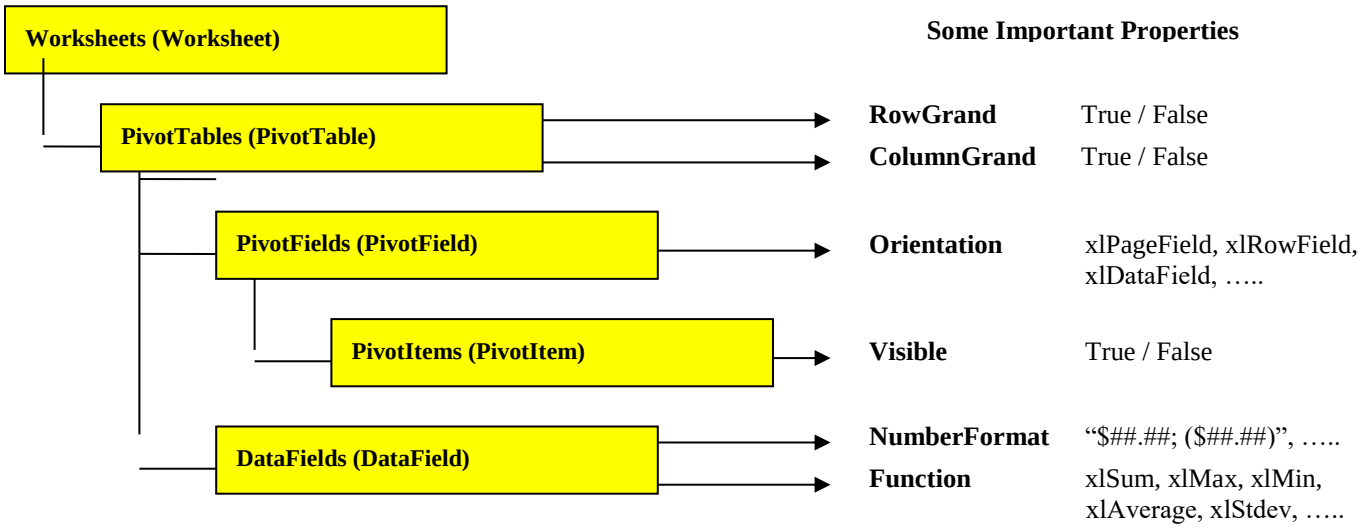


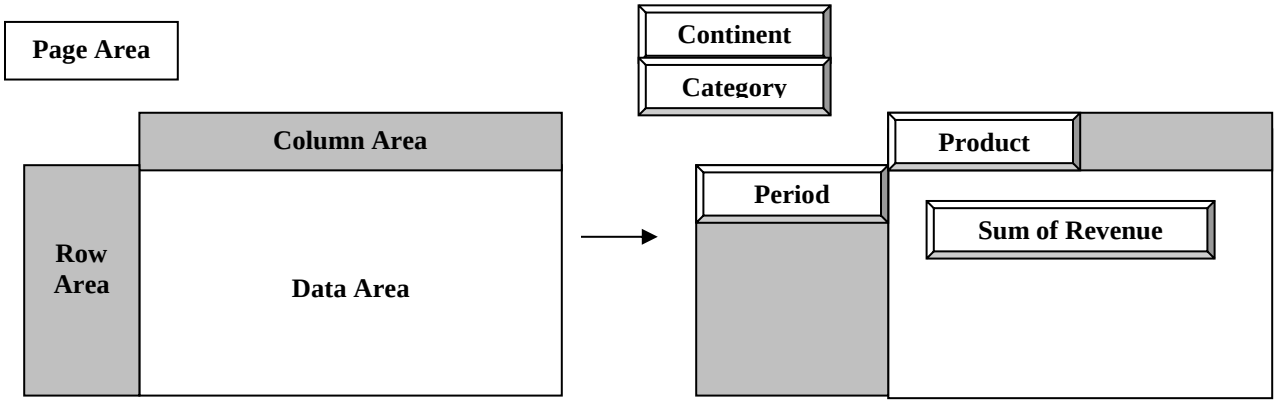
Chart2 (as created above)
(Data drawn from BikeData.mdb)

Finally, **Chart events**, such as clicking on a portion of the chart and having a message box pop up which gives an explanation of the data, can be defined for Chart objects that exist on a separate sheet (but not for charts embedded in worksheets).

PivotTables



PivotTable Structure:



The creator of the table can initially place the different fields in chosen areas within the PivotTable, using the PivotTable Wizard (in Excel under Data → PivotTable Report…)

The following are examples of how the PivotTable may be manipulated after the initial creation:

Ex: Re-assigning pivot fields to areas within a PivotTable:

```
Worksheets(1).PivotTables(1).Name = "SalesDataTable"
With Worksheets(1).PivotTables("SalesDataTable")
        .PivotFields("Category").Orientation = xlPageField
        .PivotFields("Product").Orientation = xlColumnField
        .PivotFields("Period").Orientation = xlRowField
        .PivotFields("Revenue").Orientation = xlDataField
        .PivotFields("Continent").Orientation = xlHidden
End With
```

Ex: Formatting/manipulating fields placed within the <u>Data Area</u>

```
With Worksheets(1).PivotTables("SalesDataTable")
        .DataFields(1).Function = xlAverage   'default: xlSum (numeric values) or xlCount (text values)
        .DataFields(1).NumberFormat = "$#,#00.00; ($#,#00.00)"
End With
'  - changes name of DataField(1) to "Average of FieldName" (ex: Average of Revenue), so best to _
        use the numerical index rather than the name in order to reference the fields in the data area.
```

Ex: Showing or hiding PivotItems

```
With Worksheets(1).PivotTables("SalesDataTable")
        For index = 1 to 5
                .PivotFields("Period").PivotItems(index).Visible = False
        Next
End With
```

Ex: Turning off (or on) row and column totals

```
Worksheets(1).PivotTables("SalesDataTable").RowGrand = False  ' or True
Worksheets(1).PivotTables("SalesDataTable").ColumnGrand = False  ' or True
```