

Automating Microsoft Excel with VisualAPL

One way to incorporate the utility of Microsoft Excel in an application system is to 'automate' it with VisualAPL. Microsoft Excel is an ActiveX server, so it can be accessed from any .Net language including VisualAPL.

Because Microsoft Excel is based on Win32, the previous generation Microsoft Windows application programming interface (API), a .Net solution incorporating Microsoft Excel will not be "fully-managed".

This document provides a brief overview of the syntax that can be used to manipulate Microsoft Excel from VisualAPL. It is not meant to be a comprehensive survey of these techniques. Ultimately the programmer must study the Microsoft Excel object model in order to add significant value to an application system by automating Microsoft Excel.

For more details about the Excel object model and how it can be automated from a .Net language, see [http://msdn.microsoft.com/en-us/library/syyd7czh\(v=VS.80\).aspx](http://msdn.microsoft.com/en-us/library/syyd7czh(v=VS.80).aspx).

For a relatively simple example of an application system which combines VisualAPL, Microsoft Excel, Windows Presentation Foundation and C# see <http://forum.apl2000.com/viewtopic.php?t=478>.

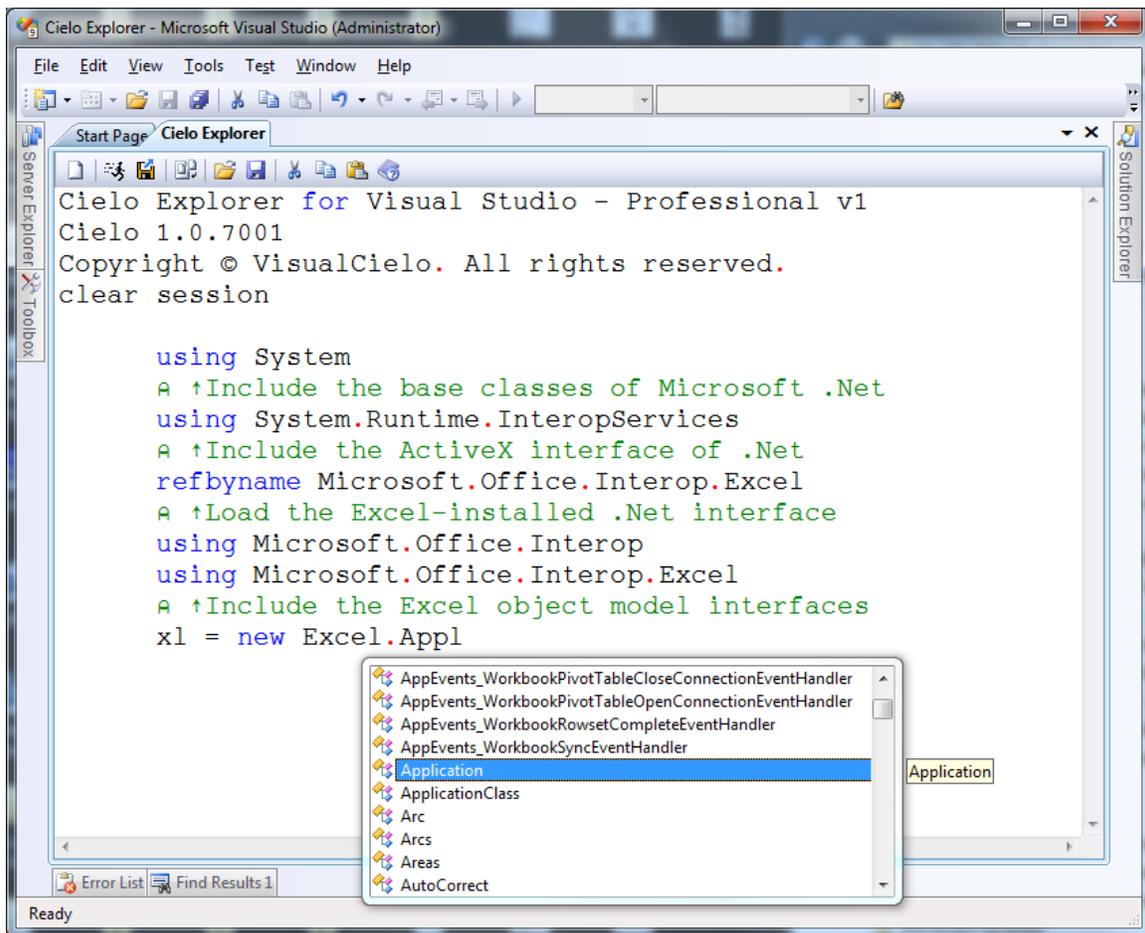
The remainder of this document uses the APLNext VisualAPL Cielo Explorer interactive session in Visual Studio 2008 to illustrate the automation of Microsoft Excel. Remember that any VisualAPL statements perfected in the Cielo Explorer may be incorporated into a compiled VisualAPL .Net assembly or a Cielo Explorer script.

Using the Cielo Explorer makes it very easy to try various VisualAPL statements and see the results immediately. The Visual Studio 'debugger' is fully active within the Cielo Explorer, so a source code statement with errors will be immediately identified.

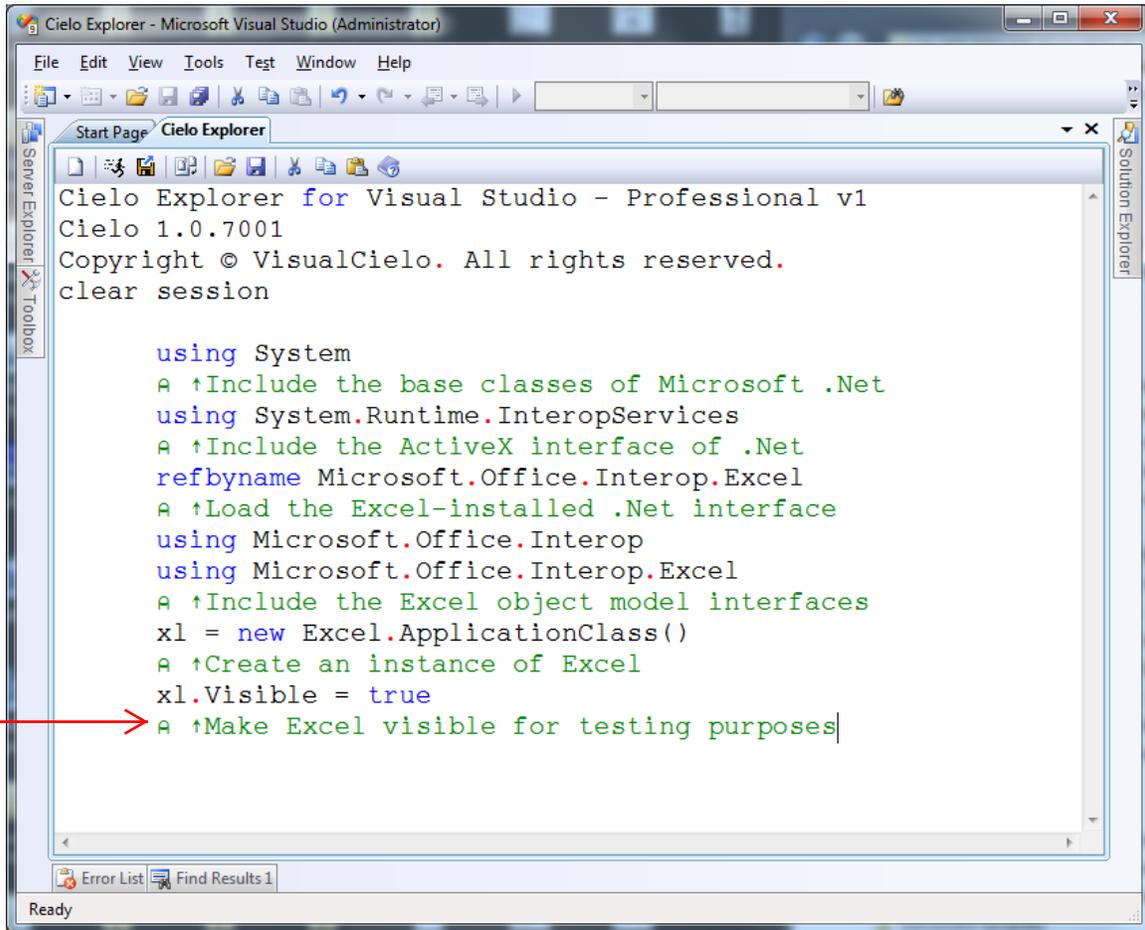
The screen captures of the Cielo Explorer in action have incorporated VisualAPL comments which explain the operations involved.

Open Visual Studio 2008 and make the Cielo Explorer window visible. Enter the VisualAPL statements indicated below to automate Microsoft Excel. These statements will be successful only if Microsoft Excel has been installed on the programmer's workstation.

Visual Studio 'Intellisense' context-sensitive documentation is available in the Cielo Explorer, however for certain Microsoft Excel objects, Visual Studio is not able to retrieve complete Intellisense information. This is why it is important for the programmer to study the Microsoft Excel object model to know what methods, properties and events are available.



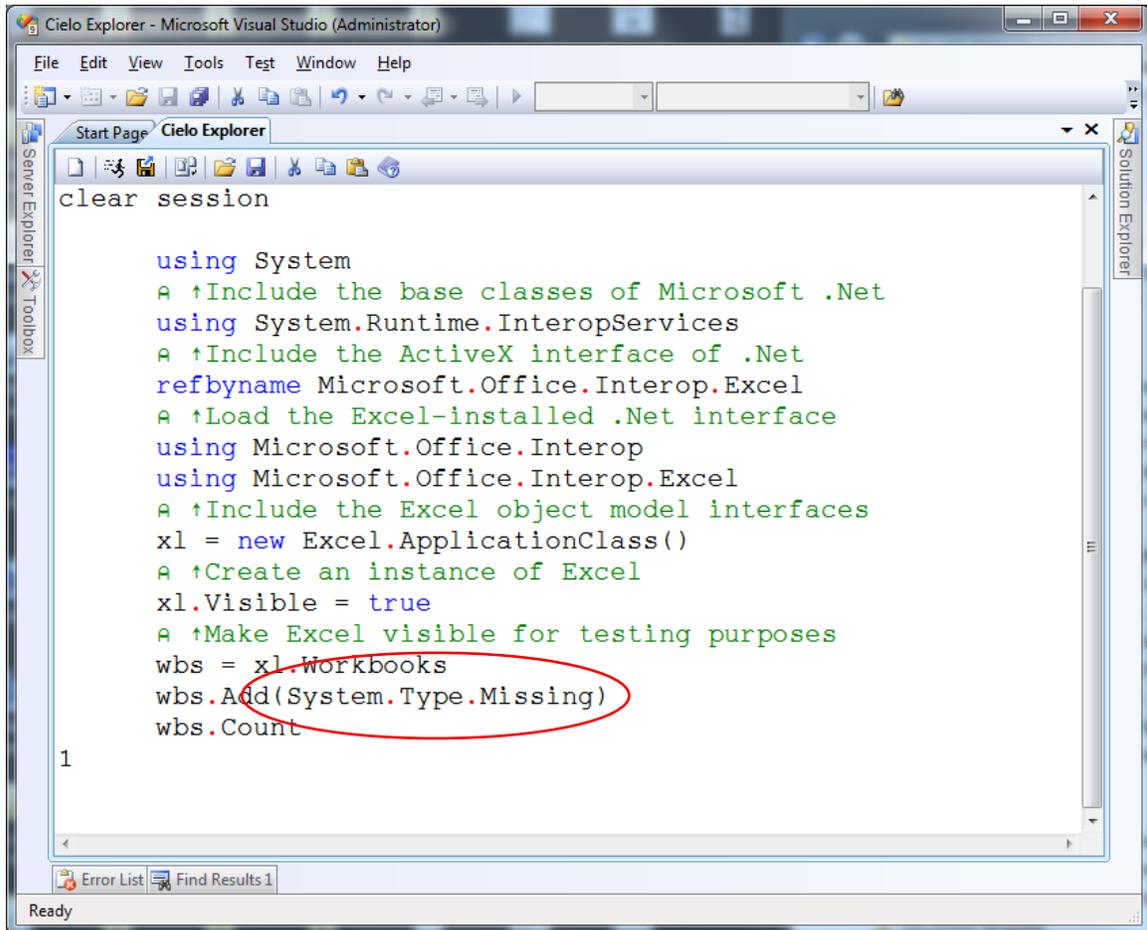
When testing an application system that is automating Excel, it is generally beneficial to make the instance of Excel visible.



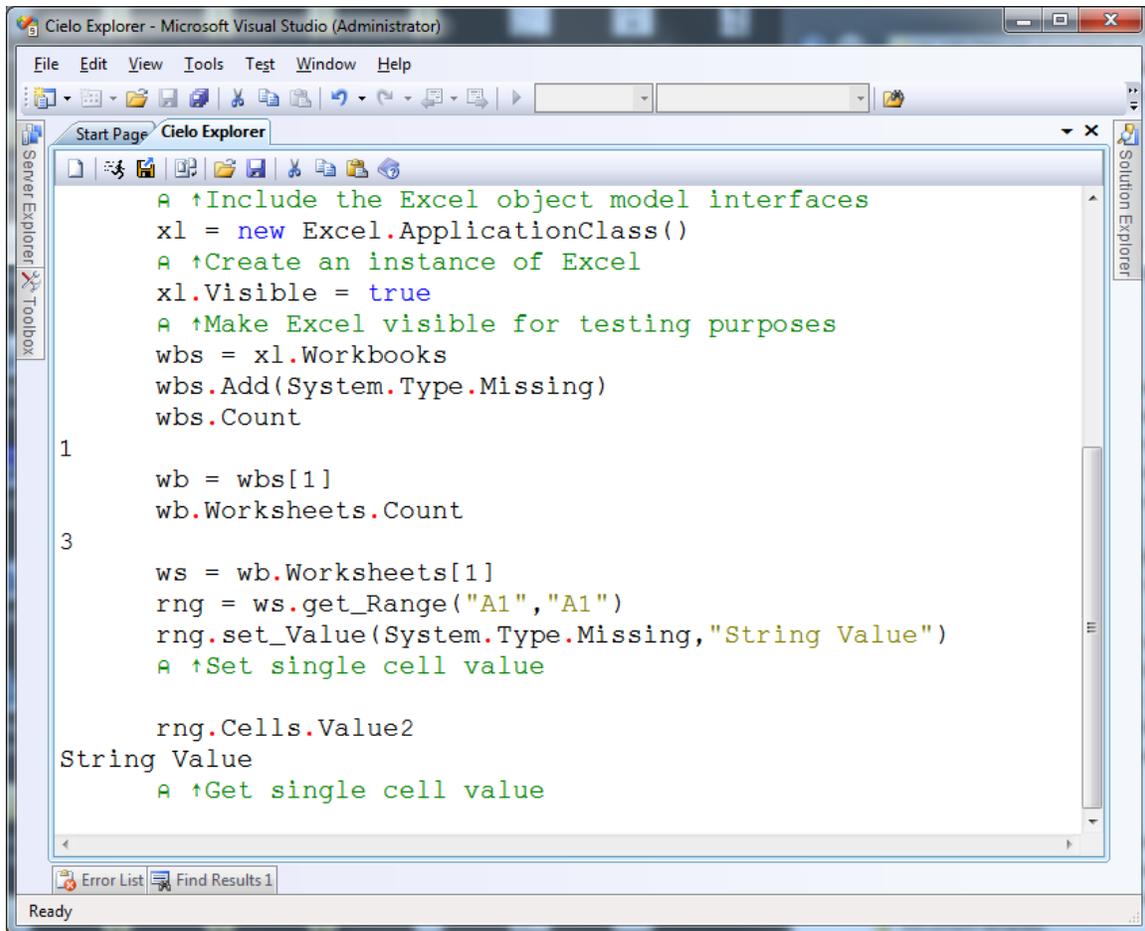
```
Cielo Explorer for Visual Studio - Professional v1
Cielo 1.0.7001
Copyright © VisualCielo. All rights reserved.
clear session

using System
A ↑Include the base classes of Microsoft .Net
using System.Runtime.InteropServices
A ↑Include the ActiveX interface of .Net
refbyname Microsoft.Office.Interop.Excel
A ↑Load the Excel-installed .Net interface
using Microsoft.Office.Interop
using Microsoft.Office.Interop.Excel
A ↑Include the Excel object model interfaces
xl = new Excel.ApplicationClass()
A ↑Create an instance of Excel
xl.Visible = true
A ↑Make Excel visible for testing purposes|
```

The 'System.Type.Missing' object is convenient when using certain Excel methods which have numerous arguments, many of which need not be specified explicitly. Before using System.Type.Missing, check the Excel object model documentation to assure that these additional arguments are actually not critical to the application system automating Excel.



The Excel object model incorporates many subordinate objects such as indexable collections for workbooks, worksheets and cells as well as properties like Count and methods such as Add(), set_Value() and get_Value().



The screenshot shows the Microsoft Visual Studio (Administrator) interface with a C# code file named 'Cielo Explorer'. The code demonstrates how to interact with the Excel object model. It includes comments in green and code in black. The code starts by creating an instance of the Excel.ApplicationClass, making it visible, and then accessing the Workbooks collection. It adds a new workbook, accesses the first workbook, and then the first worksheet. It then sets the value of cell A1 to 'String Value' and finally accesses the Value2 property of the cell.

```

A ↑Include the Excel object model interfaces
xl = new Excel.ApplicationClass()
A ↑Create an instance of Excel
xl.Visible = true
A ↑Make Excel visible for testing purposes
wbs = xl.Workbooks
wbs.Add(System.Type.Missing)
wbs.Count

1
wb = wbs[1]
wb.Worksheets.Count

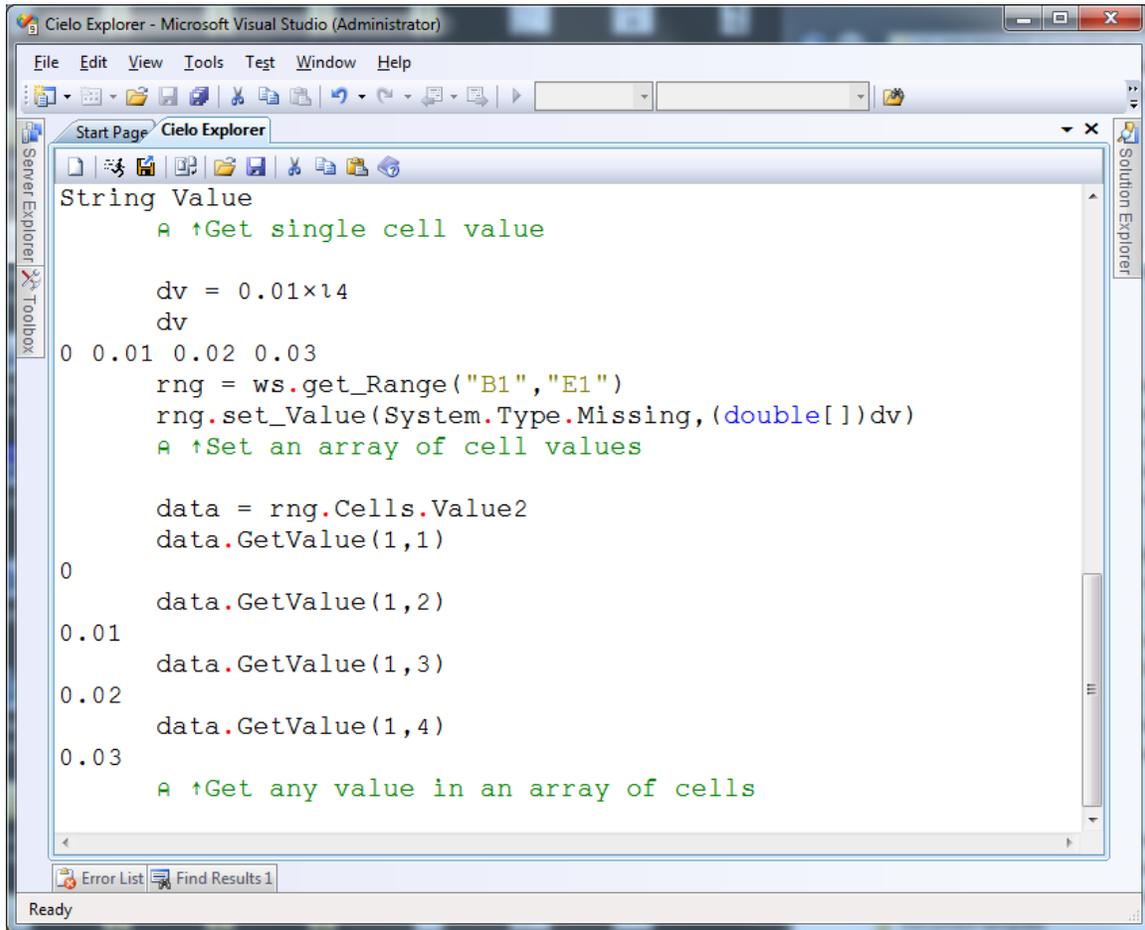
3
ws = wb.Worksheets[1]
rng = ws.get_Range("A1","A1")
rng.set_Value(System.Type.Missing,"String Value")
A ↑Set single cell value

rng.Cells.Value2
String Value
A ↑Get single cell value

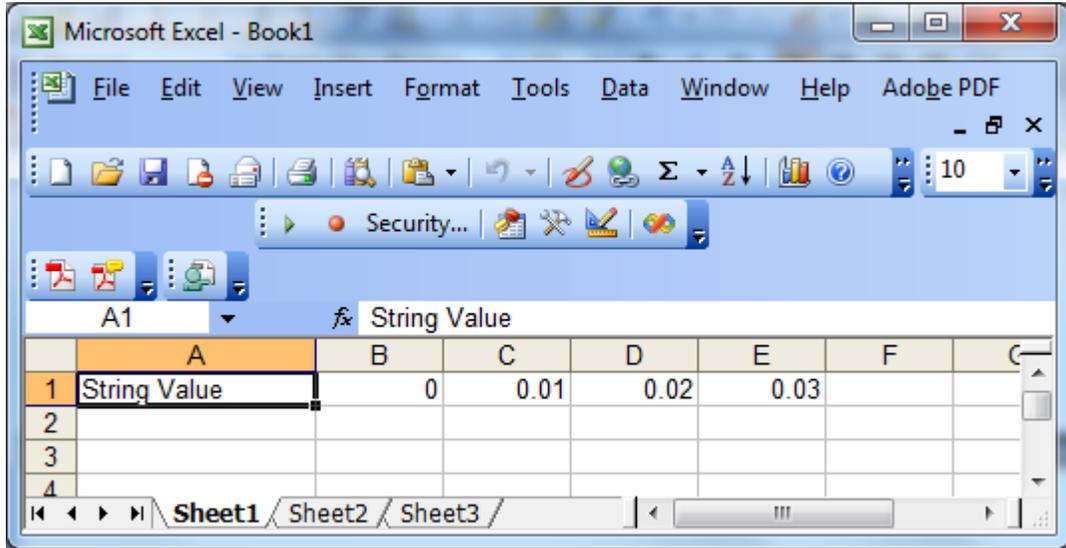
```

At the bottom of the window, there is an 'Error List' and 'Find Results 1' pane, and the status bar shows 'Ready'.

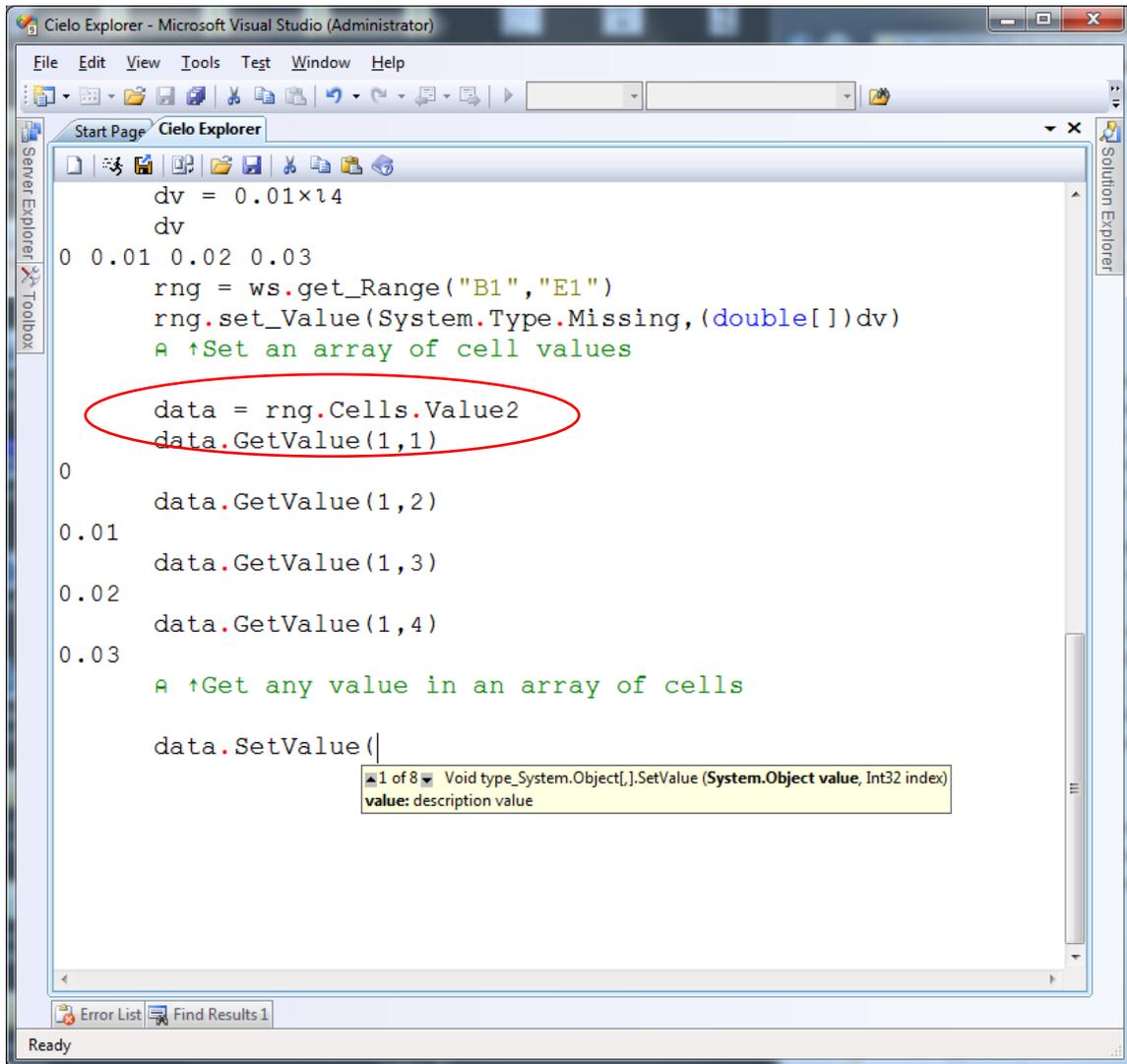
Excel ranges may reference individual cells or arrays of cells supporting some array-based operations.



Here is the Excel application after the above Excel automation operations have been completed.



It is possible to abstract the data in an Excel range into a 'data' object.



The data object associated with an Excel range object can be manipulated by getting or setting values within it. The data object can then be used to update the Excel range object.

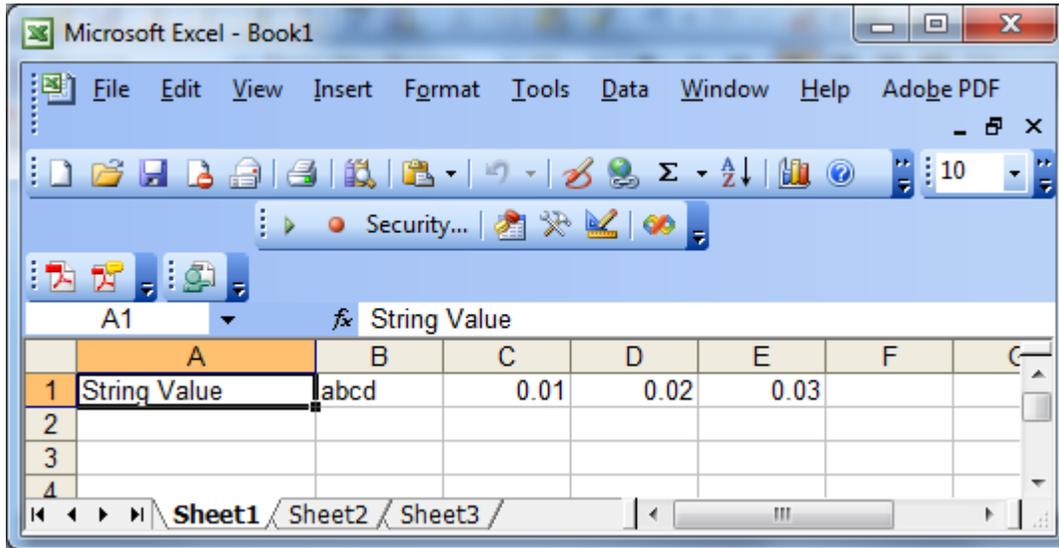
```
String Value
  A ↑Get single cell value

  dv = 0.01×14
  dv
0 0.01 0.02 0.03
  rng = ws.get_Range("B1","E1")
  rng.set_Value(System.Type.Missing,(double[])dv)
  A ↑Set an array of cell values

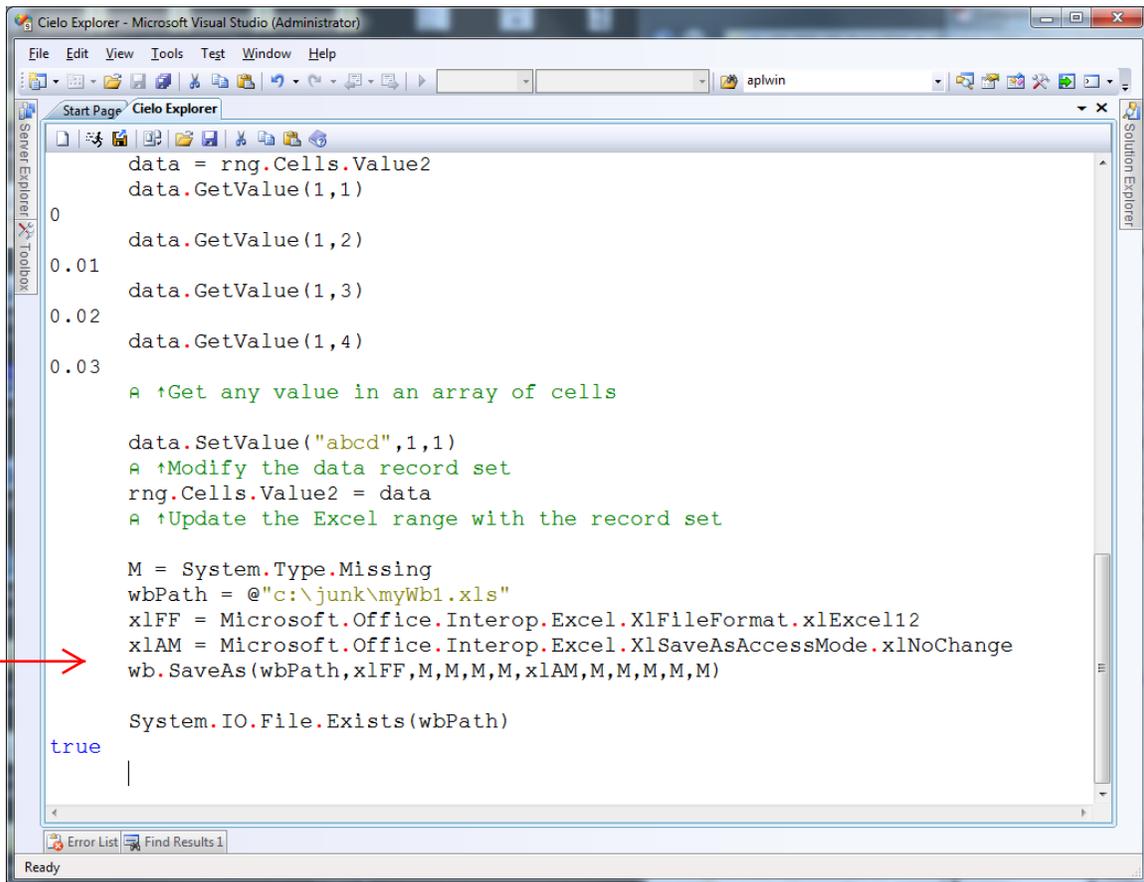
  data = rng.Cells.Value2
  data.GetValue(1,1)
0
  data.GetValue(1,2)
0.01
  data.GetValue(1,3)
0.02
  data.GetValue(1,4)
0.03
  A ↑Get any value in an array of cells

  data.SetValue("abcd",1,1)
  A ↑Modify the data record set
  rng.Cells.Value2 = data
  A ↑Update the Excel range with the record set
```

Here is the Excel application after the data object updated the range object.



The Excel Workbook.SaveAs() method has a complex argument structure. Generally only certain of these arguments need to be explicitly specified.



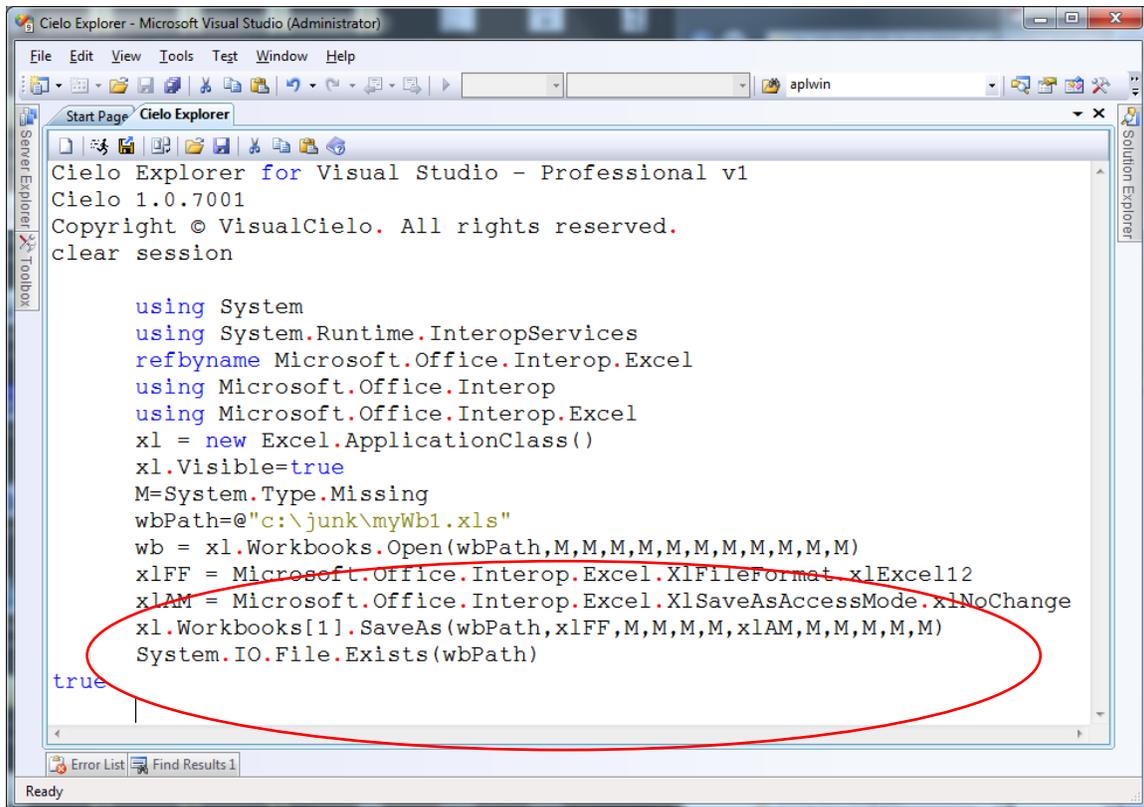
```
data = rng.Cells.Value2
data.GetValue(1,1)
0
data.GetValue(1,2)
0.01
data.GetValue(1,3)
0.02
data.GetValue(1,4)
0.03
A †Get any value in an array of cells

data.SetValue("abcd",1,1)
A †Modify the data record set
rng.Cells.Value2 = data
A †Update the Excel range with the record set

M = System.Type.Missing
wbPath = @"c:\junk\myWb1.xls"
xlFF = Microsoft.Office.Interop.Excel.XlFileFormat.xlExcel12
xlAM = Microsoft.Office.Interop.Excel.XlSaveAsAccessMode.xlNoChange
wb.SaveAs(wbPath,xlFF,M,M,M,M,xlAM,M,M,M,M,M)

System.IO.File.Exists(wbPath)
true
|
```


The Excel object model provides alternate syntax for many statements. Here is an example of saving the current workbook using indexing on the Workbooks collection.



```
Cielo Explorer for Visual Studio - Professional v1
Cielo 1.0.7001
Copyright © VisualCielo. All rights reserved.
clear session

using System
using System.Runtime.InteropServices
refbyname Microsoft.Office.Interop.Excel
using Microsoft.Office.Interop
using Microsoft.Office.Interop.Excel
xl = new Excel.ApplicationClass()
xl.Visible=true
M=System.Type.Missing
wbPath=@"c:\junk\myWb1.xls"
wb = xl.Workbooks.Open(wbPath,M,M,M,M,M,M,M,M,M,M)
xlFF = Microsoft.Office.Interop.Excel.XlFileFormat xlExcel12
xlAM = Microsoft.Office.Interop.Excel.XlSaveAsAccessMode.xlNoChange
xl.Workbooks[1].SaveAs(wbPath,xlFF,M,M,M,M,xlAM,M,M,M,M)
System.IO.File.Exists(wbPath)

true
```