

A SunCam online continuing education course

Programming MS Excel in Visual Basic (VBA) Part 4-Error Handling, Arrays, Dialog Boxes, Charts

by

Kwabena Ofosu, Ph.D., P.E., PTOE



Abstract

This course is the fourth of a four-part series on computer programming in *Excel Visual Basic for Applications* (VBA), tailored to practicing engineers. In this course the topics, error handling, arrays, dialog boxes, and charts and graphs are presented. Several examples relevant to engineering are used to illustrate and demonstrate the concepts and methods learned in this class. A mini-project is used to demonstrate the programming concepts and methods in a situation encountered by practicing engineers.

Computer Programming in Visual Basic (VBA) – Part 1, Part 2, and Part 3 are not required prerequisites to this course. It would however be helpful to understand the basic principles of computer programming as well as the fundamentals of the Excel VBA language as presented in Part 1, Part 2 and Part 3 of this series.



TABLE OF CONTENTS

Abstract
List of Tables
1. DIALOGUE BOXES
1.1 Input Box
1.2 File and Folder Dialog Boxes
1.3 msoFileDialogOpen
1.4 Customizing the Dialog Box
1.5 msoFileDialogFolderPicker
1.6 msoFileDialogFilePicker
1.7 msoFileDialogSaveAs
2. ARRAYS
2.1 Declaring an Array
2.2 Array Elements
2.3 Matrices
2.4 Dynamic Arrays
2.5 For Each Next Statement
3. ERROR HANDLING
3.1 Compiler Errors
3.2 Run-time Errors
3.3 Logic Errors
3.4 On Error Statements
3.5 On Error Goto 0
3.6 On Error Resume Next
3.7 On Error Goto < label >:
3.8 Error Handling Blocks
3.9 The Resume Statement
3.10 Caution on Error Handlers
3.11 Seventh VBA Project
4. GRAPHS



4.1 Chart Objects	
4.2 Creating Graphs	
5. CONCLUSION	
REFERENCES	



List of Tables

Table 6: Values for input box <i>Type</i> argument
--



1. DIALOGUE BOXES

A dialog box is a secondary window that enables an application to retrieve input from the user. Dialogue boxes are used to ask users questions, provide information to the user, or communicate feedback to or from the user. A **modal dialog box** requires the user to complete and close before continuing. In a **modeless dialog box** the user may choose not to complete it but the program execution will proceed.

Generally a dialogue box consists of the following:

Title bar: The title bar identifies the application or feature from which it originated.

Main instruction: This is some question or statement on the dialog box that communicates its primary purpose. It may also include an icon or symbol.

Content Area: The content area provides further descriptive information regarding the dialog box and its purpose.

Command Area: This consists of controls such as buttons or a textbox where the user communicates with the application.

Footnote area: This is an optional element that provides further guidance particularly for less experienced users.

For example, Open a new Excel session. Click on **Home** to open the **Home ribbon**.



Click on the **Find & Select** icon. Select **Replace**

mpatibili Traf	-	e] - Microsof Acrobat	ft Excel							x 9 - 0 x 9 - (?) a
enter 🕶			0	Conditional Fo ormatting ▼ as Styl		terenterenterenterenterenterenterentere	Delete Form	at 🛃	Fill T	Sort & Find & Filter * Select *
				2.9.					ab ac	<u>F</u> ind <u>R</u> eplace
	L	K	L	M	N	0	P	Q	*	<u>G</u> o To Go To <u>S</u> pecial Form <u>u</u> las Co <u>m</u> ments <u>C</u> onditional Formatting Co <u>n</u> stants Data <u>V</u> alidation
									\} ₽₽	Select Objects Selection Pane



The *Find and Replace* dialog box opens. This dialogue box can be used to search for a letter, word, phrase, sentence etc., in a document or code window, and replace it (or them) throughout the document with the entry in the *Replace with* control. It is required that the user make some selection or the other in the command area in order to dismiss the dialog box and proceed, even if no information is supplied in the *Find what* and *Replace with* controls. This is therefore modal dialog box.

			<mark></mark> ≫. ⊒ ≇ 1			Genera		v 00.00 0.€ 0	Conditiona Formatting
Font	5		Alig	nment		G.	Number	- Fa	
×									
D	E	F	G	Н	l I	J	K	l	L N
	Find and Find Find wh Replace	Replace	Replace	Find Al		INext	? ? ✓ Options >> Close		

In this chapter a selection of VBA dialog boxes will be reviewed.



1.1 Input Box

An input box is a special dialog box that requests a value from a user. The value is stored in some variable which is then called within some procedure as part of some manipulation or calculation, etc. In its simplest form, the input box is of the following structure.

Data Entry	×
Enter a number between 1 and 10	ОК
	Cancel

The user must enter the requested input and click on *OK* to dismiss the input box and enable the program to proceed. The user may decide to *Cancel* in which case the input box is dismissed and no user input is supplied to the program.

The syntax for the input box is

avariable = Application.InputBox(Prompt, Title, Default, Left, Top, HelpFile, HelpContextId, Type)

where,

Prompt is a required string data type, and is the message displayed in the main area of the dialog box.

Title is an optional variant data type, and is the title for the input box.

Default is an optional variant data type that specifies a value that will appear in the text box when the input box opens. If this argument is omitted, the text box will be empty. Note that this value can be a range object, for example, a range of cells on a spreadsheet.



Left is an optional variant type that specifies the horizontal position of the top left corner of the input box in relation to the upper left corner of your computer's monitor. If omitted, the input box will be centered horizontally.

Top is an optional variant type that specifies the vertical position of the input box in relation to the upper left corner of the computer screen. If omitted, the input box will be centered vertically.

HelpFile is an optional variant type that specifies the name of the Help file for the input box. If the *HelpFile* and *HelpContextID* arguments are provided, a Help button will appear on the dialog box.

Type is an optional variant type that specifies the return data type. If omitted, the input box returns a text data type. The values the *Type* argument can realize are as listed in Table 6.

For simple input boxes, the Application class is optional.

Type value	Returned data type
0	A formula
1	A number
2	Text (a string)
4	A logical value (True or False)
8	A cell reference, as a Range object
16	An error value, such as #N/A
64	An array of values

Table 1: Values for input box Type argument

Example 1: Simple Input Box:

Open a new Excel session.

Select Developer, and click on Design Mode to activate it.

Click on Insert, Select CommandButton from ActiveX controls to add one to the spreadsheet.

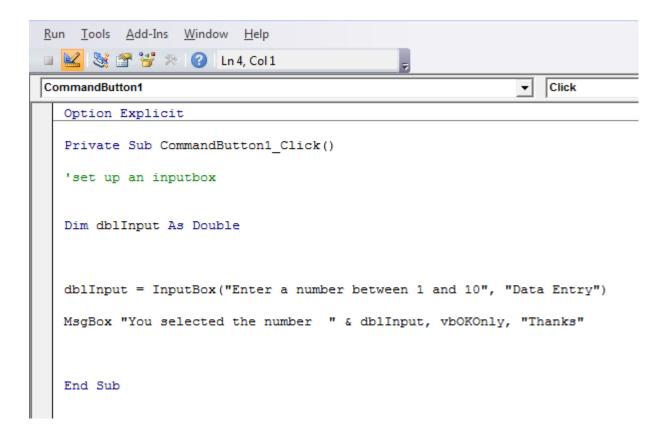


Trace out the command button onto the spreadsheet.

	Visual Macros Basic Macro Security Code					s COM Add-Ins	Insert Des Mo		Code	Sou	urce
C	ommandB	utton1 🔫	· (=	f_{x}	=EMI	BED("Forn	ns.Commar	ndButton.1	","")		
	А	В	С		D	E	F	G	Н		
1											
2	Input Box	es					_				
3							Ĭ	0	Ĭ		
4							Comm	andButton	1 6		
5									- [_		
6							J	-			
7											
8											
9											



Double click on the command button to enter the code window. Type the following code.



Clicking on the command button will activate an input box. This is an input box with a prompt and a title only (the first two arguments in the input box syntax). Note that the *Application* class has also been omitted. The user entry will be stored in the variable *dblInput*. The value of the variable holding the user entry value will then be displayed on a simple message box.

Save the file. Click on Debug, Select Compile VBAProject. Correct any errors and recompile, otherwise save and go to the spreadsheet. Click on Design Mode to deactivate it.



Click on the command button.

		I							-	
	f_{x}									
С	D	E	F	G	Н	1	J	К	L	М
			Comm	andButton	1					
			comm	andbutton						
				D	ata Entry					x
					Enter a numl	er between	1 and 10			
						Jer between	1 010 10		ОК	
									Cance	1
										_



Follow the prompt and enter a value and click on *OK*.

Data Entry	—
Enter a number between 1 and 10	OK Cancel
9	

CommandButton1	
	Thanks
	Thanks 💽

Click on *OK* to dismiss the message box. Save the file.



Example 2:Detailed Input Box

Create a login application where an employee must enter their full name, and then their four digit company ID number to enter the system. A message box will notify the user of a successful login.

Activate Design Mode.

Add a second command button to the spreadsheet.

Right click on the command button.

Select Properties.

Change the Caption property to *CommandButton2*.

Close Properties.

А	В	С	D	E	F	G	Н	
nput Box	(es							
					Comm	andButton	1	
								-
					9	0		
					Comm	andButtor	12	-
					0			
								-

Double click on the command button to go to the code window.



In this application, an input box will ask the user for their full name. A second input box will ask the user to enter their four digit ID number. Once the information is confirmed, a message box notifies the user of a successful login.

Type the following code.

```
Private Sub CommandButton2_Click()
Dim strYourName As String
Dim lngYourID As Long
'use input box to enter employee name
strYourName = Application.InputBox(prompt:="Enter your full name ", Title:="Employee Login", Type:=2)
'use input box to capture employee compnay ID number
lngYourID = Application.InputBox(prompt:="Enter your four digit employee number ", Title:="Employee Login", Type:=1)
'use message box to inform user that login is correct, access granted
MsgBox "Name: " & strYourName & vbNewLine & _
    "Employee ID: " & lngYourID & vbNewLine & vbNewLine & _
    "Employee login information confirmed.", vbOKOnly, "Access Granted"
End Sub
```

Note that in this case the full blown input box syntax is being used. Therefore the *Application* class is appended to the input box call. Note the data types being used to store the user inputs. Also note the argument values and how they are assigned with the ":=". Particularly, review the *Type* argument values.

Save the file. Click on Debug, Select Compile VBAProject. Correct any errors and recompile, otherwise Save and go to the spreadsheet. Click on Design Mode to deactivate it. Test the application.



Click on CommandButton2.

The command button event procedure executes, and the first input box opens.

Notice the change in the appearance of the input box due to the full blown syntax structure being used.

Employee Login	
Co OK Cancel]

Enter a name and click OK.

Employee Login		? 💌
Enter your full nar	ne	
James Williams		
	ОК	Cancel



The next input box pops up. Enter your four digit ID number.

	Employee Login		-?- - ×-
	Enter your four digit 9865	employee number	
Со	[ок	Cancel

Click on *OK*. The message box opens, confirming the user information.

CommandButton2	
commandbuttoriz	Access Granted
	Name: James Williams
	Employee ID: 9865
	Employee login information confirmed.



Click on *OK* to dismiss the message box. The application has been successfully tested.

Note that in this example, any name and four digit ID combination will result in a successful login. To ensure that only valid employees and ID numbers are granted access, the names and ID numbers of current employees would have to be used to set up a conditional statement. The variables capturing the user inputs would then be checked against the names and IDs in the conditional statement's logic, and if a match is found login, access will be granted, otherwise access will be denied. Given that two inputs will be checked, a nested conditional statement or a composite conditional statement, or some combination of both would be applicable.

1.2 File and Folder Dialog Boxes

File and folder dialog boxes enable a user to select a file or folder to open, or a folder to save a file to, through a dialog box. This is accomplished by using the VBA *FileDialog* object. There are a number of choices that can be applied. In each case, a unique value is passed to the *FileDialog* object.

1.3 msoFileDialogOpen

This choice of the VBA *FileDialog* object enables a user to select one or more files to open.

Open a new spreadsheet, and activate the Design mode. Insert a new ActiveX command button. Double click on the button to go to the code window.

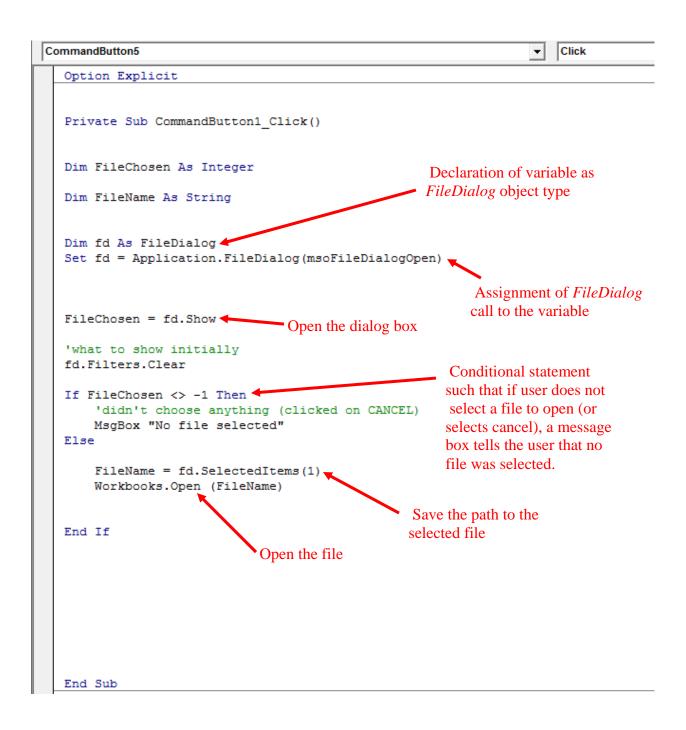


Type the following code.

```
CommandButton5
                                                                     Click
                                                                  Ŧ
  Option Explicit
  Private Sub CommandButton1_Click()
  Dim FileChosen As Integer
  Dim FileName As String
  Dim fd As FileDialog
  Set fd = Application.FileDialog(msoFileDialogOpen)
  FileChosen = fd.Show
   'what to show initially
  fd.Filters.Clear
  If FileChosen <> -1 Then
      'didn't choose anything (clicked on CANCEL)
      MsgBox "No file selected"
  Else
      FileName = fd.SelectedItems(1)
      Workbooks.Open (FileName)
  End If
  End Sub
```



Note the following highlights of the code.





Save the file.

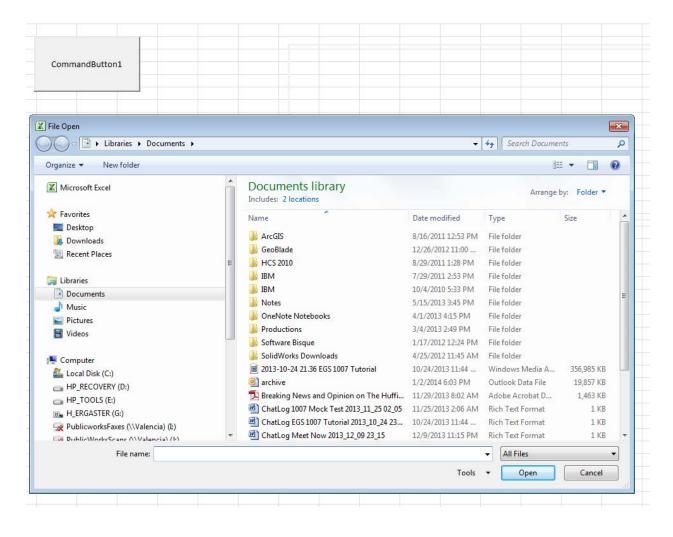
Click on Debug, Select Compile VBAProject.

Correct any errors and recompile, otherwise save and go to the spreadsheet.

Click on Design Mode to deactivate it

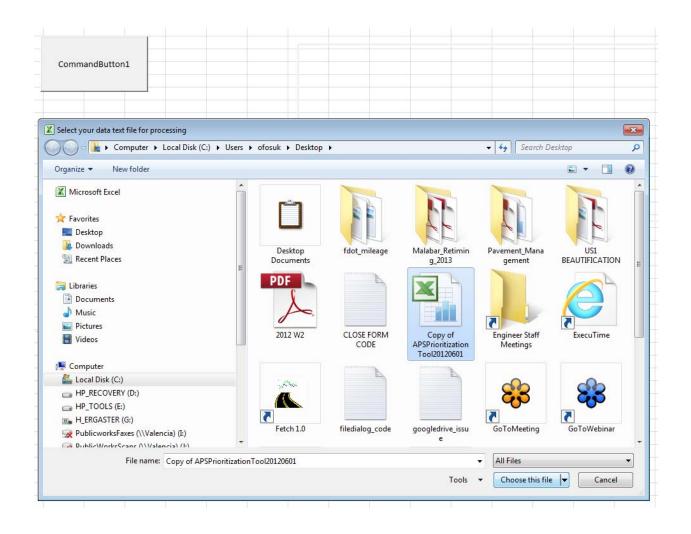
Click on the command button.

The dialog box opens.





Navigate to a file and select it, and click on the Open button to open the file.





The selected file opens. The dialog box is dismissed.

			M - 1-											
X		⊻7 - (Copy of A	PSPrioritizatio
	File	Ho	me Insert	Page Layou	ut Formulas	Data	Review	View	Developer	Traffic	Ac	robat		
- 4			Record Macro	D	<i>*</i> 📃	and the second	N 🖻	Properties		Map Prope				
L			Use Relative	References	× ⊳ -ŵ	X		View Code		Expansion	Packs	🔤 Export		
	asic	Macros	🔔 Macro Securi	ty	Add-Ins COM Add-Ins		Design Mode	Run Dialog	Source (Refresh Da	ta		Document Panel	
			Code		Add-Ins		Contro	ls		XML			Modify	
		A5	+ (0	f_x										
		Α	B C		E	F	G	Н	I J	K	L	N	1 N	0
1			Prioritization To		lation of Access		_	als, NCHRP 3	3-62, 2006					
2	_			S	Supplementa	al Work	sheet							
3	Lo	cation:												
4		ppleme	ntal Sketch											
-4	Ē									h				
5														
6		ppleme	ntal Notes							î				
		hhhhhh	ı											



Reactivate the dialog box

This time click on the *Cancel* button.

The dialog box is dismissed, and the message box pops up telling the user that a file was not selected.

CommandButton1	
	Microsoft Excel
	No file selected
	ОК

Click on the *OK* button to dismiss the message box.



1.4 Customizing the Dialog Box

The appearance of the dialog box can be customized by adjusting properties of the *FileDialog* object.

Copy and paste the command button.

Change the Caption of the new command button to CommandButton2 via its Properties.

Double click on the command button to enter the code window.

Type the following code that will adjust the following properties of the dialog box

- The Title.
- The default directory that opens in the dialog box.
- Clear any filters. The filter property enables only a certain type of file to be visible and selectable based on the filter value applied. Add a new filter such that only text files in a directory will be visible in the dialog box.
- Change the caption on the *Open* button.



```
End Sub
Private Sub CommandButton2 Click()
Dim fd As FileDialog
Set fd = Application.FileDialog(msoFileDialogOpen)
Dim FileChosen As Integer
Dim FileName As String
FileChosen = fd.Show
fd.Title = "Select your data text file for processing"
'initial path to display
fd.InitialFileName = "C:\Desktop"
fd.InitialView = msoFileDialogViewList
'what to show initially
fd.Filters.Clear
'we will not be using any filters here. Show us all files
'no lest show only the text files
fd.Filters.Add "Text files", "*.txt"
'fd.Filters.Add "Excel macros", "*.xlsx"
fd.FilterIndex = 1
fd.ButtonName = "Choose this file"
If FileChosen <> -1 Then
    'didn't choose anything (clicked on CANCEL)
    MsgBox "No file selected"
Else
    FileName = fd.SelectedItems(1)
   Workbooks.Open (FileName)
End If
End Sub
```



Save the file.

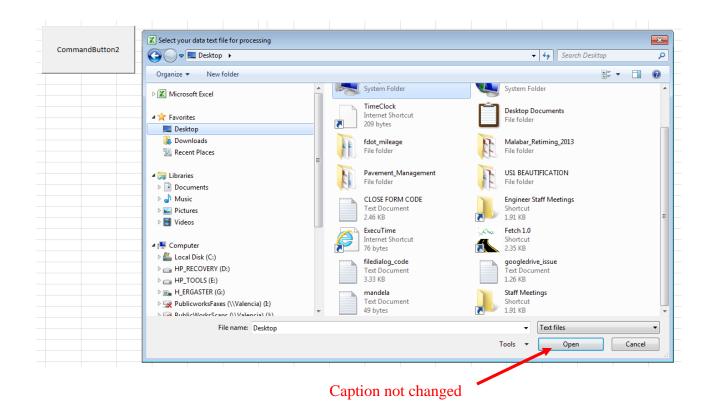
Click on Debug, Select Compile VBAProject.

Correct any errors and recompile, otherwise save and go to the spreadsheet.

Click on Design Mode to deactivate it.

Test the command button.

Click on CommandButton2.

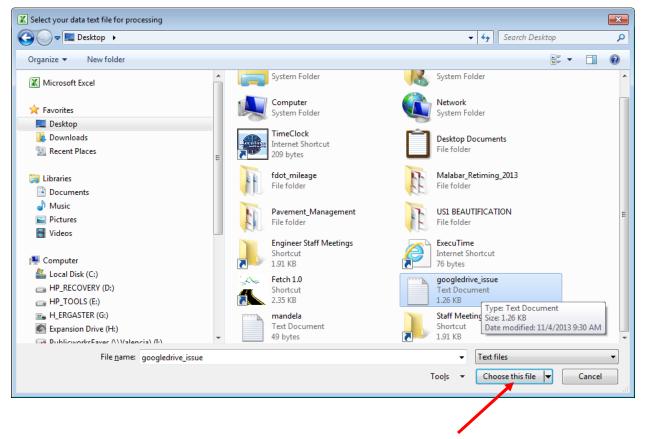


Note that due to the text file filter, regardless of which folder the user navigates to, only text files in that folder are visible and can be selected.

Review and confirm the customized features.

Navigate to a text file and open it (or select Cancel)





Caption changes to customized caption once a file is selected



The file opens (in Excel)

25.0-1								
25-Oct								
Hi Ofosusam,This is Kay with Google Drive Su	innort we were i	ust chatting a fey	v minutes ago ab	out the issues you y	vere having when	trying to share a vid	eo. Here are some s	tens that honefully y
in crosusuit, mis is kuy with coogle brive su	ipport, we were j	ust chutting u ret	initiates ago ab	out the issues you t	fere naving when	a ying to share a vie	co. Here are some s	teps that hoperany v
In regards to steps 5-8, if you don't have a fol	der that says x86	then just skip al	l of those steps.					
0 1 1 1								
Also, if you can, try downloading Chrome or a	another browser	to see if you are	able to share you	r videos there, it co	uld be an issue wi	th some settings or	extensions that spe	cific browser.
I hope that helps! Please feel free to reply if	you have further	questions.						
I hope that helps! Please feel free to reply if	you have further	questions.						
		questions.						
		questions.						
		questions.						
		questions.						
		questions.						
I hope that helps! Please feel free to reply if Thanks again for contacting Google Drive Sup		questions.						

The test is complete and successful.



1.5 msoFileDialogFolderPicker

This choice of the *FileDialog* object enables a user to select a path to a folder.

Activate the Design mode. Insert an ActiveX command button. Double click on the button to go to the code window. Type the following code.

```
Private Sub CommandButton3_Click()
Dim FileChosen As Integer
Dim fd As FileDialog
Set fd = Application.FileDialog(msoFileDialogFolderPicker)
FileChosen = fd.Show
If FileChosen <> -1 Then
    'didn't choose anything (clicked on CANCEL)
    MsgBox "No file selected"
Else
    MsgBox fd.SelectedItems(1)
End If
End Sub
```

This code is such that the folder path will be displayed on a message box.



Save the file.

Click on Debug, select Compile VBAProject.

Correct any errors and recompile, otherwise save and go to the spreadsheet.

Click on Design Mode to deactivate it.

Test the command button.

Click on CommandButton3.

Navigate to a folder.

K Browse							—
🚱 🗢 🖡 🕨 Kwabena Ofosu 🕨 Downloads 🕨			-	4 9	Search Down	nloads	٩
Organize 👻 New folder						•	0
K Microsoft Excel	^	Name	Date modified	Ту	pe	Size	
 ★ Favorites ■ Desktop Downloads 2000 Recent Places 2000 Recent Places 2000 Documents 2000 Music 2000 Pictures 2000 Pictures 2000 Videos 	I	Excel	7/12/2013 3:06 PM	File	e folder		
Computer Local Disk (C:) HP_RECOVERY (D:) HP_TOOLS (E:) H_ERGASTER (G:) PublicworksFaxes (\\Valencia) (L) PublicWorksFaxes (\\Valencia) (L) Folder name: Excel	+		Too <u>l</u> s	•	ОК	Cance	

Note that the contents of the folder are not visible, but only the path to the folder.

As with the previous example, the dialog box may be customized as needed.



Click OK.

The message box displays the path of the folder that was selected in the dialog box.

G	Н	I.	J	K	L	Μ	N
 Comm	andButtor	1					
Comma	andButton	2		Microsoft I	Fyrel		
Comm	andButton	2			.ofosuk\Dow	nloads\Exce	
comm	andbutton			-		ОК	

Click *OK* to dismiss the message box.

The test is complete and successful.



1.6 msoFileDialogFilePicker

This choice enables a user to select a path to one or more files, but will not open the file(s).

Activate the Design Mode. Insert an ActiveX command button. Double click on the command button to go to the code window. Type the following code.

```
Private Sub CommandButton4_Click()
Dim FileChosen As Integer
Dim fd As FileDialog
Set fd = Application.FileDialog(msoFileDialogFilePicker)
FileChosen = fd.Show
If FileChosen <> -1 Then
    'didn't choose anything (clicked on CANCEL)
    MsgBox "No file selected"
Else
    MsgBox fd.SelectedItems(1)
End If
End Sub
```



Save the file.

Click on Debug, Select Compile VBAProject.

Correct any errors and recompile, otherwise save and go to the spreadsheet.

Click on Design Mode to deactivate it.

Test the command button.

Click on CommandButton4.

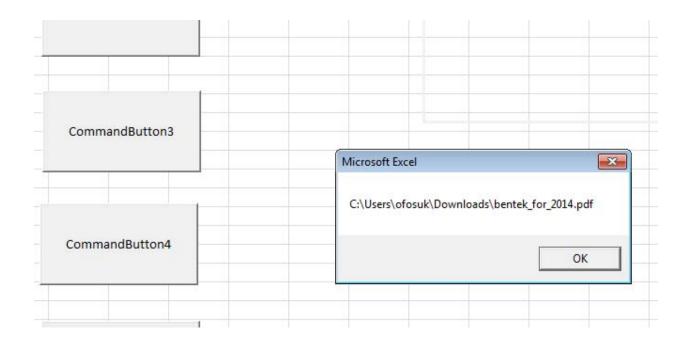
Navigate to a file, and select it.

G H I	Computer > Local Disk (C:) > 1	Users & ofocule & Downloads &	-	✓ Search Downlood	nds.
	Comparer y Edear Disk (C.) y	uses v olosak v bownioads v		···	143
	Organize 🔻 New folder			8	i 🔹 🗍 🌘
-	K Microsoft Excel	Name	Date modified	Туре	Size
CommandButton1		Excel	7/12/2013 3:06 PM	File folder	
	🚖 Favorites	5 B1384938	6/19/2013 1:54 PM	Adobe Acrobat D	447 KB
	Desktop	TA BASS PRO SHOPS MOT	11/14/2013 8:29 AM	Adobe Acrobat D	2,129 KB
	Downloads	benefits_for_2012	11/26/2011 9:14 PM	Adobe Acrobat D	34 KB
	🔠 Recent Places	bentek_for_2013	5/21/2013 8:17 AM	Adobe Acrobat D	29 KB
		bentek_for_2014	11/13/2013 9:42 PM	Adobe Acrobat D	29 KB
CommandButton2	詞 Libraries	T Brevard_3 PB-Melbourne UAB2	10/29/2013 8:16 AM	Adobe Acrobat D	189 KB
	Documents	Brevard_5 Func Class South-Draft	10/29/2013 8:20 AM	Adobe Acrobat D	225 KB
	J Music	CamStudio_2.7_r316_setup	5/16/2013 1:41 PM	Application	3,027 KB
	Pictures	1 ccv	5/14/2013 9:34 AM	Adobe Acrobat D	353 KB
	Videos 🖬	📃 🔁 cond prob test question	6/14/2013 11:11 AM	Adobe Acrobat D	233 KB
		🔁 dewatering and groundwater-I	1/24/2013 8:53 AM	Adobe Acrobat D	8,989 KB
CommandButton3	🖳 Computer	EGS1007_12619_FALL2013	8/26/2013 6:21 AM	Adobe Acrobat D	479 KB
	🚢 Local Disk (C:)	EGS2025_10759_FALL2013	8/26/2013 6:20 AM	Adobe Acrobat D	499 KB
	HP_RECOVERY (D:)	🔁 EM 1110-1-1905 - Bearing Capacity of Soi	1/23/2013 5:58 PM	Adobe Acrobat D	10,866 KB
	HP_TOOLS (E:)	🔁 EM1110-2-2504 Design of sheet pile retai	1/23/2013 5:57 PM	Adobe Acrobat D	798 KB
	R H_ERGASTER (G:)	🔁 FHWA Func Class Guidelines 2013	10/29/2013 8:25 AM	Adobe Acrobat D	2,731 KB
CommandButton4	🙀 PublicworksFaxes (\\Valencia) (E)	🔁 Florida Laws and Rules Course	1/4/2013 10:57 AM	Adobe Acrobat D	1,112 KB
commandbatton4	DublicWorksCane (\Valencia) (1)	AD GCEA coors2010Commin	6/10/2012 1.41 DM	Microroft Accorr	1 7/0 1/0
	File name: bentek_for_2014			✓ All Files	-
			Tools	- OK -	Cancel

As with the previous examples, the dialog box may be customized as needed.



The message box displays the path of the file that was selected in the dialog box.



Click *OK* to dismiss the dialog box.

The test is complete and successful.



1.7 msoFileDialogSaveAs

This choice enables a user to select a location to save a file to.

Activate the Design Mode. Insert an ActiveX command button. Double click on the command button to go to the code window. Type the following code.

```
End Sub
Private Sub CommandButton5_Click()
Dim FileChosen As Integer
Dim fd As FileDialog
Set fd = Application.FileDialog(msoFileDialogSaveAs)
FileChosen = fd.Show
If FileChosen <> -1 Then
    'didn't choose anything (clicked on CANCEL)
    MsgBox "No file selected"
Else
    ThisWorkbook.SaveAs (fd.SelectedItems(1))
End If
End Sub
```



Save the file.

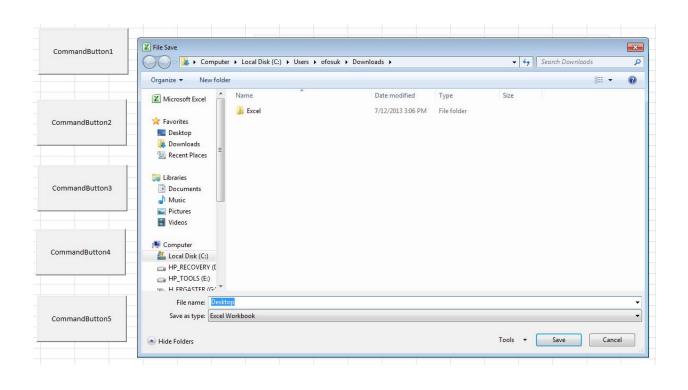
Click on Debug, select Compile VBAProject.

Correct any errors and recompile, otherwise save and go to the spreadsheet.

Click on Design Mode to deactivate it.

Test the command button.

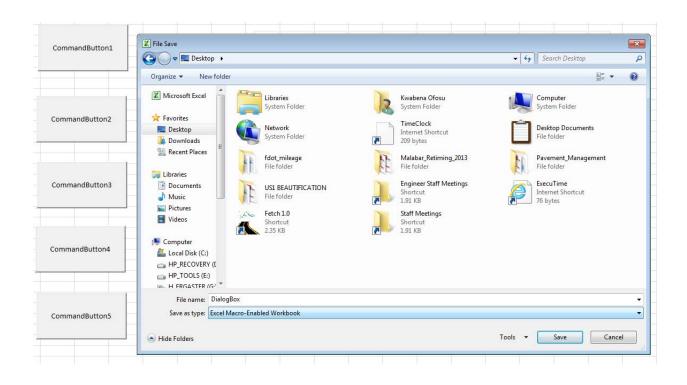
Click on CommandButton5.





Save a copy of the current file.

Navigate to a directory, and enter a name for the new file. Select an appropriate file suffix in the *Save as type* drop down



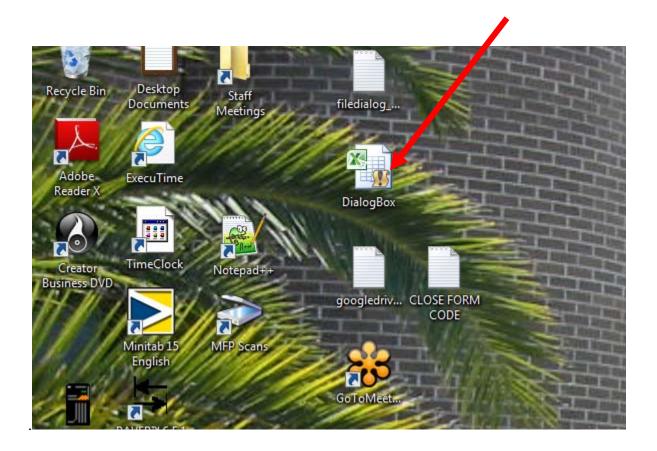
As with the previous examples, the dialog box may have been customized as needed.



Click on Save.

The dialog box dismisses.

The new file is created and saved with the name and directory location as selected and entered in the dialog box.



The test is complete and successful.



2. ARRAYS

An array is a variable that holds multiple values of the same data type. The individual values are the **elements** of the array. Each element has a unique **index** (or address) to identify it. For example, consider the test scores of students in a computer programming class. The score of each student can be stored individually and separately as a variable. Alternately, the scores of all of the students can be saved collectively as one variable using an array. Each individual test score would therefore be an element of the array.

In large and complex programs, arrays can dramatically reduce the number of variables and the amount of code written.

2.1 Declaring an Array

An array is declared in a *Dim* statement by giving it a variable name and appending the number of elements it will hold (this is called the **dimension list**) in parenthesis, and stating the data type. The syntax for declaring an array is as follows

Dim arryname(n) As datatype

where

n is the highest index, starting from zero. This part of the syntax is the dimension list.

For example, an array to hold the test scores (in percent) of a class of 25 will be of the form

Dim arrTests(24) As Integer

This array contains 25 elements, namely, *element 0*, *element 1*, *element 2*, ..., *element 24*.

Alternately, the declaration may be of the form



Dim arryname(firstindex To lastindex) As datatype

Therefore using this alternate dimension list format, the array for the test scores can be declared as

Dim arrTests(1 To 25) As Integer

In this case the first element is of index 1, and the last element is of index 25, for a total of 25 elements.

The choice of how to set up the dimension list is entirely up to the programmer.

2.2 Array Elements

An array element is identified or called as follows

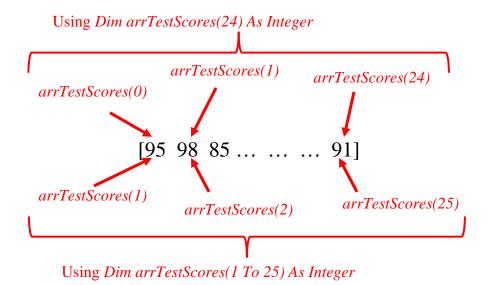
arrayname(i)

where

arrayname is the name of the array as declared in the declaration statement, and i is the index of the element.

The array elements for the arrTests array can be depicted as a list of items as follows





This array consists of one row of values and twenty five columns. This is a one-dimensional array. A one-dimensional array is also called a **vector**.

2.3 Matrices

A matrix is a multidimensional array. A matrix has more than one row, and more than one column.

The declaration of a matrix is as follows

Dim arryname(m, n) As datatype

where

m is the highest row index, whereby the first row is of index 0 n is the highest column index, whereby the first column is of index 0

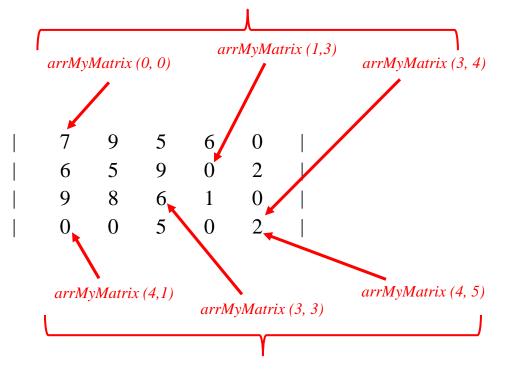
Alternately, the matrix may be declared as



Dim arryname(firstrowindex To lastrowindex, firstcolumnindex To lastcolumnindex) As datatype

For example, consider a *4 x 5* matrix, i.e. 4 rows and 5 columns, called *arrMyMatrix*, and holds integer values. The elements of the matrix are as follows





Using Dim arrMyMatrix(1 To 4, 1 To 5) As Integer



2.4 Dynamic Arrays

In some applications, it is unknown beforehand what size of array shall be needed. One strategy is to declare an excessively large array and use only those elements that are needed to meet the objectives of the project. This strategy however, is inefficient and may have serious impacts on the computer's memory. Another alternative is to use a **dynamic array**.

A dynamic array can be sized and resized, at any point in run time. The process to create a dynamic array involves

1. Declare the array without a dimension list

Dim arryname() As datatype

2. At the point in the code where the array needs to be sized/ resized, use the *ReDim* statement to size/ resize it with the relevant dimension(s). Note that the dimension(s) may be a value(s) stored in some variable(s) that is calculated or generated by the program in run time.

ReDim arryname(m, n)

Example: Consider an array that is used to compile the speeds of vehicles passing over a pneumatic traffic counting device installed on the highway, every 15 minutes. The device counts each vehicle and logs its speed. The number of vehicles that will pass in the next 15-minute period is random and unknown, therefore a static array will be inefficient and difficult to use for this application. A dynamic array can be created as follows.

Dim arrTrafficSpeeds() As Integer

Assuming that after a 15-minute period, 275 vehicles were recorded by the counter, the array can now be resized to accept the speed data for that 15-minute period.



ReDim arrTrafficSpeeds(1 To 275)

Assume that in the subsequent 15-minute period after that, 305 vehicles were recorded by the counter, the *ReDim* statement will be called again to resize the array for the current number of elements for processing as follows

ReDim arrTrafficSpeeds(1 To 305)

and so on and so forth.

It must be noted that the *ReDim* Statement clears all pre-existing elements and dimensions such that the array is now "empty", before resizing it to the new dimensions. In some situations this may be undesirable. To resize an array while keeping the pre-existing data, the *Preserve* keyword must be added to the *ReDim*.

Example: Consider that the traffic engineer would like to compile the speeds of the vehicles in a given hour into an array but the traffic counting device compiles the volumes every 15 minutes. Consider the four 15-minute traffic volumes in the coming hour will be 200, 300, 200, and 150 vehicles respectively. The hourly volume can be compiled from the 15-minute volumes as follows

At the beginning of the first quarter,

Dim arrTrafficSpeeds() As Integer

At the end of the first quarter,

ReDim arrTrafficSpeeds(1 To 200)



At the end of the second quarter, resize the array such that the pre-existing data will be preserved and together with the current period volume will all fit in the array.

ReDim Preserve arrTrafficSpeeds(1 To 500)

At the end of the third quarter, resize such that pre-existing and current can fit in the array. *ReDim Preserve arrTrafficSpeeds(1 To 700)*

At the end of the fourth quarter, resize such that pre-existing and current can fit in the array. *ReDim Preserve arrTrafficSpeeds*(1 To 850)

This completes the total for the hour.

Consider that now the engineer wants to reduce this new "hourly" array back down to contain only the first quarter dimensions, elements, and data. This can be done by *ReDim* but by reducing the size of the array.

ReDim Preserve arrTrafficSpeeds(1 To 200)

Only the first quarter elements will remain. All other elements will be eliminated.

2.5 For Each ... Next Statement

A *For Each* ... *Next* loop can be used to loop through each element in an array. For example consider vector of 20 elements that is multiplied by a scalar value of 5. Ordinarily a *For Next* loop can be used follows



For i = 1 *To* 20

arrMyVector(i) = 5**arrMyVector(i)*

Next

Alternately, the For Each loop can be used as follows

For Each i In arrMyVector()

arrMyVector(*i*) = 5**arrMyVector*(*i*)

Next

Note the keyword In. Also, the keyword Next is optional.

The *For Each* loop will enable a fixed number of iterations be performed even though the programmer may not know the number of iterations needed. For example, if the array is a dynamic array and its dimensions are continuously changing at various points in the program, it may be unclear what the current dimensions are, making the use of a *For Next* loop problematic. On the other hand, the *For Each* loop will apply the target operation(s) on all elements in the array regardless.

The general form of the For Each loop is therefore

For Each indexvariable In nameofarray()

'perform some operation on all the array elements

Next



3. ERROR HANDLING

It is common, even among seasoned programmers, that due to some error(s) in the code, the program may not work as expected, or it may run partially and prematurely terminate, or not run at all. Identifying errors and addressing them is called **debugging**. In programming, error handling refers to techniques and practices used to anticipate errors in advance, and to provide code to deal with them as they arise.

In general, errors can be grouped into three categories namely, **compiler errors**, **run-time errors**, and **logic errors**.

3.1 Compiler Errors

A compiler error occurs when a command, instruction, statement or expression cannot be executed or cannot be converted to machine language by the compiler. Common causes of compiler errors include errors in the syntax, misspelled keywords, incomplete branching or looping structures, undeclared variables, and inadmissible use of mathematical operators and functions among others. For example, consider the following error messages for compiler errors resulting from a missing *Next* statement for a For-loop, and an incomplete *If* statement, respectively.

Microsoft Visual Basic for Applications	Microsoft Visual Basic for Applications
Compile error: For without Next	Compile error: Expected: expression
OK Help	OK Help



The error causes the program to go into break mode.

Clicking on the *OK* button sends the programmer to the procedure in which the compiler error was detected.

```
Private Sub CommandButton1_Click()
Dim i As Integer
For i = 1 To 10
Cells(i, 1).Value = i
Next
End Sub
Private Sub CommandButton2_Click()
Dim i, j As Integer
For i = 18 To 35
For j = 1 To 5
Cells(i, j).Value = i + j
Next j
'Next i
End Sub
```

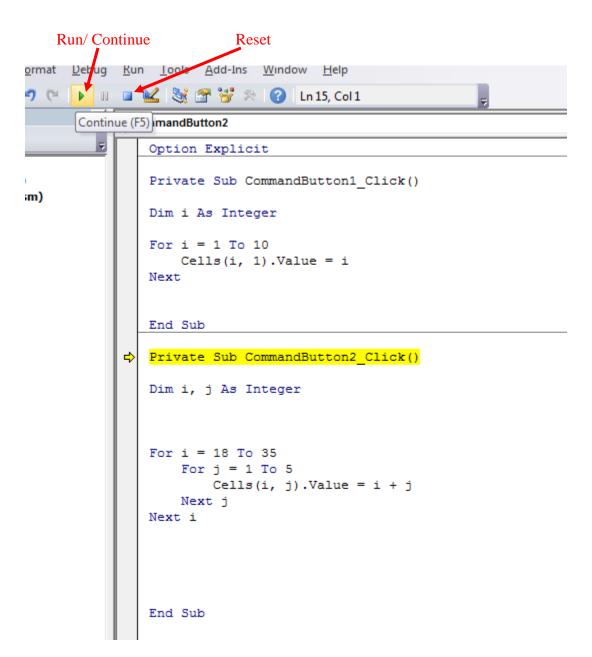


Clicking on the Help button (from the error message box) opens the *Excel Help* which provides more detailed information on the nature of the error, possible solutions, as well as links to user forums and search engine results on the topic.

🕐 Excel Help		_		23
📀 🔿 😒 🏠 🚑 Aš 🧼 🍳				Ŧ
✓ Ø Search ✓				
Excel Developer Home > Visual Basic for Applications Language Reference for Office 2010 > Visual	Basic Language Reference > Error Messages			
B3.Office	Search help Ding			
More	on Office.com: images templates			
For without Next This page is specific to the Visual Basic for Applications (VBA) Language Every For statement must have a matching Next statement. This error I A For statement is used without a corresponding Next statement. For additional information, select the item in question and press F1 (in W	has the following cause and solution: . Check for an incorrectly matched ForNext structure inside the outer ForNext st	ucture.		
Did this article help you? Yes No Not what I was looking for				
Developer Reference	Conne	cted to O	ffice.co	., mc



Once the offending code has been resolved – in this case removing the comment sign in front of the last *Next*, the programmer may click on *Run/ Continue* to resume from the current cursor location (which can be adjusted and must be chosen appropriately), or select *Reset*, in which case the program will have to be restarted from the beginning.





Before running a program the code may be checked for compiler errors by clicking on the **Debug** menu, and selecting **Compile VBAProject**. The code is compiled by the compiler (in other words it is converted to machine language (that the computer understands) without actually running the program. If the code cannot be compiled for any reason, an error message opens. From the error message the programmer can select to be taken to the region of the code where the error occurred, or go through the error message's help button to research additional information on the nature of the error, probable causes, and possible solutions.

着 Microsoft Visual Basic for Applications -	SecondVBAProject.xlsm - [UserForm1 (Code)]
File Edit View Insert Format	Debug Run Tools Add-Ins Window Help
: 🛛 🔄 - 📙 🔏 🖻 😭 💾 🔊 (*	Compi <u>l</u> e VBAProject
Project - VBAProject	Step Into F8
III III III	[J⊒ Step <u>O</u> ver Shift+F8
Solver (SOLVER.XLAM)	≧≣ Step O <u>u</u> t Ctrl+Shift+F8
UBAProject (FUNCRES.XLAM)	*≣ <u>R</u> un To Cursor Ctrl+F8 ncy
VBAProject (SecondVBAProject.xls	Add Watch tton1 Click()
Sheet1 (Sheet1)	Edit Watch Ctrl+W 1s button
Sheet2 (Sheet2)	6J Quick Watch Shift+F9 hold the input dat for line item 1
	Toggle Breakpoint F9 rSubtotal1 As Currency
E Forms	<u>Clear All Breakpoints</u> Ctrl+Shift+F9 ouble
	Set Next Statement Ctrl+F9
	Show Next Statement rSubtotal2 As Currency
Properties - UserForm1	Dim dblQuantity2 As Double 'declare variables to hold the input dat for line item 2 Dim curUnitPrice3, curSubtotal3 As Currency
UserForm1 UserForm Alphabetic Categorized	Dim dblQuantity3 As Double
(Name) UserForm1 BackColor &H8000000F& BorderColor &H80000012&	<pre>'assign variables with relevant textbox values</pre>
BorderStyle 0 - fmBorderStyleNone Caption PROJECT CALCULATOR Cycle 0 - fmCycleAllForms DrawBuffor 22000	'line item 1 curUnitPrice1 = UserForm1.TextBox1.Value dblQuantity1 = UserForm1 TextBox2 Value
IN RESIDENCE IN THE RESERVED	



3.2 Run-time Errors

Once a program has been successfully compiled and execution proceeds, it may happen that some value is processed or some resource is accessed in a manner that is inadmissible to the programming language. This will cause the program execution to terminate and abort with a runtime error.

For example, dividing some value or variable by zero will cause a run-time error as the value is mathematically indeterminate.

Cells(i, 1).Value = i Next	
E Microsoft Visual Basic	
P Run-time error '6':	
D Overflow	
i	
F	
Continue End Debug Help	
Next 1	

Note that in VBA each run-time error has a number identifier and a description in the main content area of the error message box. A comprehensive list of all the run-time errors built into the VBA language, and their descriptions can be found on the Microsoft website. (Microsoft, 2014a).



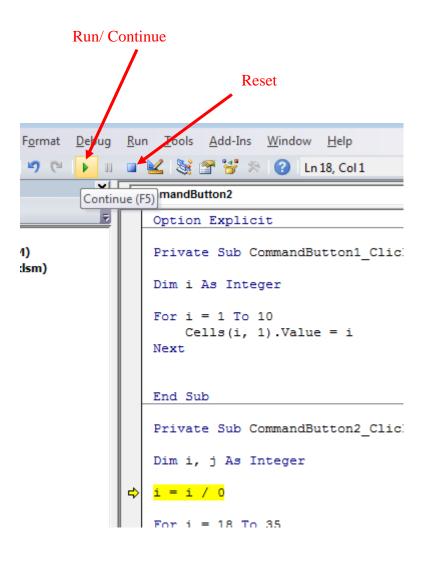
The error causes the program to go into break mode.

Clicking on the Debug button sends the programmer to the procedure in which the error occurred, and highlights the line of code on which the program was aborted.

```
End Sub
Private Sub CommandButton2_Click()
Dim i, j As Integer
i = i / 0
For i = 18 To 35
    For j = 1 To 5
        Cells(i, j).Value = i + j
        Next j
Next i
End Sub
```



Once the offending code has been resolved, the programmer may click on *Run/ Continue* to resume from the current cursor location (which can be adjusted and must be chosen appropriately), or select *Reset*, in which case the program will have to be restarted from the beginning manually.





Clicking on the *Help* button (from the error message box) opens the *Excel Help* which provides more detailed information on the nature of the error, examples, possible solutions, as well as hyperlinks and links to user forums and search engine results on the topic.

🕐 Excel Help 🗆 🗉	23
📀 🛞 🛞 🕼 🚔 🗛 🧇 🍳	Ŧ
- p Search -	
Excel Developer Home > Visual Basic for Applications Language Reference for Office 2010 > Visual Basic Language Reference > Error Messages	<u>^</u>
More on Office.com: images templates	
Overflow (Error 6)	
This page is specific to the Visual Basic for Applications (VBA) Language Reference for Office 2010.	
An overflow results when you try to make an assignment that exceeds the limitations of the target of the assignment. This error has the following causes and solutions:	E
The result of an assignment, calculation, or data type conversion is too large to be represented within the range of values allowed for that type of variable.	=
Assign the value to a variable of a type that can hold a larger range of values.	
An assignment to a property exceeds the maximum value the property can accept.	
Make sure your assignment fits the range for the property to which it is made.	
You attempt to use a number in a calculation, and that number is coerced into an integer, but the result is larger than an integer. For example:	
	- 11
Dim x As Long x = 2000 * 365 ' Error: Overflow	
To work around this situation, type the number, like this:	
Dim x As Long x = 2000 * 365 ' Error: Overflow	
Dim x As Long Developer Reference	.com

Clicking on the *End* button (from the error message box) stops the program execution altogether and sends the program to design time.



3.3 Logic Errors

Logic errors, also called data entry errors, and commonly called **bugs**, occur when the program runs "normally" but produces unexpected or undesirable results. In other words upon review, the programmer knows that the results are incorrect, but from the point of view of the compiler, the code is fine. Logic errors also exist if the program behaves erratically, for example results are saved to a spreadsheet that was not the intent of the programmer.

Due to the fact that the program will be running "normally", there will be no compiler or runtime errors for logic errors. This makes logic errors difficult to identify. The programmer must have some domain knowledge on the underlying theories and mathematical models being implemented in the program. The program must be tested repeatedly and the results thoroughly scrutinized and verified/ validated.

Some common causes of logic errors include

- omission of relevant code
- calling the wrong variables or functions
- declaring the wrong data type for a variable
- incorrect choice of branching and looping structures
- incorrect variables and/ or logic in conditional statements
- incorrect (loop) variables in loops
- incorrect sequence of instructions
- incorrect array dimensioning



3.4 On Error Statements

During run-time, if an error occurs, the applicable built-in run-time error message box will appear. Ordinarily, an end user will not know what the error message means or what steps should be taken to address it. As a result it is considered unacceptable programming practice for built-in error messages to open up to an end user. It is therefore the programmer's responsibility to anticipate potential errors that may occur and add code that will address them in such a manner that the built-in error messages do not open to an end user. This is the basis of error handling. Errors that are anticipated and addressed such that the built-in error message box does not appear to the user are referred to as **handled errors**, otherwise they are referred to as **unhandled errors**.

The *On Error* statement is the programming feature in VBA for error handling. It has three forms, namely

- On Error Goto 0
- On Error Resume Next
- On Error Goto <label>:

3.5 On Error Goto 0

In this mode VBA will display the applicable built-in standard run-time error message box and allow the user to enter the remedial code in break mode or to terminate the VBA program. It is technically the same as having no error handler. This is the default in VBA

Example: Consider a contractor compiling unit costs of work items on a project for the actual work done and the actual expenses incurred. The contractor would like to compare these with the budgeted unit prices used to prepare the bid for the job, in order to evaluate the current financial position.



The *Actual Unit Cost* for each item will be calculated by dividing the *Quantity* of work done by the *Actual Cost* for that item.

F	ile Home Insert Page	Layout	Formulas	Data Revi	iew View Develo	per Trat	ffic Acrobat				
	Calibri	- 11	I [™] A [*] A	. ≡ = =	📕 🗞 🖓 Wrap T	ext	General	*			
Paste v Format Painter B I U v ⊡ v 🎂 v 🚣 v ≣ ≣ ≣ 🛱 🛱 Merge & Center v \$								00. 0. → 0.♦ 00.	Conditional Form Formatting + as Tab		
	Clipboard 🗔 Font 🗔 Alignment 🗔 Number 🗔										
	D14 •										
	А	В	С	D	E		F		G		
1	Monitoring Actual Project Cost	ts									
2											
3	Activity	Unit	Quantity	Actual Cost	Actual Cost per Unit	Budgeted Cost per Unit		Margin/ Loss			
4											
5	Painting	SY	480	\$785.00			\$2.34				
6	Tiling	SY	248	\$1,250.00			\$4.25				
7	Pipes	LF	0	\$0.00			\$2.69				
8	Ceiling	SY	1,897	\$1,680.00			\$1.38				
9	Windows	EA	68	\$3,645.00			\$45.65				
10	Lighting Fixtures	EA	88	\$1,380.00			\$8.92				
11											
12											

A command button with a *For* loop will be used to iterate down the list. The profit or loss for the work done for that item is calculated by multiplying the difference of the *Budgeted Unit Cost* and the *Actual Unit Cost* by the *Quantity* of work done for that item. Note that no pipe work was done and therefore that item has a quantity of zero which will cause an error when it divides the actual cost for pipe work.

	A	В	С	D	E	F	G	Н	1	J
1	Monitoring Actual Project Cos	ts								
2										
3	Activity	Unit	Quantity	Actual Cost	Actual Cost per Unit	Budgeted Cost per Unit	Margin/Loss			10.11.4
4									Comma	ndButton1
5	Painting	SY	480	\$785.00	\$1.64	\$2.34	(\$338.21)		<u></u>	
6	Tiling	SY	248	\$1,250.00	\$5.04	\$4.25	\$195.99			
7	Pipes	LF	0	\$0.00		\$1.69				
8	Ceiling	SY	1,897	\$1,680.00		\$1.38				
9	Windows	EA	68	\$3,645.00		\$45.65				
10	Lighting Fixtures	EA	88	\$1,380.00		\$8.92				
11										
12										



The *On Error Goto 0* statement is the same as having no error handler at all. On executing the program, therefore, the standard run-time error Number 6 pops up, and the program goes into break mode to enable the user correct the code and re-run the program or terminate altogether.

🚰 Microsoft Visual Basic - vba tutorial.xlsm	r · · · · · · · · · · · · · · · · · · ·
	Microsoft Visual Basic
Eile Edit View Insert Format	Type a question for help
	Run-time error '6':
Project - VBAProject	Overflow
🕀 😻 Internet_Assistant (HTML.XLA	
🕀 💩 SOLVER (SOLVER.XLAM)	
🗄 🚳 sumif (SUMIF.XLAM)	
VBAProject (FUNCRES.XLAM)	Continue End Debug Help
🖻 🐯 VBAProject (vba_tutorial.xlsm	conditional condit
Module1	
Sheet1 (Sheet1)	'declare variable to move up or down the table
Sheet10 (Sheet10)	Dim i, lastrow As Integer
■ Sheet12 (Sheet12)	'identify the last row with data on your spreadsheet
■ Sheet13 (Sheet14)	lastrow = Range ("A65536").End (x1Up).Row
Sheet14 (Sheet13)	Tastiow - Kange (Rosso) .End (Riop) .Kow
Sheet15 (Sheet15)	' compute in a loop for each activity on the list
Sheet16 (Sheet16)	For i = 5 To lastrow
Sheet17 (Sheet17)	'extract data from spreadsheet
Sheet2 (Sheet2)	quantity = Sheets("Sheet17").Cells(i, 3).Value
Sheet3 (Sheet3)	<pre>actualcost = Sheets("Sheet17").Cells(i, 4).Value</pre>
Sheet4 (Sheet4)	<pre>budcostperunit = Sheets("Sheet17").Cells(i, 6)</pre>
Sheet5 (Sheet5)	'calculate actual cost per unit
Sheet6 (Sheet6)	accostperunit = actualcost / quantity
Bheet7 (Sheet7)	Sheets("Sheet17").Cells(i, 5).Value = accostperunit
Sheet8 (Sheet8)	'calculate profit or los
Sheet9 (Sheet9)	profitorloss = (accostperunit - budcostperunit) * quantity
·····································	Sheets("Sheet17").Cells(i, 7).Value = profitorloss
	Next



Obviously, this is not desirable from the point of view of a non-expert end user. The programmer needs to add code to handle this error, knowing ahead of time that if any work item is not done and has *Quantity* of zero, this error will continually occur.

3.6 On Error Resume Next

This error handler causes the program to ignore the built-in error message and resume execution of the code on the next line. Following the sequence of code execution in a procedure in VBA - from top to bottom, the error handler code must be placed towards the top of the procedure, or above the code where the programmer thinks may be prone to some error, such that any code below it that triggers an error message will then be subject to the pre-existing error handler.

It is important to note however that this error handler simply ignores the error and continues code execution on the next line and does not resolve the problem generating the error whatsoever.

Example:

Continuing from the contractor's unit pricing analysis in the previous section, add an *On Error Resume Next* handler in anticipation of errors resulting from the *Actual Unit Price* calculation, particularly the case where division by zero may occur.

F	ile Home Inse	rt Page	Layout	Formulas	Data Revi	iew View Develo	per Traffic Acrobat				
	$ \begin{array}{c c} & & \\ \hline \\ & \\ &$										
	Paste 🗸 Format Painter 🖪 🖌 🗓 ▾ 🔄 ་ 🏠 ▾ 🖺 苇 🚍 🗐 🏗 🏗 🖼 Merge & Center ▾ 💲 ▾ % , 號 🐝										
	Clipboard 🖙 Font 🖙 Alignment 🖙 Number 🖙										
	D14	• (=	f_{x}								
	A		В	С	D	E	F		G		
1	Monitoring Actual P	roject Cost	s								
2											
3	Activity		Unit	Quantity	Actual Cost	Actual Cost per Unit	Budgeted Cost per Unit	Ma	argin/ Loss		
4											
5	Painting		SY	480	\$785.00		\$2.34				
6	Tiling		SY	248	\$1,250.00		\$4.25				
7	Pipes		LF	0	\$0.00		\$2.69				
8	Ceiling		SY	1,897	\$1,680.00		\$1.38				
9	Windows		EA	68	\$3,645.00		\$45.65				
10	Lighting Fixtures		EA	88	\$1,380.00		\$8.92				
11											
12											



```
Click
CommandButton2
  Option Explicit
  Private Sub CommandButton1 Click()
  'declare variables for the calculation
  Dim quantity As Double
  Dim actualcost, accostperunit, budcostperunit, profitorloss As Currency
  'declare variable to move up or down the table
  Dim i, lastrow As Integer
  'identify the last row with data on your spreadsheet
  lastrow = Range("A65536").End(xlUp).Row
  ' compute in a loop for each activity on the list
  For i = 5 To lastrow
       'if an error occurs in the calcs just skip and continue
      'code execution on the next line
      On Error Resume Next 🗲
      'extract data from spreadsheet
      quantity = Sheets("Sheet17").Cells(i, 3).Value
      actualcost = Sheets("Sheet17").Cells(i, 4).Value
      budcostperunit = Sheets("Sheet17").Cells(i, 6)
      'calculate actual cost per unit
      accostperunit = actualcost / quantity
      Sheets("Sheet17").Cells(i, 5).Value = accostperunit
      'calculate profit or los
      profitorloss = (accostperunit - budcostperunit) * quantity
      Sheets("Sheet17").Cells(i, 7).Value = profitorloss
  Next
  End Sub
```



No error message is displayed for the division by zero as it has been handled by the *On Error Resume Next* handler. Notice that for the item where this error occurred, no new output values were calculated. The program simply ignored the error and went to the next line to continue. The values calculated from the previous item were brought over and displayed per the code for the offending line item. The error handler did not attempt to address the problem, or adjust the values displayed for that line item. This may be acceptable in some cases, however in most cases the programmer may want to not only handle the error, but also attempt to fix the problem or adjust the relevant output values to reflect the situation that, as in this case, output values were not calculated for that line item.

_	. (e	fx			200			
1	A	В	С	D	E	F	G	н
1	Monitoring Actual Proje	ct Costs						
2 3 4	Activity	Unit	Quantity	Actual Cost	Actual Cost per Unit	Budgeted Cost per Unit	Margin/Loss	
5	Painting	SY	480	\$785.00	\$1.64	\$2.34	(\$338.21)	
6	Tiling	SY	248	\$1,250.00	\$5.04	\$4.25	\$195.99	
7	Pipes	LF	0	\$4,985.00	\$5.04	\$2.69	\$0.00	
8	Ceiling	SY	1,897	\$1,680.00	\$0.89	\$1.38	(\$937.88)	-
9	Windows	EA	68	\$3,645.00	\$53.60	\$45.65	\$540.80	
10	Lighting Fixtures	EA	88	\$1,380.00	\$15.68	\$8.92	\$595.04	
11								
12								
13								
14								
15								

3.7 On Error Goto < label >:

In this error handler, once the error is handled, execution "jumps" to after the line listed in the label. The code between the error location and the label is ignored.

As with the On Error Goto 0, and On Error Goto Next, the error is not fixed, it is simply bypassed.

Example:

Consider the contractor's *Actual Unit Price* calculations for a project as reviewed in the previous section. Add the line label called *ErrHandler*. In other words if an error occurs after the *On Error Goto <label>*, the program will look for the line labeled *ErrHandler* and continue



executing the code after it. Note the syntax used in the line label. Code is added such that if or when an error occurs in the calculations as a result of inadmissible data, the user is alerted with a message box and any further calculations are stopped. In other words the program is terminated.

```
End Sub
Private Sub CommandButton2 Click()
'declare variables for the calculation
Dim quantity As Double
Dim actualcost, accostperunit, budcostperunit, profitorloss As Currency
'declare variable to move up or down the table
Dim i, lastrow As Integer
'identify the last row with data on your spreadsheet
lastrow = Range("A65536").End(xlUp).Row
' compute in a loop for each activity on the list
For i = 5 To lastrow
    'if an error occurs in the calcs just skip and continue
    'code execution on the next line
    On Error GoTo ErrHandler:
    'extract data from spreadsheet
    quantity = Sheets("Sheet17").Cells(i, 3).Value
    actualcost = Sheets("Sheet17").Cells(i, 4).Value
   budcostperunit = Sheets("Sheet17").Cells(i, 6)
    'calculate actual cost per unit
   accostperunit = actualcost / quantity
    Sheets("Sheet17").Cells(i, 5).Value = accostperunit
    'calculate profit or los
    profitorloss = (accostperunit - budcostperunit) * quantity
    Sheets("Sheet17").Cells(i, 7).Value = profitorloss
Next
ErrHandler:
    ' error handling code
    'In this case don't do anything. We could set up code
   'here to fix the problem
   MsgBox ("An error has been detected. Program shall terminate.")
End Sub
```



	~ (9	<i>f</i> _x -338	.21							
	A	В	С	D	E	F	G	Н	1	J
1	Monitoring Actual Project Cost	s								
2										
3	Activity	Unit	Quantity	Actual Cost	Actual Cost per Unit	Budgeted Cost per Unit	Margin/ Loss			
4									Comman	dButton1
5	Painting	SY	480	\$785.00	\$1.64	\$2.34	(\$338.21)			
6	Tiling	SY	248	\$1,250.00	\$5.04	\$4.25	\$195.99			
7	Pipes	LF	0	\$0.00		\$1.69				
8	Ceiling	SY	1,897	\$1,680.00		\$1.38			-	
9	Windows	EA	68	\$3,645.00		\$45.65			Comman	dButton2
10	Lighting Fixtures	EA	88	\$1,380.00		\$8.92				
11										
12										
13										
14					Microsoft Exc	el	×			
15										
16					An error has	been detected. Program shal	ll terminate.			
17										
18										
19							ОК			
20										
21										

Note that if the handler label was placed inside the *For*-loop, execution would have continued for the other entries down the table, and only the line(s) causing the error would be skipped.

	• (9	<i>f</i> _x -338	3.21							
	A	В	С	D	E	F	G	Н	1	J
1	Monitoring Actual Project Cost	is								
2										
3	Activity	Unit	Quantity	Actual Cost	Actual Cost per Unit	Budgeted Cost per Unit	Margin/Loss		c	10.000
4		2							Comman	ndButton1
5	Painting	SY	480	\$785.00	\$1.64	\$2.34	(\$338.21)			
6	Tiling	SY	248	\$1,250.00	\$5.04	\$4.25	\$195.99			
7	Pipes	LF	0	\$0.00		\$1.69				
8	Ceiling	SY	1,897	\$1,680.00		\$1.38				10 11 0
9	Windows	EA	68	\$3,645.00		\$45.65			Comma	ndButton2
10	Lighting Fixtures	EA	88	\$1,380.00		\$8.92			<u></u>	
11										



3.8 Error Handling Blocks

When on On Error statement is executed, the error handler is said to be **enabled**.

When an error occurs and execution is transferred to another location by way of the *On Error Goto <label>:* statement, the code that executes is called an **active error handler**.

The section of code to which execution is transferred through the *On Error Goto <label>:* statement is called an **error handling block**. The code of the error handling block may be designed to fix the problem causing the error and resume "normal" execution, or to terminate the procedure.

3.9 The Resume Statement

The *Resume* statement causes a program to resume execution at a specific line. The *Resume* statement can only be used in an error handling block. Apart from exiting a procedure, the *Resume* statement is the only way to exit out of an error handling block. There are three syntactic forms of the Resume statement, namely,

- Resume
- Resume Next
- Resume <label>

Resume causes execution to resume on the line where the error was detected. The error handling block associated with the *Resume* must therefore fix the problem that caused the error to begin with.

Resume Next will cause execution to continue on the line immediately following the line of code where the error occurred.

Resume <*label*> causes code execution to resume at a specified labeled location. Therefore this format enables some section of code to be skipped if an error occurs.

Example:

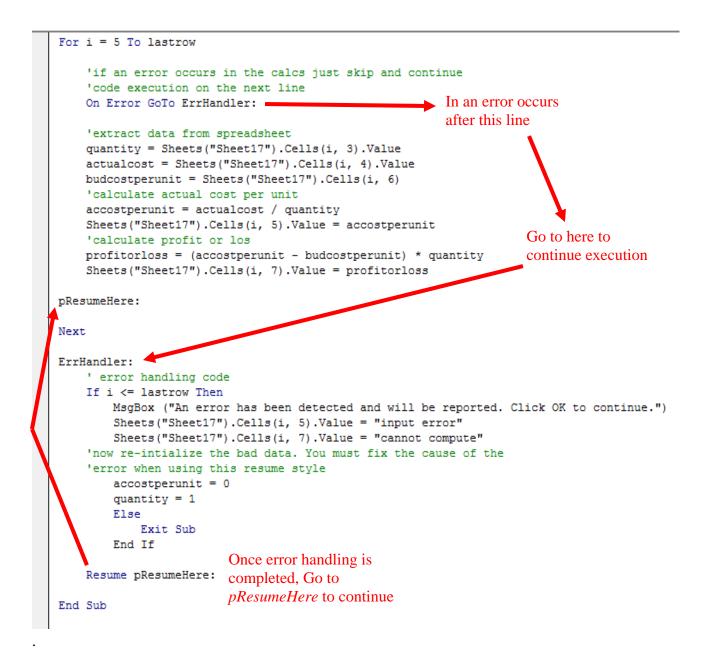
Add the following error handling block and *Resume* statement to the code for the calculation of *Actual Unit Cost* in the previous section.



```
For i = 5 To lastrow
    'if an error occurs in the calcs just skip and continue
    'code execution on the next line
    On Error GoTo ErrHandler:
    'extract data from spreadsheet
    quantity = Sheets("Sheet17").Cells(i, 3).Value
    actualcost = Sheets("Sheet17").Cells(i, 4).Value
    budcostperunit = Sheets("Sheet17").Cells(i, 6)
    'calculate actual cost per unit
    accostperunit = actualcost / quantity
    Sheets("Sheet17").Cells(i, 5).Value = accostperunit
    'calculate profit or los
    profitorloss = (accostperunit - budcostperunit) * quantity
    Sheets("Sheet17").Cells(i, 7).Value = profitorloss
pResumeHere:
Next
ErrHandler:
    ' error handling code
    If i <= lastrow Then
        MsgBox ("An error has been detected and will be reported. Click OK to continue.")
        Sheets("Sheet17").Cells(i, 5).Value = "input error"
        Sheets("Sheet17").Cells(i, 7).Value = "cannot compute"
    'now re-intialize the bad data. You must fix the cause of the
    'error when using this resume style
        accostperunit = 0
        quantity = 1
        Else
            Exit Sub
        End If
    Resume pResumeHere:
End Sub
```



Note that when an error occurs in the *Actual Unit Cost* calculation on any line, execution is transferred to the line after the label *ErrHandler*, the error handling block. Here text entries "input error" and "cannot compute" are assigned to spreadsheet cells which would have otherwise displayed the *Actual Unit Cost* and the *Profit/Loss* respectively. The *Resume* statement with label *pResumeHere* then exits the program out of the error handling block and back into the *For*-loop so that the calculations proceed for the next line item in the list, and down the table.





The results are therefore as follows.

1	Monitoring Actual Project Costs						
2	Activity	Unit	Quantity	Actual Cost	Actual Cost per Unit	Budgeted Cost per Unit	Margin/ Loss
3							
4							
5	Painting	SY	480	\$785.00	\$1.64	\$2.34	(\$338.21)
6	Tiling	sv	248	\$1,250,00	\$5.04	\$4.25	\$195.99
7	Pipes	LF	0	\$0.00	input error	\$1.69	cannot compute
8	Ceiling	SY	1,897	\$1,680.00	\$0.89	\$1.38	(\$937.88)
9	Windows	EA	68	\$3,645.00	\$53.60	\$45.65	\$540.80
10	Lighting Fixtures	EA	88	\$1,380.00	\$15.68	\$8.92	\$595.04
11							
12							
10							

3.10 Caution on Error Handlers

Error handling techniques are often used to skip or jump over problematic code, and code which the programmer can conclude is not meeting expectations, but may be finding difficulty in debugging. Such uses are considered misuse and abuse of the error handling methods. The programmer must identify and fix the bugs, and test the code to confirm that it is working as planned. It is recommended that potential situations that may cause errors should be thoroughly researched and if possible, accommodated for in the code. If this is not possible, then error handlers should be used in anticipation of such potential problems.

3.11 Seventh VBA Project

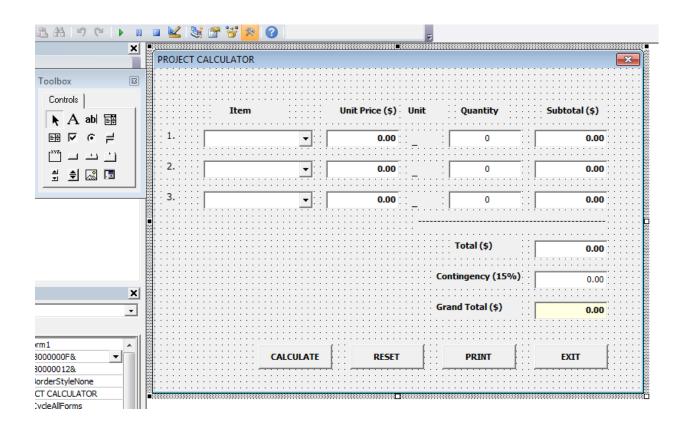
Problem Statement

In this project the project calculator of the *Fifth VBA Project* will be developed further, incorporating array methods in the calculations as well as error handling to deal with unforeseen cases and inadmissible inputs.



Solution

The framework for the application is as follows.



Review the code and identify all the functions that were created and the function calls that were embedded in the other procedures. Select and delete all functions and function calls embedded in other procedures.



For instance, the function *clcSubTotal1*,

(General) ClcSubTotal1
	Option Explicit
I 1	Private Function clcTotal() As Currency
	'this function will calculate the total by summing the sub totals
	clcTotal = clcSubTotal1 + clcSubTotal2 + clcSubTotal3
1	
1	
	End Function
	Private Function clcSubTotal1() As Currency
1	'this function will calculate the subtotal for line item 1
1	
1	'first pull the information needed
1	
	declare variables that will be needed
	Dim curUnitPrice, curSubTotal As Currency
	Dim dblQuantity As Double
	'now get the values
	'get the unit price variable value from the user form unit price text box
	curUnitPrice = UnitPrice1.Value
	get the quantity variable value from the user form quantity text box
	dblQuantity = Quantity1.Value
	'we can now calculate the sub total
1	curSubTotal = curUnitPrice * dblQuantity
1	
1	'now, as required for functions, assign the result to the
1	'a variable bearing the function name, so the function
	'can return the value
	clcSubTotal1 = curSubTotal
	End Function
	Private Function clcSubTotal2() As Currency
	'this function will calculate the subtotal for line item 1



'and its calls within the UnitPrice1 and Quantity1 change procedures.

```
ComboBox1
                                                               Ŧ
                                                                  Change
Ē
      Option Explicit
     Private Sub ComboBox1 Change()
     If ComboBox1.Value = "Concrete" Then
          'if concrete is selected
         UnitPrice1.Value = "85.35"
         Label31.Caption = "CY"
     ElseIf ComboBox1.Value = "Asphalt" Then
         'if asphalt is selected
         UnitPrice1.Value = "35.45"
         Label31.Caption = "TON"
     ElseIf ComboBox1.Value = "Road Markings" Then
          'if road markings is selected
:
         UnitPrice1.Value = "6.95"
         Label31.Caption = "LF"
     Else
          'if none of the above
          UnitPrice1.Value = "0.00"
         Label31.Caption = " "
     End If
       calculate/ recalculate the sub total 1 value anytime
       you make a change.
      call the function, and display its value on the form
          Subtotal1.Value = clcSubTotal1
       call the function to calculate the Total, and the contigency
          Total.Value = clcTotal
         GrandTotal.Value = clcTotal + 0.15 * clcTotal
      End Sub
     Private Sub Quantity1 Change()
      'calculate/ recalculate the sub total 1 value anytime
      'you make a change.
      'call the function
          'call the function, and display its value on the form
         Subtotal1.Value = clcSubTotal1
      'call the function to calculate the Total, and the contingency
         Total.Value = clcTotal
          GrandTotal.Value = clcTotal + 0.15 * clcTotal
      End Sub
```



Repeat for functions *clcSubTotal2*, clc*SubTotal3*, clc*Total*.

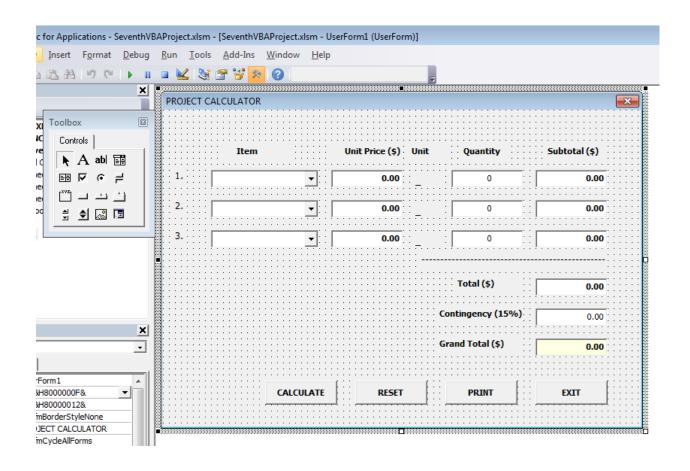
Save the workbook. Click on Debug. Select Compile VBAProject.

If there no errors, particularly errors referring to undefined variables and unknown functions, then all functions have been deleted. If not review the code again and remove any functions and function calls.

Resave the workbook.

Open the design window.

Rename the Calculate and Print buttons CommandButton5 and CommandButton6 respectively.





Declare the following **global** arrays

- A vector to hold the *Item* values
- A matrix to hold the *Unit Prices*, the *Quantities*, and the *Subtotals*.
- A vector to hold the *Total*, *Contingency*, and the *Grand Total*.

```
ComboBox1
ComboBox1
Change
Option Explicit
Dim strArrItems(1 To 3) As String
Dim curArrCalculationValues(1 To 3, 1 To 3) As Currency
Dim curArrTotals(1 To 3) As Currency
Private Sub ComboBox1_Change()
If ComboBox1.Value = "Concrete" Then
'if concrete is selected
UnitExist? Value = "SE 25"
```

Global variables or in this case global arrays are declared outside of any procedure, but must be assigned values from within a procedure. Unlike local variables that "die" once a procedure has executed to completion, global variables are still "alive" as long as the module is active. In this case the array values shall be printed from the application window "long" after the procedures that generated their values have run to completion and "died".



Double click on the *Calculate* button to open its click procedure.

×	C	ommandButton5	-	Click
		'the Total and grand Total revert to zeros		
		Total.Value = "0.00"		
n)		GrandTotal = "0.00"		
		'the Units labels revert to underscore		
		Label31.Caption = " "		
		Label32.Caption = " "		
		Label33.Caption = " "		
		Label33.Caption = "_"		
		End Sub		
		Private Sub CommandButton4_Click()		
		'this is the Exit button		
		Unload UserForm1		
		End Sub		
×				
		Private Sub CommandButton5_Click()		
<u> </u>				
1				
III				
III				
III				
——III				
		End Sub		
——III				
		Private Sub UserForm Activate()		
		'on activation of the form do the following		
		'populate the ComboBox 1 list		
		ComboBox1.AddItem "Concrete"		
iter		ComboBox1.AddItem "Asphalt"		
		ComboBox1.AddItem "Road Markings"		
		CONDODORT. AUGICEN ROAU MAIAINYS		
[]]		Inonulate CombePow? list		

Assign the *Item* values element by element.

Assign *Unit Prices*, *Quantities*, and *SubTotals* values to the matrix, element by element. The currency conversion function is used to force the text values to currency type before assigning to the matrix element. For convenience the *Quantities* are currency type but integer



type or double type may have been used. In that case the *Quantities* may have been declared in a separate matrix of the selected data type.

The SubTotals are the product of Unit Price and Quantity.

The *Total* is the sum of the *SubTotals*. The *Contingency* is 15% of the *Total* in dollars. The *Grand Total* is the sum of the *Total* and the *Contingency*.

```
End Sub
```

```
Private Sub CommandButton5 Click()
'assign Item elements to the vector
strArrItems(1) = ComboBox1.Value
strArrItems(2) = ComboBox2.Value
strArrItems(3) = ComboBox3.Value
'assign unit prices, quantities, and subttoals
'unit prices
curArrCalculationValues(1, 1) = CCur(UnitPrice1.Value)
curArrCalculationValues(2, 1) = CCur(UnitPrice2.Value)
curArrCalculationValues(3, 1) = CCur(UnitPrice3.Value)
'quantities
curArrCalculationValues(1, 2) = CCur(Quantity1.Value)
curArrCalculationValues(2, 2) = CCur(Quantity2.Value)
curArrCalculationValues(3, 2) = CCur(Quantity3.Value)
'sub totals
curArrCalculationValues(1, 3) = curArrCalculationValues(1, 1) *
                                        curArrCalculationValues(1, 2)
curArrCalculationValues(2, 3) = curArrCalculationValues(2, 1) *
                                        curArrCalculationValues(2, 2)
curArrCalculationValues(3, 3) = curArrCalculationValues(3, 1) *
                                        curArrCalculationValues(3, 2)
'total
curArrTotals(1) = curArrCalculationValues(1, 3) +
            curArrCalculationValues(2, 3) + curArrCalculationValues(3, 3)
'contingency
curArrTotals(2) = 0.15 * curArrTotals(1)
'grand total
curArrTotals(3) = curArrTotals(1) + curArrTotals(2)
```



Continue the *CommandButton5* procedure by adding code to display the results in the form controls.

```
CUTATTCALCULATIONVALUES(3, 3) = CUTATTCALCULATIONVALUES(3, 1) * CUTATTCALCULATIONVAL
'total
curArrTotals(1) = curArrCalculationValues(1, 3) + curArrCalculationValues(2, 3) + _
                   curArrCalculationValues(3, 3)
'contingency
curArrTotals(2) = 0.15 * curArrTotals(1)
'grand total
curArrTotals(3) = curArrTotals(1) + curArrTotals(2)
'send results to form
'sub totals
Subtotal1.Value = curArrCalculationValues(1, 3)
Subtotal2.Value = curArrCalculationValues(2, 3)
Subtotal3.Value = curArrCalculationValues(3, 3)
'totals
Total = curArrTotals(1)
Contingency = curArrTotals(2)
GrandTotal = curArrTotals(3)
End Sub
```

Save the workbook. Click on Debug. Select Compile VBAProject. Correct any errors. Resave. If there are no errors, test the application.



Open the application. Enter items, and quantities. Click on *Calculate*.

PROJECT	CALCULATOR					×
	Item	Unit Price (\$)	Unit	Quantity	Subtotal (\$)	
1.	Concrete 💌	85.35	сү	65	5547.75	
2.	Asphalt 💌	35.45	TON	85	3013.25	
3.	Road Markings 👻	6.95	LF	455	3162.25	
				Total (\$)	11723.25	
				Contingency (15%) Grand Total (\$)	1758.4875	
	CALCULATE	RESET		PRINT	EXIT	

The test is a success.

The original version of this application used twelve variables to enable the calculations. This current version, by taking advantage of arrays uses only three to achieve the same result. This version, therefore, is more efficient and has less impact on the computer's resources.



The next step is to add error handling code. Assume a user inadvertently entered a non-numeric value in for a quantity. The following will occur.

DROIECT	CALCULATOR					
PROJECT	CALCOLATON					
	Item		Unit Price (\$)	Unit	Quantity	Subtotal (\$)
1.	Concrete	•	85.35	CY	u5	5547.75
2.	Asphalt	•	35.45	TON	85	3013.25
				104		
3.	Road Marking	s 🔻	6.95	LF	455	3162.25
					Total (\$)	11723.25
						11725.25
					Contingency (15%)	1758.4875
					Grand Total (\$)	13481.7375
		CALCULATE	RESET		PRINT	EXIT
	_					

Microsoft Visual Basic
Run-time error '13':
Type mismatch
Continue End Debug Help



Add the following error handler into the command button procedure.

```
End Sub
Private Sub CommandButton5_Click()
On Error Resume Next
'assign Item elements to the vector
strArrItems(1) = ComboBox1.Value
strArrItems(2) = ComboBox2.Value
strArrItems(3) = ComboBox3.Value
'assign unit prices, quantities, and subttoals
'unit prices
curArrCalculationValues(1, 1) = CCur(UnitPrice1.Value)
curArrCalculationValues(2, 1) = CCur(UnitPrice2.Value)
curArrCalculationValues(3, 1) = CCur(UnitPrice3.Value)
'quantities
curArrCalculationValues(1, 2) = CCur(Quantity1.Value)
curArrCalculationValues(2, 2) = CCur(Quantity2.Value)
curArrCalculationValues(3, 2) = CCur(Quantity3.Value)
'sub totals
curArrCalculationValues(1, 3) = curArrCalculationValues(1, 1) *
                                        curArrCalculationValues(1, 2)
curArrCalculationValues(2, 3) = curArrCalculationValues(2, 1) *
                                        curArrCalculationValues(2, 2)
curArrCalculationValues(3, 3) = curArrCalculationValues(3, 1) *
                                        curArrCalculationValues(3, 2)
'total
curArrTotals(1) = curArrCalculationValues(1, 3) +
          curArrCalculationValues(2, 3) + curArrCalculationValues(3, 3)
'contingency
```

This means that whenever an error of any kind occurs, it will be ignored and the code will resume execution on the next line.



Test the application.

PROJECT	CALCULATOR						—
	Item		Unit Price (\$)	Unit	Quantity	Subtotal (\$)	
1.	Road Markings	•	6.95	LF	575	3996.25	
2.	Asphalt	•	35.45	TON	у5	0	-
3.	Concrete	•	85.35	CY	65	5547.75	
					Total (\$)	9544	
					Contingency (15%)	1431.6	
					Grand Total (\$)	10975.6	
	CAL	CULATE	RESET		PRINT	EXIT	

The error is handled and the procedure continues to completion yielding a valid result that omits the offending line from the calculations.



Alternately, the *On Error Goto <label>* can be used to alert the user and prematurely stop the procedure. In this case the *On Error Goto <label>* will be used in conjunction with a *Resume* statement. The Resume label will be placed such that the after the offending line of code sends the execution to the error handling block for fixing or bypassing the problem, the user is alerted and the program prematurely terminates execution.

At the top of the procedure enter the *On Error Goto* <*label>* call.

```
End Sub
Private Sub CommandButton5 Click()
'On Error Resume Next
On Error GoTo myHandlerLine: 🗲
'assign Item elements to the vector
strArrItems(1) = ComboBox1.Value
strArrItems(2) = ComboBox2.Value
strArrItems(3) = ComboBox3.Value
'assign unit prices, quantities, and subttoals
'unit prices
curArrCalculationValues(1, 1) = CCur(UnitPrice1.Value)
curArrCalculationValues(2, 1) = CCur(UnitPrice2.Value)
curArrCalculationValues(3, 1) = CCur(UnitPrice3.Value)
'quantities
curArrCalculationValues(1, 2) = CCur(Quantity1.Value)
curArrCalculationValues(2, 2) = CCur(Quantity2.Value)
curtrrCalculationValues(3, 2) = CCur(Ouantitu3 Value)
```

The error handling block associated with the line label *myHandlerLine* shall be inserted at an appropriate location such that the program will "jump" over any block of code the programmer thinks may cause an error. In this case the intent is to display a message box and prematurely stop execution after the bad data is detected. Therefore the error handling block will be placed towards the bottom of the procedure.



Add the error handling block. Note the "Exit Sub" line. Without this, the error handler will fire regardless of whether an error is picked up or not. It is analogous to a wall blocking off the error handling block, and can only be "jumped" over when an error is picked up and execution is transferred to after the label line.

```
'sub totals
Subtotal1.Value = curArrCalculationValues(1, 3)
Subtotal2.Value = curArrCalculationValues(2, 3)
Subtotal3.Value = curArrCalculationValues(3, 3)
'totals
Total = curArrTotals(1)
Contingency = curArrTotals(2)
GrandTotal = curArrTotals(3)
Exit Sub
myHandlerLine: MsgBox "You have inadmissible data! Check your data " &
               "inputs", vbOKOnly + vbCritical, "Premature Termination"
               Resume myResumeLine:
End Sub
```

Drivata Sub CommandButtons Click()



The *Resume* label is placed such that the totals will be calculated for the "good data" up until an error is picked up and handled.

```
CutAtionicutationvalues(2, 1) Cout(Oniotitice.value)
curArrCalculationValues(3, 1) = CCur(UnitPrice3.Value)
'quantities
curArrCalculationValues(1, 2) = CCur(Quantity1.Value)
curArrCalculationValues(2, 2) = CCur(Quantity2.Value)
curArrCalculationValues(3, 2) = CCur(Quantity3.Value)
myResumeLine:
'sub totals
curArrCalculationValues(1, 3) = curArrCalculationValues(1, 1) *
                                      curArrCalculationValues(1, 2)
curArrCalculationValues(2, 3) = curArrCalculationValues(2, 1) *
                                       curArrCalculationValues(2, 2)
curArrCalculationValues(3, 3) = curArrCalculationValues(3, 1) *
                                        curArrCalculationValues(3, 2)
'total
curArrTotals(1) = curArrCalculationValues(1, 3) +
           curArrCalculationValues(2, 3) + curArrCalculationValues(3, 3)
'contingency
curArrTotals(2) = 0.15 * curArrTotals(1)
'grand total
curArrTotals(3) = curArrTotals(1) + curArrTotals(2)
```



Test the application.

PROJECT	CALCULATOR						×
	Item		Unit Price (\$)	Unit	Quantity	Subtotal (\$)	
1.	Road Marking	s 🔻	6.95	LF	685	0.00	
2.	Concrete	•	85.35	CY	95	0.00	
3.	Asphalt	•	35.45	TON	9t	0.00	
				/	Total (\$)	0.00	
						0.00	
					Contingency (15%)	0.00	
					Grand Total (\$)	0.00	
	_	CALCULATE	RESET		PRINT	EXIT	



The error handler activates and displays the message box.

PROJECT C	CALCULATOR						Σ
	_						
	Item		Unit Price (\$)	Unit	Quantity	Subtotal (\$)	
1.	Road Markin	igs 🔻	6.95	LF	685	0.00	
2.	Concrete	•	85.35	CY	95	0.00	
3.	Asphalt	Premature Term	ination			0.00	
		🔀 You	have inadmissible o	data! Chec	k your data inputs		
		<u> </u>				0.00	
					ОК	0.00	
	l			G	irand Total (\$)	0.00	
			_				
		CALCULATE	RESET		PRINT	EXIT	



Dismiss the message box. The calculations for the "good data" are executed and displayed, whereas that of the bad data is skipped. Upon picking up an error, the message box is displayed after which the program prematurely terminates, as designed.

PROJECT	CALCULATOR						X
	Item		Unit Price (\$)	Unit	Quantity	Subtotal (\$)	
1.	Road Markings	•	6.95	LF	685	4760.75	
2.	Concrete	•	85.35	CY	95	8108.25	
3.	Asphalt	•	35.45	TON	9t	0	
					Total (\$)	12869	
					Contingency (15%)	1930.35	
					Grand Total (\$)	14799.35	
	CA	LCULATE	RESET		PRINT	EXIT	

The test is a success.



Double click on the *Print* button.

PROJ	ECT CALCO	JLATOR							×
3 #									
		Item	•••••••••	Unit Duice (¢)		Oursetite	• • • • • • •	Cubtotal (¢)	
		Item		Unit Price (\$)		Quantity		Subtotal (\$)	
							· · ·		
. 1 •∶			-	0.00	:: <u> </u>	: 0	::	0.00	
2.				0.00		0	: : [0.00	
	::: !				:: 	.:			
							· · r		
.			<u>▼</u> ::	0.00	:: <u> </u>	: 0	::	0.00	
						Total	· · · · · · ·		
							::	0.00	
						Contingency		15	%
ਗ਼ 📲 💠 🗄									70
						Grand Total (\$)		
							'	0.00	
11 🖉 🗄 🖓									
			ALCULATE	RESET		PRINT		EXIT	[
			ALCOLATE	RESET		- FRUIT		LAIT	



On clicking this button, the data will be saved to a spreadsheet. Since the global variables are still "alive", they can be called to display their values on the spreadsheet as an alternative to calling the form controls' values. In terms of the amount of code needed, this is a more efficient method.

```
-----
End Sub
Private Sub CommandButton6 Click()
'declare variables to designate the element address
Dim i, j As Integer
'use a For loop and nested For loop to run through the arrays and copy
'the elements to the spreadsheet
'the items
For i = 1 To 3
   Sheets("Sheet1").Cells(i, 1).Value = strArrItems(i)
Next
'unit price, quantity, subtotall
For i = 1 To 3
   For j = 1 To 3
       Sheets("Sheet1").Cells(i, j).Value = curArrCalculationValues(i, j)
   Next
Next
End Sub
Private Sub UserForm Activate()
```



Save the workbook. Click on Debug. Select Compile VBAProject. Correct any errors. Resave. If there are no errors, test the application.

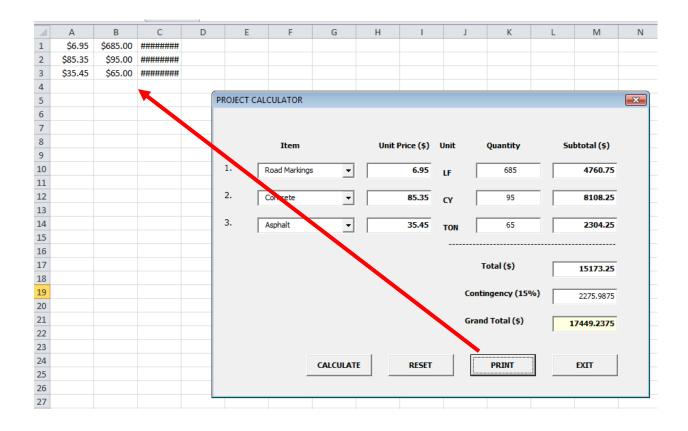
Open the application. Enter items, and quantities. Click on *Calculate*.

PROJECT	CALCULATOR						-X
	Item		Unit Price (\$)	Unit	Quantity	Subtotal (\$)	
1.	Road Markings	•	6.95	LF	685	4760.75	
2.	Concrete	•	85.35	сү	95	8108.25	
3.	Asphalt	•	35.45	TON	65	2304.25	
					Total (\$)	15173.25	
					Contingency (15%)	2275.9875	
					Grand Total (\$)	17449.2375	
	C/	ALCULATE	RESET		PRINT	EXIT	



Press Print

Confirm the values are correct and saved to the cells specified in the code.



Press *Exit* to close out of the application. The test is a success.



4. GRAPHS

4.1 Chart Objects

Graphs can be created by using the VBA Chart object. Examples of charts and graphs that can be created in Excel-VBA include scatterplot, pie chart, bar chart, area graph, and their 3-dimensional counterparts among others. A comprehensive list of chart types supported by Excel-VBA can be found on the Microsoft website. (Microsoft, 2014b).

4.2 Creating Graphs

A general procedure for creating graphs involves the following

Step 1: Create the chart object. The syntax is

Charts.Add

Step 2: Specify where the chart shall be displayed. To display on a spreadsheet, for example, the syntax is of the form

ActiveChart.Location Where : = xlLocationAsObject, Name : = "nameofspreadsheet"

Step 3: By default Excel will plot any pre-existing series of data on the spreadsheet on the chart. Delete any such series plotted on the chart. A loop can be utilized to remove the data series by series. The syntax will be of the form

Do Until ActiveChart.SeriesCollection.Count = 0

ActiveChart.SeriesCollection(1).Delete

Loop



Step 4: Add new series to the chart and assign data to it. Obtaining the data from a spreadsheet, for example, will be of the form

With ActiveChart.SeriesCollection.NewSeries

```
'give the series a name
.Name = ActiveSheet.Range (rangeofcells)
'assign "y" values
.Values = ActiveSheet.Range (rangeofcells)
'assign "x' values
.XValues = ActiveSheets.Range (rangeofcells)
'assign other properties e.g. title, gridlines etc
:
:
'if multiple series of data are being plotted on the graph, add
'name, y-values, x-values for each series, as needed.
:
:
:
```

End With



Example:

Consider the following data. For *Y2* through *Y6*, plot each *Y* versus *X* to create one graph with multiple series of data plotted on it.

A	В	C	D	E	. F.	G
Х	Y1	Y2	Y3	Y4	Y5	Y6
1	10	40	70	100	130	160
2	20	48	76	104	132	160
3	30	56	82	108	134	160
4	40	64	88	112	136	160
5	50	72	94	116	138	160
6	60	80	100	120	140	160
7	70	88	106	124	142	160
8	80	96	112	128	144	160
9	90	104	118	132	146	160
10	100	112	124	136	148	160
11	110	120	130	140	150	160
	X 1 2 3 4 5 6 7 8 9 10	X Y1 1 10 2 20 3 30 4 40 5 50 6 60 7 70 8 80 9 90 10 100	X Y1 Y2 1 10 40 2 20 48 3 30 56 4 40 64 5 50 72 6 60 80 7 70 88 8 80 96 9 90 104 10 100 112	X Y1 Y2 Y3 1 10 40 70 2 20 48 76 3 30 56 82 4 40 64 88 5 50 72 94 6 60 80 100 7 70 88 106 8 80 96 112 9 90 104 118 10 100 112 124	X Y1 Y2 Y3 Y4 1 10 40 70 100 2 20 48 76 104 3 30 56 82 108 4 40 64 88 112 5 50 72 94 116 6 60 80 100 120 7 70 88 106 124 8 80 96 112 128 9 90 104 118 132 10 100 112 124 136	X Y1 Y2 Y3 Y4 Y5 1 10 40 70 100 130 2 20 48 76 104 132 3 30 56 82 108 134 4 40 64 88 112 136 5 50 72 94 116 138 6 60 80 100 120 140 7 70 88 106 124 142 8 80 96 112 128 144 9 90 104 118 132 146 10 100 112 124 136 148

Add the following code under a command button click procedure

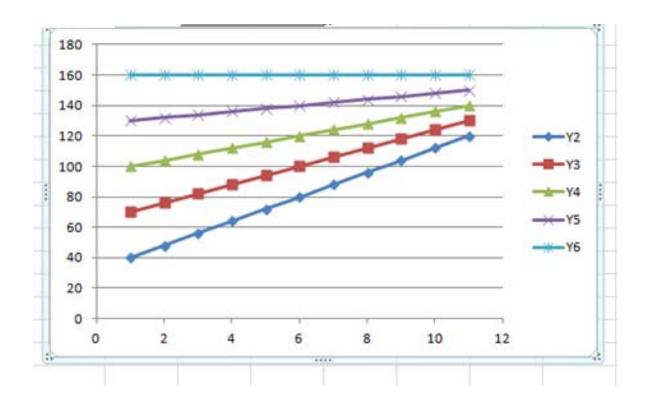


```
CommandButton3
  Private Sub CommandButton3 Click()
  'create the chart object
  Charts.Add
   'next define the type of chart you want
  ActiveChart.ChartType = xlXYScatterLines
  'then specify where you want the chart display
  ActiveChart.Location Where:=xlLocationAsObject, Name:="Sheet13"
  'now as we did not specify the data, Excel will attempt to
  'plot any data it see of the spreadsheet.
   'so let us delete all this so we can add our own that we want
  Do Until ActiveChart.SeriesCollection.Count = 0
              ActiveChart.SeriesCollection(1).Delete
  Loop
  'now lets add out X-Y data
  'lest do Y2 vs X
  With ActiveChart.SeriesCollection.NewSeries
      .Name = ActiveSheet.Range("C1")
       .Values = ActiveSheet.Range("C2:C12")
       .XValues = ActiveSheet.Range("A2:A12")
  End With
   'lets add Y3 vs X
  With ActiveChart.SeriesCollection.NewSeries
       .Name = ActiveSheet.Range("d1")
       .Values = ActiveSheet.Range("d2:d12")
       .XValues = ActiveSheet.Range("A2:A12")
  End With
  'lets add Y4 vs X
  With ActiveChart.SeriesCollection.NewSeries
      .Name = ActiveSheet.Range("e1")
      .Values = ActiveSheet.Range("e2:e12")
      .XValues = ActiveSheet.Range("A2:A12")
  End With
   'lets add Y5 vs X
  With ActiveChart.SeriesCollection.NewSeries
       .Name = ActiveSheet.Range("f1")
      .Values = ActiveSheet.Range("f2:f12")
      .XValues = ActiveSheet.Range("A2:A12")
  End With
  'lets add Y6 vs X
  With ActiveChart.SeriesCollection.NewSeries
       .Name = ActiveSheet.Range("g1")
       .Values = ActiveSheet.Range("g2:g12")
       .XValues = ActiveSheet.Range("A2:A12")
  End With
   'STUDENTS! see how can add the series using a Loop instead
   'manually cutting and pasting the code. You will have to
  'use Cells(i,j) to assign the ranges.
  End Sub
```



The code results in the following.

	А	В	С	D	E	F	G
1	Х	Y1	Y2	Y3	Y4	Y5	Y6
2	1	10	40	70	100	130	160
3	2	20	48	76	104	132	160
4	3	30	56	82	108	134	160
5	4	40	64	88	112	136	160
6	5	50	72	94	116	138	160
7	6	60	80	100	120	140	160
8	7	70	88	106	124	142	160
9	8	80	96	112	128	144	160
10	9	90	104	118	132	146	160
11	10	100	112	124	136	148	160
12	11	110	120	130	140	150	160





5. CONCLUSION

This course has presented a broad overview of fundamental concepts and principles of computer programming, and presented them in situations encountered by practicing engineers and scientists. All codes were developed using the *Visual Basic for Applications* (VBA) programming language.

In this course the topics, error handling, arrays, dialog boxes, and graphs were presented. Several examples relevant to engineering were used to illustrate and demonstrate the concepts and methods learned in this class. A mini-project was used to demonstrate the programming concepts and methods in a situation encountered by practicing engineers.

This course has enabled participants to identify situations where programming is relevant and will be of advantage to the professional. Practitioners are strongly encouraged to look out for situations in their domains of expertise where programming solutions are applicable and will be of benefit to their work and their organization.

Computer programming requires a careful and meticulous approach, and can only be mastered and retained by practice and repetition.

Good Luck and Happy Programming.



REFERENCES

- Bradley, J. C., & Millspaugh, A. C. (1999). *Programming in Visual Basic 6.0.* Irwin McGraw-Hill.
- FunctionX Inc. (2013). VBA for Microsoft Office Excel 2007. Retrieved December 21, 2013, from FunctionX Tutorials: www.functionx.com/
- Microsoft. (2013). *Excel 2013 developer reference*. Retrieved October 15, 2013, from Office Dev Center: http://msdn.microsoft.com/en-us/library/office/ee861528.aspx
- Microsoft. (2014a). *Error Trapping with Visual Basic for Applications*. Retrieved January 11, 2014, from http://support.microsoft.com/kb/146864
- Microsoft. (2014b). *XlChartType Enumeration (Excel)*. Retrieved January 15, 2014, from Office Dev Center: http://msdn.microsoft.com/en-us/library/office/ff838409.aspx
- Wise Owl Business Solutions Ltd. (2013). *Using FileDialogs in VBA*. Retrieved January 3, 2014, from Wise Owl: http://www.wiseowl.co.uk/blog/s209/type-filedialog.htm

Images were all drawn/ prepared by K. Ofosu