Programming with the Peltier Tech Utility

The Peltier Tech Utility was designed so that much of its functionality is available via VBA as well as through the Excel user interface. This document explains how to access this functionality.

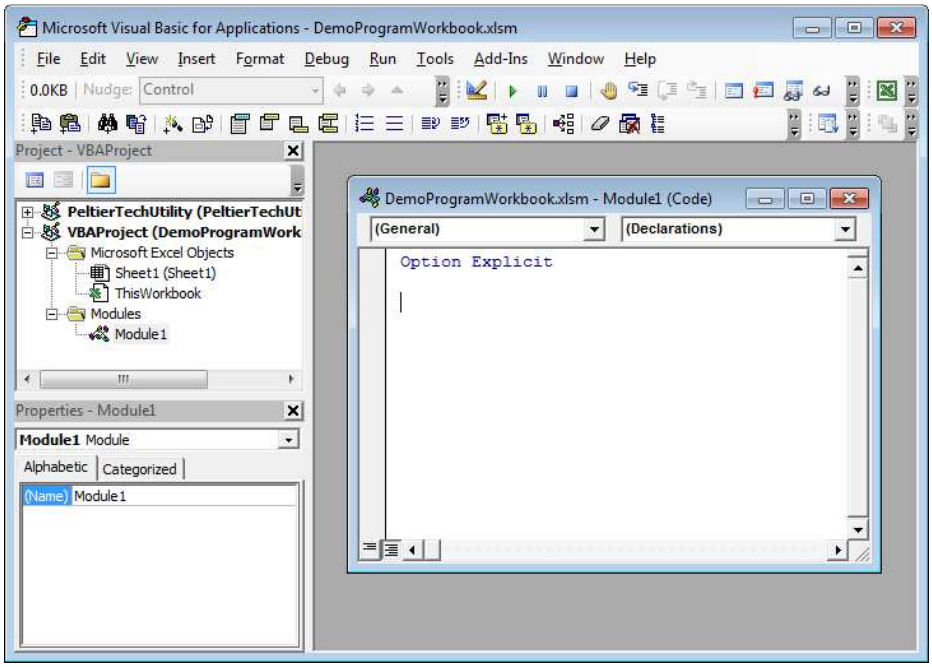
Contents

Getting Started

This document is not meant to teach you all about VBA programming, but it will cover some important topics beyond the use of the Peltier Tech Utility.

The Visual Basic Editor

In Excel, press Alt+F11 to open the VBIDE (Visual Basic Integrated Design Environment), or simply the VB Editor, shown below.



Every Excel workbook and add-in has a VB Project, shown in the Project Explorer pane. They are listed by project name, followed by workbook name in parentheses. Here we see PeltierTechUtility (PeltierTechUtility.xlam) and VBAProject (DemoProgramWork.xlam). In the examples here, we will call routines in the Peltier Tech Utility from DemoProgramWork.xlam.

Early Binding vs. Late Binding

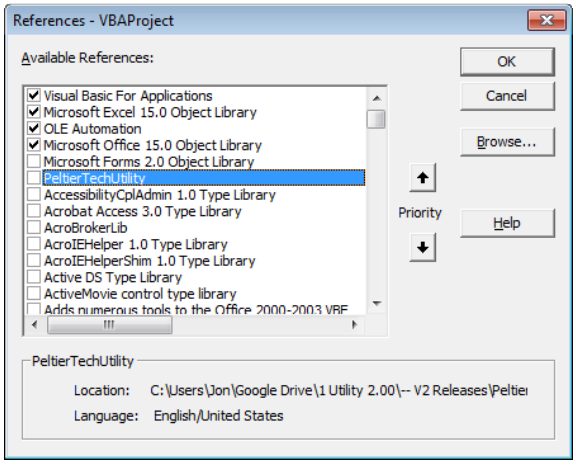
There are two ways to connect your project to a project that has code you want to run. Early Binding means you connect at design time, that is, while writing the code that calls the other project's code. Early binding has several advantages. First, when using early binding, you get all of the benefits of using an integrated design environment, such as IntelliSense and the Object Browser. Second, the code will usually run slightly faster under early binding, though usually the difference is not perceptible to your users. The main drawback to using early binding is that, if the linked project isn't present, your code will experience a compile error, and will not run at all.

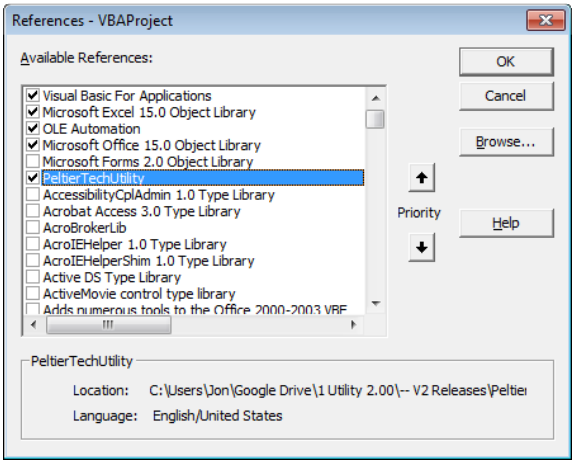
Late Binding means you connect to the other project at run time, that is, not until your code is actually running and needs the code in the other project. Late binding has none of the design advantages, nor the supposed speed advantage, of early binding. However, if the other project is not available, your code will run until it hits the call to the other project, then raise a run-time error. You can write defensive code that knows how to deal with run-time errors.

Many developers will write their code using early binding, to take advantage of its benefits, then convert it before deployment to late binding, to avoid paralyzing compile errors.

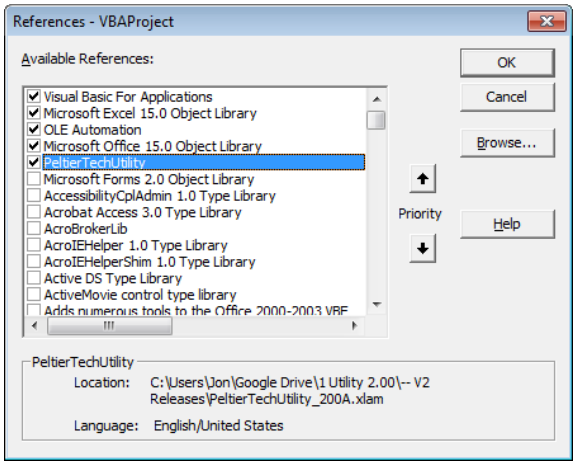
References

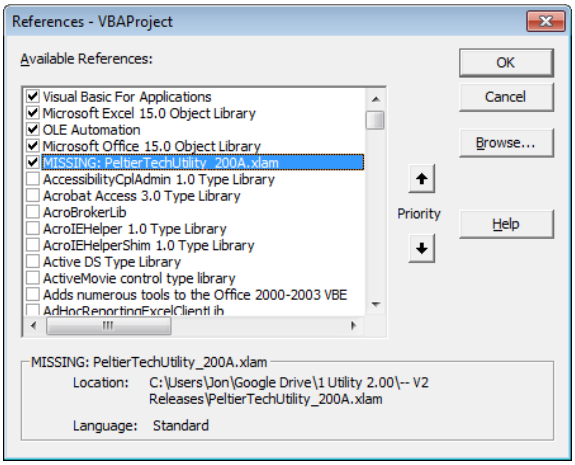
To use early binding, you need to set a reference to the VB project containing the code you want to use. In the VB Editor, go to Tools menu > References, and find the name of the project you want to use. Check the box in front of the project name, and click OK.





If you open the References dialog and a reference to the project is already set, its check box is checked and it is listed with all of the other references. If a reference has been set but the referenced project is no longer available, its check box will be checked, but the project name will be prefixed with "MISSING:", and you will get compile errors when trying to run the code.

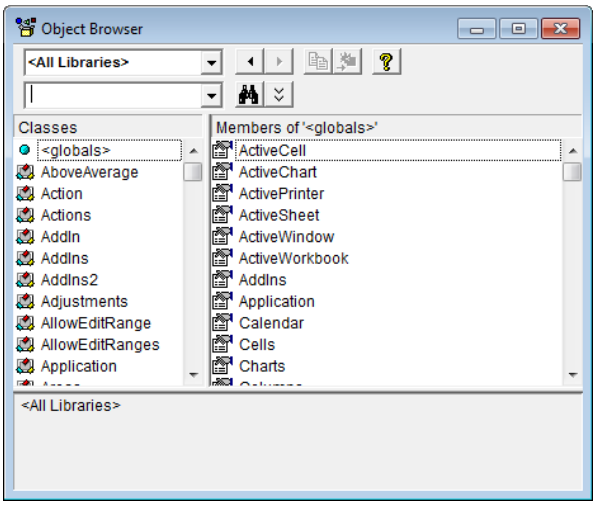


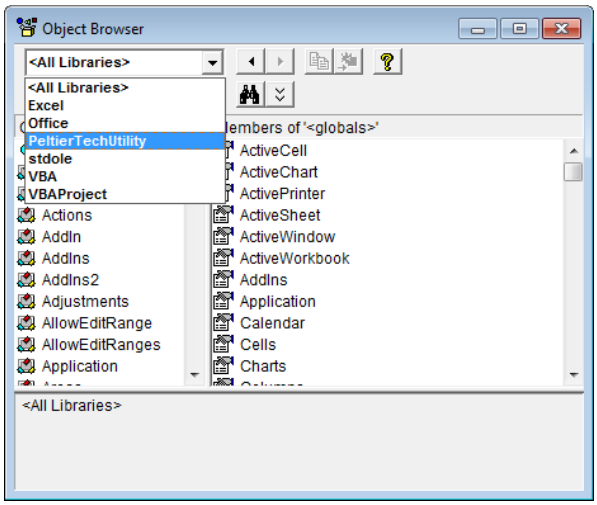


If you open a project that references another project, if the referenced project is closed, Excel looks for it, and if Excel finds it, Excel opens it. If your project references another project and both are open, Excel will not let you close the referenced project because of the reference.

The Object Browser

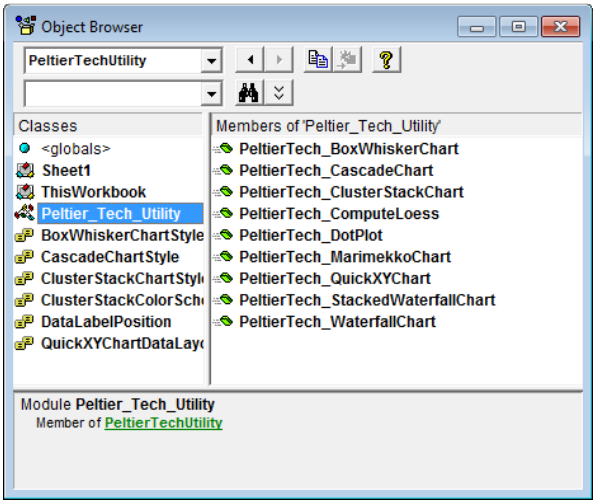
To open the Object Browser, in the VB Editor, go to View menu > Object Browser, or click F2. The Object Browser (below left) is a window in the VB Editor that shows you all of the objects in Excel's libraries, and in libraries of referenced projects. You can select a library from the top left drop down.

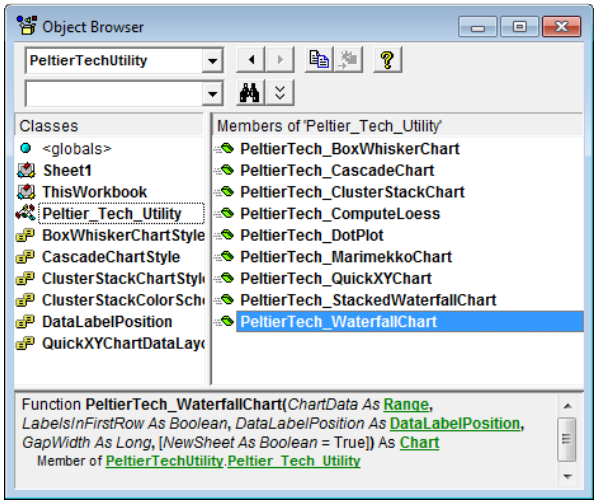




With the PeltierTechUtility library selected in the dropdown (below left), the Classes pane shows all of the classes in the library, including the uninteresting Sheet1 and ThisWorkbook, but also Peltier_Tech_Utility, which is a module containing accessible code routines, and a variety of custom named constants. Peltier_Tech_Utility is selected in the Classes pane, so the routines you can access are shown in the Members pane.

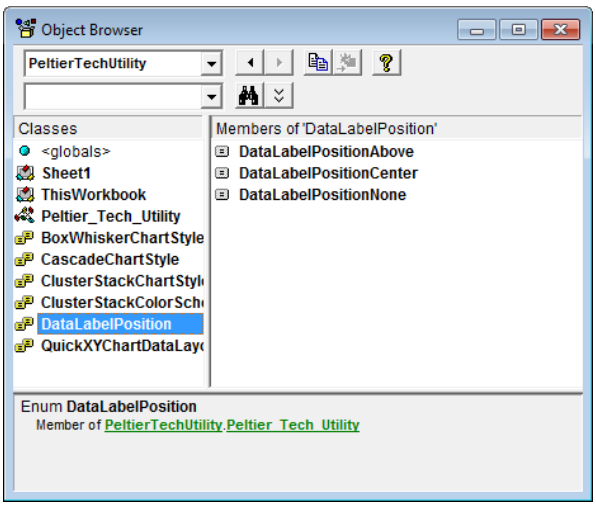
When a procedure is selected in the Members pane, the syntax of that procedure appears at the bottom of the Object Browser. Any of the green underlined objects in the syntax can be clicked to see its definition in the Classes pane.

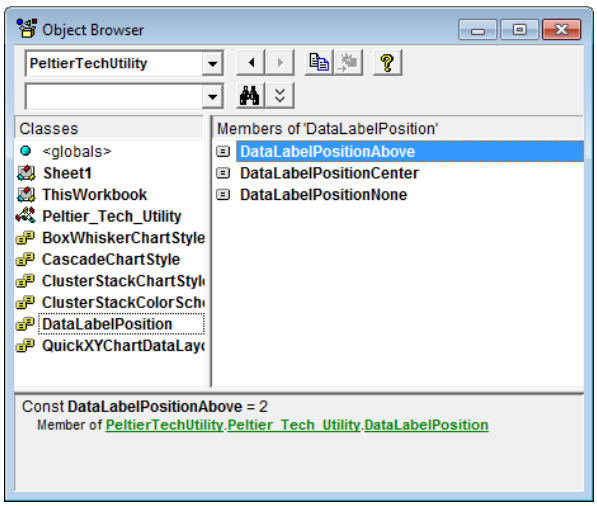




For example, one of the arguments in the PeltierTech_WaterfallChart function is the named constant type DataLabelPosition. Named constants are handy because they give mnemonic names to a simple list of values. The named constant type DataLabelPosition has values of DataLabelPositionNone (zero), DataLabelPositionCenter (1), and DataLabelPositionAbove (2). Click

on the constant in the Members pane, and the value is displayed at the bottom of the Object Browser.

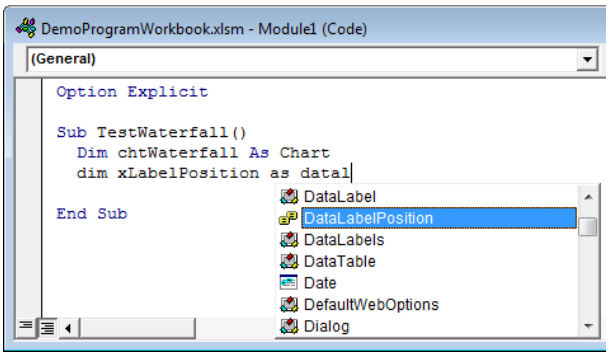


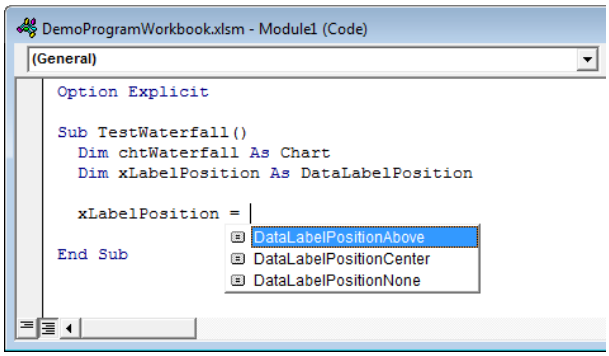


IntelliSense

IntelliSense is the feature of the VB Editor that pops up syntax hints when you are entering a function or variable type, and autocompletes keywords as you type. For example, when the name of the PeltierTech_WaterfallChart procedure is typed, the syntax pops up in a box below the procedure's name.

When a variable type is partially entered (below left, a list of possible finished names is provided (below left). When a variable is being assigned a value, a popup of the possible named values appears (below right).





Early Bound Test Procedures

The following procedures have been entered into Module1 of our workbook's VB project. The syntax shows the PeltierTech_WaterfallChart procedure as a function, but you can use it as a sub, by not assigning its value to a variable. The simplest sub syntax is shown below left, with the procedure name followed by the list of arguments. More verbose is the use of the keyword Call, followed by the procedure name, then the list of arguments in parentheses (below right).

```
Sub TestWaterfallSub()
  Dim chtWaterfall As Chart
  Dim rngData As Range
  Dim bLabels As Boolean
  Dim xLabelPosition As DataLabelPosition
  Dim lGapWidth As Long
  Dim bNewSheet As Boolean

  Set rngData = ActiveSheet.Range("B3:C9")
  bLabels = True
  xLabelPosition = DataLabelPositionAbove
  lGapWidth = 50
  bNewSheet = True

  PeltierTech_WaterfallChart rngData, _
      bLabels, xLabelPosition, lGapWidth, _
      bNewSheet

End Sub
```

```
Sub TestWaterfallSubCall()
  Dim chtWaterfall As Chart
  Dim rngData As Range
  Dim bLabels As Boolean
  Dim xLabelPosition As DataLabelPosition
  Dim lGapWidth As Long
  Dim bNewSheet As Boolean

  Set rngData = ActiveSheet.Range("B3:C9")
  bLabels = True
  xLabelPosition = DataLabelPositionAbove
  lGapWidth = 50
  bNewSheet = True

  Call PeltierTech_WaterfallChart(rngData, _
      bLabels, xLabelPosition, lGapWidth, _
      bNewSheet)

End Sub
```

The use of the procedure as a function is shown below left, where a variable of the appropriate type is set equal to the result of the function. The code below right illustrates why you would want to use the procedure as a function. After the variable chtWaterfall is assigned to the chart that results from the PeltierTech_WaterfallChart function, the calling procedure goes on to modify properties of the variable. In this case, the colors of some of the chart elements are changed.

```
Sub TestWaterfallFunction()
  Dim chtWaterfall As Chart
  Dim rngData As Range
  Dim bLabels As Boolean
  Dim xLabelPosition As DataLabelPosition
  Dim lGapWidth As Long
  Dim bNewSheet As Boolean

  Set rngData = ActiveSheet.Range("B3:C9")
  bLabels = True
  xLabelPosition = DataLabelPositionAbove
  lGapWidth = 50
  bNewSheet = True

  Set chtWaterfall = _
      PeltierTech_WaterfallChart(rngData, _
      bLabels, xLabelPosition, lGapWidth, _
      bNewSheet)

End Sub
```

```
Sub TestWaterfallFunctionFormat()
  Dim chtWaterfall As Chart
  Dim rngData As Range
  Dim bLabels As Boolean
  Dim xLabelPosition As DataLabelPosition
  Dim lGapWidth As Long
  Dim bNewSheet As Boolean

  Set rngData = ActiveSheet.Range("B3:C9")
  bLabels = True
  xLabelPosition = DataLabelPositionAbove
  lGapWidth = 50
  bNewSheet = True

  Set chtWaterfall = _
      PeltierTech_WaterfallChart(rngData, _
      bLabels, xLabelPosition, lGapWidth, _
      bNewSheet)

  With chtWaterfall
    .SeriesCollection("Ends").Format.Fill _
        .ForeColor.RGB = RGB(178, 178, 178)
        ' medium gray
    With .ChartGroups(2)
      .UpBars.Format.Fill _
          .ForeColor.RGB = RGB(51, 204, 255)
          ' teal blue
      .DownBars.Format.Fill _
          .ForeColor.RGB = RGB(255, 204, 0)
          ' orange
    End With
  End With

End Sub
```

The above procedures take advantage of early binding through the use of the DataLabelPosition variable type, its assignment to the named value DataLabelPositionAbove, and the way the procedure PeltierTech_WaterfallChart is simply called by name.

Late Bound Test Procedures

The following routines show the differences in code that is run using late binding. The variable xLabelPosition is declared as Long, not DataLabelPosition, and it is given a value of 2, not DataLabelPositionAbove. The procedure PeltierTech_WaterfallChart cannot be called by name, but has to be inserted as the first argument of Application.Run, as text, followed by the rest of the arguments.

The first three examples show the procedure called as a sib, with no variable assigned to its result.

Below left, just the procedure name is used as the first argument of Application.Run. Excel hunts through the open procedures, runs the first one it finds with that name. This might result in the wrong version of the procedure being called, if Excel finds the procedure name in a different project.

Below right, Application.Run is passed the project's workbook name plus the procedure name, separated by an exclamation point. This ensures that the procedure is run from the intended project.

```
Sub TestWaterfallSub()
  Dim chtWaterfall As Chart
  Dim rngData As Range
  Dim bLabels As Boolean
  Dim xLabelPosition As Long
  Dim lGapWidth As Long
  Dim bNewSheet As Boolean

  Set rngData = ActiveSheet.Range("B3:C9")
  bLabels = True
  xLabelPosition = 2
  lGapWidth = 50
  bNewSheet = True

  Application.Run "PeltierTech_WaterfallChart", _
      rngData, bLabels, xLabelPosition, _
      lGapWidth, bNewSheet

End Sub
```

```
Sub TestWaterfallSub2()
  Dim chtWaterfall As Chart
  Dim rngData As Range
  Dim bLabels As Boolean
  Dim xLabelPosition As Long
  Dim lGapWidth As Long
  Dim bNewSheet As Boolean

  Set rngData = ActiveSheet.Range("B3:C9")
  bLabels = True
  xLabelPosition = 2
  lGapWidth = 50
  bNewSheet = True

  Application.Run "PeltierTechUtility.xlam!" _
      & "PeltierTech_WaterfallChart", _
      rngData, bLabels, xLabelPosition, _
      lGapWidth, bNewSheet

End Sub
```

In late binding, if the procedure's workbook is not open, Excel tries to open it and run the procedure. You can qualify the workbook name with its path (below), to make sure Excel opens the correct workbook.

```
Sub TestWaterfallSub3()
  Dim chtWaterfall As Chart
  Dim rngData As Range
  Dim bLabels As Boolean
  Dim xLabelPosition As Long
  Dim lGapWidth As Long
  Dim bNewSheet As Boolean

  Set rngData = ActiveSheet.Range("B3:C9")
  bLabels = True
  xLabelPosition = 2
  lGapWidth = 50
  bNewSheet = True

  Application.Run "C:\Users\Jon\Documents\PeltierTechUtility.xlam!PeltierTech_WaterfallChart", _
      rngData, bLabels, xLabelPosition, lGapWidth, bNewSheet

End Sub
```

The following examples show how to use Application.Run as a function, not a sub, using the simplest use of the procedure name (without workbook name or path). Below left is the simple function; below right shows the variable returned by function being further modified by the calling procedure.

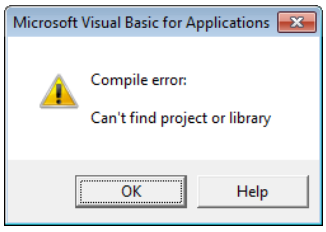
```
Sub TestWaterfallFunction()
  Dim chtWaterfall As Chart
  Dim rngData As Range
  Dim bLabels As Boolean
  Dim xLabelPosition As Long
  Dim lGapWidth As Long
  Dim bNewSheet As Boolean

  Set rngData = ActiveSheet.Range("B3:C9")
  bLabels = True
  xLabelPosition = 2
  lGapWidth = 50
  bNewSheet = True

  Set chtWaterfall = _
    Application.Run("PeltierTech_WaterfallChart", _
    rngData, bLabels, xLabelPosition, lGapWidth, _
    bNewSheet)

End Sub
```

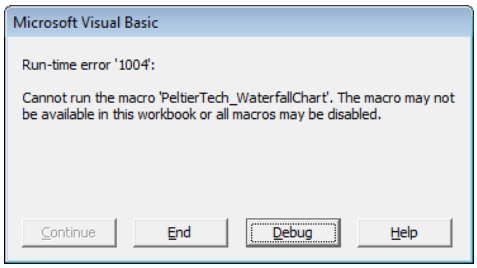
```
Sub TestWaterfallFunctionFormat()
  Dim chtWaterfall As Chart
  Dim rngData As Range
  Dim bLabels As Boolean
  Dim xLabelPosition As Long
  Dim lGapWidth As Long
  Dim bNewSheet As Boolean

  Set rngData = ActiveSheet.Range("B3:C9")
  bLabels = True
  xLabelPosition = 2
  lGapWidth = 50
  bNewSheet = True

  Set chtWaterfall = _
    Application.Run("PeltierTech_WaterfallChart", _
    rngData, bLabels, xLabelPosition, lGapWidth, _
    bNewSheet)

  With chtWaterfall
    .SeriesCollection("Ends").Format.Fill _
        .ForeColor.RGB = RGB(178, 178, 178)
        ' medium gray
    With .ChartGroups(2)
      .UpBars.Format.Fill _
          .ForeColor.RGB = RGB(51, 204, 255)
          ' teal blue
      .DownBars.Format.Fill _
          .ForeColor.RGB = RGB(255, 204, 0)
          ' orange
    End With
  End With

End Sub
```

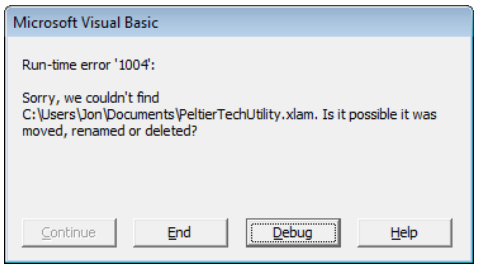
Errors

In early binding, if the workbook with your code is opened, and the referenced project is not open and Excel can't find it to open it, Excel won't be able to find any of the project's classes or methods in the project's library. When your procedure is called, Excel throws a compile error (below left).

In late binding, if the called procedure is not qualified by workbook name and cannot be found in any open projects, Excel throws a run-time error to tell you all about it (below center). If the workbook name is attached to the project name, and the workbook cannot be found, you will see a corresponding run-time error (below right).







Compile errors stops code execution in its tracks. Run-time errors, on the other hand, allow a clever programmer to anticipate and code against the possibilities that the procedure could not be run.

A simple defense against this kind of error is to simply use On Error Resume Next to bypass an error, then if desired, show the user a message if there is an error.

```
Sub TestWaterfallSubSimpleDefense()
  Dim chtWaterfall As Chart
  Dim rngData As Range
  Dim bLabels As Boolean
  Dim xLabelPosition As Long
  Dim lGapWidth As Long
  Dim bNewSheet As Boolean

  Set rngData = ActiveSheet.Range("B3:C9")
  bLabels = True
  xLabelPosition = 2
  lGapWidth = 50
  bNewSheet = True

  On Error Resume Next
  Application.Run "PeltierTech_WaterfallChart", _
      rngData, bLabels, xLabelPosition, lGapWidth, _
      bNewSheet
  If Err.Number <> 0 Then
    MsgBox "The chart could not be created."
  End If

End Sub
```

A more sophisticated approach checks to see whether the called procedure's project is open, and if not, skips the procedure altogether and pops up a message.

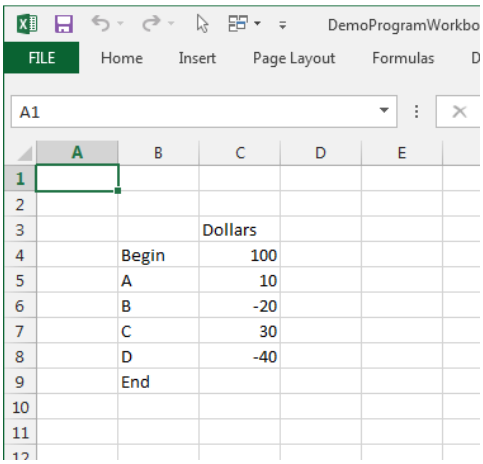
```
Sub TestWaterfallSubWkbkDefense()
  Dim chtWaterfall As Chart
  Dim rngData As Range
  Dim bLabels As Boolean
  Dim xLabelPosition As Long
  Dim lGapWidth As Long
  Dim bNewSheet As Boolean
  Dim wkbk As Workbook
  Dim sWkbkName As String
  Dim bWkbkFound As Boolean

  Set rngData = ActiveSheet.Range("B3:C9")
  bLabels = True
  xLabelPosition = 2
  lGapWidth = 50
  bNewSheet = True
  sWkbkName = "PeltierTechUtility.xlam"

  For Each wkbk In Workbooks
    If wkbk.Name = sWkbkName Then
      bWkbkFound = True
      Exit For
    End If
  Next

  If bWkbkFound Then
    Application.Run sWkbkName & "!PeltierTech_WaterfallChart", _
        rngData, bLabels, xLabelPosition, lGapWidth, _
        bNewSheet
  Else
    MsgBox "The workbook " & sWkbkName & " is not open."
  End If
End Sub
```

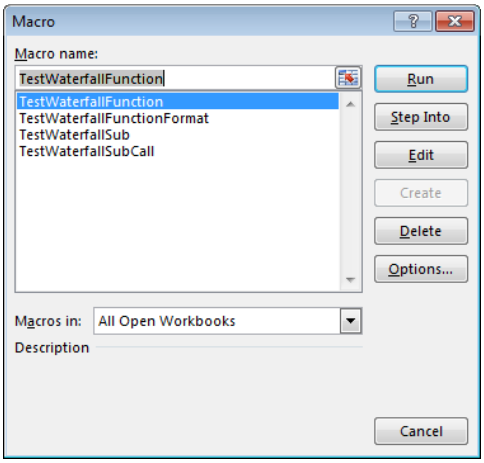
A third approach checks that the procedure's parent workbook even exists on the computer.

```
Sub TestWaterfallSubWkbkPathDefense()
  Dim chtWaterfall As Chart
  Dim rngData As Range
  Dim bLabels As Boolean
  Dim xLabelPosition As Long
  Dim lGapWidth As Long
  Dim bNewSheet As Boolean
  Dim sWkbkFullName As String

  Set rngData = ActiveSheet.Range("B3:C9")
  bLabels = True
  xLabelPosition = 2
  lGapWidth = 50
  bNewSheet = True
  sWkbkFullName = "C:\Users\Jon\Documents\PeltierTechUtility.xlam"

  If Len(Dir(sWkbkFullName)) = 0 Then
    MsgBox "The workbook " & sWkbkFullName & " is not present on this computer."
    Exit Sub
  End If

  Application.Run sWkbkFullName & "!PeltierTech_WaterfallChart", _
      rngData, bLabels, xLabelPosition, lGapWidth, _
      bNewSheet

End Sub
```

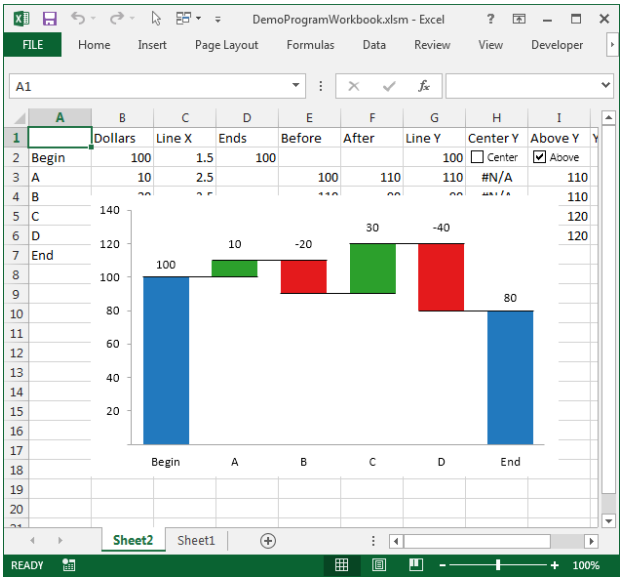
Program Output

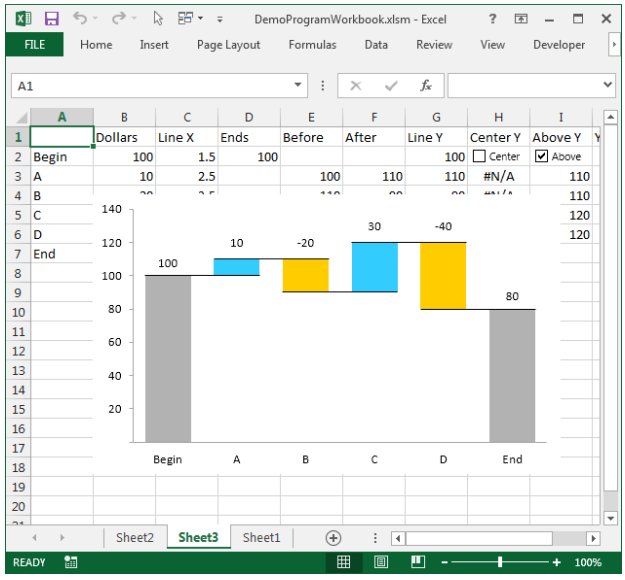
All of the sample code presented above, early or late bound, function or sub, produces the same output. Given the following data in the identified range of the active sheet (below left), you can press Alt+F8 to run your code from the Macro dialog (below right)





Output of these procedures is shown in the Excel window below left, except for the functions that assign a variable to the results of the Peltier Tech procedure and subsequently modify this result, shown below right.





Syntax of the Peltier Tech Public Procedures

The Peltier Tech procedures are descriptively named after the charts they produce or other functions they provide. The arguments are also descriptively named, and in most cases can be directly matched to elements in the dialog. Named constants are shown below the functions where applicable. You are encouraged to familiarize yourself with the dialogs, so you understand how the following arguments work.

Waterfall Chart

```
Public Function PeltierTech_WaterfallChart(ChartData As Range, LabelsInFirstRow As Boolean, _
    DataLabelPosition As DataLabelPosition, GapWidth As Long, Optional NewSheet As Boolean = True) _
    As Chart

Public Enum DataLabelPosition
  DataLabelPositionNone = 0
  DataLabelPositionCenter = 1
  DataLabelPositionAbove = 2
End Enum
```

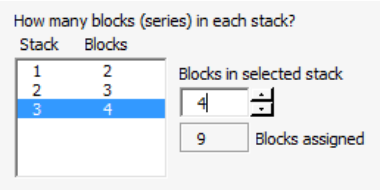
Stacked Waterfall Chart

```
Public Function PeltierTech_StackedWaterfallChart(ChartData As Range, LabelsInFirstRow As Boolean, _
    GapWidth As Long, Optional NewSheet As Boolean = True) As Chart
```

Cluster Stack Chart

```
Public Function PeltierTech_ClusterStackChart(ChartData As Range, _
    Optional ChartStyle As ClusterStackChartStyle = ClusterStackChartStyleColumn, _
    Optional ChartColorScheme As ClusterStackColorScheme = ClusterStackColorDefault, _
    Optional DataLayouts As Variant) As Chart

Public Enum ClusterStackChartStyle
  ClusterStackChartStyleColumn = 0
  ClusterStackChartStyleBar = 1
End Enum

Public Enum ClusterStackColorScheme
  ClusterStackColorDefault = 0
  ClusterStackColorAlongStacks = 1
  ClusterStackColorAcrossStacks = 2
End Enum
```

DataLayouts is derived from information in the Cluster Stack Chart dialog. It is a variant that contains a simple one-dimensional array of the blocks in the second (Blocks) column of the blocks per stack listbox of the dialog. In the example at right, DataLayouts is {2,3,4}.



Marimekko Chart

```
Public Function PeltierTech_MarimekkoChart(ChartData As Range, SegmentOrientation As XlRowCol) As Chart
```

Cascade Chart

```
Public Function PeltierTech_CascadeChart(ChartData As Range, CascadeStyle As CascadeChartStyle) As Chart

Public Enum CascadeChartStyle
  CascadeChartStackedArea = 0
  CascadeChartStackedLine = 1
  CascadeChartUnstackedLine = 2
End Enum
```

Box Plot

```
Public Function PeltierTech_BoxWhiskerChart(ChartData As Range, DataOrientation As XlRowCol, _
    LabelsInFirstRowCol As Boolean, VerticalBars As Boolean, _
    BoxWhiskerChartStyle As BoxWhiskerChartStyle) As Chart

Public Enum BoxWhiskerChartStyle
  BoxWhiskerStandardStyle = 0
  BoxWhiskerFourBoxStyle = 1
  BoxWhiskerOutlierStyle = 2
End Enum
```

Dot Plot

```
Public Function PeltierTech_DotPlot(ChartData As Range, LabelsInFirstRow As Boolean, _
    HGridLines As Boolean, VGridLines As Boolean, ConnectingLines As Boolean) As Chart
```

Quick XY Chart

```
Public Function PeltierTech_QuickXYChart(ChartData As Range, DataLayout As QuickXYChartDataLayout, _
    DataOrientation As XlRowCol, FirstRowColLabels As Boolean) As Chart

Public Enum QuickXYChartDataLayout
  DataLayoutXYY
  DataLayoutXXY
  DataLayoutYXX
  DataLayoutYYX
  DataLayoutXYXY
  DataLayoutYXYX
  DataLayoutXXYY
  DataLayoutYYXX
  DataLayoutXonY
  DataLayoutYonX
End Enum
```

In Case of Problems

If you have any problems installing or using the Peltier Tech Utility, contact Peltier Tech.

Jon Peltier
Peltier Technical Services, Inc.
http://peltiertech.com
jon@peltiertech.com
+1-774-275-0064